



**Protocol API**  
**PROFINET IO RT/IRT Device**

**V3.10.0**

**Hilscher Gesellschaft für Systemautomation mbH**  
**[www.hilscher.com](http://www.hilscher.com)**

DOC111110API15EN | Revision 15 | English | 2016-03 | Released | Public

# Table of Contents

<b>1</b>	<b>Introduction.....</b>	<b>7</b>
1.1	About this Document.....	7
1.1.1	List of Revisions.....	7
1.2	Functional Overview.....	8
1.3	System Requirements.....	8
1.4	Intended Audience.....	8
1.5	Specifications for Stack V3.10.0.....	9
1.5.1	Supported Protocols.....	9
1.5.2	Technical Data.....	9
1.5.3	Limitations.....	11
1.6	Terms, Abbreviations and Definitions.....	12
1.7	References to Documents.....	13
1.8	Legal Notes.....	14
1.8.1	Copyright.....	14
1.8.2	Important Notes.....	14
1.8.3	Exclusion of Liability.....	15
1.8.4	Export.....	15
1.8.5	Third Party Software Licenses.....	16
1.8.5.1	SNMP.....	16
<b>2</b>	<b>Fundamentals.....</b>	<b>17</b>
2.1	General Access Mechanisms on netX Systems.....	17
2.2	Accessing the Protocol Stack by Programming the Stacks PNS-IF Task's Queue.....	18
2.2.1	Getting the Receiver Task Handle of the Process Queue.....	18
2.3	Accessing the Protocol Stack via the Dual Port Memory Interface.....	19
2.3.1	Communication via Mailboxes.....	19
2.3.2	Using Source and Destination Variables correctly.....	19
2.3.2.1	How to use ulDest for Addressing rcX and the netX Protocol Stack by the System and Channel Mailbox.....	19
2.3.2.2	How to use ulSrc and ulSrcId.....	20
2.3.2.3	How to Route rcX Packets.....	21
2.3.3	Obtaining useful Information about the Communication Channel.....	22
2.4	Packet Types.....	24
2.4.1	Timeout for Response Packets.....	25
<b>3</b>	<b>Dual-Port Memory.....</b>	<b>26</b>
3.1	Cyclic Data (Input/Output Data).....	26
3.1.1	Input Process Data.....	27
3.1.2	Output Process Data.....	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 and 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
3.2.7	Packet fragmentation.....	33
3.3	Status.....	37
3.3.1	Common Status.....	37
3.3.1.1	All Implementations.....	37
3.3.1.2	Master Implementation.....	41
3.3.1.3	Slave Implementation.....	41
3.3.2	Extended Status.....	42
3.4	Control Block.....	43
<b>4</b>	<b>Getting Started.....</b>	<b>44</b>
4.1	Structure of the PROFINET Device Stack.....	44
4.2	Naming Conventions.....	47
4.3	Overview about Essential Functionality.....	47
4.4	Event Mechanism.....	48
4.5	Device Handle.....	49
<b>5</b>	<b>Exchanging Cyclic Data.....</b>	<b>50</b>
5.1	General Concepts.....	50
5.2	Behavior regarding IO data and IOPS.....	51

5.3	Exchanging cyclic Data using Callback Interface .....	52
5.3.1	Overview of the Callback Interface .....	52
5.3.2	Callback Functions .....	52
5.3.2.1	UpdateConsumerImage Callback .....	53
5.3.2.2	UpdateProviderImage Callback .....	54
5.3.2.3	Event Handler Callback .....	55
<b>6</b>	<b>Status Information .....</b>	<b>56</b>
6.1	Communication State .....	56
6.1.1	Implementation from V3.10 .....	56
6.1.2	Legacy Implementation (V3.9 and earlier) .....	56
<b>7</b>	<b>Packet Interface .....</b>	<b>57</b>
7.1	Configuring the IO-Device Stack .....	57
7.1.1	Cyclic Process Data Image .....	58
7.1.2	Configuration of Process Data Images .....	58
7.1.3	Configuration of the Submodules .....	61
7.1.4	Configuring the PROFINET IO Device Stack .....	61
7.1.4.1	Remark on Reconfiguration .....	62
7.1.5	Set Configuration Service .....	63
7.1.5.1	Set Configuration Request .....	63
7.1.5.2	Set Configuration Confirmation .....	71
7.1.5.3	Behavior when receiving a Set Configuration Command .....	72
7.1.6	Register Application Service .....	73
7.1.6.1	Register Application Request .....	74
7.1.6.2	Register Application Confirmation .....	74
7.1.6.3	Register Application for Selective Indications Only .....	74
7.1.7	Unregister Application Service .....	75
7.1.7.1	Unregister Application Request .....	75
7.1.7.2	Unregister Application Confirmation .....	75
7.1.8	Register Fatal Error Callback Service .....	76
7.1.8.1	Register Fatal Error Callback Request .....	76
7.1.8.2	Register Fatal Error Callback Confirmation .....	77
7.1.9	Unregister Fatal Error Callback Service .....	79
7.1.9.1	Unregister Fatal Error Callback Request .....	79
7.1.9.2	Unregister Fatal Error Callback Confirmation .....	80
7.1.10	Set Port MAC Address Service .....	81
7.1.10.1	Set Port MAC Address Request .....	81
7.1.10.2	Set Port MAC Address Confirmation .....	82
7.1.11	Set OEM Parameters Service .....	84
7.1.11.1	Set OEM Parameters Request .....	84
7.1.11.2	Set OEM Parameters Confirmation .....	89
7.1.12	Load Remanent Data Service .....	90
7.1.12.1	Load Remanent Data Request .....	90
7.1.12.2	Load Remanent Data Confirmation .....	92
7.1.13	Configuration Delete Service .....	93
7.1.13.1	Configuration Delete Request .....	93
7.1.13.2	Configuration Delete Confirmation .....	93
7.1.14	Set IO-Image Service .....	94
7.1.14.1	Set IO-Image Request .....	94
7.1.14.2	Set IO-Image Confirmation .....	96
7.1.15	Set IOXS Config Service .....	98
7.1.15.1	Set IOXS Config Request .....	98
7.1.15.2	Set IOXS Config Confirmation .....	100
7.1.16	Configure Signal Service .....	101
7.1.16.1	Configure Signal Request .....	101
7.1.16.2	Configure Signal Confirmation .....	104
7.1.16.3	Example: Configure Signal Request packet .....	105
7.2	Connection Establishment .....	106
7.2.1	AR Check Service .....	112
7.2.1.1	AR Check Indication .....	112
7.2.1.2	AR Check Response .....	114
7.2.2	Check Indication Service .....	115
7.2.2.1	Check Indication .....	117
7.2.2.2	Check Response .....	119
7.2.3	Connect Request Done Service .....	121
7.2.3.1	Connect Request Done Indication .....	121
7.2.3.2	Connect Request Done Response .....	122
7.2.4	Parameter End Service .....	123

7.2.4.1	Parameter End Indication .....	123
7.2.4.2	Parameter End Response .....	124
7.2.5	Application Ready Service.....	126
7.2.5.1	Application Ready Request .....	126
7.2.5.2	Application Ready Confirmation .....	127
7.2.6	AR InData Service .....	129
7.2.6.1	AR InData Indication.....	129
7.2.6.2	AR InData Response .....	130
7.2.7	Store Remanent Data Service .....	131
7.2.7.1	Store Remanent Data Indication.....	131
7.2.7.2	Store Remanent Data Response .....	132
7.3	Acyclic Events indicated by the Stack.....	133
7.3.1	Read Record Service .....	134
7.3.1.1	Read Record Indication .....	134
7.3.1.2	Read Record Response .....	136
7.3.2	Write Record Service.....	138
7.3.2.1	Write Record Indication .....	138
7.3.2.2	Write Record Response.....	139
7.3.3	AR Abort Indication service .....	142
7.3.3.1	AR Abort Indication Indication .....	142
7.3.3.2	AR Abort Indication Response.....	144
7.3.4	Save Station Name Service.....	145
7.3.4.1	Save Station Name Indication .....	146
7.3.4.2	Save Station Name Response.....	147
7.3.5	Save IP Address Service.....	148
7.3.5.1	Save IP Address Indication.....	149
7.3.5.2	Save IP Address Response.....	150
7.3.6	Start LED Blinking Service .....	151
7.3.6.1	Start LED Blinking Indication .....	151
7.3.6.2	Start LED Blinking Response .....	152
7.3.7	Stop LED Blinking Service.....	153
7.3.7.1	Stop LED Blinking Indication .....	153
7.3.7.2	Stop LED Blinking Response.....	154
7.3.8	Reset Factory Settings Service .....	155
7.3.8.1	Reset Factory Settings Indication .....	157
7.3.8.2	Reset Factory Settings Response .....	159
7.3.9	APDU Status Changed Service .....	160
7.3.9.1	APDU Status Changed Indication.....	160
7.3.9.2	APDU Status Changed Response.....	161
7.3.10	Alarm Indication Service.....	163
7.3.10.1	Alarm Indication .....	163
7.3.10.2	Alarm Indication Response.....	165
7.3.11	Release Request Indication Service.....	166
7.3.11.1	Release Request Indication.....	166
7.3.11.2	Release Request Indication Response.....	167
7.3.12	Link Status Changed Service .....	168
7.3.12.1	Link Status Changed Indication .....	168
7.3.12.2	Link Status Changed Response .....	170
7.3.13	Error Indication Service .....	171
7.3.13.1	Error Indication .....	171
7.3.13.2	Error Indication Response .....	172
7.3.14	Read I&M Service.....	173
7.3.14.1	Read I&M Indication .....	173
7.3.14.2	Read I&M Response .....	174
7.3.15	Write I&M Service.....	179
7.3.15.1	Write I&M Indication .....	179
7.3.15.2	Write I&M Response.....	180
7.3.16	Parameterization Speedup Support .....	182
7.3.16.1	Parameterization Speedup Support Indication .....	182
7.3.16.2	Parameterization Speedup Supported Response.....	183
7.3.17	Event Indication Service .....	185
7.3.17.1	Event Indication .....	185
7.3.17.2	Event Indication Response .....	186
7.4	Acyclic Events requested by the Application .....	187
7.4.1	Get Diagnosis Service .....	189
7.4.1.1	Get Diagnosis Request.....	189
7.4.1.2	Get Diagnosis Confirmation.....	190
7.4.2	Get XMAC (EDD) Diagnosis Service.....	194
7.4.2.1	Get XMAC (EDD) Diagnosis Request .....	194

7.4.2.2	Get XMAC (EDD) Diagnosis Confirmation.....	194
7.4.3	Process Alarm Service .....	197
7.4.3.1	Process Alarm Request.....	197
7.4.3.2	Process Alarm Confirmation .....	199
7.4.4	Diagnosis Alarm Service .....	200
7.4.4.1	Diagnosis Alarm Request .....	200
7.4.4.2	Diagnosis Alarm Confirmation .....	201
7.4.5	Return of Submodule Alarm Service .....	203
7.4.5.1	Return of Submodule Alarm Request.....	203
7.4.5.2	Return of Submodule Alarm Confirmation .....	204
7.4.6	AR Abort Request Service.....	206
7.4.6.1	AR Abort Request.....	206
7.4.6.2	AR Abort Request Confirmation .....	207
7.4.7	Plug Module Service.....	208
7.4.7.1	Plug Module Request .....	208
7.4.7.2	Plug Module Confirmation .....	210
7.4.8	Plug Submodule Service .....	211
7.4.8.1	Plug Submodule Request.....	212
7.4.8.2	Plug Submodule Confirmation .....	214
7.4.8.3	Extended Plug Submodule Request.....	217
7.4.8.4	Extended Plug Submodule Confirmation .....	219
7.4.9	Pull Module Service.....	221
7.4.9.1	Pull Module Request .....	221
7.4.9.2	Pull Module Confirmation.....	222
7.4.10	Pull Submodule Service .....	224
7.4.10.1	Pull Submodule Request .....	224
7.4.10.2	Pull Submodule Confirmation .....	225
7.4.11	Get Station Name Service .....	227
7.4.11.1	Get Station Name Request.....	227
7.4.11.2	Get Station Name Confirmation .....	228
7.4.12	Get IP Address Service .....	229
7.4.12.1	Get IP Address Request.....	229
7.4.12.2	Get IP Address Confirmation .....	230
7.4.13	Add Channel Diagnosis Service .....	231
7.4.13.1	Add Channel Diagnosis Request.....	231
7.4.13.2	Add Channel Diagnosis Confirmation.....	235
7.4.14	Add Extended Channel Diagnosis Service .....	237
7.4.14.1	Add Extended Channel Diagnosis Request.....	237
7.4.14.2	Add Extended Channel Diagnosis Confirmation.....	239
7.4.15	Add Generic Diagnosis Service .....	241
7.4.15.1	Add Generic Channel Diagnosis Request .....	241
7.4.15.2	Add Generic Channel Diagnosis Confirmation .....	242
7.4.16	Remove Diagnosis Service .....	244
7.4.16.1	Remove Diagnosis Request .....	244
7.4.16.2	Remove Diagnosis Confirmation .....	245
7.4.17	Get Submodule Configuration Service .....	246
7.4.18	Set Submodule State Service.....	248
7.4.18.1	Set Submodule State Request .....	249
7.4.18.2	Set Submodule State Confirmation .....	251
7.4.19	Get Parameter Service.....	253
7.4.19.1	Get Parameter Service .....	253
7.4.19.2	Get Parameter Confirmation.....	255
<b>8</b>	<b>Special Topics .....</b>	<b>258</b>
8.1	Behavior under special situations .....	258
8.1.1	Sequence of configuration evaluation.....	258
8.1.2	Configuration Lock.....	258
8.1.3	Setting Parameters by means of DCP.....	258
8.2	Multiple ARs .....	259
8.2.1	Ownership .....	259
8.2.2	Possibilities and Limitations for the Feature Shared Device.....	260
8.2.3	Ethernet MAC Addresses .....	261
8.3	Usage of Linkable Object Module .....	262
8.3.1	Config.c.....	262
8.3.1.1	Hardware Resources.....	262
8.3.1.2	Disable XMAC3 .....	263
8.3.1.3	Systime Unit .....	264
8.3.1.4	Static Task List .....	265
8.3.2	PNS_StackInit().....	267

8.3.3	Task Priorities.....	271
8.3.4	Fiber optic device .....	272
8.3.4.1	Fiber optic configuration .....	272
8.3.4.2	Medium Attachment Unit for Fiber Optic.....	278
8.3.5	PROFINET Netload Requirements.....	279
8.4	PROFINET Certification .....	281
8.4.1	RT Tests (Conformance class A, B and C).....	281
8.4.1.1	Description.....	281
8.4.1.2	General Requirements for RT Tests .....	281
8.4.1.3	Common checks before Certification (GSDML) .....	282
8.4.1.4	Basic Application Behavior .....	282
8.4.2	IRT Tests (Conformance class C only).....	283
8.4.2.1	Description.....	283
8.4.2.2	General Requirements for IRT Tests .....	283
8.4.2.3	Hardware Requirements for IRT Tests .....	283
8.4.2.4	Software Requirements for IRT Tests .....	283
8.4.2.5	GSDML Requirements for IRT Tests.....	283
8.4.3	Network Load Tests.....	284
8.4.3.1	Description.....	284
8.4.3.2	Requirements to the Application.....	284
8.4.4	How to handle I&M Data.....	285
8.4.4.1	Overview.....	285
8.4.4.2	Structure and access paths of I&M objects.....	286
8.4.4.3	Usage of I&M with Hilscher PROFINET Protocol .....	287
8.5	Second DPM channel – Ethernet Interface.....	288
8.6	Isochronous Application .....	290
8.7	PROFINET Status Code .....	292
8.7.1	The ErrorCode Field.....	293
8.7.2	The ErrorDecode Field .....	293
8.7.3	The ErrorCode1 and ErrorCode2 Fields.....	293
8.7.3.1	ErrorCode1 and ErrorCode2 for ErrorDecode = PNIO	294
8.7.4	ErrorCode1 and ErrorCode2 for ErrorDecode = PNIO .....	295
8.7.5	ErrorCode1 and ErrorCode2 for ErrorDecode is Manufacturer Specific.....	300
9	<b>Status/Error Codes Overview.....</b>	<b>301</b>
9.1	General Errors.....	301
9.2	Status/Error Codes for CMCTL Task .....	312
9.2.1	CMCTL-Task Diagnosis-Codes.....	316
9.3	Status/Error Codes for CM-Dev Task .....	317
9.3.1	CM-Dev-Task Diagnosis-Codes .....	322
9.4	Status/Error Codes for EDD Task .....	323
9.4.1	EDD-Task Diagnosis-Codes.....	323
9.5	Status/Error Codes for ACP Task .....	324
9.5.1	ACP-Task Diagnosis-Codes.....	327
9.6	Status/Error Codes for DCP Task .....	328
9.6.1	DCP-Task Diagnosis-Codes.....	331
9.7	Status/Error Codes for MGT Task.....	332
9.7.1	MGT-Task Diagnosis-Codes .....	335
9.8	Status/Error Codes for FODMI-Task.....	335
9.9	Status/Error Codes for RPC-Task.....	335
9.9.1	RPC -Task Diagnosis-Codes.....	339
10	<b>Appendix .....</b>	<b>340</b>
10.1	List of Tables .....	340
10.2	List of Figures.....	343
10.3	Contacts .....	344

# 1 Introduction

## 1.1 About this Document

This manual describes the user interface of the PROFINET IO-Device implementation on netX. The aim of this manual is to support the integration of devices based on the netX chip into own applications based on direct access to protocol stack.

### 1.1.1 List of Revisions

Rev	Date	Name	Revisions
13	2015-06-03	AM, AR	Firmware/stack version V3.8.0.0 Section <i>Packet fragmentation</i> added. Section <i>Read Record Service</i> and <i>Write Record Service</i> updated for LFW targets. Section <i>Reset Factory Settings Indication</i> : Additional description of the parameter value PNS_IF_RESET_FACTORY_SETTINGS_IF_COMMUNICATION. Section <i>How to handle I&amp;M Data</i> added. Section <i>Second DPM channel – Ethernet Interface</i> added.
14	2015-08-27	AR, BM, HH	Firmware/stack version V3.9.0.0 Limitation regarding TCP/IP task configuration added. Section <i>Possibilities and Limitations for the Feature Shared Device</i> added.
15	2016-03-07	AM, BM, AR	Firmware/stack version V3.10.0 Section <i>Packet Types</i> and <i>Timeout for Response Packets</i> added. Chapter <i>Status Information</i> added. Update of NXLOM section for startup parameters. Fix wrong information in <i>Store Remanent Data Service</i> . Describe GSDML V2.32 relations to <i>Process Alarm Service</i> . Extended section <i>Possibilities and Limitations for the Feature Shared Device</i> regarding certification requirements. Chapter <i>Isochronous Application</i> added. Section <i>Ethernet MAC Addresses</i> added. Added more details to the chapter <i>Fiber optic device</i> .

Table 1: List of Revisions

## 1.2 Functional Overview

The stack has been written in order to meet the IEC 61158 Type 10 specification. You as a user are getting a capable and a general-purpose Software package with following features:

- Realization of the PROFINET IO-Device context management
- Realization of the PROFINET IO-Device cyclic data exchange
- Realization of the PROFINET IO-Device acyclic data exchange
- Realization of the DCP protocol

## 1.3 System Requirements

This software package has following system requirements to its environment:

- netX-Chip as CPU hardware platform
- operating system rcX
- if Fast Startup shall be used the flash containing the firmware shall be fast enough
- if configuration is done via the packet interface the user application has to have access to a non-volatile memory (e.g. a flash) with a capacity to store a minimum of 8192 Byte

## 1.4 Intended Audience

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

- Knowledge of the programming language C
- Knowledge of the use of the real-time operating system rcX
- Knowledge of the Hilscher Task Layer Reference Model
- Knowledge of the IEC 61158 Type 10 specification



## 1.5 Specifications for Stack V3.10.0

### 1.5.1 Supported Protocols

RTC – Real time Cyclic Protocol, class 1 (unsynchronized), class 3 (synchronized)

RTA – Real time Acyclic Protocol

DCP – Discovery and Configuration Protocol

CL-RPC – Connectionless Remote Procedure Call

LLDP – Link Layer Discovery Protocol

SNMP – Simple Network Management Protocol

MRP – MRP Client is supported

### 1.5.2 Technical Data

Maximum number of total cyclic input data	1440 bytes
Maximum number of total cyclic output data	1440 bytes
Maximum number of submodules	255 submodules per Application Relation at the same time, 1000 submodules can be configured
Multiple Application Relations (AR)	The stack can handle up to 8 IO-ARs, one Supervisor AR and one Supervisor-DA AR at the same time. If executed on netX50 this is limited to 2 IO-ARs instead of 8. See section <i>Possibilities and Limitations for the Feature Shared Device</i> (on page 260) for further details.
Acyclic communication	Read/Write Record - up to 8 KB for NXLFW (using DPM fragmentation) - up to 32 KB for NXLOM
Alarm Types	Process Alarm, Diagnostic Alarm, Return of Submodule Alarm, Plug Alarm (implicit), Pull Alarm (implicit) Other alarm types are not supported
Identification & Maintenance	Reading and writing of I&M1-4. <sup>1</sup> Reading of I&M5 for NXLFW
Topology recognition	LLDP, SNMP V1, MIB2, physical device
Minimum cycle time	1ms for RT_CLASS_1 (all implementations) 1ms for netX50 and RT_CLASS_3 250µs for netX51 and RT_CLASS_3 250µs for netX100/500 and RT_CLASS_3
IRT Support	RT_CLASS_3
MediaRedundancy	MRP client

<sup>1</sup> If the stack is configured to handle I&M by itself (this is the default) the GSDML device description shall be adapted to indicate the support of I&M1-4.

---

Additional supported features	DCP, VLAN- and priority tagging, Shared Device (but only 1 RTC3 AR in total)
Baud rate	100 MBit/s
Data transport layer	Ethernet II, IEEE 802.3
PROFINET IO specification	V2.3, legacy startup of specification v2.2 is supported

**Firmware/stack available for netX**

netX 50	yes
netX 100, netX 500	yes
netX 51	yes

**Configuration**

Configuration is done by sending packets to the stack or by using SYCON.net configuration database.

**Diagnostic**

Some elementary diagnosis information can be read using the service "*Get Diagnosis Service*". This service might be extended in future.

**Cyclic input/output data**

The stack has the ability to convert the byte order of cyclic process data image.

### 1.5.3 Limitations

- RT over UDP not supported
- Multicast communication not supported
- Only one device instance is supported
- DHCP is not supported
- Fast Startup is implemented in the stack. However some additional hardware limitations apply to use it.
- The amount of configured IO-data influences the minimum cycle time that can be reached.
- Only 1 Input-CR and 1 Output-CR per AR are supported
- Using little endian byte order for cyclic process data instead of default big endian byte order may have a negative impact on minimum reachable cycle time
- System Redundancy (SR-AR) and Configuration-in-Run (CiR) are not supported
- Max. 255 submodules can be used simultaneously within one specific Application Relation
- SharedInput is not supported
- MRPD is not supported
- DFP and other HighPerformance-profile related features are not supported
- PDEV functionality is only supported for submodules located in slot 0
- Submodules can not be configured or used by an AR in subslot 0
- DAP and PDEV submodules only supported in slot 0.
- Loadable firmware does not support Flash Device Label
- Only for LOM:
  - The protocol stack cannot be used together with the standard two-port switch of rcX.
  - The protocol stack cannot be used together with the standard MAC of rcX.
  - Thus the protocol stack can exclusively be used with the PROFINET IO-Device switch. Consequently, in any case 3 XC channels are required at the netX 100 and the netX 500.
  - It is not possible to manually configure the integrated Hilscher TCP/IP task with its service "TCPIP\_IP\_CMD\_SET\_CONFIG\_REQ" when using PROFINET IO Device protocol stack. The service will succeed but has no effect.

## 1.6 Terms, Abbreviations and Definitions

Term	Description
AP (-task)	Application (-task) on top of the stack
DCP	Discovery and Basic Configuration Protocol
API	Application Process Identifier
AR	Application Relation
RT_CLASS_1	Real-Time communication Class 1
RT_CLASS_2	Real-Time communication Class 2
RT_CLASS_3	Real-Time communication Class 3
CIR	Configuration in Run
IO	Input/Output
RTA	Real-Time Protocol Acyclic
CR	Communication Relationship
IOCS	IO Consumer Status
IOPS	IO Provider Status
PNIOC	PROFINET IO-Controller
PNS	PROFINET IO-Device
CM	Context Management
CMDEV	Device Context Management
CMCTL	Controller Context Management
NRPM	Name Resolution Protocol Machine
RMPM	Resource Manager Protocol Machine
ALPMI	Alarm Protocol Machine Initiator
ALPMR	Alarm Protocol Machine Responder
LMPM	Link Mapping Protocol Machine
RPC	Remote Procedure Call
APMR	Acyclic Protocol Machine Receiver
APMS	Acyclic Protocol Machine Sender
CPM	Consumer Protocol Machine
PPM	Provider Protocol Machine
FSPM	FAL Service Protocol Machine
PNS_IF	PROFINET IO-Device stack's Application Interface task
LOM	Linkable Object Module
LFW	Loadable Firmware
FODMI	Fiber Optic Diagnostic Media Interface
I&M	Identification & Maintenance

Table 2: Terms, Abbreviations and Definitions

## 1.7 References to Documents

This document refers to the following documents:

- [1] PROFINET IO; Application Layer Service Definition, Application Layer Protocol Specification, Version 2.3 Edition 2 Maintenance Update 1, March 2014, Order No.: 2.722, PROFIBUS International
- [2] Operating System Manual – Realtime Communication System for netX – Kernel API Function Reference; Rev. 6; Hilscher GmbH; 2007
- [3] DPM Interface Manual for netX based Products; Hilscher Gesellschaft für Systemautomation mbH; 2012, Revision 12
- [4] TCP/IP Protocol Interface Manual; Rev. 8; Hilscher GmbH; 2009
- [5] netX I/O Synchronization; Hilscher Gesellschaft für Systemautomation mbH; 2006-2011, Rev. 5
- [6] Application note PROFINET IO Device „Synchronous application using IRT“; Hilscher Gesellschaft für Systemautomation mbH; 2014, Rev. 1
- [7] Protocol API Ethernet; Hilscher Gesellschaft für Systemautomation mbH; 2014, Rev. 8

*Table 3: References to Documents*

## **1.8 Legal Notes**

### **1.8.1 Copyright**

© 2006-2016 Hilscher Gesellschaft für Systemautomation mbH

All rights reserved.

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

### **1.8.2 Important Notes**

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

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

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

### 1.8.3 Exclusion of Liability

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

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

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

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

### 1.8.4 Export

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

## 1.8.5 Third Party Software Licenses

### 1.8.5.1 SNMP

For SNMP functionality the PROFINET IO RT/IRT Device Protocol stack uses third party software that is licensed under the following licensing conditions:

/\*\*\*\*\*

Copyright 1988, 1989 by Carnegie Mellon University

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of CMU not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

CMU DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL CMU BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

\*\*\*\*\*/



## 2 Fundamentals

### 2.1 General Access Mechanisms on netX Systems

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

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

The picture below visualizes these three ways:

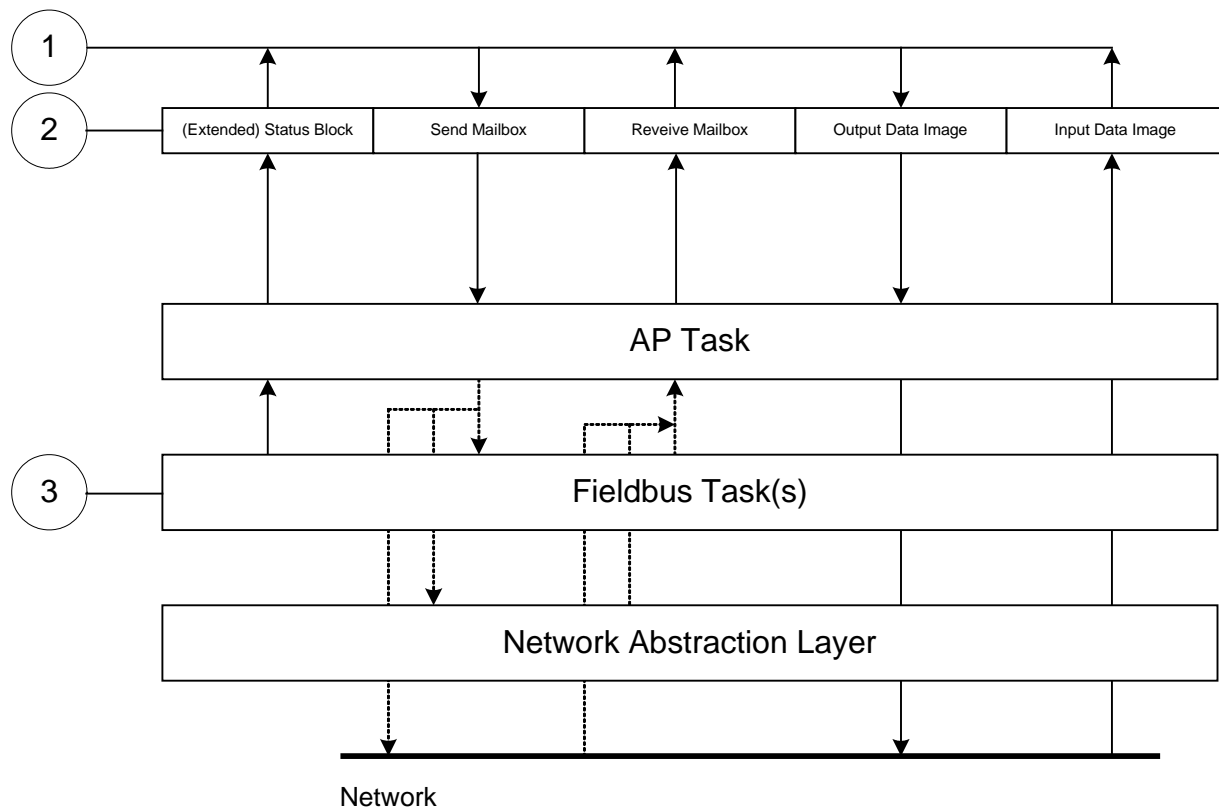


Figure 1: The 3 different Ways to access a Protocol Stack running on a netX System

This chapter explains how to program the stack (alternative 3) correctly while the next chapter describes accessing the protocol stack via the dual-port memory interface according to alternative 1 (and 2, if the user application is executed on the netX chip in the context of the rcX operating system and uses the shared DPM). Finally, chapter 5 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 5. 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 Stacks PNS-IF 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 `PNS_IF` –Task, the output of the Function `PNS_StackInit()` shall be used:

```
PROFINET_IODEVICE_STARTUPPARAMETER_T    tPNSTParam /* PNS Stack Parameters */
PROFINET_IODEVICE_TASK_RESOURCES_T      *ptPNSRsc  /* Pointer to PNS Resources */
TLR_HANDLE hQuePnsIf;                    /* PNS-IF Task Queue */

/* Fill in tPNSTParam */
...

If (TLR_S_OK == PNS_StackInit(&tPNSTParam, &ptPNSRsc))
{
    hQuePnsIf = ptPNSRsc->hQuePnsif;
}
```

For backwards compatibility, the stack still supports identifying the PNS-IF Task Queue using the macro `TLR_QUE_IDENTIFY()`. 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 the instance of the PNS Task. (See `PROFINET_IODEVICE_STARTUPPARAMETER_T`) The correct ASCII-queue names for accessing the `PNS_IF` –Task which you have to use as current value for the first parameter (`pszIdn`) is

ASCII Queue Name	Description
"QUE_PNS_IF"	Name of the <code>PNS_IF</code> –Task process queue

Table 4: Names of Queues in PROFINET Firmware

```
PROFINET_IODEVICE_STARTUPPARAMETER_T    tPNSTParam /* PNS Stack Parameters */
PROFINET_IODEVICE_TASK_RESOURCES_T      *ptPNSRsc  /* Pointer to PNS Resources */
TLR_HANDLE hQuePnsIf;                    /* PNS-IF Task Queue */

/* Fill in tPNSTParam */
...

If (TLR_S_OK == PNS_StackInit(&tPNSTParam, &ptPNSRsc))
{
    TLR_QUE_IDENTIFY("QUE_PNS_IF", tPNSTParam.ulInstance, &hQuePnsIf);
}
```

The handle `hQuePnsif` has to be used as value `ulDest` in all initiator packets the AP-Task intends to send to the `PNS_IF` –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.

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

### 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 *Acyclic Data (Mailboxes)* (page 28) in this context, is described in detail in section *General Structure of Messages or Packets for Non-Cyclic Data Exchange* (page 29) while the possible codes that may appear are listed in section *Status and Error Codes* (page 31).

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:

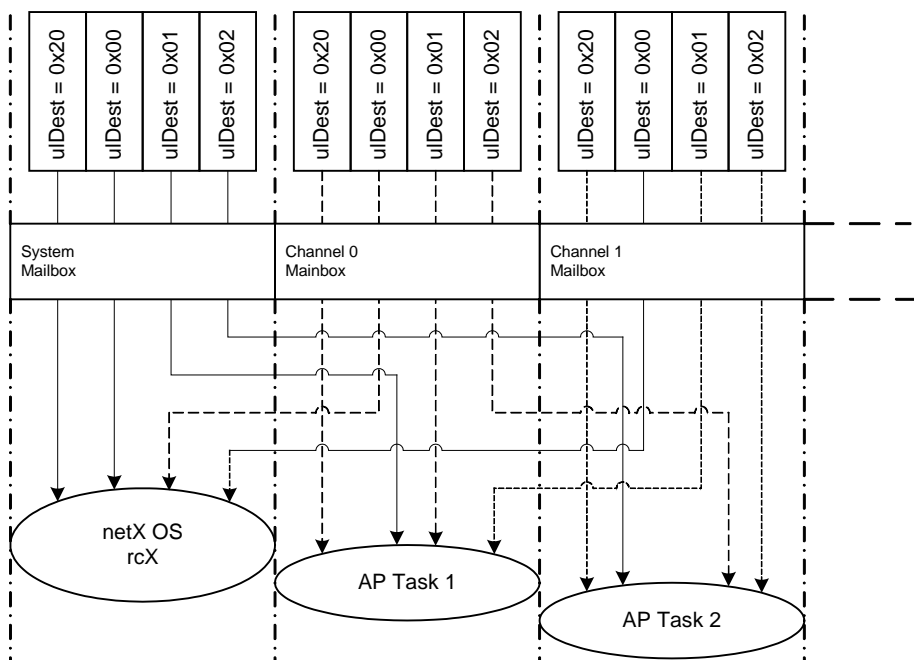


Figure 2: Use of `ulDest` in Channel and System Mailbox

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

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

Table 5: 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:

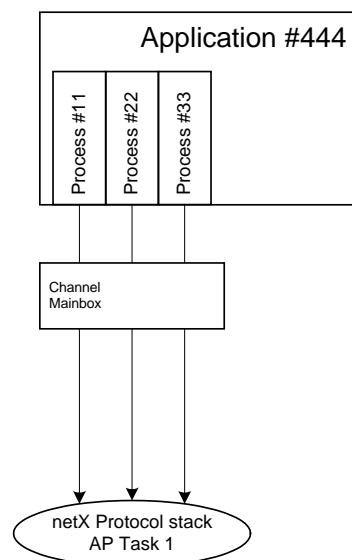


Figure 3: Using *ulSrc* and *ulSrcId*

### 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 6: 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]

Table 7: Hardware Assembly Options for different xC Ports

Check each of the hardware assembly options whether its value has been set to `RCX_HW_ASSEMBLY_ETHERNET = 0x0080`. If true, this denotes that this xCPort is suitable for running the PROFINET protocol stack. Otherwise, this port is designed for another communication protocol. In most cases, xC Port[2] will be used for Fieldbus 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

Table 8: Communication Channel Addresses in Dual-Port-Memory

In devices which support only one communication system which is usually the case (either a single Fieldbus 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 define RCX_CHANNEL_TYPE_COMMUNICATION = 0x05)
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)

Table 9: Communication Channel-related Information

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

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

## 2.4 Packet Types

Figure 4 shows the the packet types: Request Packet, Confirmation Packet, Indication Packet and Response Packet. Packets are used for communication between the application and the firmware.

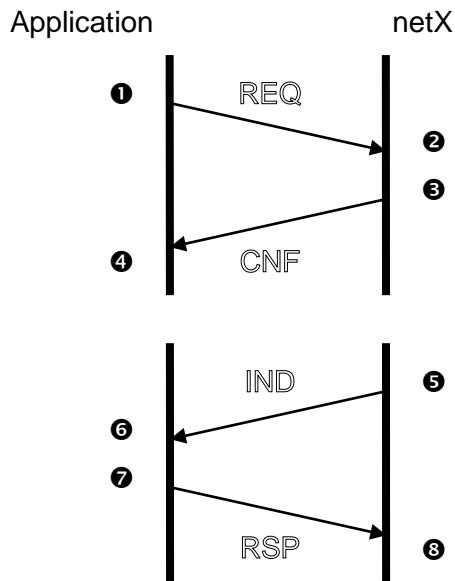


Figure 4: Packet Types

- ➊ ➋ The application sends request packets to the netX firmware.
- ➌ ➍ The netX firmware sends a confirmation packet in return.
- ➎ ➏ The application receives indication packets from the netX firmware.
- ➐ ➑ The application sends response packet to the netX firmware (may not be required).

REQ	Request	CNF	Confirmation
IND	Indication	RSP	Response

### Services requested by the application: Request Packet and Confirmation Packet

The host application can send request packets to the netX firmware (transition 1 ⇒ 2) to request a service. The netX firmware sends a confirmation packet in return, signaling success or failure (transition 3 ⇒ 4) after processing the request.

### Services indicated by the firmware to the application: Indication Packet and Response Packet

The host application has to register to the netX firmware in order to receive indication packets (transition 5 ⇒ 6). The application has to send a response packet to the netX firmware (transition 7 ⇒ 8).



## 2.4.1 Timeout for Response Packets

---

**Attention:**

The PROFINET IO Device Stack implements a timeout for some indications send to the application. If the host does not respond to any indication within a time of 3000 ms (default value), the stack will generate a response internally. If this occurs the host will be informed by the *Error Indication Service* (see page 171) with error code `TLR_E_PNS_IF_APPLICATION_TIMEOUT` in the field `ulErrorCode`. If this occurs while an AR is established, no valid data may be exchanged.

---

The default timeout value can be changed using the *Set OEM Parameters Service* with `ulParamType = PNS_IF_SET_OEM_PARAMETERS_TYPE_3`.

### 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. Each IO block uses handshake bits for access synchronization. Input and output data block handshake operates independently from each other. The PROFINET Firmware supports different operation modes for process data image synchronization. The mode of operation can be configured using the `RCX_SET_HANDSHAKE_CONFIG_REQ` (see reference [5]). The supported modes are shown in the following table.

In/Out-Handshake Mode	In/Out-Source	Description
<code>RCX_IO_MODE_BUFF_HST_CTRL</code>	0	Buffered, host controlled, unsynchronized mode (default).

Table 10: Supported process data image synchronization modes

The behavior for the different modes is as follows:

- **Buffered, host controlled, unsynchronized:** After the host has passed access to one of the blocks to the netX the netX will update the input block from an internal receive buffer if new data is available or prepare an internal send buffer using the data from output block. Afterwards the netX immediately passes back access to the host. Therefore if the host reads the input data slower or writes output data faster than the cyclic update time the master uses intermediate data may be lost.

### 3.1.1 Input Process Data

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 receive cyclic data **from** the network.

The default size of the input data image is 5760 byte. However, not all available space is actually used by the protocol stack. Depending on the specific protocol, the area actually available for user data might be much smaller than 5760 byte. An input data block may or may not be available in the dual-port memory. It is always available in the default memory map (see reference [3]).

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

Table 11: Input Data Image

### 3.1.2 Output Process Data

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 send cyclic data from the host **to** the network.

The default size of the output data image is 5760 byte. However, not all available space is actually used by the protocol stack. Depending on the specific protocol, the area actually available for user data might be much smaller than 5760 byte. An output data block may or may not be available in the dual-port memory. It is always available in the default memory map (see reference [3]).

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

Table 12: 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 acyclic data **to** the network or **to** the firmware. The receive mailbox is used to transfer acyclic 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			Type
Variable	Type	Value / Range	Description
<b>Structure Information</b>			
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
<b>Structure Information</b>			
...	...		User Data Specific To The Command

Table 13: General Structure of Packets for non-cyclic Data Exchange.

Some of the fields are mandatory; some are conditional; others are optional. However, the Head area of packet must always exist. Depending on the command, a packet may or may not have the data Area. If present, the content of the data field is specific to the command, respectively to the reply.

#### Destination Queue Handle

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 Handle

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 and Error Codes

The following status and error codes can be returned in *ulState* List of error codes see reference [3] or section *Status/Error Codes Overview* on page 301.

### 3.2.3 Differences between System and Channel Mailboxes

The mailbox system on netX provides a non-cyclic data transfer channel for field bus 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 14: 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;
```



### 3.2.7 Packet fragmentation

In some cases a large amount of data needs to be transferred between the stack and the host that does not fit into the mailbox. In this case, fragmented packet transfer is used: The data part of the affected packet is splitted into smaller fragments. Each fragment fits (including the packet header) into the mailbox. These fragments are transferred fragment by fragment through the mailbox. These fragments have to be reassembled to one “piece”.

Typically, the data length of the reassembled packet in a fragmented transfer is unknown in advance. Please refer to the used service description for details about the length.

A fragmented packet transfer is managed using the packet header fields ulExt, ulld, ulCmd and ulSta. The remaining packet header fields should be used as usual and require no special treatment. The following definitions apply to these fields:

- Two bits of the ulExt field are used to initiate and control the fragmented transfer. The affected bits are covered by the bitmask definition TLR\_PACKET\_SEQ\_MASK. The remaining bits shall be handled by the host as usual.

Definition	Value	Description
TLR_PACKET_SEQ_MASK	0x000000C0	Bitmask defining the relevant bits of ulExt for packet fragmentation
TLR_PACKET_SEQ_NONE	0x00000000	The packet shall be transferred without fragmentation
TLR_PACKET_SEQ_FIRST	0x00000080	The first fragment of a fragmented transfer
TLR_PACKET_SEQ_MIDDLE	0x000000C0	A middle fragment of a fragmented transfer
TLR_PACKET_SEQ_LAST	0x00000040	The last fragment of a fragmented transfer

Table 15: Packet Header ulExt field for fragmented transfers

- ulld is used for verification of the sequence and is incremented by the initiator of the transfer for each sent packet associated with the transfer. The fragment acknowledge, response or confirmation must contain the same id as the corresponding fragment indication or request.
- ulCmd is used to distinguish indication or request packets from response or confirmation packets.
- ulSta can be used to abort a fragmented packet transfer from host or stack. For this purpose ulSta should be set to a non-zero value.

The first step in a fragmented transfer is to split the data into smaller parts that fit into the mailbox. In order to minimize the number of fragmented requests it is recommended to make the fragments filling the complete mailbox. Thus the typical fragment length will be 1556 bytes data + 40 bytes header. This procedure is shown in Figure 5.

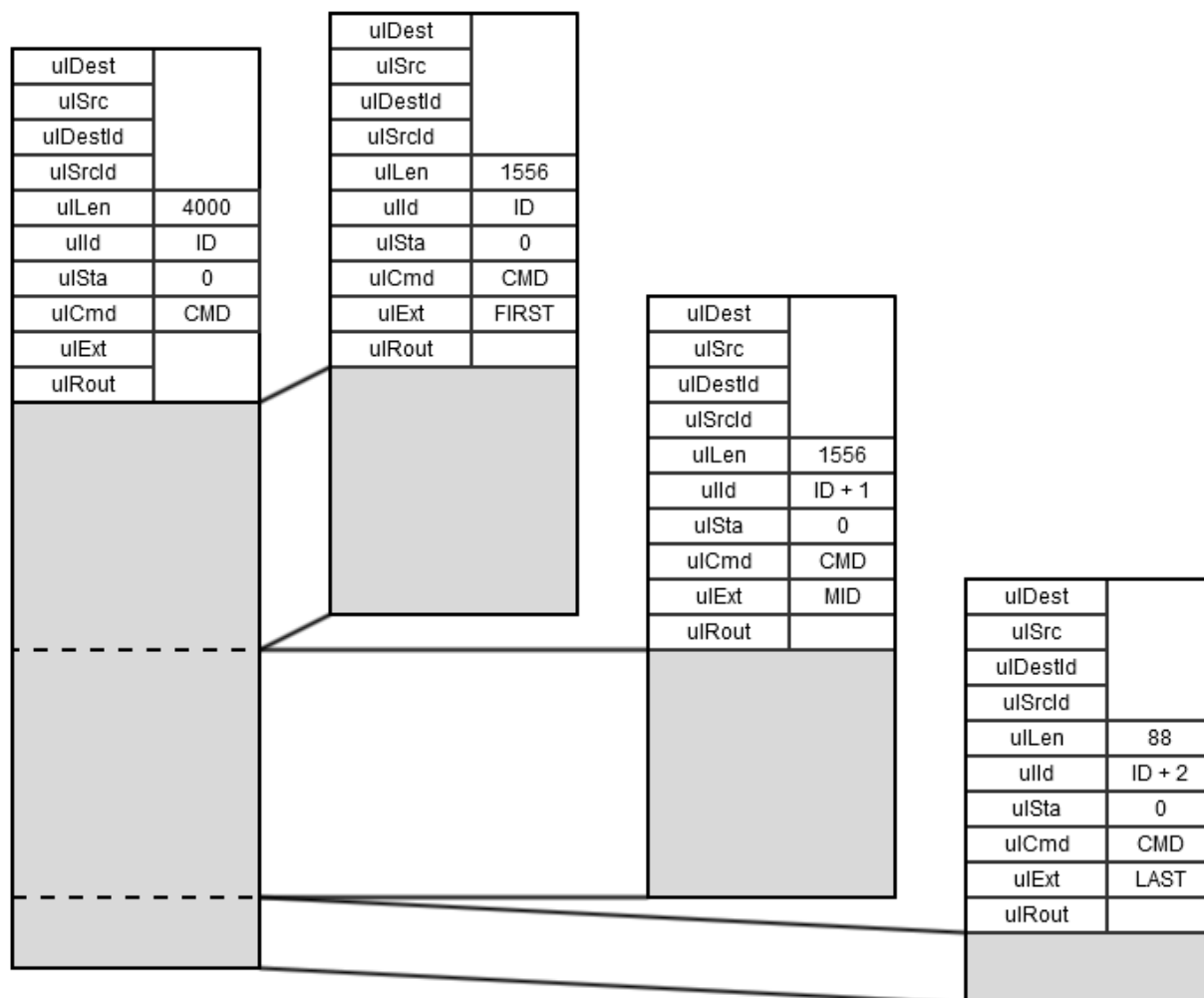


Figure 5: Splitting a large packet into fragments (figure shows 3 fragments)

After that step the data is transferred through the mailbox of the DPM by sending one fragment after another. Each fragment must be acknowledged from the receiver before the next fragment can be sent. The acknowledge packet contains the packet header only.

The sequence of a fragmented response/confirmation is shown in Figure 6.

1. The stack starts the fragmented transfer by sending the first fragment to the host with the packet header field `ulExt` & `TLR_PACKET_SEQ_MASK` set to a non-zero value.

**Attention:**

The stack starts a fragmented transfer using `TLR_PACKET_SEQ_FIRST`. Observe that, if the stack uses `TLR_PACKET_SEQ_LAST` the stack starts the fragmentation sequence, but only one fragment is required by the stack. The stack 'expects' that the host requires a fragmentation sequence to respond. The host has to use at least `TLR_PACKET_SEQ_LAST`.

2. Until the stack issues the last fragment indicated by `TLR_PACKET_SEQ_LAST`, the host receives the fragment and acknowledges it using a packet without data part. `ulld` and `ulExt` must be identical to the fragments header values.
3. After the stack has sent the last fragment indicated by `TLR_PACKET_SEQ_LAST`, the host has not to send an acknowledge. Instead the host processes the reassembled indication packet from the stack and creates the response content.
4. Now the host transfers the response packet to the stack. For this, the host might also use fragmentation. The host initiates the response transfer by sending the first response fragment to the stack.
5. As long as the response fragment does not use `TLR_PACKET_SEQ_LAST`, the stack will expect additional response fragments. Thus it will generate a fragment acknowledge using the indication command code and mirrored `ulExt` field. The host can now send the next fragment of the response using the `ulld` value from the acknowledge.
6. On the last fragment of the response, the host has to use `TLR_PACKET_SEQ_LAST`. The stack will not send another fragment acknowledge and the transfer is complete.

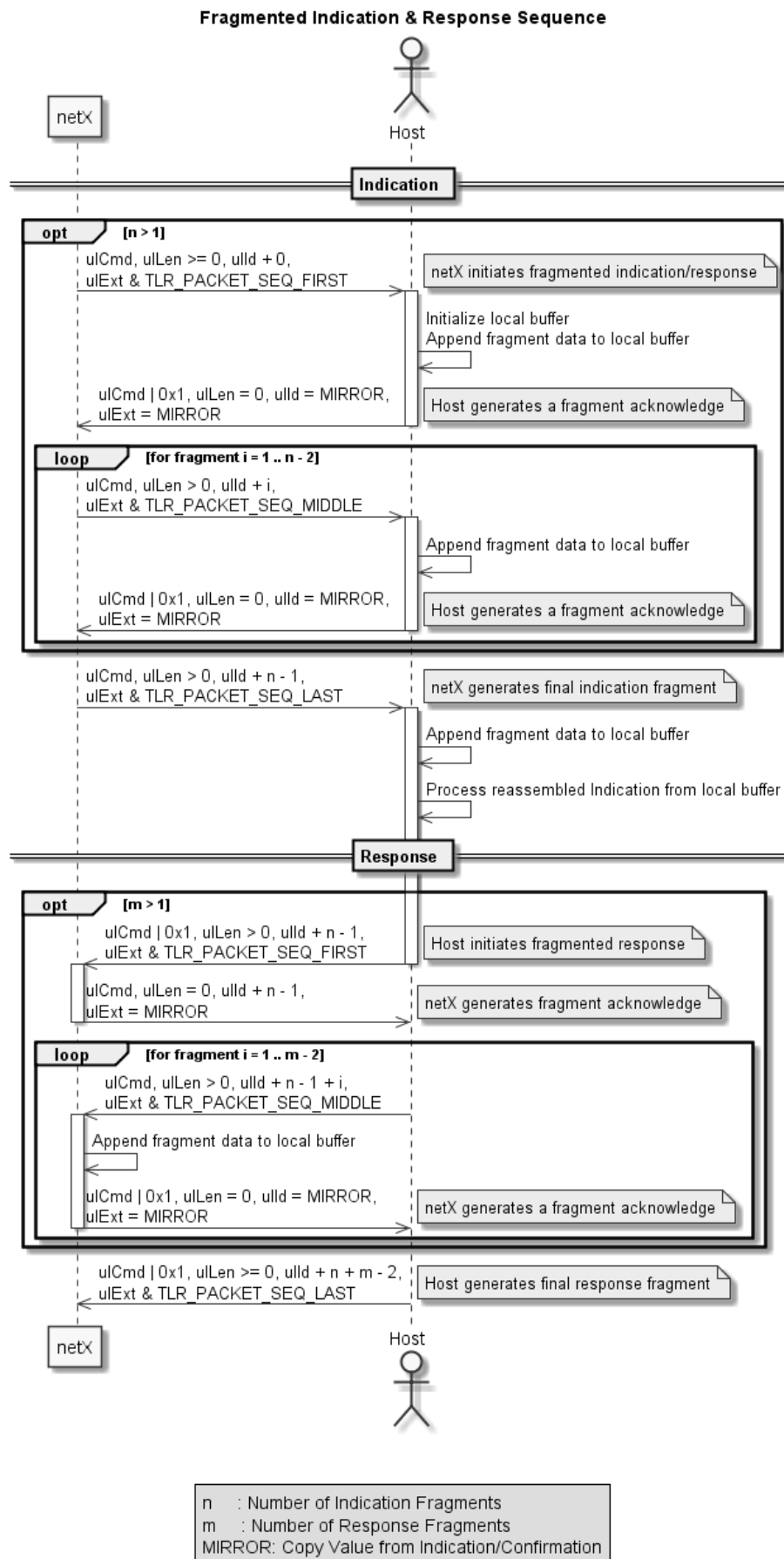


Figure 6: Sequence of fragmented indication and response

### 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 reference [3]).

#### 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	UINT8	bPDInHskMode	Input area handshake mode.
0x0021	UINT8	bPDInSource	Input Area Handshake source.
0x0022	UINT8	bPDOutHskMode	Output area handshake mode.
0x0023	UINT8	bPDOutSource	Output Area Handshake source.
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	UINT8	bErrorLogInd	<u>Counter of available Log Entries.</u>
0x002D	UINT8	bErrorPDInCnt	<u>Counter of input area handshake handling errors</u>
0x002E	UINT8	bErrorPDOutCnt	<u>Counter of output area handshake handling errors</u>
0x002F	UINT8	bErrorSyncCnt	<u>Counter of synchronization handshake handling errors</u>
0x0030	UINT8	bSyncHskMode	<u>Synchronization handshake mode.</u>
0x0031	UINT8	bSyncSource	<u>Synchronization handshake source.</u>
0x0032	UINT16	ausReserved[3]	<u>Reserved</u>

Table 16: Common Status Structure Definition

## Common Status Block Structure Reference

```
typedef struct NETX_COMMON_STATUS_BLOCK_Ttag
{
    uint32_t    ulCommunicationCOS;
    uint32_t    ulCommunicationState;
    uint32_t    ulCommunicationError;
    uint16_t    usVersion;
    uint16_t    usWatchdogTime;
    uint8_t     bPDInHskMode;
    uint8_t     bPDInSource;
    uint8_t     bPDOutHskMode;
    uint8_t     bPDOutSource;
    uint32_t    ulHostWatchdog;
    uint32_t    ulErrorCount;
    uint8_t     bErrorLogInd;
    uint8_t     bErrorPDInCnt;
    uint8_t     bErrorPDOutCnt;
    uint8_t     bErrorSyncCnt;
    uint8_t     bSyncHskMode;
    uint8_t     bSyncSource;
    uint16_t    ausReserved[3];
    {
        NETX_MASTER_STATUS    tMasterStatusBlock;
        uint32_t               aulReserved[6];
    } __RCX_PACKED_POST uStackDepended;
} 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 reference [3]). 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 reference [3]).

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 17: Communication State of Change

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

Bit	Definition / Description
0	Ready (RCX_COMM_COS_READY) 0 - ... 1 – The <i>Ready</i> flag is set as soon as the protocol stack is started properly. Then the protocol stack is awaiting a configuration. As soon as the protocol stack is configured properly, the <i>Running</i> flag is set, too.
1	Running (RCX_COMM_COS_RUN) 0 - ... 1 –The <i>Running</i> flag is set when the protocol stack has been configured properly. Then the protocol stack is awaiting a network connection. Now both the <i>Ready</i> flag and the <i>Running</i> flag are set.
2	Bus On (RCX_COMM_COS_BUS_ON) 0 - ... 1 –The <i>Bus On</i> flag is set to indicate to the host system whether or not the protocol stack has the permission to open network connections. If set, the protocol stack has the permission to communicate on the network; if cleared, the permission was denied and the protocol stack will not open network connections.
3	Configuration Locked (RCX_COMM_COS_CONFIG_LOCKED) 0 - ... 1 –The <i>Configuration Locked</i> flag is set, if the communication channel firmware has locked the configuration database against being overwritten. Re-initializing the channel is not allowed in this state. To unlock the database, the application has to clear the <i>Lock Configuration</i> flag in the control block (see page 43).
4	Configuration New (RCX_COMM_COS_CONFIG_NEW) 0 - ... 1 –The <i>Configuration New</i> flag is set by the protocol stack to indicate that a new configuration became available, which has not been activated. This flag may be set together with the <i>Restart Required</i> flag.
5	Restart Required (RCX_COMM_COS_RESTART_REQUIRED) 0 - ... 1 –The <i>Restart Required</i> flag is set when the channel firmware requests to be restarted. This flag is used together with the <i>Restart Required Enable</i> flag below. Restarting the channel firmware may become necessary, if a new configuration was downloaded from the host application or if a configuration upload via the network took place.
6	Restart Required Enable (RCX_COMM_COS_RESTART_REQUIRED_ENABLE) 0 - ... 1 – The <i>Restart Required Enable</i> flag is used together with the <i>Restart Required</i> flag above. If set, this flag enables the execution of the Restart Required 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 18: 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





### 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 reference [3]).

---

**Note:** 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.

---

```
typedef struct NETX_EXTENDED_STATUS_BLOCK_Ttag
{
    UINT8 abExtendedStatus[432];
} NETX_EXTENDED_STATUS_BLOCK_T
```

For the PROFINET IO RT/IRT Device protocol stack V3 implementation, the extended status area is currently not used.

### 3.4 Control Block

A control block is always present in both system and communication channel. In some respects, control and status block are used together in order to exchange information between host application and netX firmware. The control block is written by the application, whereas the application reads a status block. Both control and status block have registers that use the *Change of State* mechanism (see section 2.2.1 of reference [3]).

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 19: Communication Control Block

#### Communication Control Block Structure

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

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

## 4 Getting Started

### 4.1 Structure of the PROFINET Device Stack

The illustration below shows the internal structure of the tasks which together represent the PROFINET IO Device Stack V3.10.0:

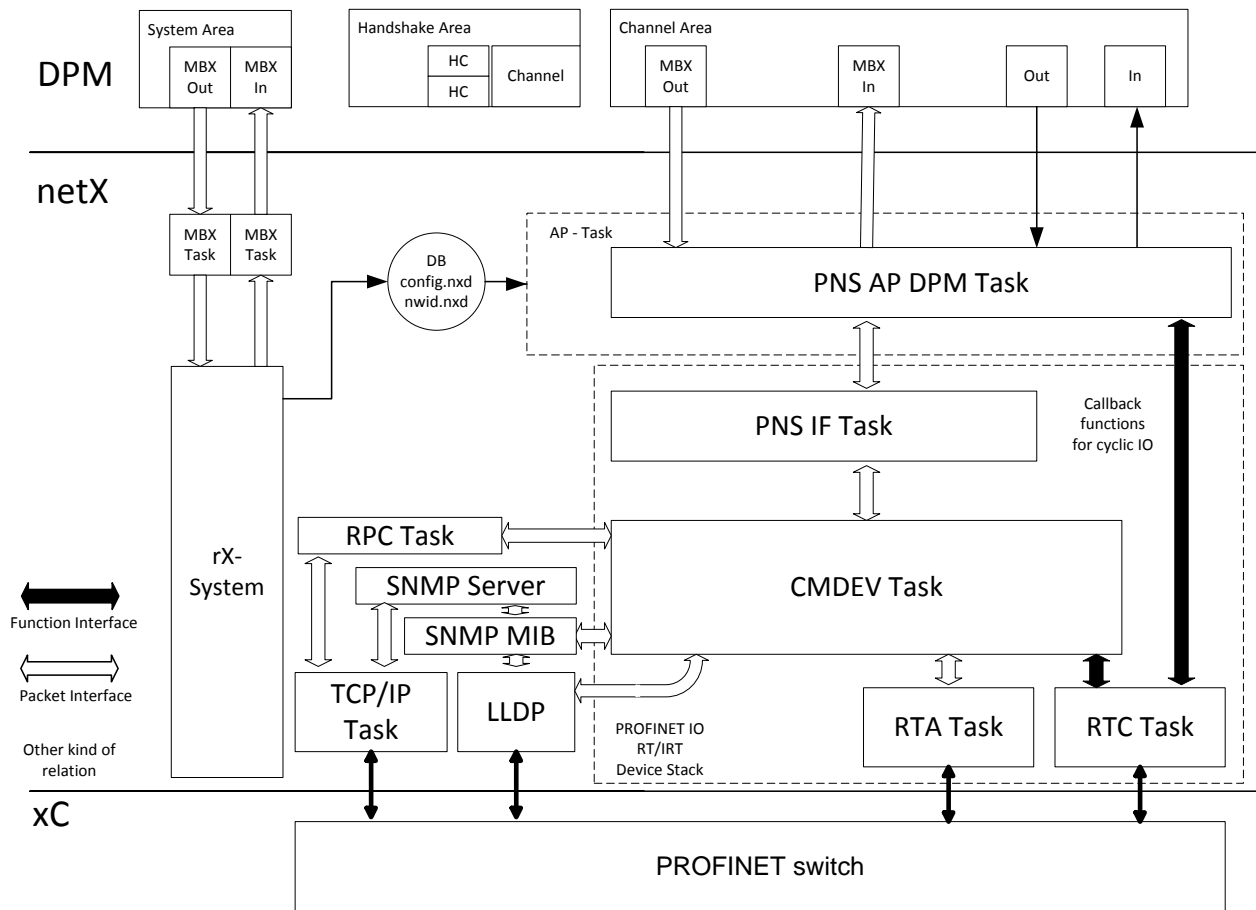


Figure 7: Task Structure of the PROFINET IO Device Stack V3.7

In this scenario, the dual-port memory is used for exchange of information, data and packets. Here, the PNS AP DPM Task takes care of mapping the PROFINET Device Stack API to the Dual-Port-Memory.

In general, the AP-Task (User application or Dual-Port-Memory Task) only interfaces to the highest layer task, namely the PNS\_IF task, which represents the application interface of the PROFINET Device stack.

- The RTA task, RTC task and the CMDEV task form the core of the PROFINET Device Stack.
- The RPC task is required to perform the RPC calls required by PROFINET specification. As transport, the connectionless DCE-RPC protocol is used
- The SNMP server task and the SNMP MIB task are handling the SNMP requests used for network diagnosis and topology detection.
- The LLDP task provides the implementation of the LLDP protocol use for neighborhood detection.
- The TCP/IP task provides TCP and UDP services

Handling of Ethernet frames is used by the RTA Task, RTC Task, LLDP Task and the TCP/IP Task. This functionality is provided by the xC Switch codes in conjunction with an rcX Ethernet Driver API. In detail, the various tasks have the following functionality and responsibilities:

### **RTA task**

The RTA task provides the following functionality:

- Processing of all PROFINET acyclic real-time telegrams. (PROFINET Alarm services)
- Processing of DCP telegrams
- Handling of Link Status Changes.

The following PN IO state machines are implemented by this task: APMS, APMR, ALPMR, ALPMI, DCPHMCS, DCPMCR, DCPUCS and DCPUCR.

### **RTC task**

The RTC task provides the following functionality:

- Processing of all PROFINET cyclic real-time telegrams. (PROFINET Cyclic services)
- Mapping of I/O-Data between application and telegrams

The following PN IO state machines are implemented by this task: CPM and PPM.

### **CMDEV task**

The CMDEV task provides the following functionality:

- PROFINET AR Management (Connection Handling, Alarm Generation, Ownership Handling)
- Handling of AR related parameter records.
- Controlling of TCP/IP-, SNMP-, RPC-, RTA- and RTC-Task

The following PN IO state machines are implemented by this task: CMDEV, CMDEV-DA, CMINA, CMRPC, CMSM, CMSU, CMWRR, OWNSM, PLUGSM, PULLSM and RSMSM.

## **xC PROFINET-Switch + rcX Ethernet Driver**

The xC PROFINET-Switch and the associated rcX Ethernet Driver provides the following functionality:

- Standard Ethernet 2-Port Switch functionality. (Local Send & Receive of frames. Forwarding of Frames between ports)
- Ethernet PHY Handling (LinkUp/Down/State)
- Handling of protocols like PTCP and MRP

This part provides the LMPM according to the PROFINET IO specification. Additionally, the following PN IO state machines are implemented by the associated Ethernet Driver:

- PTCP Delay Requestor,
- PTCP Delay Responder,
- MRP Client.

## **RPC task**

The RPC task is required for connection-less RPC used by PROFINET for acyclic services. It provides the following functionality:

- Connectionless RPC Client and RPC Server. (Using UDP Protocol)
- RPC Endpoint Mapper Services

## **LLDP task**

The LLDP task implements the LLDP protocol and the LLDP MIB Database according to the PROFINET IO and LLDP specification.

## **SNMP tasks**

The SNMP Server and MIB tasks implement the SNMP protocol and the MIB (Management Information Base).

## **PNS\_IF task**

The PNS\_IF task provides the following functionality:

- Interface between protocol stack and AP-Task
- Handling of Physical Device Parameters
- Diagnosis processing
- Handling of most PROFINET Read/Write Records
- Checking submodule IO-offsets for overlapping
- Handling I&M data (if stack shall handle it)
- Handling PROFINET Logbook

## **PNS AP DPM task**

This task implements the Dual-Port-Memory interface of the PROFINET Device Stack. It furthermore implements the functionality to evaluate SYCON.net configuration databases.

## **TCP/IP task**

The TCP/IP task provides TCP and UDP services.

## 4.2 Naming Conventions

PROFINET and the netX use different naming schemes in certain cases. To avoid problems with that this section shall clarify the naming conventions:

PROFINET	netX DPM/SHM	PROFINET Device Stack	Description
Inputs (Data)	Shall be written to the Output Area	Provider Image / Provider Data	Data of Input Submodules. Send from the Device to the Controller
Outputs (Data)	Shall be read from the Input Area	Consumer Image / Consumer Data	Data of Output Submodules. Send from the Controller to the Device

Table 20: Naming convention of Input/Output Data

## 4.3 Overview about Essential Functionality

You can find the most commonly used functionality of the PROFINET IO RT/IRT Device Protocol API within the following sections of this document:

Topic	Section No.	Section Name
Configuration of the stack	6.1.5.1	Set Configuration Request
Acyclic data transfer (Read Indication)	6.3.1.1	Read Record Indication
Acyclic data transfer (Write Indication)	6.3.2.1	Write Record Indication
Register Application	6.1.6.1	Register Application Service
Save IP	6.3.5.1	Save IP Address Indication
Check Indication	6.2.2.1	Check Indication
AR InData Indication	6.2.6.1	AR InData Indication
Error Indication	6.3.13.1	Error Indication
Get Diagnosis	6.4.1	Get Diagnosis Service
Diagnostic Alarm	6.4.4.1	Diagnosis Alarm Request
Process Alarm	6.4.3.1	Process Alarm Request

Table 21: Overview about essential Functionality

For more information how to configure and setup the protocol stack, see chapter *Packet Interface Configuring the IO-Device Stack*, and especially section *Configuring the PROFINET IO Device Stack* on page 61 might be very useful.

## 4.4 Event Mechanism

The PROFINET Device stack uses an event mechanism to indicate some import events to the application. If the stack is accessed using Dual-Port-Memory or Shared Memory API, the Events will be indicated using the Event Indication Service describe on page 185. If the Callback Interface is used to access the I/O-Data, the events are indicated using the event callback function.

The following events are currently defined:

Event Number	Meaning
0x00000000	PNS_IF_IO_EVENT_RESERVED: Reserved
0x00000001	PNS_IF_IO_EVENT_NEW_FRAME: This event <b>shall not be used anymore</b> and will be generated for compatibility only
0x00000002	PNS_IF_IO_EVENT_CONSUMER_UPDATE_REQUIRED: The Data within the consumer image shall be updated because it may not be valid any longer. This occurs for example when the connection is lost.
0x00000003	PNS_IF_IO_EVENT_PROVIDER_UPDATE_REQUIRED: The stack needs access to the provider data. This happens for example if an application ready has to be send and the stack needs to refresh the cyclic data before.
0x00000004	PNS_IF_IO_EVENT_FRAME_SENT: This <b>event shall not be used anymore</b> and will be generated for compatibility only.
0x00000005	PNS_IF_IO_EVENT_CONSUMER_UPDATE_DONE: The stack finished updating the consumer data image. The user application shall read the data now. This event has no meaning for DPM/SHM because the standard Handshake mechanism is used here.
0x00000006	PNS_IF_IO_EVENT_PROVIDER_UPDATE_DONE: The stack finished updating the frames from provider data image. The user application may write new data now. This event has no meaning for DPM/SHM because the standard Handshake mechanism is used here.

Table 22 PROFINET Device Stack Events



## 4.5 Device Handle

The PROFINET Device Stack supports handling multiple ARs at the same time. This means, that for instance Parameter Writes, Parameter Reads, Connect Sequences and Abort Sequence may occur at the same time. In order to distinguish between the different ARs, the stack provides the attribute `hDeviceHandle` in all affected packets. The device handle may hold the following values:

- `hDeviceHandle = 0`: The request is associated with no AR (Only ReadImplicit) or the request is associated with a Supervisor DA AR.
- `hDeviceHandle != 0`: The request is associated with a Supervisor AR or an IO-AR. The Type of the AR is indicated to the application using the AR Check Service.

As a consequence, the application must be able to handle multiple instances of the following indications at the same time:

- AR Check Indication
- Connect Request Done Indication
- Parameter End Indication
- AR InData Indication
- Read Record Indication
- Write Record Indication
- AR Abort Indication Indication
- APDU Status Changed Indication
- Alarm Indication
- Release Request Indication
- Read I&M Indication
- Write I&M Indication
- Parameterization Speedup Support Indication

## 5 Exchanging Cyclic Data

This section describes how the user application can access the cyclic IO-data which is exchanged with the PROFINET Controller. The PROFINET Device stack provides different ways to exchange this data. Depending on the user's application only one of these methods may be used:

- If the netX chip is used as dedicated communication processor while the user's application runs on a separate host processor of its own, I/O-Data can be accessed using the mechanism described in the *Dual Port Memory Interface* manual only. This is always the case if the stack is used as loadable firmware.
- If the user application is running on the netX chip together with the PROFINET IO-Device stack there are two possibilities to access the cyclic I/O-Data:
  - If the *Shared Memory Interface* is used, the user application has to access the I/O-Data using the shared memory interface API. As this is basically an emulation of the *Dual Port Memory Interface* for applications running locally on the netX chip, the interface is similar to using the netX as dedicated communication processor.
  - If the user application is not using the shared memory interface, the I/O-Data is accessed using a function call API. This approach removes any overhead from the Shared Memory Interface.

### 5.1 General Concepts

PROFINET uses the concept of a cyclic process data image. Each master or slave of a PROFINET network has an image of input and output data. This image is updated from communication partner images using periodic Ethernet telegrams. These frames are sent at intervals configured by the engineering system. The frames contain the I/O-Data together with their associated data status. Furthermore each frame contains a "global" frame data status field which for example can be used to mark the whole frame as invalid.

PROFINET organizes the cyclic data in a Provider-Consumer model. This means that an IO-data consumer exists for every IO-data provider. Both indicate their current state to each other in different frames. These states are the IO Provider state (IOPS) and IO Consumer State (IOCS). The IOPS indicates if the associated data is valid or invalid. For instance a faulty submodule in a device would mark its input data to be invalid by setting the IOPS appropriately. The IOCS is sent from the consumer of the back to its provider to indicate if the data was handled. For instance a DAC submodule may use this service to indicate the controller an output value which is out of the DAC's range.

In PROFINET each submodule has its I/O-data and its I/O-data states. Therefore for every submodule used by the IO-Controller there is not only the I/O-data exchanged but additionally also two I/O-data states. In terms of the Ethernet frame structure the provider data state resides directly behind the I/O-data itself and contains information if the I/O-data is good and may be evaluated or not. The consumer state is sent in the opposite direction in a different frame and contains information if the I/O-data could be handled by the consumer.

**Attention:**

Because of the concept of a cyclic process data image, it is strongly recommended to design the application in a way to cyclically read the consumer data and write the provider data regardless of the communication state. The stack will take care of setting the consumer data either to zero, to the last valid value or to some substitute value (depends on parameters) if no communication is active.

**Note:**

If the application cannot be designed to cyclically read or write the process data or if the cycle is slow (larger than 50 ms) it is recommended to read the consumer data when the `PNS_IF_IO_EVENT_CONSUMER_UPDATE_REQUIRED` occurs and to write the provider data when the `PNS_IF_IO_EVENT_PROVIDER_UPDATE_REQUIRED` event is signaled.

## 5.2 Behavior regarding IO data and IOPS

An application determines the behavior of physical outputs. The PROFINET-IO stack delivers the output data (sent by provider, e.g. controller) until the IO-image (e.g. DPM) only. An application has to validate the data and handle their outputs according to the provider states IOPS.

Following communication states must be considered:

- Bus on (e.g. start of the stack or link up)
- Communication abort (e.g. PROFINET controller sent disconnect or link down)
- IOPS is set to BAD (e.g. data provider does not send valid data)

For these cases the **default** behavior of the PROFINET stack is the copying of NULLs in the IO-image for all affected submodules.

**Attention:**

If an application does not use the IOXS interface (Set IOXS Config Service), it has no opportunity to determine invalid data because NULLs are also evaluated as valid.

On the other hand, using the IOXS interface an application determines invalid data evaluating the IOPS.

**Special** behavior of IO-data in the PROFINET stack can happen if a PROFINET-IO controller configures the device (during connection establishment) using a standardized service to define the behavior of IO-data, it is:

1. data set to NULLs (default);
2. the last valid values are delivered;
3. replacement values are set.

**Note:**

In order to support the second and third behaviors, the key "SupportedSubstitutionModes" should be set in the GSDML-file. Currently there is no engineering system on the market that supports this key.

## 5.3 Exchanging cyclic Data using Callback Interface

### 5.3.1 Overview of the Callback Interface

The PROFINET Device Stack provides a simple callback interface to local applications for accessing the cyclic input/output data. Using this interface most of the PROFINET specific logic is hidden from the user application. Basically all data exchange can be performed by calling of two callback functions provided by the stack. For more sophisticated setups the user application shall provide an event callback function which will be used by the stack to indicate events to the user application. Generally the callback interface requires the following tasks from the user application's view:

1. The user application has to allocate two memory blocks. One to hold the consuming (output) and one to hold the providing (input) data. These blocks shall be large enough to hold the IO-data and the IOPS if desired.
2. Pass the pointers to the blocks, their size and optionally user application's event callback function pointer to the stack using the *Set IO-Image Service* described in section *Set IO-Image Service* on page 94. In return this service will provide the user application with function pointers of the callback functions to call for data exchange.
3. Now call the callbacks either cyclically or if necessary:
  - To update the outgoing provider data call the *UpdateProviderImage* Callback described in section *UpdateProviderImage* Callback on page 54 whenever the outgoing data has changed.
  - To get the newest incoming consumer data, call the *UpdateConsumerImage* described in section *UpdateConsumerImage* on page 53 periodically.

### 5.3.2 Callback Functions

The callback interface consists of four functions. Three of them are used by the application to update the consumed data, update the provided data or get information about the state areas within the IO-data images. The fourth callback is provided by the user application to the stack. This callback is used by the stack to inform the application about cyclic events.

**Note:**

The callback functions may only be called by the application in task context. They shall not be called in interrupt context when the interrupt modes *SYSTEM* or *INTERRUPT* are used.

Otherwise the stack may get stuck or the operating system task scheduler will not work properly any longer.

**Note:**

The callback functions may only be called by the application in one task context. They shall not be called in different contexts (e.g. by different tasks).

Otherwise the application may get stuck or the operating system task scheduler will not work properly any longer.

**Note:**

Prior using the callback interface the application has to provide the stack with a consumer and a provider IO image and a pointer to the event callback function. The other way round the application needs the function pointers of the stack's callback functions. For this the *PNS\_IF\_SET\_IOIMAGE\_REQ* request has to be issued by the user's application before any stack initialization.

### 5.3.2.1 UpdateConsumerImage Callback


**Note:**

The Update ConsumerImage callback function may only be called by the application in task context. It shall not be called in interrupt context when the interrupt modes SYSTEM or INTERRUPT are used.

Otherwise the stack may get stuck or the operating system task scheduler will not work properly any longer.

This callback is used by the user's application to update the consumer data image from the last received cyclic frames. The consumer data will be copied from the frame into the consumer image according to the submodule configuration supplied by the user's application. If enabled, the function will also copy the associated provider states into the provider state block of the consumer image. While the stack is updating the consumer image the data of the image will be inconsistent. Therefore the user application must not access the consumer image until the stack has finished updating the image. The UpdateConsumerImage callback is defined by the following type declaration:

```
typedef TLR_RESULT (*PNS_IF_UPDATE_IOIMAGE_CLB_T) (
    TLR_HANDLE hUserParam,
    TLR_UINT    fLateConfirmation,
    TLR_UINT    uiReserved1,
    TLR_UINT    uiReserved2
);
```

The four parameters of the callback are described as follows:

Parameter	Meaning
hUserParam	This parameter is a handle obtained from the stack by using the <code>PNS_IF_SET_IOIMAGE_REQ</code> .
fLateConfirmation	If this parameter is set to zero, the callback will return after the stack has finished updating the consumer data image. (The calling task will be blocked). If this parameter is set to a non-zero value, the function will return immediately and stack will use the <code>PNS_IF_IO_EVENT_CONSUMER_UPDATE_DONE</code> event later on to notify the user application about finishing the update.
uiReserved1	Reserved for future usage. Set to zero.
uiReserved2	Reserved for future usage. Set to zero.

Table 23: Parameters of UpdateConsumerImage Callback

The callback uses the following return codes:

Return code	Meaning
0x00000000	TLR_S_OK: No Error.

Table 24: Return Codes of UpdateConsumerImage Callback

### 5.3.2.2 UpdateProviderImage Callback


**Note:**

The Update ProviderImage callback function may only be called by the application in task context. It shall not be called in interrupt context when the interrupt modes SYSTEM or INTERRUPT are used.

Otherwise the stack may get stuck or the operating system task scheduler will not work properly any longer.

This callback is used by the user's application to update the cyclic frames sent by the IO-Device to the IO-Controller from the provider data image. The provider data will be copied from provider image into the cyclic frame according to the submodule configuration supplied by the user's application. If enabled, the function will also copy the associated provider states from the provider state block of the provider image. While the stack is updating the frames, the user application must not change the content of the provider image. Otherwise inconsistent or invalid data may be transmitted to the bus. The UpdateProviderImage callback is defined by the following type declaration:

```
typedef TLR_RESULT (*PNS_IF_UPDATE_IOIMAGE_CLB_T) (
    TLR_HANDLE hUserParam,
    TLR_UINT    fLateConfirmation,
    TLR_UINT    uiReserved1,
    TLR_UINT    uiReserved2
);
```

The four parameters of the callback are described as follows:


Parameter	Meaning
hUserParam	This parameter is a handle obtained from the stack by using the PNS_IF_SET_IOIMAGE_REQ.
fLateConfirmation	<p>If this parameter is set to zero, the callback will return after the stack has finished updating the frames from provider data image. (The calling task will be blocked) and finished its current internal cycle. If this parameter is set to a non-zero value, the function will return immediately and stack will use the <b>PNS_IF_IO_EVENT_PROVIDER_UPDATE_DONE</b> event later on to notify the user application about finishing the update.</p> <div>  <p>If this parameter is set to zero the calling task may be blocked up to 1ms. Thus it is highly recommended to set this value to a non-zero value.</p> </div>
uiReserved1	Reserved for future usage. Set to zero.
uiReserved2	Reserved for future usage. Set to zero.

Table 25: Parameters of UpdateProviderImage Callback

The callback uses the following return codes:

Return code	Meaning
0x00000000	TLR_S_OK: No Error.

Table 26: Return Codes of UpdateProviderImage Callback

### 5.3.2.3 Event Handler Callback

This callback has to be provided by the user's application to the stack. It will be called by the stack upon events defined in section 4.4 Event Mechanism. As the callback will be called from stacks task context, the user must consider locking of shared resources.

**Note:**

In the Event Handler Callback it is strictly forbidden to call any OS function that waits for any objects or forces task to "sleep" state.

The Event Handler callback is defined by the following type declaration:

```
typedef TLR_RESULT (*PNS_IF_IOEVENT_HANDLER_CLB_T) (  
    TLR_HANDLE hUserParam,  
    TLR_UINT   uiEvents  
);
```

The two parameters of the callback are described as follows:

Parameter	Meaning
hUserParam	This parameter is a handle supplied by the user by the PNS_IF_SET_IOIMAGE_REQ request.
uiEvents	This argument is the event number.

The return value of the function is currently ignored by the stack. However the value 0x00000000 (TLR\_S\_OK) should be used for compatibility with future versions of the stack.

## 6 Status Information

### 6.1 Communication State

In general the Communication State is described in [4].

This section describes how the Communication State is used by PROFINET IO Device.

#### 6.1.1 Implementation from V3.10

Starting with PROFINET IO Device V3.10.0.0 the default handling of communication state is as described in the following table:

State	Description
OFFLINE	The IO Device has no valid configuration.
STOP	The IO Device has no communication to the IO Controller. Connection establishment is not in progress. The Bus state of the IO Device may be set to on or off.
IDLE	The communication establishment is in progress (at least one CMDEV state is > W_CIND and no CMDEV state is INDATA).
OPERATE	The I/O connection is established and valid I/O data is exchanged between the Controller and the Device (at least one CMDEV state is INDATA).

Table 27: Communication State (V3.10 and later)

#### 6.1.2 Legacy Implementation (V3.9 and earlier)

Implementations of version V3.9 and earlier behave as follows.

State	Description
OFFLINE	No valid configuration.
STOP	Valid configuration and (Bus off or Link down or Fatal Error).
IDLE	n.a. (Note: this state is not used at all)
OPERATE	Valid configuration and Bus on and Link up.

Table 28: Communication State (V3.9 and earlier)

If needed, legacy handling could be enabled by setting the AP task startup parameter PNS\_AP\_DPM\_STARTUP\_FLAG\_LEGACY\_COMM\_STATE in firmware/stack V3.10.



## 7 Packet Interface

### 7.1 Configuring the IO-Device Stack

This chapter explains how the PROFINET Device stack is configured at start-up. The Configuration is either read out from SYCON.net database, taken from iniBatch configuration files or provided by the application by means of Set Configuration Service (see section *Set Configuration Service* on page 63).

In detail, the following configuration functionality is provided by the PROFINET IO Device IRT Stack:

Overview over the Configuration Packets of the PROFINET IO Device IRT Stack			
Section	Packet	Command code	Page
6.1.5	Set Configuration Request	0x1FE2	63
	Set Configuration Confirmation	0x1FE3	71
6.1.10	Set Port MAC Address Request	0x1FE0	81
	Set Port MAC Address Confirmation	0x1FE1	82
6.1.11	Set OEM Parameters Request	0x1FE8	84
	Set OEM Parameters Confirmation	0x1FE9	89
6.1.12	Load Remanent Data Request	0x1FEC	90
	Load Remanent Data Confirmation	0x1FED	92
6.1.13	Configuration Delete Request	0x2F14	93
	Configuration Delete Confirmation	0x2F15	93
6.1.14	Set IO-Image Request	0x1FF0	94
	Set IO-Image Confirmation	0x1FF1	96
6.1.15	Set IOXS Config Request	0x1FF2	98
	Set IOXS Config Confirmation	0x1FF3	100

Table 29: Overview over the Configuration Packets of the PROFINET IO Device IRT Stack

### 7.1.1 Cyclic Process Data Image

The PROFINET IO protocol uses a cyclic process data model to exchange process data between the PROFINET IO Controller and Device. That is, the PROFINET IO Controller's application and the PROFINET IO Device's application periodically update their own copies of the process data. These two images are periodically synchronized vice versa by the Protocol Stacks.

In PROFINET process data is organized at the level of submodules. Each submodule can be assigned input and/or output process data. Submodules without any data are assigned with a zero length to the input process data block. Each process data block is associated with a provider data status (IOPS) and a consumer data status (IOCS). The provider data status is generated by the producer of the data and thus is exchanged in the same direction as the process data itself. It indicates whether the data is valid. In contrast to that the consumer data status is generated by the consumer of the data and thus exchanged in the opposite direction than the process data. It indicates if the consumer of the data was able to use the data. The data status reflects the validity of the process data at application level. That means in particular, that the application must indicate invalid data. This cannot be done by the protocol stack. Nevertheless, the protocol stack might force a Bad Provider/Consumer State on certain Submodules if required by the Protocol.

---

**Note:** In order to simplify the host application development, the Hilscher PROFINET Device stack is configured by default in an operation mode where it assumes valid data whenever DPM Output Area/Provider Data Area is updated. If explicit provider and/or consumer status processing is required, the stack must be configured using the Configure IOXS Service.

---

### 7.1.2 Configuration of Process Data Images

From the PROFINET IO Device viewpoint, the Provider Process Data Image is the data which is sent from the PROFINET IO Device to the PROFINET IO Controller. This is usually called Input Process Data (Inputs). This data is to be placed by the application in the Provider Process Data Image. If a Loadable Firmware or Module is used, the Provider Data Image corresponds to the DPM Output Area. For Linkable Objects, the Provider Process Data Image is defined by the Application as a gapless memory block and has to be provided to the Protocol Stack.

---

**Attention:** Remind that Process Data sent to the PROFINET IO Controller, which is typically called Inputs, is to be placed within the DPM Output Area. This Naming issue might lead to confusion.

---

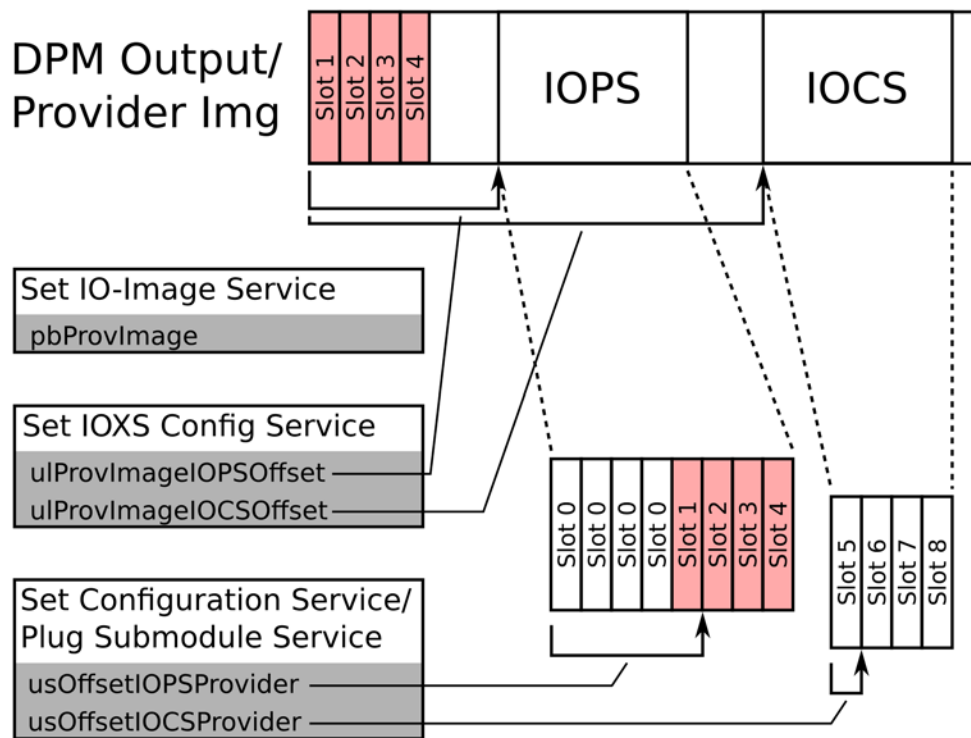


Figure 8: Provider Process Data Structure

The red color indicates data associated with input submodules. Slot 0 refers to the DAP and PDEV submodules which are also input submodules and must be always present.

The Process Data Image consists of up to three blocks: The Process Data itself, a Provider Data Status Area and a Consumer Status Area. This is shown in image provider.

In contrast to that the Consumer Process Data Image is the data which is sent from the PROFINET IO Controller to the PROFINET IO Device. (If looking from PROFINET IO Device Viewpoint). This data is typically called Output Process Data (Outputs). This data should be taken by the application from the Consumer Process Data Image. If a Loadable Firmware or Module is used, the Consumer Process Data Image corresponds to the DPM Input Area. For Linkable Objects, the Consumer Process Data Image is defined by the Application as a gapless memory block and has to be provided to the protocol stack.

**Attention:** Remind that Process Data received from the PROFINET Controller, which is typically called Outputs is to be placed within the DPM Input Area. This Naming issue might lead to confusion.

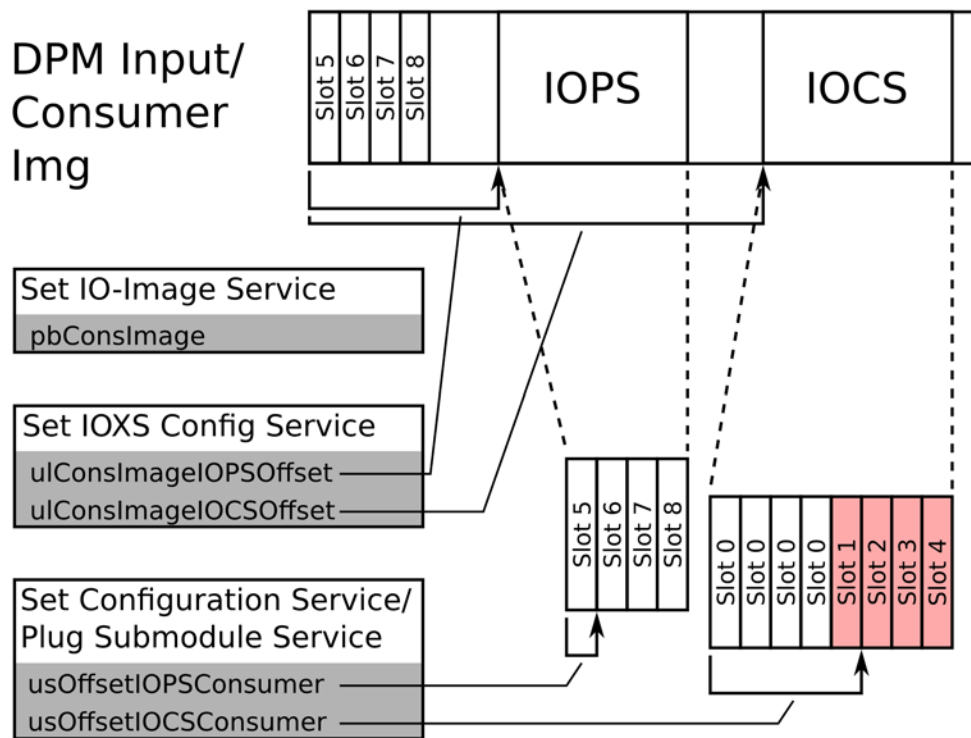


Figure 9: Consumer Process Data Structure

The red color indicates data associated with input submodules. Slot 0 refers to the DAP and PDEV submodules which are also input submodules and must always be present.

**Note:** If the host application uses consumer data states it is important to understand that the Consumer Data State of a submodule is to be placed in the opposite Process Data Image than the Process Data itself. This is because the Consumer Data States are a kind of confirmation from the receiver for the sender of the data.

In order to configure the process data image structure, the following steps must be performed:

1. If a Linkable Object is used, the first step is to provide the stack with the pointer to the process data memory blocks by means of Set IO Image Service. The memory areas must be defined by the application and shall be large enough to hold the data. This step is not required if Dual Port Memory or Shared Memory Api is used. In that case the Provider Process Data Image corresponds to the DPM Output Area and the Consumer Process Data Image corresponds to the DPM Input Area.
2. The second step is optional and only required if Provider and/or Consumer Process Data States are prepared by the Host Application. This step enabled the IOPS and/or IOCS blocks which are disabled by default. If these Blocks shall be used, the Host Application must configure the start offsets of them. These Offsets are relative to the Beginning of the corresponding Process Data Image. The Set IOXS Config Service shall be used for this purpose. These offsets are configured in units of Bytes.
3. The third step in configuring the process data images is to specify the location of the process data and the data states within their corresponding areas. These parameters are part of the Set Configuration Service and/or Plug Submodule Service. The offset of the Process Data is always specified in bytes relative to the beginning of the corresponding process data memory. The offsets of the data states are always relative to the beginning of the corresponding data state block. These offset are either in units of bytes or in units of bits. This depends on the mode configured in step 2.

### 7.1.3 Configuration of the Submodules

This option will just be evaluated if the IOXS Configuration Service has been performed before or the stack will ignore the configuration values. If IOXS is configured it is mandatory to configure correct offsets for each submodule.

The configuration is done by the Set Configuration Request for each (see section Set Configuration Request).

### 7.1.4 Configuring the PROFINET IO Device Stack

In order to configure a PROFINET IO Device using the PROFINET IO Device Stack correctly, proceed as follows:

- Configure the IO Image with the Set IO-Image Service if Dual-Port-Memory or Shared Memory Interface is not used
- If necessary, assign a MAC Address to the device. This is described in the netX Dual-Port Memory Manual. Additionally, PROFINET requires assigning of Port MAC Addresses. It is necessary to assign a MAC address if no local MAC address exists (e.g. no Security Memory).



---

**Important:**

Changing the MAC Address always requires a reboot of the PROFINET IO Device Stack!

---

- Restore the remanent data using the *Load Remanent Data Service* if remanent data is not handled by the stack but by the application.
- Set the OEM parameters using the *Set OEM Parameters Service* if the Hilscher default values are insufficient.
- Register the application using the *Register Application Service* in order to receive indications from the PROFINET IO Device stack if this is required.
- Configure the IOxS handling with the Set IOXS Config Service if IOxS access is required.
- Configure the device using the *Set Configuration Service*. This means providing the device with all parameters needed for operation. These include both basic parameters for identification such as *NameOfStation*, *DeviceID* and *VendorID* as well as the module configuration. This module configuration contains information about the APIs, modules and submodules the stack will use. When the stack returns the *Set\_Configuration* packet back to the application, the given configuration has been evaluated completely and prepared for applying. If Dual-Port-Memory or Shared Memory Interface is in use it will indicate the new configuration to the user by setting the bits “Configuration new” and “Restart required” in the communication COS register.
- Perform the Channel Initialization/ConfigReload (see reference [3]) to take over the new configuration and cause the stack to use the new parameters. After this the stack is ready to start communication with an IO-Controller.

A graphical representation of this sequence is shown in Figure 6.

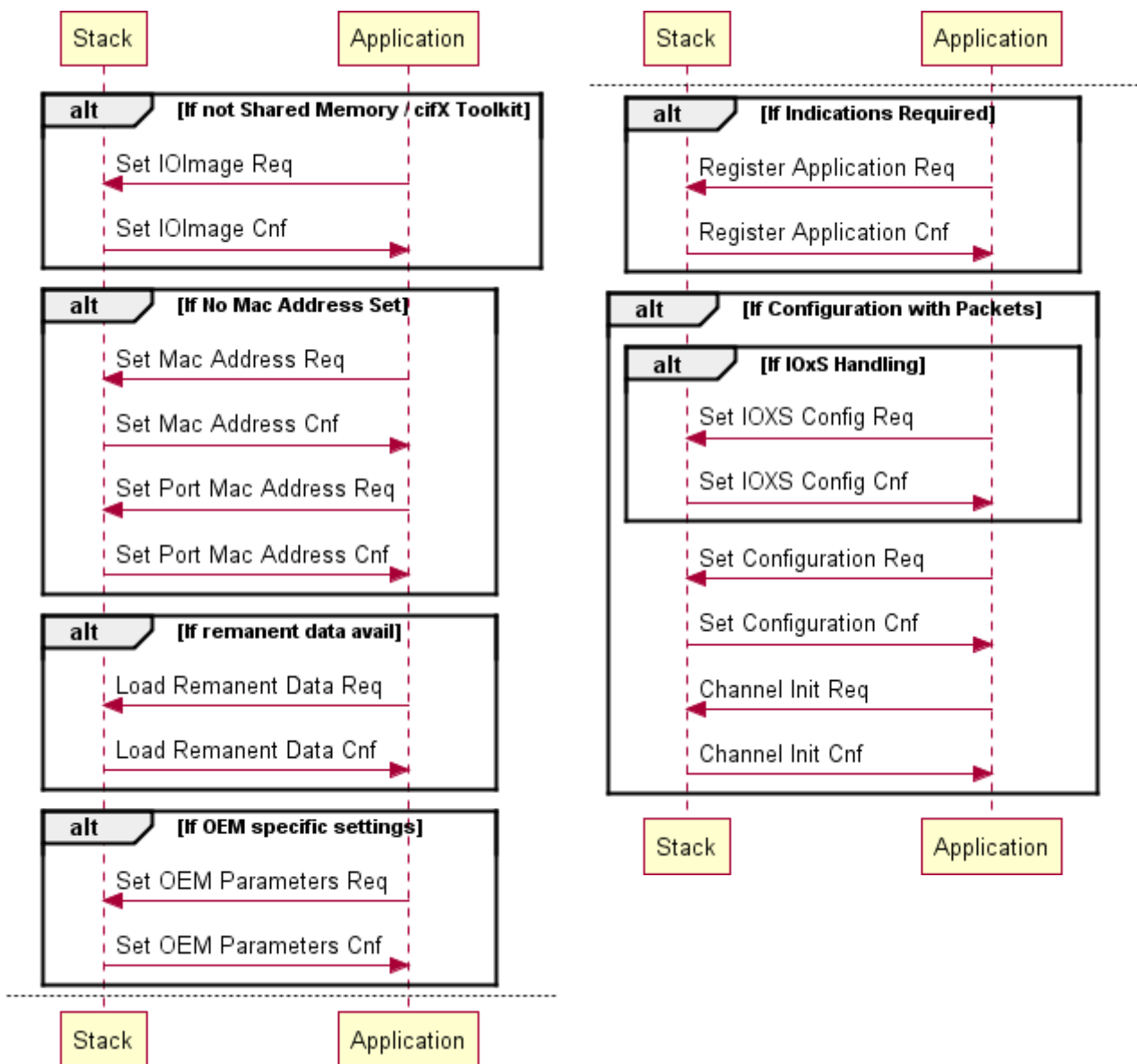


Figure 10: Configuring the PROFINET IO Device Stack

#### 7.1.4.1 Remark on Reconfiguration

It is possible to reconfigure the stack at any time. To do so, simply send a new configuration to the stack followed by a Channel Init Request (see reference [3]). Sending the new configuration without the Channel Init Request will not have an effect on any running communication. The new parameters will simply be stored. Sending the Channel Init Request will stop any communication and take over the new parameters.

## 7.1.5 Set Configuration Service

This packet has to be sent by the application. The Set Configuration Service shall be used by application to configure the stack on startup.

Using the Set Configuration packet the application provides information about the API, the modules and the submodules to the stack. This module configuration may be changed later at runtime using the Pull (see sections 6.4.9 and 6.4.10) and Plug services (see sections 6.4.7 and 6.4.8).



### Attention:

As described in Dual Port Memory manual (Ref. 4), it is **required** to send a Channel Initialization request (Channellnit) to the protocol stack after the Set Configuration Request has been performed. The stack will not use the configuration until the Channel Initialization Request has been received.



### Attention:

If the stack is configured using the Set Configuration Service, the NameOfStation and the IP Parameters must be handled by the application. In turn the application must register with the stack in order to receive the relevant indications (See section *Register Application Service*).

This attention is not applied if bit D17 (PNS\_IF\_SYSTEM\_NAME\_IP\_HANDLING\_BY\_STACK\_ENABLED) in System flags is set, see Table 32.

### 7.1.5.1 Set Configuration Request

The application has to send this request to the protocol stack. As the configuration of modules and submodules can (almost) freely be defined by the application, this packet cannot have a fixed layout. This packet has to be considered as a data container.

If the stack is accessed via Dual Port Memory, the application may need to fragment the request packet due to the limited mailbox size. The handling of the fragmented service by the application and the stack is described in reference [3].

structure PNS_IF_SET_CONFIGURATION_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Header	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32		Packet data length in bytes. This is a value depending on the amount of submodules.
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet

structure PNS_IF_SET_CONFIGURATION_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
	ulSta	UINT32	0	Status not in use for request.
	ulCmd	UINT32	0x1FE2	PNS_IF_SET_CONFIGURATION_REQ-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_SET_CONFIGURATION_REQ_DATA_T			
	ulTotalConfigPckLen	UINT32		Length (in bytes) of the entire configuration data. If the size of the whole Set_Configuration Request exceeds the DPM mailbox size sequenced mechanisms have to be used. This parameter in the very first packet shall contain the complete size of all sequenced packets (without the packet headers).
	tDeviceParameters	PNS_IF_DEVICE_PARAMETER_T		The structure describing the device parameters, see explanation below.
	tModuleConfig	PNS_IF_MODULE_CFG_REQ_DATA_T		The structure describing APIs and submodules, see explanation below.
	tSignalConfig	PNS_IF_SIGNAL_CFG_REQ_DATA_T		This optional structure is needed if little endian byte order shall be used. It describes the data structure of each submodule

Table 30: PNS\_IF\_SET\_CONFIGURATION\_REQ\_T - Set Configuration Request

tDeviceParameters is structured like this:

```
typedef struct PNS_IF_DEVICE_PARAMETER_Ttag
{
    TLR_UINT32 ulSystemFlags;
    TLR_UINT32 ulWdgTime;
    TLR_UINT32 ulVendorId;
    TLR_UINT32 ulDeviceId;
    TLR_UINT32 ulMaxAr;
    TLR_UINT32 ulCompleteInputSize;
    TLR_UINT32 ulCompleteOutputSize;
    TLR_UINT32 ulNameOfStationLen;
    TLR_UINT8  abNameOfStation[PNIO_MAX_NAME_OF_STATION];
    TLR_UINT32 ulTypeOfStationLen;
    TLR_UINT8  abTypeOfStation[PNIO_MAX_TYPE_OF_STATION];
    TLR_UINT8  abDeviceType[PNS_IF_MAX_DEVICE_TYPE_LEN +3];
    TLR_UINT8  abOrderId[PNIO_MAX_ORDER_ID];
    TLR_UINT32 ulIpAddr;
    TLR_UINT32 ulNetMask;
    TLR_UINT32 ulGateway;
    TLR_UINT16 usHwRevision;
    TLR_UINT16 usSwRevision1;
    TLR_UINT16 usSwRevision2;
    TLR_UINT16 usSwRevision3;
    TLR_UINT8  bSwRevisionPrefix;
    TLR_UINT8  bReserved;
    TLR_UINT16 usMaxDiagRecords;
    TLR_UINT16 usInstanceId;
    TLR_UINT16 usReserved;
} PNS_IF_DEVICE_PARAMETER_T;
```



Data	structure tDeviceParameters		
ulSystemFlags	UINT32		Flags for system behavior. See below.
ulWdgTime	UINT32	Default value: 1000  Allowed values: 0, 20...65535	Watchdog time (in milliseconds). 0 = Watchdog timer has been switched off
ulVendorId	UINT32	1..65279 (= 0xFEFF)*, Hilscher: 286 (decimal)/ 0x011E (hex)	Vendor ID: Vendor identification number of the manufacturer, which has been assigned to the vendor by the PROFIBUS Nutzerorganisation e. V. All Hilscher products use the value 0x011E.
ulDeviceId	UINT32	1 ... (2 <sup>16</sup> - 1)*, e.g. for cifX 50-RE:: 259 (decimal)/ 0x103 (hex)	Device ID This is an identification number of the device, freely eligibly by the manufacturer, which is fixed and unique for every device.
ulMaxAr	UINT32	0	Currently not used. Set to zero.
ulCompleteInputSize	UINT32	Default value:128 Allowed values: 0.. 1440 Bytes	Maximum amount of allowed input data. The sum of data of all submodules configured by the user must not exceed this value. This field references input data as data received by the IO-Device.
ulCompleteOutputSize	UINT32	Default value:128 Allowed values: 0.. 1440 Bytes	Maximum amount of allowed output data. The sum of data of all submodules configured by the user must not exceed this value. This field references output data as data sent by the IO-Device.
ulNameOfStationLen	UINT32	0..240	Length of NameOfStation
abNameOfStation[240]	UINT8[]		The NameOfStation as ASCII char-array. If bit D17 in the system flags ulSystemFlags is set the stack doesn't evaluate this parameter, the stack will use the NameOfStation saved in remanent data.
ulTypeOfStationLen	UINT32	1..240	Length of TypeOfStation
abTypeOfStation[240]	UINT8[]		The TypeOfStation as ASCII char-array.
abDeviceType[28]	UINT8[]		The DeviceType as ASCII char-array. The last 3 bytes are reserved padding bytes and shall be set to zero.
abOrderId[20]	UINT8[]		The OrderID as ASCII char-array.
ulIpAddr	UINT32	Valid IP address, default: 0.0.0.0	IP address. If bit D17 in the system flags ulSystemFlags is set, the stack does not evaluate this parameter. The stack will use the IP address saved in remanent data.
ulNetMask	UINT32	Valid network mask, default: 0.0.0.0	Network mask. If bit D17 in the system flags ulSystemFlags is set, the stack does not evaluate this parameter. The stack will use the network mask saved in remanent data.
ulGateway	UINT32	Valid gateway address, default: 0.0.0.0	Gateway address. If bit D17 in the system flags ulSystemFlags is set, the stack does not evaluate this parameter. The stack will use the gateway address saved in remanent data.

usHwRevision	UINT16	0..0xFFFF, default: 0	Hardware Revision
usSwRevision1	UINT16	0..0xFFFF, default: 0	Software Revision 1
usSwRevision2	UINT16	0..0xFFFF, default: 0	Software Revision 2
usSwRevision3	UINT16	0..0xFFFF, default: 0	Software Revision 3
bSwRevisionPrefix	UINT8	'V', 'R', 'P', 'U', 'T'	Software Revision Prefix Possible values and their meanings are: 'V': Released version 'R': Revision 'P': Prototype 'U': Under field test 'T': Test device
bReserved	UINT8	0	Reserved, set to zero.
usMaxDiagRecords	UINT16	1..0xFFFF, default: 0	The maximum number of user diagnosis records the stack is able to handle in parallel.  <b>Note:</b> This field has influence on the amount of memory the stack requires.
usInstanceId	UINT16	0..0xFFFF, default: 0	Instance ID. This parameter must match to value <code>ObjectUUID_LocalIndex</code> in the GSDML file corresponding to the IO-Device. The value 1 is recommended.
usReserved	UINT16	0	Reserved for future use, set to zero

Table 31: Structure *tDeviceParameters*

Bit No	Behavior to influence	Flag Define/ Desired Behavior
D0	Bus State after applying configuration  For more information concerning this topic see section 4.4.1 "Controlled or Automatic Start" of the <i>netX DPM Interface Manual</i> .	0 - <code>PNS_IF_SYSTEM_START_AUTO_START</code> : The stack will enable the Bus (Network access) right after Channel Initialization.  1 - <code>PNS_IF_SYSTEM_START_APPL_CONTROLLED</code> : The stack disables the Bus after Channel Initialization. Application shall use Start/Stop Communication Service to enable Bus.
D3	Byte order of Process Data	0 - <code>PNS_IF_SYSTEM_BYTEORDER_BIG_ENDIAN</code> : The stack uses big endian byte order for process data.  1 - <code>PNS_IF_SYSTEM_BYTEORDER_LITTLE_ENDIAN</code> : The stack uses little endian byte order for process data. <b>ATTENTION: Using this mode causes a lot of CPU usage and may have a bad impact on minimum reachable cycle time.</b>
D8	Identification & Maintenance Handling	0 - <code>PNS_IF_SYSTEM_STACK_HANDLE_I_M_DISABLED</code> : I&M Requests are only parsed by the stack and will be forwarded to the application using Read I&M Indication and Write I&M Indication. The application must handle the I&M0-3 Data sets.  1 - <code>PNS_IF_SYSTEM_STACK_HANDLE_I_M_ENABLED</code> : I&M Requests are handled internally by the stack. I&M 0-4 is supported on DAP and PDEV submodules.
D9	Automatic Application Ready if no application registered	0 - <code>PNS_IF_SYSTEM_ARDY_WOUT_APPL_REG_DISABLED</code> : The Stack will never generate an Application Ready Sequence if no application is registered: Therefore the application must register with the stack in order to establish an AR.

		<p>1 - PNS_IF_SYSTEM_ARDY_WOUT_APPL_REG_ENABLED: The stack will generate an Application Ready automatically if no application is registered. Use this flag with care: the stack has no information about the Readiness of the application. <b>Therefore sending Application Ready automatically can be dangerous.</b></p>
D11	Generate Check Indications for matching submodules	<p>0 - PNS_IF_SYSTEM_CHECK_IND_ALL_MODULES_DISABLED : The stack will not generate a Check Indication for expected submodules that match the configuration.</p> <p>1 - PNS_IF_SYSTEM_CHECK_IND_ALL_MODULES_ENABLED: The stack will generate a Check Indication for expected submodules that match the configuration. E.g. the application will receive a Check Indication for each submodule owned by an AR.</p>
D12	Handle Link Down as Fatal Error	<p>0 - PNS_IF_SYSTEM_HANDLE_LINK_DOWN_AS_FATAL_DISABLED: The Stack will not treat link down events as fatal error.</p> <p>1 - PNS_IF_SYSTEM_HANDLE_LINK_DOWN_AS_FATAL_ENABLED: The Stack will treat link down events as fatal error. Among other things, the stack will disable the bus (Network access) in case of a link down.</p>
D13	Generate Check Indications for unused modules	<p>0 - PNS_IF_SYSTEM_CHECK_IND_UNUSED_MODULES_DISABLED: The stack does not generate a Check Indication for modules not used by any AR.</p> <p>1 - PNS_IF_SYSTEM_CHECK_IND_UNUSED_MODULES_ENABLED: The stack will generate a Check Indication for each submodule not used by any AR right before generating the Connect Request Done Indication. If another controller starts establishing an AR while these Check Indications are generated, the stack will stop generating these Check Indications and restart after processing another ARs expected submodules.</p>
D14	Remanent Data Handling	<p>0 - PNS_IF_SYSTEM_DISABLE_STORE_REMANENT_DISABLE: The stack will store the remanent data by itself if DPM/SHM is in use and non volatile storage (flash memory) is available.</p> <p>1 - PNS_IF_SYSTEM_DISABLE_STORE_REMANENT_ENABLED: The stack will never store remanent data by itself. This needs to be implemented in the application.</p>
D15	Generate Link State change Indications	<p>0 - PNS_IF_SYSTEM_ENABLE_LINK_STATE_INDICATION: The stack will generate a Link Status Changed Indication if the link state changes on any port.</p> <p>1 - PNS_IF_SYSTEM_DISABLE_LINK_STATE_INDICATION: The stack not generate Link Status Changed Indication.</p>
D16	Check if IO-data offsets in IO-image	<p>0 - PNS_IF_SYSTEM_ENABLE_IO_OFFSET_CHECKING: The stack will check if IO-data offsets in IO-image (or DPM) are configured properly</p> <p>1 - PNS_IF_SYSTEM_DISABLE_IO_OFFSET_CHECKING: The stack will newer check if IO-data offsets in IO-image (or DPM) are configured properly. <b>ATTENTION: disabling this check may lead to inconsistent IO-data in case of a faulty application configuration. Disabling this check will lead to invalid stack response to the service RCX_GET_DPM_IO_INFO_REQ!</b></p>
D17	Device name and IP-parameters Handling  <b>Note: This feature is supported starting with stack version V3.5.46.0</b>	<p>0 - PNS_IF_SYSTEM_NAME_IP_HANDLING_BY_STACK_DISABLED: The stack will never store IP-parameters (IP-address, network mask and gateway) and device name by itself . An application should take care to store them on indications PNS_IF_SAVE_STATION_NAME_IND, PNS_IF_SAVE_IP_ADDR_IND and reset them on PNS_IF_RESET_FACTORY_SETTINGS_IND</p> <p>1 - PNS_IF_SYSTEM_NAME_IP_HANDLING_BY_STACK_ENABLED: The stack will store IP-parameters and device name by itself . An</p>

		<p>Application shouldn't take care to store them.</p> <p><b>The stack should have a possibility to save the Remanent Data: either DPM/SHM is in use and non volatile storage (flash memory) is available or application (using the linkable object module) supports Load Remanent Data Service and Store Remanent Data Service.</b></p>
--	--	---

Table 32: System flags to use for configuration of the Stack.

tModuleConfig is structured like this:

```
typedef struct PNS_IF_MODULE_CFG_REQ_DATA_Ttag
{
    /* number of API elements to follow */
    TLR_UINT32          ulNumApi;
} PNS_IF_MODULE_CFG_REQ_DATA_T;
```

- It contains the number of API elements of the tModuleConfig structure as first element. This is stored in variable ulNumApi. If you want to configure the device for using m APIs, you set ulNumApi to the value m.
- For every additional API a structure describing it and its submodules follows the last submodule structure of the preceding API. Assume this API should be configured for using n submodules.

structure PNS_IF_API_STRUCT_T				
Area	Variable	Type	Value / Range	Description
	ulApi	UINT32	0..m-1	The number of the API profile to be configured. 0 indicates "manufacturer specific". Currently only one single API is supported, so only the value 0 makes sense.
	ulNumSubmoduleItems	UINT32	1..n	Number of submodule-items this API contains. These items follow directly behind this entry.

Table 33: Structure PNS\_IF\_API\_STRUCT\_T

For every submodule of the API a structure describing it follows the field `ulNumSubmoduleItems`.

structure PNS_IF_SUBMODULE_STRUCT_T				
Area	Variable	Type	Value / Range	Description
	usSlot	UINT16		The slot this submodule belongs to.
	usSubslot	UINT16		The subslot this submodule belongs to.
	ulModuleID	UINT32		The ModuleID of the module this submodule belongs to.
	ulSubmoduleID	UINT32		The SubmoduleID of this submodule.
	ulProvDataLen	UINT32	0..1440	The length of data provided by this submodule. This length describes the data sent by IO-Device and received by IO-Controller.
	ulConsDataLen	UINT32	0..1440	The length of data consumed by this submodule. This length describes the data sent by IO-Controller and received by IO-Device.
	ulDPMOffsetIn	UINT32		Offset in DPM InputArea or in Input-image (in pbConsImage, see 6.1.14) where consumed data for the submodule shall be copied to. This data is received by IO-Device and sent by IO-Controller. If the length of data in this direction is 0 this value shall be set to 0. See section 6.1.3 for further information
	ulDPMOffsetOut	UINT32		Offset in DPM OutputArea or in Output-image (in pbProvImage, see 6.1.14) where provided data of the submodule shall be taken from. This data is sent by the IO-Device and received by the IO-Controller. If the length of thze data in this direction is 0 this value shall be set to 0. See section 6.1.3 for further information
	usOffsetIOPSP rovider	UINT16		Offset where the stack shall take the IOPS provider state for this submodule relative to beginning of IOPS block in DPM output area from. See section 6.1.3 for further information
	usOffsetIOPSC onsumer	UINT16		Offset where the stack shall put the IOPS consumer state of this submodule relative to beginning of IOPS block in DPM input area to. See section 6.1.3 for further information
	usOffsetIOCSP rovider	UINT16		Offset where the stack shall take the IOCS provider state for this submodule relative to beginning of IOCS block in DPM output area from. See section 6.1.3 for further information
	usOffsetIOCSC onsumer	UINT16		Offset where the stack shall put the IOCS consumer state for this submodule relative to beginning of IOCS block in DPM input area to. See section 6.1.3 for further information
	ulReserved	UINT32	0	Reserved for future use. Set to zero.

Table 34: Structure `PNS_IF_SUBMODULE_STRUCT_T`

- If you configure more than one API ( $m > 1$ ), then `PNS_IF_API_STRUCT_T` and  $n$  times `PNS_IF_SUBMODULE_STRUCT_T` will follow for each additional API.



**Note:**

**Here the value  $n$  can be chosen individually for each API, thus the number of submodules of the different APIs of the configuration may differ!**

If little endian byte order shall be used for the process data image, the user has to provide the stack with detailed information about the structure of the cyclic input and output data. For this the user has to create one `IO_SIGNALS_CONFIGURE_SIGNAL_REQ_DATA_T` element for each submodule and each direction the submodule has data for. (Two structures are needed for combined input/output modules. No structure is required for submodules which do not have any input/output data)

`tSignalConfig` is structured like this:

```
typedef struct PNS_IF_SIGNAL_CFG_REQ_DATA_Ttag
{
    /* number of signals to follow */
    TLR_UINT32          ulNumSignals;

    /* array of ulNumSignals count IO_SIGNALS_CONFIGURE_SIGNAL_REQ_DATA_T
     * structures follows */
} PNS_IF_SIGNAL_CFG_REQ_DATA_T
```

- It contains the number of `IO_SIGNALS_CONFIGURE_REQ_DATA_T` elements included in `tSignalConfig` as first element. This is stored in the variable `ulNumSignals`. If you want to configure the device for using `n` Signals, you set `ulNumSignals` to the value `n`.
- After this field an (packed) array of `n` elements of type `IO_SIGNALS_CONFIGURE_REQ_DATA_T` follows. The structure of `IO_SIGNALS_CONFIGURE_REQ_DATA_T` is the same as the data part of the Configure Signal Request described in section 6.1.16.1.

### 7.1.5.2 Set Configuration Confirmation

This confirmation will be returned to the application.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T      PNS_IF_SET_CONFIGURATION_CNF_T;
```

#### Packet Description

structure PNS_IF_SET_CONFIGURATION_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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 below.
	ulCmd	UINT32	0x1FE3	PNS_IF_SET_CONFIGURATION_CNF -Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 35: PNS\_IF\_SET\_CONFIGURATION\_CNF\_T - Set Configuration Confirmation

### 7.1.5.3 Behavior when receiving a Set Configuration Command

The following rules apply for the behavior of the PROFINET IO RT/IRT Device protocol stack when receiving a set configuration command:

- The configuration packet's name is
  - `PNS_IF_SET_CONFIGURATION_REQ` for the request packet and
  - `PNS_IF_SET_CONFIGURATION_CNF` for the confirmation packet.
- The configuration data are checked for consistency and integrity.
- In case of failure no data are accepted.
- In case of success the configuration parameters are stored internally (within the RAM).
- The new configuration is not processed until a channel init is performed!
- No automatic registration of the application at the stack happens.
- The confirmation packet `PNS_IF_SET_CONFIGURATION_CNF` only transfers simple status information, but does not repeat the whole parameter set.



### 7.1.6 Register Application Service

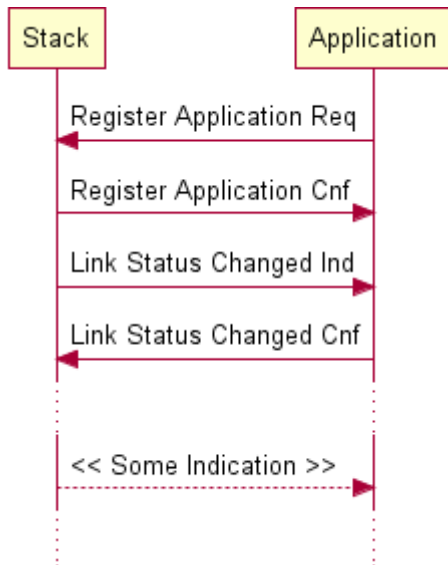


Figure 11: Register Application Service Packet sequence

Using the Register Application Service the user application provides the stack an endpoint to send indications to. If the user application is registered at the stack, it will receive the following indications (if the event triggering any of these indications occurs):

- Save Station Name Indication
- Save IP Address Indication
- Reset Factory Settings Indication
- AR Check Indication
- Check Indication
- Connect Request Done Indication
- Parameterization Speedup Support Indication
- Parameter End Indication
- Store Remanent Data Indication
- AR InData Indication
- APDU Status Changed Indication
- Read Record Indication
- Write Record Indication
- Read I&M Indication
- Write I&M Indication
- Alarm Indication
- AR Abort Indication Indication
- Release Request Indication
- Start LED Blinking Indication
- Stop LED Blinking Indication
- Link Status Changed Indication
- Error Indication
- Event Indication

**Note:**

It is required that the application returns all indications it receives as valid responses to the stack. Especially it is not allowed to change any field in packet header except `ulSta`, `ulCmd` and `ulLen`. Otherwise the stack will not be able to handle and assign the response successfully.

The service is described in reference [3].

### 7.1.6.1 Register Application Request

This service is described in “DPM Interface Manual for netX based Products” [4]. The stack will generate an initial Link Status Changed Indication when this request is received (See Figure 7: Register Application Service).

### 7.1.6.2 Register Application Confirmation

This service is described in reference [3].

### 7.1.6.3 Register Application for Selective Indications Only

Starting with PROFINET IO-Device protocol stack version V3.5.50.0 the stack offers the “selective indications only” function. With this function it is possible for an application to handle only the indications the application is interested in. For all other indications it is possible to implement a default handling in the application. In that case, the stack behaves in the same way as “if no application would be registered”.

To activate this service there is no special handling required. Just the regular Register Application service is used as described in section *Register Application Request* on page 74.

If the application does not want to handle a specific indication it is required that the application returns this indications as responses to the stack changing only **two fields** in the packet header and does not change anything in the data part of the other fields in the header of the indication:

```
ulSta = TLR_E_NO_APPLICATION_REGISTERED; /*unhandled indication*/
ulCmd = ulCmd | 1;                      /*response command*/
```

To meet the conditions of PROFINET certification in such a case, the application has to fulfill the following conventions:

Unhandled indication	Conventions
Store Remanent Data	Flag <code>PNS_IF_SYSTEM_DISABLE_STORE_REMANENT_DISABLE</code> has to be used in Set Configuration Request
Save Station Name and Save IP Address	Flag <code>PNS_IF_SYSTEM_NAME_IP_HANDLING_BY_STACK_ENABLED</code> has to be used in Set Configuration Request
Read I&M and Write I&M	Flag <code>PNS_IF_SYSTEM_STACK_HANDLE_I_M_ENABLED</code> has to be used in Set Configuration Request
Event Indication	The application has to update cyclically the provider data because of skipping the event <code>PNS_IF_IO_EVENT_PROVIDER_UPDATE_REQUIRED</code>
Parameter End	Flag <code>PNS_IF_SYSTEM_ARDY_WOUT_APPL_REG_ENABLED</code> has to be used in Set Configuration Request
Start LED Blinking Stop LED Blinking	If the the application is working with the Linkable Object Module it must be possible to identify the PROFINET device visually so the application has to implement LED blinking.

## **7.1.7 Unregister Application Service**

Using this service the application can unregister with the PROFINET Device stack: the stack will not generate indications any more.

This service is described in reference [3].

### **7.1.7.1 Unregister Application Request**

This service is described in reference [3].

### **7.1.7.2 Unregister Application Confirmation**

This service is described in reference [3].

## 7.1.8 Register Fatal Error Callback Service

With this Service the user application can register a fatal error callback function to the stack. In case of a fatal error the stack will call this function to allow the application to perform some needed actions (e.g. set modules to a safe state).

Some examples of fatal errors that lead to calling the callback function:

- Resource problem inside the stack (empty packet pool, full packet queue, no free memory)
- Detection of a configuration error (e.g. inconsistent information about modules and its parameters)

---

**Note 1:** If the application is not running locally on the netX this functionality is NOT available. In this case the stack is informed about the error with a packet as described in section *Error Indication Service* of this document. Therefore this service is NOT available if Dual port Memory is used.

---



---

**Note 2:** It is necessary that the applications callback function returns. It is not allowed for the callback function to enter any kind of while(1) loop.

---

### 7.1.8.1 Register Fatal Error Callback Request

With this request the application hands over the pointer of its error callback function. It is also possible for the application to add one user parameter which will be handed over while calling the callback function. It may contain the pointer to applications task resources or anything else application may need.

#### Packet Structure Reference

```
/* Request packet */
typedef struct PNS_REG_FATAL_ERROR_CALLBACK_REQ_DATA_Ttag
{
    PNS_FATAL_ERROR_CLB    pfnClbFnc;
    TLR_VOID*              pvUserParam;
} PNS_REG_FATAL_ERROR_CALLBACK_REQ_DATA_T;

typedef struct PNS_REG_FATAL_ERROR_CALLBACK_REQ_Ttag
{
    TLR_PACKET_HEADER_T    tHead;
    PNS_REG_FATAL_ERROR_CALLBACK_REQ_DATA_T    tData;
} PNS_REG_FATAL_ERROR_CALLBACK_REQ_T;
```

## Packet Description

structure PNS_REG_FATAL_ERROR_CALLBACK_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... 2 <sup>32</sup> -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 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for request.
	ulCmd	UINT32	0x1FDA	PNS_REGISTER_FATAL_ERROR_CALLBACK_REQ-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_REG_FATAL_ERROR_CALLBACK_REQ_DATA_T			
	pfnClbFnc	PNS_FATAL_ERROR_CLB		The pointer to the callback function. Definition see below.
	pvUserParam	void *		This user specific parameter is handed over to the callback function if it is called.

Table 36: PNS\_REG\_FATAL\_ERROR\_CALLBACK\_REQ\_T - Register Fatal Error Callback Request



### Note:

The definition of the type of callback function shall be as follows:

```
typedef TLR_VOID(*PNS_IF_FATAL_ERROR_CLB)(      TLR_UINT32 ulErrorCode,
TLR_VOID* pvUserParam);
```

### 7.1.8.2 Register Fatal Error Callback Confirmation

With this packet the stack informs the application about the success of registering the fatal error callback function.

### Packet Structure Reference

```
/* Confirmation packet */
typedef TLR_EMPTY_PACKET_T      PNS_REG_FATAL_ERROR_CALLBACK_CNF_T;
```

## Packet Description

structure PNS_REG_FATAL_ERROR_CALLBACK_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1FDB	PNS_REGISTER_FATAL_ERROR_CALLBACK_CNF - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch

Table 37: PNS\_REG\_FATAL\_ERROR\_CALLBACK\_CNF\_T - Register Fatal Error Callback Confirmation

## 7.1.9 Unregister Fatal Error Callback Service

Using this service the user application can unregister a previously registered error callback function from the stack. After unregistering this callback the application will only report fatal errors using the service described in 6.3.13.

### 7.1.9.1 Unregister Fatal Error Callback Request

Using this request deletes the registered fatal error callback handle inside the stack.

#### Packet Structure Reference

```
/* Request packet */
typedef TLR_EMPTY_PACKET_T PNS_UNREG_FATAL_ERROR_CALLBACK_REQ_T;
```

#### Packet Description

structure PNS_UNREG_FATAL_ERROR_CALLBACK_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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, untouched
	ulSta	UINT32	0	Status not used for requests. Set to zero.
	ulCmd	UINT32	0x1FDE	PNS_UNREGISTER_FATAL_ERROR_CALLBACK_REQ - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	0	Routing, do not touch

Table 38: PNS\_UNREG\_FATAL\_ERROR\_CALLBACK\_REQ\_T - Unregister Fatal Error Callback Request

### 7.1.9.2 Unregister Fatal Error Callback Confirmation

Using this packet the stack informs the application about the success of unregistering the fatal error callback.

#### Packet Structure Reference

```
/* Confirmation packet */
typedef TLR_EMPTY_PACKET_T                                PNS_UNREG_FATAL_ERROR_CALLBACK_CNF_T;
```

#### Packet Description

structure PNS_UNREG_FATAL_ERROR_CALLBACK_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1FDF	PNS_UNREGISTER_FATAL_ERROR_CALLBACK_CNF - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch

Table 39: PNS\_UNREG\_FATAL\_ERROR\_CALLBACK\_CNF\_T - Unregister Fatal Error Callback Confirmation



### 7.1.10 Set Port MAC Address Service

Using this service the user application can set the two MAC addresses which are necessary for working with LLDP (Link Layer Discovery Protocol).

This protocol is part of the stack and cannot be disabled. It requires a different MAC-address for each single Ethernet port of the hardware. Therefore a hardware with 2 Ethernet ports needs at least 3 MAC-addresses.

The well defined default MAC-addresses for a netX with MAC-address from the Hilscher address range are "Chassis MAC-address +1" and "Chassis MAC-address +2" for the 2 Ethernet ports".

If for any reason this rule is not acceptable, the user can force the stack to use any other Port MAC-address for LLDP.



#### Note:

If the user application wants to set the "Chassis MAC-address" it has to use the "Set MAC Address" (RCX\_SET\_MAC\_ADDR\_REQ) service (see section "4.16 Set MAC Address" of reference [3]).

#### 7.1.10.1 Set Port MAC Address Request

This packet is optional. The two addresses are generated by default by simply adding the values 1 respectively 2 to the Chassis MAC address.

**Important:** This packet must be sent prior to the Set Configuration Service packet, otherwise it will be rejected and the former choice of the Port MAC address will be fixed.

**Important:** This packet may be sent to the stack either once or never at all. Multiple use of this packet is not allowed.

#### Packet Structure Reference

```
typedef TLR_UINT8 PORT_MAC_ADDR_T[6];

typedef struct PNS_IF_SET_PORT_MAC_REQ_DATA_Ttag
{
    PORT_MAC_ADDR_T atPortMacAddr[2];
} PNS_IF_SET_PORT_MAC_REQ_DATA_T;

typedef struct PNS_IF_SET_PORT_MAC_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_SET_PORT_MAC_REQ_DATA_T tData;
} PNS_IF_SET_PORT_MAC_REQ_T;
```

## Packet Description

structure PNS_IF_SET_PORT_MAC_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	12	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32		Status not used for request.
	ulCmd	UINT32	0x1FE0	PNS_IF_SET_PORT_MAC_REQ- Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_SET_PORT_MAC_REQ_DATA_T			
	atPortMacAddr[2]	PORT_MAC_ADDR_T		Structure containing the two port MAC addresses. atPortMacAddr[0] stores MAC-address for Port 0, atPortMacAddr[1] stores MAC-address for Port .

Table 40: PNS\_IF\_SET\_PORT\_MAC\_REQ\_T - Set Port MAC Address Request

### 7.1.10.2 Set Port MAC Address Confirmation

The stack informs the application about the success or failure of setting the port MAC address.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_SET_PORT_MAC_CNF_T;
```

**Packet Description**

structure PNS_IF_SET_PORT_MAC_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1FE1	PNS_IF_SET_PORT_MAC_CNF - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch

*Table 41: PNS\_IF\_SET\_PORT\_MAC\_CNF\_T - Set Port MAC Address Confirmation*

### 7.1.11 Set OEM Parameters Service

For some special needs (like I&M) of the user this request is used. It is designed for multiple purposes and may be extended in the future.

The stack is able to answer to Information and Maintenance Requests. Normally well defined Hilscher default values are used. If this is not sufficient for the application the application can change this default values using this service.



**Note:**

This service has to be used before using the *Set Configuration Service* to configure the stack.

#### 7.1.11.1 Set OEM Parameters Request

Using this packet the application can hand over some parameter values to the stack e.g. to use for handling I&M0 requests.



**Important:**

This packet must be sent prior to the *Set Configuration Service* packet, otherwise it will be rejected and the well defined Hilscher default values will be used.



**Important:**

This packet may be sent to the stack **either once or never at all**. Multiple use of this packet is not allowed.

#### Packet Structure Reference

```
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_1      0x01
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_2      0x02
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_3      0x03
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_4      0x04
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_5      0x05
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_6      0x06
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_7      0x07

typedef struct PNS_IF_SET_OEM_PARAMETERS_TYPE_1_Ttag
{
    TLR_UINT8  abSerialNumber[16];
    TLR_UINT16 usProfileId;
    TLR_UINT16 usRevisionCounter;
    TLR_UINT16 usProfileSpecificType;
} PNS_IF_SET_OEM_PARAMETERS_TYPE_1_T;

typedef struct PNS_IF_SET_OEM_PARAMETERS_TYPE_2_Ttag
{
    TLR_UINT8  abSerialNumber[16];
} PNS_IF_SET_OEM_PARAMETERS_TYPE_2_T;

typedef struct PNS_IF_SET_OEM_PARAMETERS_TYPE_3_Ttag
{
    TLR_UINT32 ulTimeout;
} PNS_IF_SET_OEM_PARAMETERS_TYPE_3_T;

typedef struct PNS_IF_SET_OEM_PARAMETERS_TYPE_4_Ttag
{
    TLR_UINT16 usSystemNameLen;
    TLR_UINT8  abSystemName[255];
} PNS_IF_SET_OEM_PARAMETERS_TYPE_4_T;
```

```

typedef struct PNS_IF_SET_OEM_PARAMETERS_TYPE_5_Ttag
{
    TLR_UINT32 ulIMFlag;
} PNS_IF_SET_OEM_PARAMETERS_TYPE_5_T;

typedef struct PNS_IF_SET_OEM_PARAMETERS_TYPE_6_Ttag
{
    TLR_UINT16 usIRTCycleCounterOffset;
} PNS_IF_SET_OEM_PARAMETERS_TYPE_6_T;

typedef struct PNS_IF_SET_OEM_PARAMETERS_TYPE_7_Ttag
{
    TLR_BOOLEAN fUseOlfLinkStateCommandCode;
} PNS_IF_SET_OEM_PARAMETERS_TYPE_7_T;

typedef struct PNS_IF_SET_OEM_PARAMETERS_REQ_DATA_Ttag
{
    TLR_UINT32 ulParameterType;
} PNS_IF_SET_OEM_PARAMETERS_REQ_DATA_T;

typedef union PNS_IF_SET_OEM_PARAMETERS_UNION_Ttag
{
    PNS_IF_SET_OEM_PARAMETERS_TYPE_1_T      tType1Param;
    PNS_IF_SET_OEM_PARAMETERS_TYPE_2_T      tType2Param;
    PNS_IF_SET_OEM_PARAMETERS_TYPE_3_T      tType3Param;
    PNS_IF_SET_OEM_PARAMETERS_TYPE_4_T      tType4Param;
    PNS_IF_SET_OEM_PARAMETERS_TYPE_5_T      tType5Param;
    PNS_IF_SET_OEM_PARAMETERS_TYPE_6_T      tType6Param;
    PNS_IF_SET_OEM_PARAMETERS_TYPE_7_T      tType7Param;
} PNS_IF_SET_OEM_PARAMETERS_UNION_T;

typedef struct PNS_IF_SET_OEM_PARAMETERS_REQ_Ttag
{
    TLR_PACKET_HEADER_T      tHead;
    PNS_IF_SET_OEM_PARAMETERS_REQ_DATA_T      tData;
    PNS_IF_SET_OEM_PARAMETERS_UNION_T      tParam;
} PNS_IF_SET_OEM_PARAMETERS_REQ_T;

```

## Packet Description

structure PNS_IF_SET_OEM_PARAMETERS_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32} - 1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	4 + n	Packet data length in bytes. n depends on the parameter type contained in the packet.
	ulId	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32		Status not used for request.
	ulCmd	UINT32	0x1FE8	PNS_IF_SET_OEM_PARAMETERS_REQ-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

structure PNS_IF_SET_OEM_PARAMETERS_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Data	structure PNS_IF_SET_OEM_PARAMETERS_REQ_DATA_T			
	ulParamType	UINT32		PNS_IF_SET_OEM_PARAMETERS_TYPE_1, ..., PNS_IF_SET_OEM_PARAMETERS_TYPE_7 This parameter specifies which structure follows. See below.
	union PNS_IF_SET_OEM_PARAMETERS_UNION_T			
				Depending on the chosen Paramtype the corresponding structure shall be used here. See below.

Table 42: PNS\_IF\_SET\_OEM\_PARAMETERS\_REQ\_T - Set OEM Parameters Request

### Coding for ulParamType = PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_1

If ulParamType is set to PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_1 the structure tType1Param of PNS\_IF\_SET\_OEM\_PARAMETERS\_UNION\_T shall be used. It is defined as follows:

structure PNS_IF_SET_OEM_PARAMETERS_TYPE_1_T				
Area	Variable	Type	Value / Range	Description
	abSerialNumber	UINT8[16]		The serial number to be used for the I&M 0 responses. The serial number shall be left aligned and space (hex 0x20) padded.
	usProfileId	UINT16		The ProfileId to be used for I&M responses.
	usRevisionCounter	UINT16		Revision counter to be used for I&M responses.
	usProfileSpecificType	UINT16		The ProfileSpecificType to be used for I&M responses.

Table 43: PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_1\_T - Set OEM Parameters for ulParamType = 1

### Coding for ulParamType = PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_2

If ulParamType is set to PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_2 the structure tType2Param of PNS\_IF\_SET\_OEM\_PARAMETERS\_UNION\_T shall be used. It is defined as follows:

structure PNS_IF_SET_OEM_PARAMETERS_TYPE_2_T				
Area	Variable	Type	Value / Range	Description
	abSerialNumber	UINT8[16]		The serial number to be used for I&M 0 responses. The serial number shall be left aligned and space (hex 0x20) padded.

Table 44: PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_2\_T - Set OEM Parameters for ulParamType = 2

### Coding for ulParamType = PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_3

If ulParamType is set to PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_3 the structure tType3Param of PNS\_IF\_SET\_OEM\_PARAMETERS\_UNION\_T shall be used. It is defined as follows:

structure PNS_IF_SET_OEM_PARAMETERS_TYPE_3_T				
Area	Variable	Type	Value / Range	Description
	ulTimeout	UINT32	0x00000BB8 (3000) ...	The timeout (in milliseconds) specifies how long the stack should wait for application packets. The value can be 3000 ms or higher.

Table 45: PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_3\_T - Set OEM Parameters for ulParamType = 3

### Coding for ulParamType = PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_4



#### Note:

Parametertype 4 is deprecated. Do not use this setting.

This PROFINET stack is an implementation with RFC 3418 support, which is SNMP MIB2 and uses the sysName object for the SystemName.

If ulParamType is set to PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_4 the structure tType4Param of PNS\_IF\_SET\_OEM\_PARAMETERS\_UNION\_T shall be used. It is defined as follows:

structure PNS_IF_SET_OEM_PARAMETERS_TYPE_4_T				
Area	Variable	Type	Value / Range	Description
	usSystemNameLen	UINT16	0..255	Length of System Name
	abSystemName	UINT8[255]		The System Name is used to identify the name of the local system (LLDP services).

Table 46: PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_4\_T - Set OEM Parameters for ulParamType = 4

Explanation: The LLDP TLV SystemName has a direct connection to the SNMP MIB2 field "sysName". According to the LLDP specification (IEEE802.1AB): "The system name field shall contain an alpha-numeric string that indicates the system's administratively assigned name. The system name should be the system's fully qualified domain name. If implementations support IETF RFC 3418, the sysName object should be used for this field."

### Coding for ulParamType = PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_5



#### Note:

Parametertype 5 is deprecated. Do not use this setting.

According to certification requirements to pass the PROFINET certification the records I&M1, I&M2, I&M3 shall be supported for one submodule by each PROFINET IO Device.

If ulParamType is set to PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_5 the structure tType5Param of PNS\_IF\_SET\_OEM\_PARAMETERS\_UNION\_T shall be used. It is defined as follows:

structure PNS_IF_SET_OEM_PARAMETERS_TYPE_5_T				
Area	Variable	Type	Value / Range	Description
	ulIMFlag	UINT32	0..15	The ulIMFlag is used to indicate which kind of I&M records should be supported. If a flag is set, the corresponding I&M record is enabled

Table 47: PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_5\_T - Set OEM Parameters for ulParamType = 5

```
#define PNS_IF_STACK_HANDLE_IM_1      0x01
#define PNS_IF_STACK_HANDLE_IM_2      0x02
#define PNS_IF_STACK_HANDLE_IM_3      0x04
#define PNS_IF_STACK_HANDLE_IM_4      0x08
```

### Coding of `ulIMFlag`

Bit No	Flag Define/ Desired Behavior
D0	If this flag is set using <code>PNS_IF_STACK_HANDLE_IM_1</code> , the stack will support the handling of I&M1 Record.
D1	If this flag is set using <code>PNS_IF_STACK_HANDLE_IM_2</code> , the stack will support the handling of I&M2 Record.
D2	If this flag is set using <code>PNS_IF_STACK_HANDLE_IM_3</code> , the stack will support the handling of I&M3 Record.
D3	If this flag is set using <code>PNS_IF_STACK_HANDLE_IM_4</code> , the stack will support the handling of I&M4 Record.
D4 – D15	Reserved, set to zero

Table 48: Coding of `ulIMFlag`



#### Note:

This service is only available if the handling of I&M Requests is enabled by the stack. See `ulSystemFlags` in Set Configuration Request (6.1.5.1) packet.

### Coding for `ulParamType = PNS_IF_SET_OEM_PARAMETERS_TYPE_6`

If `ulParamType` is set to `PNS_IF_SET_OEM_PARAMETERS_TYPE_6` the structure `tType6Param` of `PNS_IF_SET_OEM_PARAMETERS_UNION_T` shall be used. It is defined as follows:

structure <code>PNS_IF_SET_OEM_PARAMETERS_TYPE_6_T</code>				
Area	Variable	Type	Value / Range	Description
	<code>usIRTCycleCounterOffset</code>	UINT16	0..	The offset in DPM / IO Image where the IRT cycle counter is copied to. Note: Only available if stack has an RTC3-AR established!

Table 49: `PNS_IF_SET_OEM_PARAMETERS_TYPE_6_T` - Set OEM Parameters for `ulParamType = 6`

### Coding for `ulParamType = PNS_IF_SET_OEM_PARAMETERS_TYPE_7`

If `ulParamType` is set to `PNS_IF_SET_OEM_PARAMETERS_TYPE_7` the structure `tType7Param` of `PNS_IF_SET_OEM_PARAMETERS_UNION_T` shall be used. It is defined as follows:

structure <code>PNS_IF_SET_OEM_PARAMETERS_TYPE_7_T</code>				
Area	Variable	Type	Value / Range	Description
	<code>fUseOlfLinkStateCommandCode</code>	TLR_BOOLEAN	0 1	Stack shall use <code>RCX_LINK_STATUS_CHANGE_IND</code> Stack shall use <code>PNS_IF_LINK_STATE_CHANGE_IND</code>

Table 50: `PNS_IF_SET_OEM_PARAMETERS_TYPE_7_T` - Set OEM Parameters for `ulParamType = 7`



### 7.1.11.2 Set OEM Parameters Confirmation

The stack informs the application about the success or failure of setting the OEM parameters.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_SET_OEM_PARAMETERS_CNF_T;
```

#### Packet Description

structure PNS_IF_SET_OEM_PARAMETERS_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1FE9	PNS_IF_SET_OEM_PARAMETERS_CNF - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch

Table 51: PNS\_IF\_SET\_OEM\_PARAMETERS\_CNF\_T - Set OEM Parameters Confirmation

## 7.1.12 Load Remanent Data Service

Using this service the user can hand over the required remanent data to the stack. This special service is only usable if bit D14 will be set in the system flags of the *Set Configuration Request*.



### Note:

The packet may be rejected by the stack with error code “invalid length” or “invalid parameter version” after a firmware update. The remanent data contains a fingerprint of the firmware version and its content may change in future versions of the stack. To avoid problems with invalid parameters this security check was implemented.

In this case the application shall continue with startup and ignore the return value.

### 7.1.12.1 Load Remanent Data Request

The user can hand over the remanent data needed by the stack with this packet. The stack will check the data and, if it is correct, this data will be used.

If the stack is accessed via Dual Port Memory it may be necessary for the application to fragment the request packet. This happens due to the limited size of the mailbox. How the fragmented service shall be handled by the application and the stack is described in reference [3].



### Important:

This packet must be sent prior to the *Set Configuration Service* packet, otherwise this packet will be rejected and the default values will be used.



### Important:

This packet may be sent to the stack either once or never at all. Multiple use of this packet is not allowed.

## Packet Structure Reference

```
typedef struct
{
    /* this is only the first byte, many others may follow */
    TLR_UINT8  abData[1];
} PNS_IF_LOAD_REMANENT_DATA_REQ_DATA_T;

typedef struct
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_LOAD_REMANENT_DATA_REQ_DATA_T  tData;
} PNS_IF_LOAD_REMANENT_DATA_REQ_T;
```

**Packet Description**

structure PNS_IF_LOAD_REMANENT_DATA_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	1 + n	Packet data length in bytes.
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32		Status not used for request.
	ulCmd	UINT32	0x1FEC	PNS_IF_LOAD_REMANENT_DATA_REQ- Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons. If the fragmented service is used see reference [3].
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_LOAD_REMANENT_DATA_REQ_DATA_T			
	abData[1]	UINT8		The remanent data which was reported by the stack. This is only the first byte as a place holder. All remaining bytes have to follow this one.

Table 52: PNS\_IF\_LOAD\_REMANENT\_DATA\_REQ\_T - Load Remanent Data Request

### 7.1.12.2 Load Remanent Data Confirmation

The stack informs the application about the success or failure of restoring the remanent data.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_LOAD_REMANENT_DATA_CNF_T;
```

#### Packet Description

structure PNS_IF_LOAD_REMANENT_DATA_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1FED	PNS_IF_LOAD_REMANENT_DATA_CNF - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch

Table 53: PNS\_IF\_LOAD\_REMANENT\_DATA\_CNF\_T - Load Remanent Data Confirmation

## 7.1.13 Configuration Delete Service

Using the Configuration Delete Service the user application can delete the current configuration. This service will **not** delete configuration files like SYCON.net database or iniBatch files.

**Note:**

If the stack performs cyclic data exchange, the configuration will be deleted, however the cyclic data exchange will not be interrupted and a valid data exchange is still assured.

### 7.1.13.1 Configuration Delete Request

This service is described in reference [3].

### 7.1.13.2 Configuration Delete Confirmation

This service is described in reference [3].

## 7.1.14 Set IO-Image Service

This service has to be used if the user application uses the callback interface for accessing the cyclic I/O-data. The service request will provide the PROFINET IO-Device stack with pointers to the consumer and provider data images and the user application's event callback. In return the service's confirmation will provide the user application with pointers to update input, update output and update extended status block callbacks. This request is essential for any user application running locally on the netX and not using the Shared memory interface. It has to be issued before using the Set Configuration Service. See also section 6.1.1 which contains an example.



### Note:

Furthermore this service does not apply if the stack is used as loadable firmware or used in conjunction with Shared memory Interface.

### 7.1.14.1 Set IO-Image Request

#### Packet Structure Reference

```
/* uiEvents */
#define PNS_IF_IO_EVENT_RESERVED          0x00000000
#define PNS_IF_IO_EVENT_NEW_FRAME         0x00000001
#define PNS_IF_IO_EVENT_CONSUMER_UPDATE_REQUIRED 0x00000002
#define PNS_IF_IO_EVENT_PROVIDER_UPDATE_REQUIRED 0x00000003
#define PNS_IF_IO_EVENT_FRAME_SENT        0x00000004
#define PNS_IF_IO_EVENT_CONSUMER_UPDATE_DONE 0x00000005
#define PNS_IF_IO_EVENT_PROVIDER_UPDATE_DONE 0x00000006

typedef TLR_RESULT (*PNS_IF_IOEVENT_HANDLER_CLB_T) (
    TLR_HANDLE hUserParam,
    TLR_UINT uiEvents
);

typedef struct PNS_IF_SET_IOIMAGE_REQ_DATA_Ttag
{
    TLR_UINT32          ulConsImageSize;
    TLR_UINT32          ulProvImageSize;
    TLR_UINT8*          pbConsImage;
    TLR_UINT8*          pbProvImage;

    PNS_IF_IOEVENT_HANDLER_CLB_T pfnEventHandler;
    TLR_HANDLE hUserParam;
} PNS_IF_SET_IOIMAGE_REQ_DATA_T;

typedef struct PNS_IF_SET_IOIMAGE_REQ_Ttag
{
    TLR_PACKET_HEADER_T tHead;
    PNS_IF_SET_IOIMAGE_REQ_DATA_T tData;
} PNS_IF_SET_IOIMAGE_REQ_T;
```

## Packet Description

structure PNS_IF_SET_IOIMAGE_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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		Status not used for request.
	ulCmd	UINT32	0x1FF0	PNS_IF_SET_IOIMAGE_REQ-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_SET_IOIMAGE_REQ_DATA_T			
	ulConslmImageSize	UINT32	0 ... $2^{16} - 1$	The size of the consumer data image.
	ulProvImageSize	UINT32	0 ... $2^{16} - 1$	The size of the provider data image.
	pbConslmImage	UINT8*		Pointer to consumer data image. Shall point to a memory area provided by the user's application where the incoming consumer data shall be copied to by the stack. See section 6.1.1 for further information
	pbProvImage	UINT8*		Pointer to provider data image. Shall point to a memory area provided by the user's application where the outgoing provider data shall be taken from by the stack. See section 6.1.1 for further information
	pfnEventHandler	PNS_IF_IOEVENT_HANDLER_CLB_T		Pointer to the user's application event callback function to be called by the stack for various events.
	hUserParam	HANDLE		Will be passed as first parameter to the user's event callback function. Typically the user applications resource parameter.

Table 54: PNS\_IF\_SET\_IOIMAGE\_REQ\_T – Set IO-Image Request

### 7.1.14.2 Set IO-Image Confirmation

#### Packet Structure Reference

```
typedef TLR_RESULT (*PNS_IF_UPDATE_IOIMAGE_CLB_T) (
    TLR_HANDLE hUserParam,
    TLR_UINT uiTimeout,
    TLR_UINT uiReserved1,
    TLR_UINT uiReserved2
);

typedef TLR_RESULT (*PNS_IF_UPDATE_EXTSTA_BLOCK_CLB_T)(
    TLR_HANDLE hUserParam,
    PNS_IF_EXTENDED_STATUS_BLOCK_T* ptExtSta,
    TLR_UINT uiOffsetCons,
    TLR_UINT uiOffsetProv
);

typedef struct  PNS_IF_SET_IOIMAGE_CNF_DATA_Ttag
{
    TLR_HANDLE hCallbackParam;
    PNS_IF_UPDATE_IOIMAGE_CLB_T pfnUpdateConsumerImage;
    PNS_IF_UPDATE_IOIMAGE_CLB_T pfnUpdateProviderImage;
    PNS_IF_UPDATE_EXTSTA_BLOCK_CLB_T pfnUpdateExtStaBlock;
} PNS_IF_SET_IOIMAGE_CNF_DATA_T;

typedef struct PNS_IF_SET_IOIMAGE_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;
    PNS_IF_SET_IOIMAGE_CNF_DATA_T tData;
} PNS_IF_SET_IOIMAGE_CNF_T;
```

#### Packet Description

structure PNS_IF_SET_IOIMAGE_CNF_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	16	Packet data length in bytes.
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32		See below
	ulCmd	UINT32	0x1FF1	PNS_IF_SET_IOIMAGE_CNF-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons



structure PNS_IF_SET_IOIMAGE_CNF_T				Type: Request
Area	Variable	Type	Value / Range	Description
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_SET_IOIMAGE_CNF_DATA_T			
	hCallbackParam	HANDLE		This handle shall be passed by the user's application as first parameter to the callback functions.
	pfnUpdateConsumerImage	PNS_IF_UPDATE_IOIMAGE_CLB_T		Pointer to the stack's update consumer callback function. This callback shall be used by the user's application to update its consumer data image with newest cyclic data.
	pfnUpdateProviderImage	PNS_IF_UPDATE_IOIMAGE_CLB_T		Pointer to the stack's update provider callback function. This callback shall be used by the user's application to instruct the stack to update the cyclic data from the provider data image.
	pfnUpdateExtStaBlock	PNS_IF_UPDATE_EXTSTA_BLOCK_CLB_T		Pointer to the stack's update extended status block callback function. This callback may be used by the user's application to fill an extended status block structure.

Table 55: PNS\_IF\_SET\_IOIMAGE\_CNF\_T – Set IO-Image Confirmation

## 7.1.15 Set IOXS Config Service

This service shall be used by the user application to enable reading/writing data states from/to the consumer/provider data image. When enabling this functionality, the stack will copy the provider data states of consumer data into the consumer data image and take the provider data states of provider data from provider data image. In this case the user application is responsible for setting these states appropriate. Consumer data states are currently not supported. For a detailed description of the structure of the data state block in the input/output image, refer to the *Dual Port Memory interface manual*. See also section 6.1.2 which contains an example.

### 7.1.15.1 Set IOXS Config Request

#### Packet Structure Reference

```
typedef enum PNS_IF_IOPS_MODE_Etag
{
    PNS_IF_IOPS_DISABLED = 0,
    PNS_IF_IOPS_BITWISE,
    PNS_IF_IOPS_BYTEWISE,
} PNS_IF_IOPS_MODE_E;

typedef enum PNS_IF_IOCS_MODE_Etag
{
    PNS_IF_IOCS_DISABLED = 0,
} PNS_IF_IOCS_MODE_E;

typedef struct PNS_IF_SET_IOXS_CONFIG_DATA_Ttag
{
    TLR_UINT32    ulIOPSMode;
    TLR_UINT32    ulConsImageIOPSOOffset;
    TLR_UINT32    ulReserved1;
    TLR_UINT32    ulProvImageIOPSOOffset;

    TLR_UINT32    ulIOCSMode;
    TLR_UINT32    ulConsImageIOCSOffset;
    TLR_UINT32    ulReserved2;
    TLR_UINT32    ulProvImageIOCSOffset;
} PNS_IF_SET_IOXS_CONFIG_DATA_T;

typedef struct PNS_IF_SET_IOXS_CONFIG_REQ_Ttag
{
    TLR_PACKET_HEADER_T    tHead;
    PNS_IF_SET_IOXS_CONFIG_DATA_T    tData;
} PNS_IF_SET_IOXS_CONFIG_REQ_T ;
```

## Packet Description

structure PNS_IF_SET_IOXS_CONFIG_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32} - 1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	32	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		Status not used for request.
	ulCmd	UINT32	0x1FF2	PNS_IF_SET_IOXS_CONFIG_REQ- Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_SET_IOXS_CONFIG_DATA_T			
	ulIOPSMODE	UINT32	0 ... 2	Desired encoding of the IOPS states. Either disabled, bit list (valid invalid) or byte list (complete IOPS)
	ulConslImageIOPSOOffset	UINT32	0 ... $2^{16} - 1$	The offset where the consumer data provider state block shall start in the consumer data image. The offset is relative to the beginning of the consumer data image. See section 6.1.2 for further information
	ulReserved1	UINT32	0	Reserved. Set to zero for compatibility reasons.
	ulProvImageIOPSOOffset	UINT32	0 ... $2^{16} - 1$	The offset where the provider data provider state block shall start in the provider data image. The offset is relative to the beginning of the provider data image. See section 6.1.2 for further information
	ulIOCSMODE	UINT32	0 ... 2	Desired encoding of the IOCS states. Either disabled, bit list (valid invalid) or byte list (complete IOCS)
	ulConslImageOCSSOffset	UINT32	0 ... $2^{16} - 10$	The offset where the consumer data consumer state block shall start in the consumer data image. The offset is relative to the beginning of the consumer data image. See section 6.1.2 for further information
	ulReserved2	UINT32	0	Reserved. Set to zero for compatibility reasons.
	ulProvImageOCSSOffset	UINT32	0 ... $2^{16} - 1$	The offset where the provider data consumer state block shall start in the provider data image. The offset is relative to the beginning of the provider data image. See section 6.1.2 for further information

Table 56: PNS\_IF\_SET\_IOXS\_CONFIG\_REQ\_T – Set IOXS Config Request

## 7.1.15.2 Set IOXS Config Confirmation

### Packet Structure Reference

```
typedef struct PNS_IF_SET_IOXS_CONFIG_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
} PNS_IF_SET_IOXS_CONFIG_CNF_T;
```

### Packet Description

structure PNS_IF_SET_IOXS_CONFIG_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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 below.
	ulCmd	UINT32	0x1FF3	PNS_IF_SET_IOXS_CONFIG_CNF- Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 57: PNS\_IF\_SET\_IOXS\_CONFIG\_CNF\_T – Set IOXS Config Confirmation

## 7.1.16 Configure Signal Service

The following section only applies if the stack was configured to use little endian byte order for cyclic process data image. This request shall be used to provide the stack with information about the data structure of the submodules.

### 7.1.16.1 Configure Signal Request

The request packet has to be sent to the stack to provide it with information about the data structure. This shall be done after the submodule has plugged. The submodule will not be used for data exchange until valid structure data has been provided. The request has to be used for each direction separately.

#### Packet Structure Reference

```
typedef enum
{
    IO_SIGNALS_TYPE_BIT = 0,           /* 0 */
    IO_SIGNALS_TYPE_BOOLEAN,          /* 1 */
    IO_SIGNALS_TYPE_BYTE,              /* 2 */
    IO_SIGNALS_TYPE_SIGNED8,           /* 3 */
    IO_SIGNALS_TYPE_UNSIGNED8,         /* 4 */
    IO_SIGNALS_TYPE_WORD,              /* 5 */
    IO_SIGNALS_TYPE_SIGNED16,          /* 6 */
    IO_SIGNALS_TYPE_UNSIGNED16,        /* 7 */
    IO_SIGNALS_TYPE_SIGNED24,          /* 8 */
    IO_SIGNALS_TYPE_UNSIGNED24,        /* 9 */
    IO_SIGNALS_TYPE_DWORD,             /* 10 */
    IO_SIGNALS_TYPE_SIGNED32,          /* 11 */
    IO_SIGNALS_TYPE_UNSIGNED32,        /* 12 */
    IO_SIGNALS_TYPE_SIGNED40,          /* 13 */
    IO_SIGNALS_TYPE_UNSIGNED40,        /* 14 */
    IO_SIGNALS_TYPE_SIGNED48,          /* 15 */
    IO_SIGNALS_TYPE_UNSIGNED48,        /* 16 */
    IO_SIGNALS_TYPE_SIGNED56,          /* 17 */
    IO_SIGNALS_TYPE_UNSIGNED56,        /* 18 */
    IO_SIGNALS_TYPE_LWORD,             /* 19 */
    IO_SIGNALS_TYPE_SIGNED64,          /* 20 */
    IO_SIGNALS_TYPE_UNSIGNED64,        /* 21 */
    IO_SIGNALS_TYPE_REAL32,            /* 22 */
    IO_SIGNALS_TYPE_REAL64,            /* 23 */
    IO_SIGNALS_TYPE_STRING,            /* 24 */
    IO_SIGNALS_TYPE_WSTRING,           /* 25 */
    IO_SIGNALS_TYPE_STRING_UUID,       /* 26 */
    IO_SIGNALS_TYPE_STRING_VISIBLE,    /* 27 */
    IO_SIGNALS_TYPE_STRING_OCTET,      /* 28 */
    IO_SIGNALS_TYPE_REAL32_STATE8,     /* 29 */
    IO_SIGNALS_TYPE_DATE,              /* 30 */
    IO_SIGNALS_TYPE_DATE_BINARY,       /* 31 */
    IO_SIGNALS_TYPE_TIME_OF_DAY,       /* 32 */
    IO_SIGNALS_TYPE_TIME_OF_DAY_NODATE, /* 33 */
    IO_SIGNALS_TYPE_TIME_DIFF,         /* 34 */
    IO_SIGNALS_TYPE_TIME_DIFF_NODATE,  /* 35 */
    IO_SIGNALS_TYPE_NETWORK_TIME,      /* 36 */
    IO_SIGNALS_TYPE_NETWORK_TIME_DIFF, /* 37 */
    IO_SIGNALS_TYPE_F_MSGTRAILER4,     /* 38 */
    IO_SIGNALS_TYPE_F_MSGTRAILER5,     /* 39 */
    IO_SIGNALS_TYPE_ENGINEERING_UINT,  /* 40 */
    IO_SIGNALS_TYPE_MAX
} IO_SIGNALS_TYPES_E;

#define IO_SIGNALS_DIRECTION_CONSUMER (1)
#define IO_SIGNALS_DIRECTION_PROVIDER (2)

typedef struct IO_SIGNALS_CONFIGURE_SIGNAL_REQ_DATA_Ttag
{
    /* see fieldbus specific API Manual for a definition how this */
}
```

```

/* fieldbus specific fields shall be filled. */
TLR_UINT32      ulFieldbusSpecific1; /* e.g. Slave Handle */
TLR_UINT32      ulFieldbusSpecific2;
TLR_UINT32      ulFieldbusSpecific3; /* e.g. Slot */
TLR_UINT32      ulFieldbusSpecific4; /* e.g. SubSlot */
TLR_UINT32      ulFieldbusSpecific5;
TLR_UINT32      ulFieldbusSpecific6;
TLR_UINT32      ulFieldbusSpecific7;
TLR_UINT32      ulFieldbusSpecific8;
/* signal direction described in this packet */
TLR_UINT32      ulSignalsDirection;
/* amount of signals contained in abSignals */
TLR_UINT32      ulTotalSignalCount;
/* array of signals - packet definition only contains the first signal, all other
follow */
struct
{
    /* type of signal - see IO_SIGNALS_TYPES_E */
    TLR_UINT8      bSignalType;
    /* amount of signal (e.g. 16 for a "16 Byte Input module") */
    TLR_UINT8      bSignalAmount;
} atSignals[1];
} IO_SIGNALS_CONFIGURE_SIGNAL_REQ_DATA_T;

typedef struct IO_SIGNALS_CONFIGURE_SIGNAL_REQ_Ttag
{
    TLR_PACKET_HEADER_T      tHead;
    IO_SIGNALS_CONFIGURE_SIGNAL_REQ_DATA_T      tData;
} IO_SIGNALS_CONFIGURE_SIGNAL_REQ_T;

```

## Packet Description

structure IO_SIGNALS_CONFIGURE_SIGNAL_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32} - 1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	40 * 2 * Number of Signals	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		Status not used for request.
	ulCmd	UINT32	0x6100	IO_SIGNALS_CONFIGURE_SIGNAL_REQ- Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	0	Set to zero.
Data	structure IO_SIGNALS_CONFIGURE_SIGNAL_REQ_DATA_T			
	ulFieldbusSpecific1	UINT32	0	Unused, set to zero.
	ulFieldbusSpecific2	UINT32	0	The API of the submodule. Currently only 0 supported.
	ulFieldbusSpecific3	UINT32	0..0x7FFF	The slot of the submodule.
	ulFieldbusSpecific4	UINT32	1..0x7FFF	The subslot of the submodule
	ulFieldbusSpecific5	UINT32	0	Unused. Set to zero
	ulFieldbusSpecific6	UINT32	0	Unused. Set to zero
	ulFieldbusSpecific7	UINT32	0	Unused. Set to zero
	ulFieldbusSpecific8	UINT32	0	Unused. Set to zero
	ulSignalsDirection	UINT32	1..2	Either IO_SIGNALS_DIRECTION_CONSUMER or IO_SIGNALS_DIRECTION_PROVIDER
	ulTotalSignalCount	UINT32	1..1024	Count of Signals elements. (n)
	atSignals	{UINT8 ;UINT8} * n		Array of pairs of bytes which describe the signal. The first byte is the signal type, the second byte is the number of elements of specified kind of signal. (e.g. (20,4) is an array of four signed 64 bit integers)

Table 58: Structure IO\_SIGNALS\_CONFIGURE\_SIGNAL\_REQ\_T

### 7.1.16.2 Configure Signal Confirmation

The confirmation packet has no data part.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T IO_SIGNALS_CONFIGURE_SIGNAL_CNF_T;
```

#### Packet Description

structure IO_SIGNALS_CONFIGURE_SIGNAL_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		Source Queue-Handle
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		Result of operation.
	ulCmd	UINT32	0x6101	IO_SIGNALS_CONFIGURE_SIGNAL_CNF- Command
	ulExt	UINT32	x	Not in use. Ignore.
	ulRout	UINT32	x	Routing, ignore.

Table 59: IO\_SIGNALS\_CONFIGURE\_SIGNAL\_CNF\_T – Configure Signal Confirmation



### 7.1.16.3 Example: Configure Signal Request packet

For an imaginary 12 byte input submodule using the data structure

```
struct {
    UINT32    aulData1[2];
    UINT8     abData2[2];
    UINT16    usData3;
}
```

Which corresponds to the following IO data subsection in a GSDML file

```
<IOData IOPS_Length="1" IOCS_Length="1">
  <Input>
    <DataItem DataType="Unsigned32" TextId="aulData1a"/>
    <DataItem DataType="Unsigned32" TextId="aulData1b"/>
    <DataItem DataType="OctetString" Length="2" TextId="abData2"/>
    <DataItem DataType="Unsigned16" TextId="abData3"/>
  </Input>
</IOData>
```

The Configure Signal Request packet has to be filled in as follows:

```
IO_SIGNALS_CONFIGURE_SIGNAL_REQ_T* ptRequest =
malloc(sizeof(IO_SIGNALS_CONFIGURE_SIGNAL_REQ) + 2 * (4-1));

memset(ptRequest, 0x00, sizeof(IO_SIGNALS_CONFIGURE_SIGNAL_REQ) + 2 * (3-1));

ptRequest->tHead.ulCmd = IO_SIGNALS_CONFIGURE_SIGNAL_REQ;
ptRequest->tHead.ulLen = sizeof(IO_SIGNALS_CONFIGURE_SIGNAL_REQ) + 2 * (3-1);

ptRequest->tData.ulFieldbusSpecific2 = 0; /* api */
ptRequest->tData.ulFieldbusSpecific3 = 1; /* slot */
ptRequest->tData.ulFieldbusSpecific4 = 1; /* subslot */

ptRequest->tData.ulSignalsDirection = IO_SIGNALS_DIRECTION_PROVIDER;
ptRequest->tData.ulTotalSignalCount = 3;

ptRequest->tData.atSignals[0].bSignalType      = IO_SIGNALS_TYPE_UNSIGNED32;
ptRequest->tData.atSignals[0].bSignalAmount    = 2;

ptRequest->tData.atSignals[1].bSignalType      = IO_SIGNALS_TYPE_BYTE;
ptRequest->tData.atSignals[1].bSignalAmount    = 2;

ptRequest->tData.atSignals[2].bSignalType      = IO_SIGNALS_TYPE_UNSIGNED16;
ptRequest->tData.atSignals[2].bSignalAmount    = 1;
```

## 7.2 Connection Establishment

After the stack has been configured and switched to bus on (if auto start has been disabled), the device is ready for operation and waits for incoming events from the network. If the device is activated for the first time, usually NameOfStation has not been configured yet. Depending on the PROFINET Controller configuration, the NameOfStation needs to be assigned by the engineering or will be assigned by the controller according to the topology. At next, the controller will verify the IP parameters of the device and adapt them according to its configuration. These steps are done using the DCP protocol as shown in the next two figures.

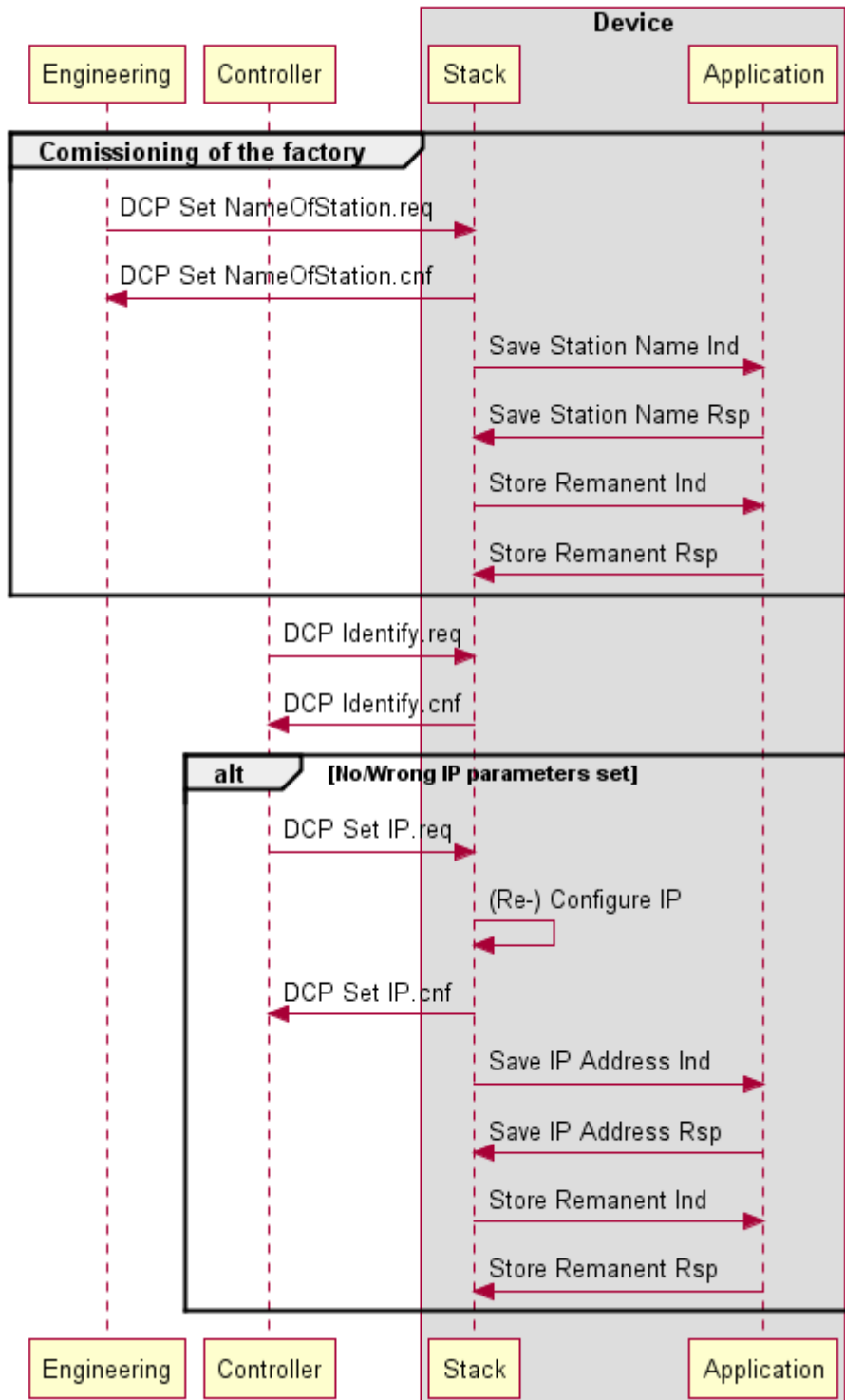


Figure 12: Initial Configuration by DCP without topology information at controller side.

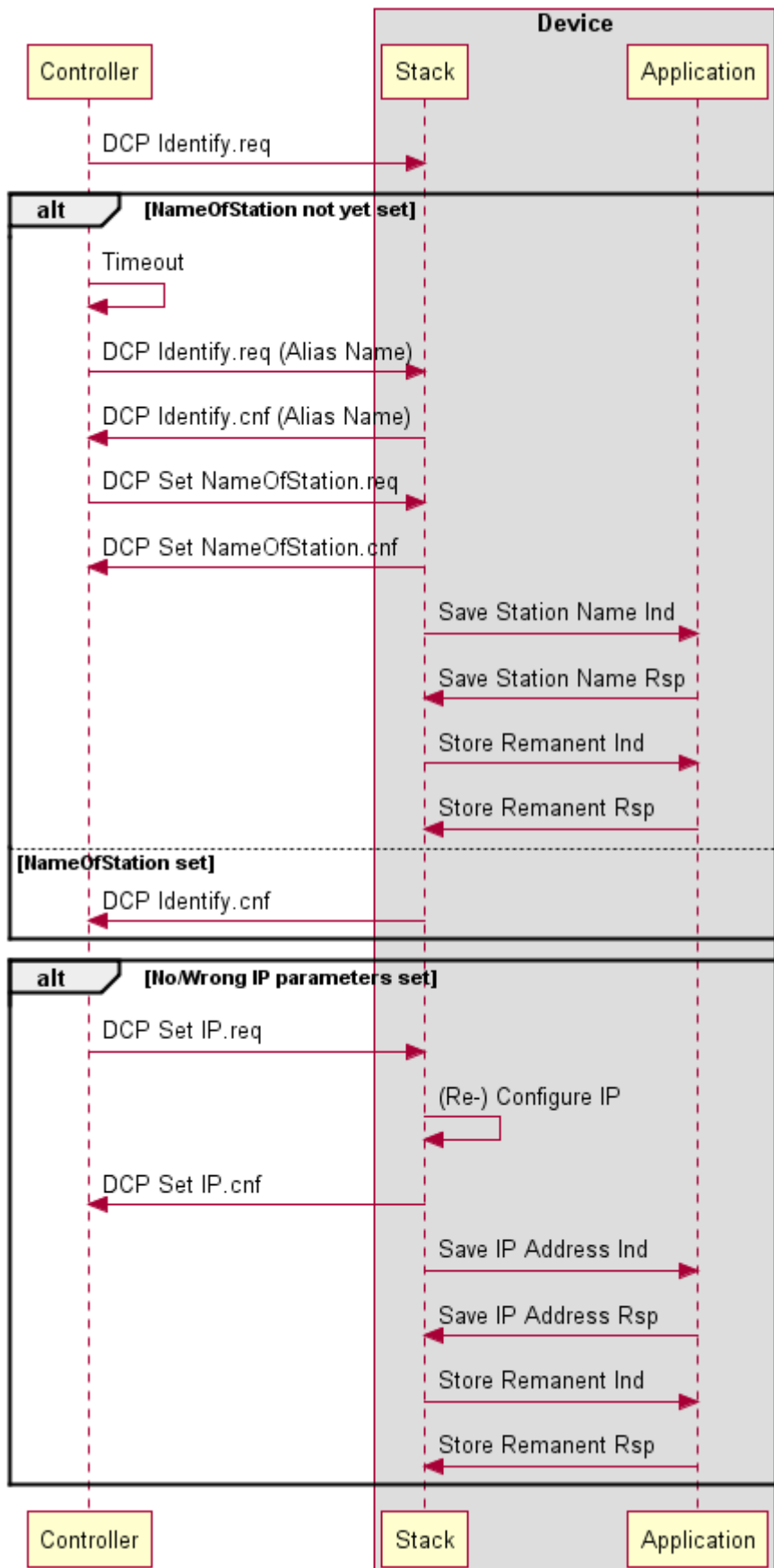


Figure 13: Initial Configuration by DCP using topology information at controller side.

Afterwards, the IP stack of the device is active and the RPC layer accepts incoming requests from the network. The PROFINET controller will now establish an AR. If the application registered with the stack previously, several indications will be generated. These must be handled properly in order to establish the connection successfully.

A typical interaction between stack and application is shown in the next two figures:

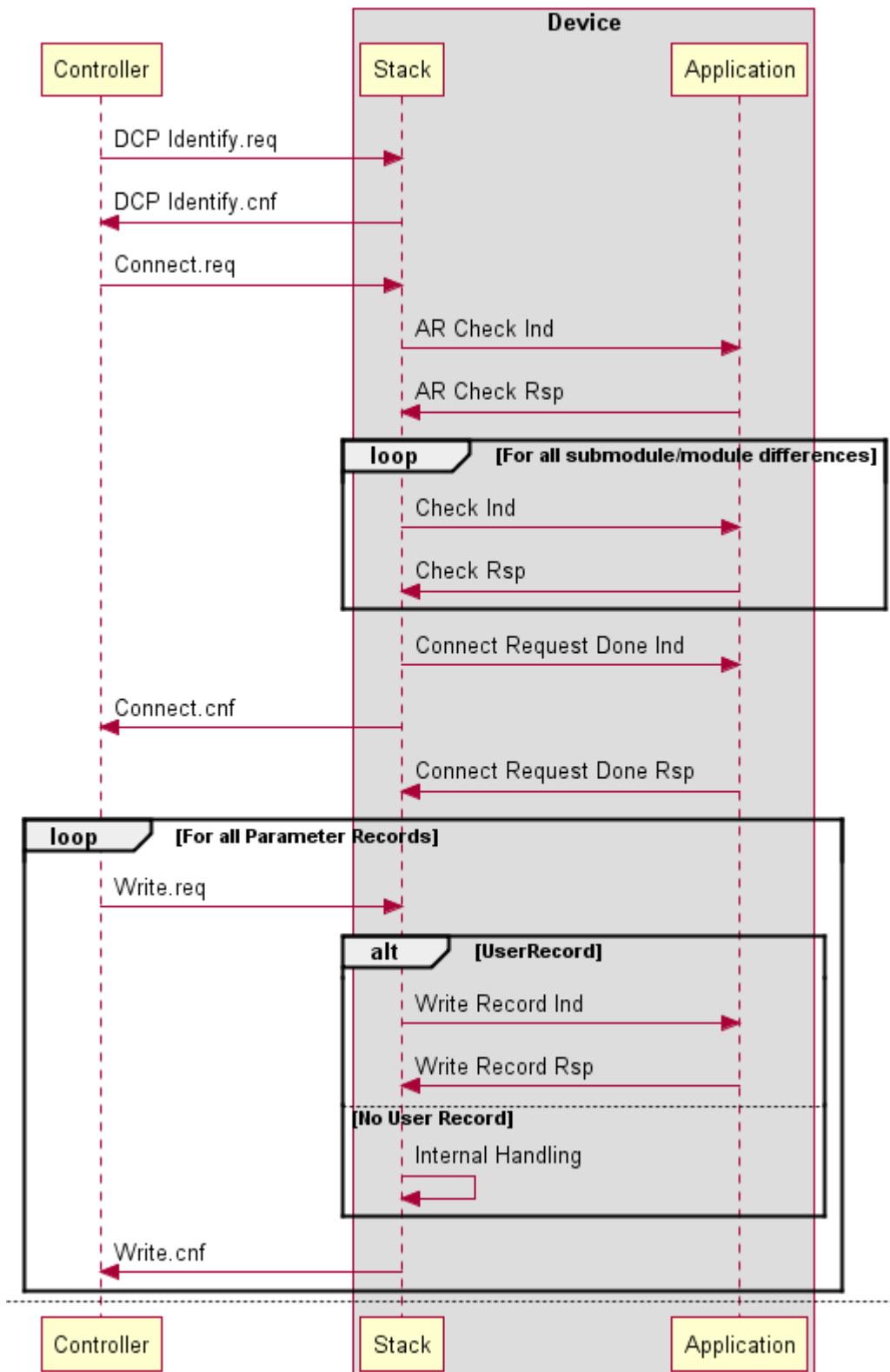


Figure 14: Sequence between Controller, Stack and Application during Connection Establishment

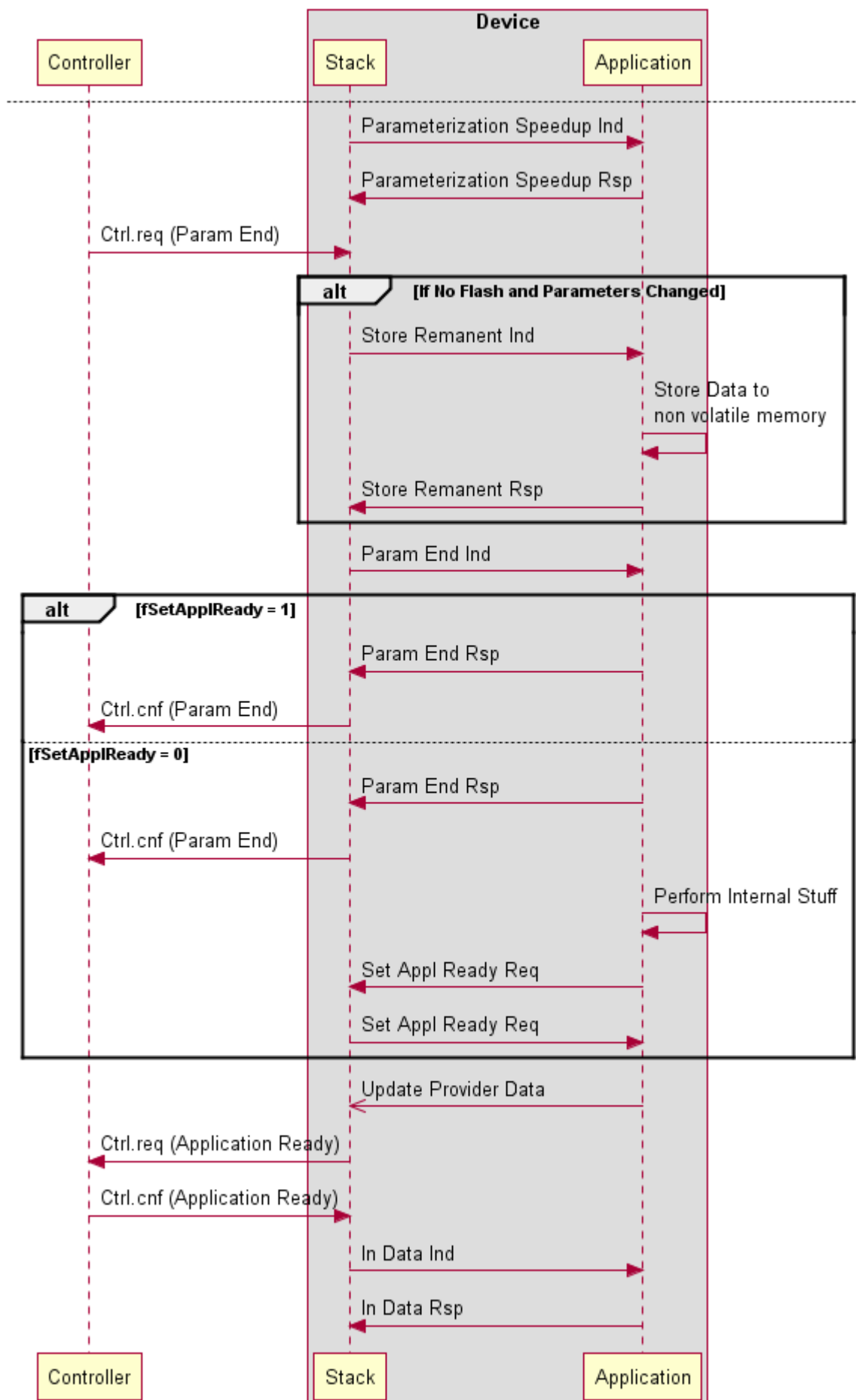


Figure 15: Sequence between Controller, Stack and Application during Connection Establishment (continued)

The very first indication will be the **AR\_Check\_Indication** (see section 6.2.1). This informative indication signifies the beginning of the AR establishment phase.

In the next step the stack will compare the module configuration requested by the controller with the current module configuration of the stack. Depending on the stack configuration parameters, the stack will now generate a **Check\_Indication** (see section 6.2.2.1) for each used, wrong, unused or missing (sub) module. If desired, the application has the opportunity to reconfigure the stack. For this, the Plug/Pull Services shall be used before returning the **Check\_Response** back to the stack. Additionally, the **Check\_Response** provides the ability to report a substitute (sub) module.

After all **Check\_Indications** have been processed, the Stack will generate the `Connect.cnf` to the PROFINET Controller and issue a **Connect\_Request\_Done\_Indication** (see section 6.2.3.1) to the application. This informative indication signifies the end of the first phase.

The PROFINET Controller will now parameterize the device by writing the parameter records of the valid (sub-)modules. If parameter records have been specified in the device description file (GSDML), **Write\_Record\_Indications** (see section 6.3.2.1) will be generated for each record written by the controller. This data has to be validated and evaluated by the application.

When the PROFINET Controller has finished writing the parameters, it will generate a ParamEnd Control message. At first, the stack will check if FastStartUp (FSU) shall be used for this AR. This will be indicated to the Application by issuing a **Parameterization\_Speedup\_Indication** with non-zero UUID. In that case, the Application is requested to store the UUID and the parameter records into non volatile memory in order to restore them on next power up. The UUID value shall be used to detect parameter changes in that case. Additionally, a **Parameter\_End\_Indication** (see section 6.2.4.1) will be generated by the stack. At this time, the application should apply all written parameter data to its internal logic/hardware (Only if parameters changed or Non-FSU mode is in use. Otherwise the parameter should have been restored from non-volatile memory at power on already). When finished, the application should generate the **Parameter\_EndResponse** from the indication. If the application will need some further initialization time, the application may explicitly request Application Ready by setting `fSendApplReady` to False and generating an `Set_ApplReady_Req` later on (see section 6.2.5).

**Attention:**

Before the stack actually generates the Application Ready request to the Controller it will also wait for the application to update the provider data. This is necessary as the cyclic telegrams sent by the device must contain valid data before the application ready is issued.

Finally, the **AR\_InData\_Indication** (see section 6.2.6.1) informs the application about the fact that the first cyclic frame from the IO-Controller has been received after the Application Ready Confirmation. The AR is now established and valid data-exchange takes place.

**Attention:**

As the stack supports multiple ARs, multiple connection establishment sequences may occur in the same time. The device handle parameter of the indications shall be used to differentiate between these ARs.

In detail, the following functionality concerning connection establishment is provided by the PROFINET IO Device IRT Stack:

Overview over the Connection Establishment Packets of the PROFINET IO Device IRT Stack			
No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
6.2.1	AR Check Indication	0x1F14	112
	AR Check Response	0x1F15	114
6.2.2	Check Indication	0x1F16	115
	Check Response	0x1F17	119
6.2.3	Connect Request Done Indication	0x1FD4	121
	Connect Request Done Response	0x1FD5	122
6.2.4	Parameter End Indication	0x1F0E	124
	Parameter End Response	0x1F0F	124
6.2.5	Application Ready Request	0x1F10	126
	Application Ready Confirmation	0x1F11	127
6.2.6	AR InData Indication	0x1F28	129
	AR InData Response	0x1F29	130
6.2.7	Store Remanent Data Indication	0x1FEA	131
	Store Remanent Data Response	0x1FEB	132

Table 60: Overview over the Connection Establishment Packets of the PROFINET IO Device IRT Stack

## 7.2.1 AR Check Service

The AR Check Service is used by the protocol stack to indicate that a new AR is going to be established. It will be generated by the protocol stack right after receiving and checking the PROFINET Connect Request.

### 7.2.1.1 AR Check Indication

This packet contains information about the AR to be established. It is sent from the stack to the application.

**Note:**

Prior to stack/firmware version 3.5.21.0 the field “tCmInitiatorObjUUID” was not reported. In that case the packet length was 256.

### Packet Structure Reference

```
typedef struct PNS_IF_AR_CHECK_IND_DATA_Ttag
{
    TLR_UINT32    hDeviceHandle;
    TLR_UINT16    usARType;
    TLR_UINT32    ulARProperties;
    TLR_UINT32    ulRemoteIpAddr;
    TLR_UINT16    usRemoteNameOfStationLen;
    TLR_UINT8     abRemoteNameOfStation[PNIO_MAX_NAME_OF_STATION];
    PNS_IF_UUID_T tCmInitiatorObjUUID;
} PNS_IF_AR_CHECK_IND_DATA_T;

typedef struct PNS_IF_AR_CHECK_IND_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T    tHead;
    /** packet data */
    PNS_IF_AR_CHECK_IND_DATA_T    tData;
} PNS_IF_AR_CHECK_IND_T;
```



## Packet Description

Structure PNS_IF_AR_CHECK_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	272	Packet data length in bytes
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indications.
	ulCmd	UINT32	0x1F14	PNS_IF_AR_CHECK_IND-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_AR_CHECK_IND_DATA_T			
	hDeviceHandle	UINT32		The device handle.
	usARType	UINT16		Connect-Request's AR-Type
	ulARProperties	UINT32		Connect-Request's AR-Properties.
	ulRemoteIpAddr	UINT32		IO-Controller's IP address.
	usRemoteNameOfStationLen	UINT16	1..240	Length of IO-Controller's NameOfStation.
	abRemoteNameOfStation[240]	UINT8		IO-Controller's NameOfStation as ASCII byte-array.
	tCmInitiatorObjUUID	PNS_IF_UUID_T		The ContextManagement Initiator Object UUID used by IO-Controller for this AR. <b>Note:</b> This field is contained in the indication starting with stack/firmware version 3.5.21.0

Table 61: PNS\_IF\_AR\_CHECK\_IND\_T - AR Check Indication

### 7.2.1.2 AR Check Response

The application has to return an AR check response after receiving the AR Check Indication.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    UINT32 hDeviceHandle;
} PNS_IF_HANDLE_DATA_T;

typedef struct PNS_IF_HANDLE_PACKET_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_HANDLE_DATA_T         tData;
} PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T          PNS_IF_AR_CHECK_RSP_T;
```

#### Packet Description

structure PNS_IF_AR_CHECK_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	Status has to be ok for this service.
	ulCmd	UINT32	0x1F15	PNS_IF_AR_CHECK_RES-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	Structure PNS_IF_HANDLE_DATA_T			
	hDeviceHandle	UINT32		The device handle.

Table 62: PNS\_IF\_AR\_CHECK\_RSP\_T - AR Check Response

## 7.2.2 Check Indication Service

If the PROFINET IO-Controller expects different submodules than configured in the PROFINET IO-Device stack, the Check Indication Service will be used to indicate this to the application. The indication contains all parameters the controller expected along with the current module and submodule state. If the submodule state is indicated as PNIO\_SUBSTATE\_WRONG\_SUBMODULE the application may use the submodule state PNIO\_SUBSTATE\_SUBSTITUTE\_SUBMODULE in the response to tell the stack that the configured submodule is able to perform like the requested submodule. Alternatively, if the application is designed to adapt to the requested configuration, the application may reconfigure the stack on check indication.



### Note:

A Check Indication Service may also be issued after a Extended Plug Submodule Request. In that case the recently plugged submodule has been associated with a new AR which expects another submodule.



### Note:

A Check Indication Service may also be issued after on submodule AR ownership change. In that case a submodule has been associated with an existing AR which expects another submodule.

In most applications the device will not adapt to the controllers configuration, e. g. (modular) I/O devices, sensors, gateways etc. The sequence for this is shown in the following figure.

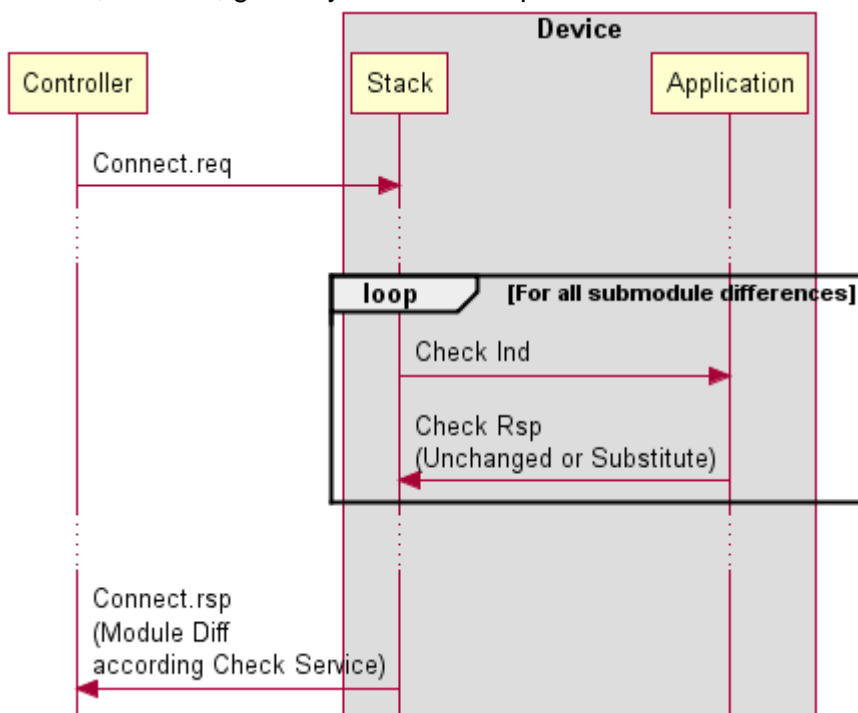


Figure 16: Check Service Packet sequence for non adapting applications

Special applications may require an adaption to the controller expectations. This includes for example:

- PROFIdrive: Here, the controller selects the desired PROFIdrive telegram type by means of submodule id.
- Selectable Data Length for Sensors: The maximum length of the presented barcode may be selected by module/submodule id.

- Configuration of a Gateway by adapting to the submodule configuration expected by the controller.

The packet sequence then is

1. Check indication
2. Pull request and confirmation: *Pull Module Request* and *Pull Module Confirmation* or *Pull Submodule Request* and *Pull Submodule Confirmation*.
3. Plug request and confirmation: *Plug Submodule Request* and *Plug Submodule Confirmation* or *Extended Plug Submodule Request* and *Extended Plug Submodule Confirmation*.
4. Check response

The sequence of adaption is shown in the next figure.

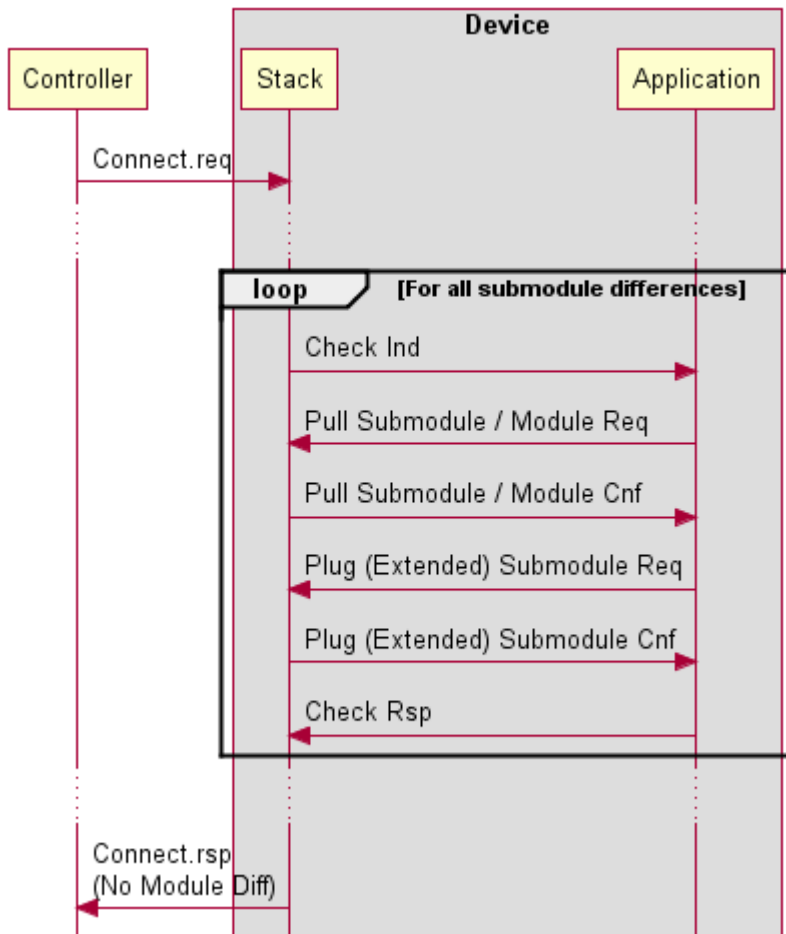


Figure 17: Check Service Packet sequence for adapting applications

### 7.2.2.1 Check Indication

This packet indicates the parameters of a wrong / missing (sub)module to the application. If enabled by the corresponding startup flags in Set Configuration Service, this indication will also be generated for unused or correct submodules.

#### Packet Structure Reference

```
typedef struct PNS_IF_CHECK_IND_IND_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulSubslot;
    TLR_UINT32 ulModuleId;
    TLR_UINT16 usModuleState;
    TLR_UINT32 ulSubmodId;
    TLR_UINT16 usSubmodState;
    TLR_UINT16 usExpInDataLen;
    TLR_UINT16 usExpOutDataLen;
} PNS_IF_CHECK_IND_IND_DATA_T;

typedef struct PNS_IF_CHECK_IND_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_CHECK_IND_IND_DATA_T  tData;
} PNS_IF_CHECK_IND_T;
```

## Packet Description

Structure PNS_IF_CHECK_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	28	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indications.
	ulCmd	UINT32	0x1F16	PNS_IF_CHECK_IND-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_CHECK_IND_IND_DATA_T			
	hDeviceHandle	UINT32		The device handle.
	ulApi	UINT32		The API of the wrong submodule.
	ulSlot	UINT32		The slot of the wrong submodule.
	ulSubslot	UINT32		The subslot of the wrong submodule.
	ulModuleId	UINT32		The Module ID the IO-Controller expected
	usModuleState	UINT16		The Module State suggested by the stack.
	ulSubmodId	UINT32		The Submodule ID the IO-Controller expected.
	usSubmodState	UINT16		The Submodule state suggested by the stack.
	usExpInDataLength	UINT16		The length of input data IO-Controller expects for the submodule. This is only informative for application.
	usExpOutDataLength	UINT16		The length of output data IO-Controller expects for the submodule. This is only informative for application.

Table 63: PNS\_IF\_CHECK\_IND\_T - Check Indication

### 7.2.2.2 Check Response

With this response packet the application influences the ModuleDiffBlock, which the IO-Device stack sends to IO-Controller.

Possible values for the fields usModuleState and usSubmodState are defined below the packet description in additional tables.

**Note:** If the application adapted the module configuration to the one the IO-Controller expected, this shall be indicated to the stack using the PNIO\_XXX\_CORRECT\_XXX (sub)module states.

#### Packet Structure Reference

```
typedef struct PNS_IF_CHECK_IND_RSP_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulSubslot;
    TLR_UINT32 ulModuleId;
    TLR_UINT16 usModuleState;
    TLR_UINT32 ulSubmodId;
    TLR_UINT16 usSubmodState;
} PNS_IF_CHECK_IND_RSP_DATA_T;

typedef struct PNS_IF_CHECK_RSP_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_CHECK_IND_RSP_DATA_T  tData;
} PNS_IF_CHECK_RSP_T;
```

Structure PNS_IF_CHECK_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	28	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	The status has to be okay for this service.
	ulCmd	UINT32	0x1F17	PNS_IF_CHECK_RES-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_AR_CHECK_IND_RSP_DATA_T			
	hDeviceHandle	UINT32		The device handle

Structure PNS_IF_CHECK_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
	ulApi	UINT32		The API of the wrong submodule.
	ulSlot	UINT32		The slot of the wrong submodule.
	ulSubslot	UINT32		The subslot of the wrong submodule.
	ulModuleId	UINT32		The ModuleID the IO-Controller expected
	usModuleState	UINT16	See below	The ModuleState. Ignored by Stack
	ulSubmodId	UINT32		The SubmoduleID the IO-Controller expected.
	usSubmodState	UINT16	See below	The SubmoduleState. Only the value PNIO_SUBSTATE_SUBSTITUTE_MODULE is honored by the stack. Any other value will lead to default behavior.

Table 64: PNS\_IF\_CHECK\_RSP\_T - Check Response

Definitions to use for the field usModuleState:

Definition / (Value)	Description
PNIO_MODSTATE_NO_MODULE (0x0)	No module plugged into the slot specified.
PNIO_MODSTATE_WRONG_MODULE (0x1)	A wrong module is plugged into the specified slot.
PNIO_MODSTATE_PROPER_MODULE (0x2)	A proper (the correct) module is plugged into the specified slot but is already used by another IO-Controller and is therefore not accessible.
PNIO_MODSTATE_SUBSTITUTE_MODULE (0x3)	A substitute module is plugged into the specified slot. Substitute modules can offer the same functionality as the originally requested module.
PNIO_MODSTATE_UNUSED_MODULE (0x4)	A module is plugged into the specified slot but is not requested/used by the IO-Controller.

Table 65: Field usModuleState

Definitions to use for the field usSubmodState:

Definition / (Value)	Description
PNIO_SUBSTATE_NO_SUBMODULE (0x0)	No submodule plugged into the slot/subslot specified.
PNIO_SUBSTATE_WRONG_SUBMODULE (0x1)	A wrong submodule is plugged into the specified slot/subslot.
PNIO_SUBSTATE_PROPER_SUBMODULE (0x2)	A proper (the correct) submodule is plugged into the specified slot/subslot but is already used by another IO-Controller. It cannot be used now.
PNIO_SUBSTATE_APPL_READY_PENDING (0x4)	The correct submodule is plugged into the specified slot/subslot but it is not ready for data exchange yet.
PNIO_SUBSTATE_SUBSTITUTE_MODULE (0x7)	A substitute submodule is plugged into the specified slot/subslot. Substitute submodules can offer the same functionality as the originally requested submodule.
PNIO_SUBSTATE_UNUSED_MODULE (0x8)	A submodule is plugged into the specified slot/subslot but it is not requested/used by the IO-Controller.

Table 66: Field usSubmodState



## 7.2.3 Connect Request Done Service

The stack sends the indication Connect Request Done to the application to indicate that no more Check Indications will be sent by the stack. Now the application has to examine all information provided by the stack to have exact knowledge about the submodules and the frame offsets of their IO-data. The application has to collect the information from the Check\_Indication (see section 6.2.2) in order to do so.

### 7.2.3.1 Connect Request Done Indication

The indication packet is informative only.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
} PNS_IF_HANDLE_DATA_T;

typedef struct PNS_IF_HANDLE_PACKET_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_HANDLE_DATA_T        tData;
} PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T          PNS_IF_CONNECTREQ_DONE_IND_T;
```

#### Packet Description

Structure PNS_IF_CONNECTREQ_DONE_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... 2 <sup>32</sup> -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 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indications.
	ulCmd	UINT32	0x1FD4	PNS_IF_CONNECT_REQ_DONE_IND-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_HANDLE_DATA_T			
	hDeviceHandle	UINT32		The device handle

Table 67: PNS\_IF\_CONNECTREQ\_DONE\_IND\_T - Connect Request Done Indication

### 7.2.3.2 Connect Request Done Response

The application has to return this packet after receiving a Connect Request Done indication. The device handle has to match that of the indication.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
} PNS_IF_HANDLE_DATA_T;

typedef struct PNS_IF_HANDLE_PACKET_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_HANDLE_DATA_T         tData;
} PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T          PNS_IF_CONNECTREQ_DONE_RSP_T;
```

#### Packet Description

Structure PNS_IF_CONNECTREQ_DONE_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	The status has to be okay for this service.
	ulCmd	UINT32	0x1FD5	PNS_IF_CONNECT_REQ_DONE_RES-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_HANDLE_DATA_T			
	hDeviceHandle	UINT32		The device handle

Table 68: PNS\_IF\_CONNECTREQ\_DONE\_RSP\_T - Connect Request Done Response

## 7.2.4 Parameter End Service

When the IO-Controller has finished parameterizing the IO-Device, it sends the Parameter End command. This is indicated by the stack to the application.

With receipt of this indication the application shall start configuring its submodules with the parameters from the write indications having been received. If no write indication was received nothing has to be done.

If the application is ready, the packet shall be answered with field `fSendApplicationReady` set to true in order to indicate to the stack, that the application is ready for cyclic process data exchange. If for any reason the application requires additional time to get ready, the application shall set the field `fSendApplicationReady` to false and respond to the indication immediately. When the application gets ready, it shall use the Application Ready Service described in section 6.2.5 to indicate this to stack.



### Note:

Before the stack will signal the Application Ready to the IO-Controller, the User Application is required to provide the Stack with valid I/O Data. Therefore the Application is required to write the I/O Data after returning Parameter End Response with `fSendApplicationReady` to true (See section 6.2.5).



### Note:

For this service, a timeout is implemented in the stack. If the application does not answer within the timeout the stack will automatically generate a negative response the IO-Controller. See section *Timeout for Response Packets* on page 25 for the timeout value.

### 7.2.4.1 Parameter End Indication

This packet indicates the receipt of Parameter End from IO-Controller.



### Note:

The combination of `usSubslot`, `usSlot` and `ulApi` all being zero at the same time signifies Parameter End for all submodules.

### Packet Structure Reference

```
typedef struct PNS_IF_PARAM_END_IND_DATA_Ttag
{
    TLR_UINT32    hDeviceHandle;
    TLR_UINT32    ulApi;        /* valid only if usSlot    != 0 */
    TLR_UINT16    usSlot;       /* valid only if usSubslot != 0 */
    TLR_UINT16    usSubslot;    /* 0: for all (sub)modules, != 0: for this specific
                                   submodule */
} PNS_IF_PARAM_END_IND_DATA_T;

typedef struct PNS_IF_PARAM_END_IND_Ttag
{
    TLR_PACKET_HEADER_T    tHead;
    PNS_IF_PARAM_END_IND_DATA_T    tData;
} PNS_IF_PARAM_END_IND_T;
```

## Packet Description

Structure PNS_IF_PARAM_END_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	12	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indications.
	ulCmd	UINT32	0x1F0E	PNS_IF_PARAM_END_IND-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	0	Routing not in use, set to zero for compatibility reasons
Data	structure PNS_IF_PARAM_END_IND_DATA_T			
	hDeviceHandle	UINT32		The device handle
	ulApi	UINT32		The API of the module whose parameterization is finished. Only valid if usSlot != 0.
	usSlot	UINT16		The slot of the module whose parameterization is finished. Only valid if usSubslot != 0.
	usSubslot	UINT16	0 != 0	Parameterization is finished for all modules. Parameterization is finished for the module specified by ulApi, usSlot and usSubslot.

Table 69: PNS\_IF\_PARAM\_END\_IND\_T - Parameter End Indication

### 7.2.4.2 Parameter End Response

This packet has to be returned to the stack. Depending on the field `fSendApplicationReady` the stack will automatically send the command Application Ready to the IO-Controller. If `ulSta` is set to `TLR_S_OK` the stack will automatically send the Application Ready to IO-Controller.



#### Note:

It is not allowed to send Application ready until the IOPS and IOCS of all submodules are set to good. If application is ready but for any reason the data status for a specific submodule is not good it is forbidden to send the application ready indication to IO-Controller.

## Packet Structure Reference

```
typedef struct PNS_IF_PARAM_END_RSP_DATA_Ttag
{
    TLR_UINT32      hDeviceHandle;
    TLR_BOOLEAN     fSendApplicationReady; /* set to TRUE to send ApplReady automatically */
} PNS_IF_PARAM_END_RSP_DATA_T;

typedef struct PNS_IF_PARAM_END_RSP_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_PARAM_END_RSP_DATA_T  tData;
} PNS_IF_PARAM_END_RSP_T;
```

## Packet Description

Structure PNS_IF_PARAM_END_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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 below.
	ulCmd	UINT32	0x1F0F	PNS_IF_PARAM_END_RES-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_PARAM_END_RSP_DATA_T			
	hDeviceHandle	UINT32		Handle to the IO-Device.
	fSendApplicationReady	BOOL32	FALSE (0)  TRUE	The stack shall not automatically send ApplicationReady. This will be initiated by Application using the Service described in section 6.2.5.  The stack shall automatically send ApplicationReady.

Table 70: PNS\_IF\_PARAM\_END\_RSP\_T - Parameter End Response

## 7.2.5 Application Ready Service

With this service the application shall indicate to the stack, that it is ready for process data exchange and that the stack shall signal Application Ready to the IO-Controller. This is only necessary, if the application returned the Parameter End Response with fSendApplicationReady set to false.



### Note:

If the application does not signal Application Ready to the Stack/ Controller at all, cyclic process data cannot be exchanged. Therefore the Application is required to indicate Application Ready at some time point (Many Controllers abort the Application Relation if ApplicationReady is not signaled within a timeout).



### Note:

If the Application handles the Provider-State on itself, it is important to set up the Provider States before sending Application Ready. The IO-Controller will evaluate the Provider States on Application Ready and will ignore all Submodules with Bad Provider State for Cyclic Process Data Exchange



### Note:

As Application Ready also signals valid process data to the IO-Controller, the stack is required to update its internal buffer at least once from the DPM Output Area / Provider Image. Therefore the Application shall either use xChannellIOWrite (DPM) or the UpdateProviderData callback function (Linkable Object) to allow the Stack to do so. Even if the Device has no Input data this shall be done. Calling xChannellIOWrite may be called with data length zero (Just to toggle the handshake flags). It may happen that xChannellIOWrite reports an error (COM-Flag not set) which can be ignored.

### 7.2.5.1 Application Ready Request

This request packet has to be sent by application to the stack if Application Ready shall be sent to the PROFINET IO-Controller.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
} PNS_IF_HANDLE_DATA_T;

typedef struct PNS_IF_HANDLE_PACKET_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_HANDLE_DATA_T         tData;
} PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T      PNS_IF_APPL_READY_REQ_T;
```

## Packet Description

Structure PNS_IF_APPL_READY_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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	0	Status not in use for indications.
	ulCmd	UINT32	0x1F10	PNS_IF_SET_APPL_READY_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_APPL_READY_REQ_DATA_T			
	hDeviceHandle	UINT32		The device handle representing the AR for which Application Ready should be signaled. Use the device handle from associated Parameter End Indication.

Table 71: PNS\_IF\_APPL\_READY\_IND\_T - Application Ready

### 7.2.5.2 Application Ready Confirmation

The stack will respond with this packet.

### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
} PNS_IF_HANDLE_DATA_T;

typedef struct PNS_IF_HANDLE_PACKET_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_HANDLE_DATA_T         tData;
} PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T          PNS_IF_APPL_READY_CNF_T;
```

## Packet Description

Structure PNS_IF_APPL_READY_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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 below.
	ulCmd	UINT32	0x1F11	PNS_IF_SET_APPL_READY_CNF-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_HANDLE_DATA_T			
	hDeviceHandle	UINT32		The device handle.

Table 72: PNS\_IF\_APPL\_READY\_CNF\_T - Application Ready Confirmation



## 7.2.6 AR InData Service

With this service the stack indicates to the application that the first cyclic frame from the IO-Controller was received after ApplicationReady was sent by the IO-Device stack.

### 7.2.6.1 AR InData Indication

This packet indicates the receipt of the first cyclic frame to the application.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
} PNS_IF_HANDLE_DATA_T;

typedef struct PNS_IF_HANDLE_PACKET_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_HANDLE_DATA_T        tData;
} PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T          PNS_IF_AR_IN_DATA_IND_T;
```

#### Packet Description

Structure PNS_IF_AR_IN_DATA_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	Status not in use for indications.
	ulCmd	UINT32	0x1F28	PNS_IF_AR_INDATA_IND-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_HANDLE_DATA_T			
	hDeviceHandle	UINT32		The device handle

Table 73: PNS\_IF\_AR\_IN\_DATA\_IND\_T - AR InData Indication

### 7.2.6.2 AR InData Response

The application has to return this packet.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
} PNS_IF_HANDLE_DATA_T;

typedef struct PNS_IF_HANDLE_PACKET_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_HANDLE_DATA_T         tData;
} PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T          PNS_IF_AR_IN_DATA_RSP_T;
```

#### Packet Description

Structure PNS_IF_AR_IN_DATA_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	The status has to be okay for this service.
	ulCmd	UINT32	0x1F29	PNS_IF_AR_INDATA_RES-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	Structure PNS_IF_HANDLE_DATA_T			
	hDeviceHandle	UINT32		The device handle

Table 74: PNS\_IF\_AR\_IN\_DATA\_RSP\_T - AR InData Response

## 7.2.7 Store Remanent Data Service

Using this service the stack indicates the presence of remanent data to the user. The application is responsible for storing this data into a permanent storage. After the next power cycle the application has to restore the stacks remanent data using the Load Remanent Data Service before configuring the stack.



### Note:

This service is used by the protocol stack to request remanent storage of protocol parameters by the application. The service is used if the protocol stack is used as a RAM-based LFW or as LOM. Additionally, the application can configure the stack to use this service.

### 7.2.7.1 Store Remanent Data Indication

Using this packet the stack indicates the presence of remanent data to the user application.

The packet itself is only defined to contain 1 byte. The correct amount of data is given by the stack in the packet header's field `ulLen`. The application should be able to handle up to 8192 Byte of remanent data.

When large amounts of data are transferred and DPM is used, these data will be divided to multiple response packets which have to be evaluated one by another.

### Packet Structure Reference

```
typedef struct PNS_IF_STORE_REMANENT_DATA_IND_DATA_Ttag
{
    TLR_UINT8 abData[1];
} PNS_IF_STORE_REMANENT_DATA_IND_DATA_T;

typedef struct PNS_IF_STORE_REMANENT_DATA_IND_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_STORE_REMANENT_DATA_IND_DATA_T    tData;
} PNS_IF_STORE_REMANENT_DATA_IND_T;
```

### Packet Description

Structure PNS_IF_STORE_REMANENT_DATA_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... $2^{32} - 1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	1 + n	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indications.

Structure PNS_IF_STORE_REMANENT_DATA_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
	ulCmd	UINT32	0x1FEA	PNS_IF_STORE_REMANENT_DATA_IND-command
	ulExt	UINT32	x	Used for fragmentation, do not change.
	ulRout	UINT32	x	Routing, do not change.
Data	structure PNS_IF_HANDLE_DATA_T			
	abData[1]	UINT8[]		The remanent data to be stored by application. Only the first byte is shown in the packet definition. The application has to store the amount of bytes reported by this packet header's field ulLen.

Table 75: PNS\_IF\_STORE\_REMANENT\_DATA\_IND\_T - Store Remanent Data Indication

### 7.2.7.2 Store Remanent Data Response

The application has to return this packet on reception of the PNS\_IF\_STORE\_REMANENT\_DATA\_IND\_T indication.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_STORE_REMANENT_DATA_RES_T;
```

#### Packet Description

Structure PNS_IF_STORE_REMANENT_DATA_RES_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	The status has to be okay for this service.
	ulCmd	UINT32	0x1FEB	PNS_IF_STORE_REMANENT_DATA_RES-command
	ulExt	UINT32	x	Used for fragmentation. Fill in the value from the indication packet.
	ulRout	UINT32	x	Routing, do not touch.

Table 76: PNS\_IF\_STORE\_REMANENT\_DATA\_RES\_T - Store Remanent Data Response

## 7.3 Acyclic Events indicated by the Stack

This section describes the acyclic events/services that the stack indicates to the application. Depending on the service (indication packet), the application has to perform an action or simply has to return a response packet.

Services like “*APDU Status Changed*” or “*Alarm Indication*” will only appear while cyclic data exchange is active. Services like “*Link Status Changed*”, “*Error Indication*” and “*Start/Stop LED Blinking*” are independent of the state of cyclic data exchange.

Table 77 lists all packets of acyclic events indicated by the stack.

Packet overview of acyclic events indicated by the PROFINET IO Device stack			
No. of section	Packet	Command code	Page
6.3.1	Read Record Indication	0x1F36	134
	Read Record Response	0x1F37	136
6.3.2	Write Record Indication	0x1F3A	138
	Write Record Response	0x1F3B	139
6.3.3	AR Abort Indication Indication	0x1F2A	142
	AR Abort Indication Response	0x1F2B	144
6.3.4	Save Station Name Indication	0x1F1A	146
	Save Station Name Response	0x1F1B	147
6.3.5	Save IP Address Indication	0x1FB8	149
	Save IP Address Response	0x1FB9	150
6.3.6	Start LED Blinking Indication	0x1F1E	151
	Start LED Blinking Response	0x1F1F	152
6.3.7	Stop LED Blinking Indication	0x1F20	153
	Stop LED Blinking Response	0x1F21	154
6.3.8	Reset Factory Settings Indication	0x1F18	157
	Reset Factory Settings Response	0x1F19	159
6.3.9	APDU Status Changed Indication	0x1F2E	160
	APDU Status Changed Response	0x1F2F	161
6.3.10	Alarm Indication	0x1F30	163
	Alarm Indication Response	0x1F31	165
6.3.11	Release Request Indication	0x1FD6	166
	Release Request Indication Response	0x1FD7	167
6.3.12	Link Status Changed Indication	0x1F70	168
	Link Status Changed Response	0x1F71	169
6.3.13	Error Indication	0x1FDC	171
	Error Indication Response	0x1FDD	172
6.3.14	Read I&M Indication	0x1F32	173
	Read I&M Response	0x1F33	174
6.3.15	Write I&M Indication	0x1F34	179
	Write I&M Response	0x1F35	180
6.3.16	Parameterization Speedup Support Indication	0x1FF8	182
	Parameterization Speedup Supported Response	0x1FF9	183
6.3.17	Event Indication	0x1FFE	185
	Event Indication Response	0x1FFF	186

Table 77: Packet overview of acyclic events indicated by the PROFINET IO Device stack

### 7.3.1 Read Record Service

With the Read Record Service the stack indicates the receipt of a Read Record Request from the IO-Controller. The application will be provided with all parameters needed to answer the request like slot, subslot and index. The application has to return the Read Record Response packet to the stack so that the stack can send the Read Response to the PROFINET IO-Controller.


**Note:**

For this service, a timeout is implemented in the stack. If the application does not answer within the timeout the stack will automatically generate a negative response to the IO-Controller. See section *Timeout for Response Packets* on page 25 for the timeout value.


**Note:**

The LOM target supports up to 32 KB of read record response data payload. The maximum amount of the data the actual packet can hold is indicated by the indication's `ulLenToRead` field. The user must not exceed this upper size limit in order to avoid memory corruption.


**Note:**

From stack version V3.8.0.0: The LFW target supports up to 8 KB of read record response data payload. For this the data must be transferred using the fragmentation transfer described in section *Packet fragmentation* on page 33.

#### 7.3.1.1 Read Record Indication

This packet indicates the receipt of a Read Record Request by the stack to the application.

##### Packet Structure Reference

```
typedef struct PNS_IF_READ_RECORD_IND_DATA_Ttag
{
    TLR_UINT32    hRecordHandle;
    TLR_UINT32    hDeviceHandle;
    TLR_UINT32    ulSequenceNum;
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_UINT32    ulIndex;
    TLR_UINT32    ulLenToRead;
} PNS_IF_READ_RECORD_IND_DATA_T;

typedef struct PNS_IF_READ_RECORD_IND_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T    tHead;
    /** packet data */
    PNS_IF_READ_RECORD_IND_DATA_T    tData;
} PNS_IF_READ_RECORD_IND_T;
```

## Packet Description

Structure PNS_IF_READ_RECORD_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	32	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indications.
	ulCmd	UINT32	0x1F36	PNS_IF_READ_RECORD_IND-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_READ_RECORD_IND_DATA_T			
	hRecordHandle	UINT32		A stack internal identifier which belongs to this indication.
	hDeviceHandle	UINT32		The device handle.
	ulSequenceNum	UINT32		The sequence number used by IO-Controller for this Read Record Request.
	ulApi	UINT32		The API the IO-Controller wants to read.
	ulSlot	UINT32		The slot the IO-Controller wants to read.
	ulSubslot	UINT32		The subslot the IO-Controller wants to read.
	ulIndex	UINT32		The index the IO-Controller wants to read.
	ulLenToRead	UINT32	1..n	The number of bytes the IO-Controller requested. If the stack is accessed using DPM or SHM n may be up to 1024 Bytes. If the Protocol Stacks AP Queue is directly programmed, n may be up to 32768.

Table 78: PNS\_IF\_READ\_RECORD\_IND\_T - Read Record Indication

### 7.3.1.2 Read Record Response

The application has to send this packet as response to a Read Record Indication.


**Note:**

The application will not receive any feedback if the stack was able to successfully send the Read Record Response to the IO-Controller. However, if the Read Record Response packet has not been correctly received at the IO-Controller or the stack for some other reason was not able to answer to the IO-Controller, this will be indicated to application using the Error Indication service (see section 6.3.13).

#### Packet Structure Reference

```
typedef struct PNS_IF_READ_RECORD_RSP_DATA_Ttag
{
    TLR_UINT32    hRecordHandle;
    TLR_UINT32    hDeviceHandle;
    TLR_UINT32    ulSequenceNum;
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_UINT32    ulIndex;
    TLR_UINT32    ulReadLen;
    /* PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2 */
    TLR_UINT32    ulPnio;
    TLR_UINT16    usAddValue1;
    TLR_UINT16    usAddValue2;
    TLR_UINT8     abRecordData[PNS_IF_MAX_RECORD_DATA_LEN];
} PNS_IF_READ_RECORD_RSP_DATA_T;

typedef struct PNS_IF_READ_RECORD_RSP_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T    tHead;
    /** packet data */
    PNS_IF_READ_RECORD_RSP_DATA_T    tData;
} PNS_IF_READ_RECORD_RSP_T;
```

#### Packet Description

Structure PNS_IF_READ_RECORD_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	40 + n	Packet data length in bytes. n is the value of ulReadLen.
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	The status has to be okay for this service.
	ulCmd	UINT32	0x1F37	PNS_IF_READ_RECORD_RES-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons



Structure PNS_IF_READ_RECORD_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_READ_RECORD_RSP_DATA_T			
	hRecordHandle	UINT32		A stack internal identifier which belongs to this indication.
	hDeviceHandle	UINT32		The device handle
	ulSequenceNum	UINT32		The sequence number passed in the indication.
	ulApi	UINT32		The API the IO-Controller wants to read.
	ulSlot	UINT32		The slot the IO-Controller wants to read.
	ulSubslot	UINT32		The subslot the IO-Controller wants to read.
	ulIndex	UINT32		The index the IO-Controller wants to read.
	ulReadLen	UINT32	0..n	The record data length read. N shall be smaller or equal than value of indication's ulLenToRead field.
	ulPnio	UINT32		PROFINET error code, consists of ErrorCode, ErrorDecode, ErrorCode1 and ErrorCode2. See section "PROFINET Status Code".
	usAddValue1	UINT16		Additional Value 1. This value is reserved for usage within Profiles and shall be set to zero by default.
	usAddValue2	UINT16		Additional Value 2. This value is reserved for usage within Profiles and shall be set to zero by default.
	abRecordData[1024]	UINT8[]		Read record data. The array must not be larger than the value in indication's ulLenToRead field.

Table 79: PNS\_IF\_READ\_RECORD\_RSP\_T - Read Record Response

### 7.3.2 Write Record Service

With the Write Record Service the stack indicates the receipt of a Write Record Request from the IO-Controller. The application will be provided with parameters contained in the request like slot, subslot, index and the data. It has to handle the data (e.g. send it to configure modules). It has to return the Write Record Response packet to the stack so that the stack can answer the request from the IO-Controller.

If the Write Record is received by the application during connection establishment it is recommended to not directly configure the submodule with the parameters but to wait for the Parameter End Indication.



#### Note:

From stack version V3.8.0.0: The LFW target supports up 8 KB of write record indication data payload. For this the data must be transferred using the fragmentation transfer described in section *Packet fragmentation* on page 33. The application might pre-calculate the size of the unfragmented packet using the data field `ulLenToWrite` which is transferred in the first fragment.

#### 7.3.2.1 Write Record Indication

With this packet the receipt of a Write Request is indicated by the stack to application. It contains the data sent by the PROFINET IO-Controller.

#### Packet Structure Reference

```
typedef struct PNS_IF_WRITE_RECORD_IND_DATA_Ttag
{
    TLR_UINT32    hRecordHandle;
    TLR_UINT32    hDeviceHandle;
    TLR_UINT32    ulSequenceNum;
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_UINT32    ulIndex;
    TLR_UINT32    ulLenToWrite;
    TLR_UINT8     abRecordData[PNS_IF_MAX_RECORD_DATA_LEN];
} PNS_IF_WRITE_RECORD_IND_DATA_T;

typedef struct PNS_IF_WRITE_RECORD_IND_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_WRITE_RECORD_IND_DATA_T    tData;
} PNS_IF_WRITE_RECORD_IND_T;
```

#### Packet Description

Structure PNS_IF_WRITE_RECORD_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons

Structure PNS_IF_WRITE_RECORD_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	32 +n	Packet data length in bytes. n is the value of ulLenToWrite.
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indications.
	ulCmd	UINT32	0x1F3A	PNS_IF_WRITE_RECORD_IND-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_WRITE_RECORD_IND_DATA_T			
	hRecordHandle	UINT32		A stack internal identifier which belongs to this indication.
	hDeviceHandle	UINT32		The device handle
	ulSequenceNum	UINT32		The sequence number used by IO-Controller for this Write Record Request.
	ulApi	UINT32		The API the IO-Controller wants to write to.
	ulSlot	UINT32		The slot the IO-Controller wants to write to.
	ulSubslot	UINT32		The subslot the IO-Controller wants to write to.
	ulIndex	UINT32		The index the IO-Controller wants to write to.
	ulLenToWrite	UINT32	1..n	The length of write record data. If the stack is accessed using DPM or SHM n may be up to 1024 Bytes. If the Protocol Stacks AP Queue is programmed directly, n may be up to 32768.
	abRecordData[1024]	UINT8[]		The write record data. The actual length of the array depends on ulLenToWrite field.

Table 80: PNS\_IF\_WRITE\_RECORD\_IND\_T - Write Record Indication

### 7.3.2.2 Write Record Response

In order to respond to a Write Record Indication, the application has to send this packet to the stack.



#### Note:

The application will not get any feedback if the stack was able to successfully send the Write Record Response to the IO-Controller. However, if a problem concerning the Write Record Response packet occurred or the protocol stack for some other reason was not able to answer to the IO-Controller this will be indicated to application using the Error Indication Service (see section 6.3.13).

### Packet Structure Reference

```
typedef struct PNS_IF_WRITE_RECORD_RSP_DATA_Ttag
{
    TLR_UINT32    hRecordHandle;
    TLR_UINT32    hDeviceHandle;
    TLR_UINT32    ulSequenceNum;
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
```

```

TLR_UINT32    ulIndex;
TLR_UINT32    ulWriteLen;
/* PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2 */
TLR_UINT32    ulPnio;
TLR_UINT16    usAddValue1;
TLR_UINT16    usAddValue2;
} PNS_IF_WRITE_RECORD_RSP_DATA_T;

typedef struct PNS_IF_WRITE_RECORD_RSP_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T            tHead;
    /** packet data */
    PNS_IF_WRITE_RECORD_RSP_DATA_T tData;
} PNS_IF_WRITE_RECORD_RSP_T;

```

## Packet Description

Structure PNS_IF_WRITE_RECORD_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	40	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 below.
	ulCmd	UINT32	0x1F3B	PNS_IF_WRITE_RECORD_RES-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_WRITE_RECORD_RSP_DATA_T			
	hRecordHandle	UINT32		A stack internal identifier which belongs to this indication.
	hDeviceHandle	UINT32		The device handle
	ulSequenceNum	UINT32		The sequence number passed in the indication.
	ulApi	UINT32		The API the IO-Controller wants to write to.
	ulSlot	UINT32		The slot the IO-Controller wants to write to.
	ulSubslot	UINT32		The subslot the IO-Controller wants to write to.
	ulIndex	UINT32		The index the IO-Controller wants to write to.
	ulWriteLen	UINT32		The record data length written.
	ulPnio	UINT32		PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See section "PROFINET Status Code".
	usAddValue1	UINT16		Additional Value 1. This value is reserved for usage within Profiles and shall be set to zero by default.

Structure PNS_IF_WRITE_RECORD_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
	usAddValue2	UINT16		Additional Value 2. This value is reserved for usage within Profiles and shall be set to zero by default.

Table 81: PNS\_IF\_WRITE\_RECORD\_RSP\_T - Write Record Response

### 7.3.3 AR Abort Indication service

With this service the stack informs the application that a formerly established connection to the IO-Controller no longer exists. Possible reasons are:

- The stack did not receive cyclic frames from IO-Controller
- The IO-Controller closed the connection (RPC Release, RPC Abort, abort alarm)
- The application disallowed communication (Set Bus state OFF)
- The application reconfigured the stack using Channel Init or Configuration Reload
- The System Redundancy Takeover Timeout occurred.



#### Note:

If an AR disconnects it is required to invalidate (set to zero) or apply substitute values to the consumer process data in most cases. In order to do so, the stack needs write access to the consumer process data image. Therefore, if the application does not update the consumer data periodically or does not use the event service, it is required, that the application allows the stack to update the consumer process data image by updating from the consumer process data image.

#### 7.3.3.1 AR Abort Indication Indication

With this packet the stack informs the application that the established connection no longer exists. This is informative for the application.

#### Packet Structure Reference

```
typedef struct PNS_IF_AR_ABORT_IND_IND_DATA_Ttag
{
    TLR_UINT32    hDeviceHandle;
    TLR_UINT32    ulPnio;
} PNS_IF_AR_ABORT_IND_IND_DATA_T;

typedef struct PNS_IF_AR_ABORT_IND_IND_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_AR_ABORT_IND_IND_DATA_T    tData;
} PNS_IF_AR_ABORT_IND_IND_T;
```

#### Packet Description

Structure PNS_IF_AR_ABORT_IND_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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

Structure PNS_IF_AR_ABORT_IND_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
	ulSta	UINT32	0	Status not in use for indications.
	ulCmd	UINT32	0x1F2A	PNS_IF_AR_ABORT_IND-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_AR_ABORT_IND_IND_DATA_T			
	hDeviceHandle	UINT32		The device handle.
	ulPnio	UINT32		PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See section “ <i>PROFINET Status Code</i> ”..

Table 82: PNS\_IF\_AR\_ABORT\_IND\_IND\_T - AR Abort Indication

### 7.3.3.2 AR Abort Indication Response

The application has to return this packet to the stack.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
} PNS_IF_HANDLE_DATA_T;

typedef struct PNS_IF_HANDLE_PACKET_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_HANDLE_DATA_T         tData;
} PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T          PNS_IF_AR_ABORT_IND_RSP_T;
```

#### Packet Description

Structure PNS_IF_AR_ABORT_IND_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	The status has to be okay for this service.
	ulCmd	UINT32	0x1F2B	PNS_IF_AR_ABORT_RES-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	Structure PNS_IF_HANDLE_DATA_T			
	hDeviceHandle	UINT32		The device handle.

Table 83: PNS\_IF\_AR\_ABORT\_IND\_RSP\_T - AR Abort Indication Response



### 7.3.4 Save Station Name Service

With this service the stack indicates to the application that a DCP Set NameOfStation request has been received. The new NameOfStation is automatically set by the stack. If the application configured the stack using packets, the application must be able to store the station name into non-volatile memory. The station name must be restored on power up by means of the *Set Configuration Request* packet. The correct application behavior is shown in the following figure.

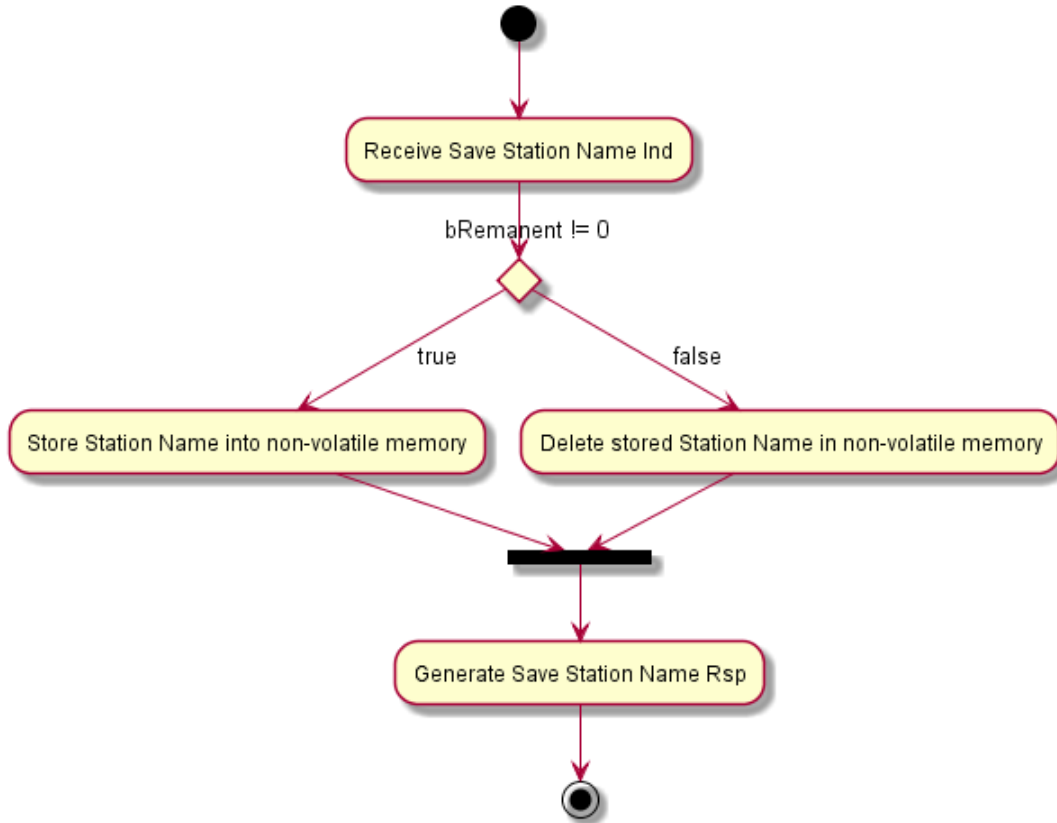


Figure 18: Desired Application behavior on Save Station Name Indication



**Note:**

If the flag `bRemanent` is not set the application shall delete the permanently stored Station Name and use an empty Station Name after the next PowerUp cycle.

**This is a certification relevant action required to be implemented by application.**

**ote:** The application can configure the stack to save the NameOfStation on its own. See Table 32 Bit D17.

### 7.3.4.1 Save Station Name Indication

The stack sends this packet to the application to indicate the change of NameOfStation.

#### Packet Structure Reference

```
typedef struct PNS_IF_SAVE_STATION_NAME_IND_DATA_Ttag
{
    TLR_UINT16  usNameLen;
    TLR_UINT8   bRemanent;
    TLR_UINT8   abNameOfStation[PONIO_MAX_NAME_OF_STATION];
} PNS_IF_SAVE_STATION_NAME_IND_DATA_T;

typedef struct PNS_IF_SAVE_STATION_NAME_IND_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_SAVE_STATION_NAME_IND_DATA_T  tData;
} PNS_IF_SAVE_STATION_NAME_IND_T;

/* Response packet */
typedef TLR_EMPTY_PACKET_T          PNS_IF_SAVE_STATION_NAME_RSP_T;
```

#### Packet Description

Structure PNS_IF_SAVE_STATION_NAME_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	243	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indications.
	ulCmd	UINT32	0x1F1A	PNS_IF_SAVE_STATION_NAME_IND-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_SAVE_STATION_NAME_IND_DATA_T			
	usNameLen	UINT16	0..240	Length of the new NameOfStation.
	bRemanent	UINT8	0 1	Do not save the new NameOfStation remanent. Save the NameOfStation remanent.
	abNameOfStation[240]	UINT8[]		The new NameOfStation as ASCII byte-array. For the station name, only small characters are allowed.

Table 84: PNS\_IF\_SAVE\_STATION\_NAME\_IND\_T - Save Station Name Indication

### 7.3.4.2 Save Station Name Response

The application acknowledges the indication with this packet.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_SAVE_STATION_NAME_RSP_T;
```

#### Packet Description

Structure PNS_IF_SAVE_STATION_NAME_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	See below.
	ulCmd	UINT32	0x1F1B	PNS_IF_SAVE_STATION_NAME_RES-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 85: PNS\_IF\_SAVE\_STATION\_NAME\_RSP\_T - Save Station Name Response

### 7.3.5 Save IP Address Service

With this service the stack indicates to the application that a DCP Set IP request was received. The new IP is automatically set by the stack. The application is informed about the new IP address. If the application configured the stack using packets, the application must be able to store the IP address parameters into non-volatile memory. The IP address parameters must be restored on power up by means of Set Configuration Request packet.

The correct application behavior is shown in the following figure.

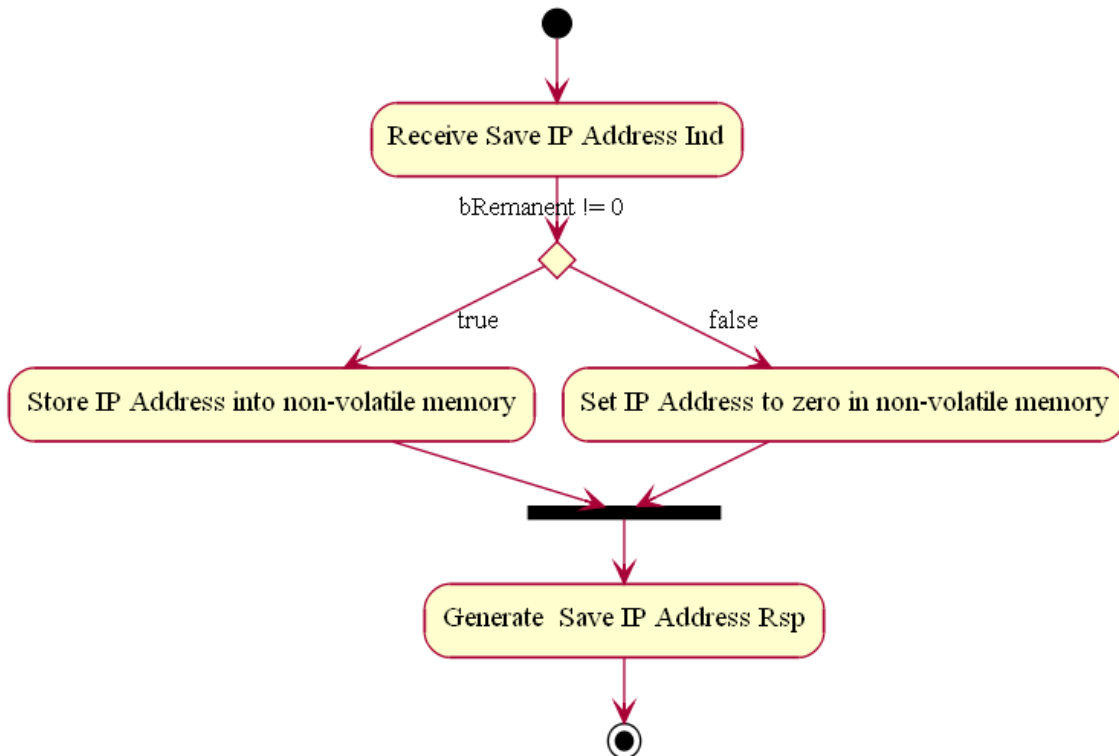


Figure 19: Desired application behavior on Save IP Address Indication



**Note:**

If the flag `bRemanent` is not set the application shall set the permanently stored IP parameters to `0.0.0.0` and use IP `0.0.0.0` after the next PowerUp cycle.

**This is a certification relevant action required to be implemented by application.**



**Note:**

The application can configure the stack to save the IP parameters by itself. See Table X30 Bit D17.

### 7.3.5.1 Save IP Address Indication

This packet indicates the receipt of a DCP Set IP request by the stack.

#### Packet Structure Reference

```
typedef struct PNS_IF_SAVE_IP_ADDR_IND_DATA_Ttag
{
    TLR_UINT32    ulIpAddr;
    TLR_UINT32    ulNetMask;
    TLR_UINT32    ulGateway;
    TLR_UINT8     bRemanent;
} PNS_IF_SAVE_IP_ADDR_IND_DATA_T;

typedef struct PNS_IF_SAVE_IP_ADDR_IND_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_SAVE_IP_ADDR_IND_DATA_T tData;
} PNS_IF_SAVE_IP_ADDR_IND_T;
```

#### Packet Description

Structure PNS_IF_SAVE_IP_ADDRESS_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	13	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indications.
	ulCmd	UINT32	0x1FB8	PNS_IF_SAVE_IP_ADDR_IND-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_SAVE_IP_ADDRESS_IND_DATA_T			
	ulIpAddr	UINT32		The new IP address.
	ulNetMask	UINT32		The new network mask.
	ulGateway	UINT32		The new gateway address.
	bRemanent	UINT8	0 1	Do not save the new IP parameters remanent. Save the IP parameters remanent.

Table 86: PNS\_IF\_SAVE\_IP\_ADDRESS\_IND\_T - Save IP Address Indication

### 7.3.5.2 Save IP Address Response

The application has to return this packet.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_SAVE_IP_ADDR_RSP_T;
```

#### Packet Description

Structure PNS_IF_SAVE_IP_ADDRESS_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	See below.
	ulCmd	UINT32	0x1FB9	PNS_IF_SAVE_IP_ADDRESS_RSP-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 87: PNS\_IF\_SAVE\_IP\_ADDRESS\_RSP\_T - Save IP Address Response

### 7.3.6 Start LED Blinking Service

The stack informs the application about the receipt of a DCP Set Signal request with this service. The application shall start blinking with an appropriate LED immediately. The LED shall blink with the specified frequency.



#### Note:

If the stack is used as LOM and is configured to use LEDs and if a valid Blinking LED-Name (see section) is given the stack will automatically cause the LED to blink. This indication is only informative.

#### 7.3.6.1 Start LED Blinking Indication

With this indication packet the stack informs the application about the receipt of a DEC SET Signal Request.

#### Packet Structure Reference

```
typedef struct PNS_IF_START_LED_BLINKING_IND_DATA_Ttag
{
    TLR_UINT32 ulFrequency;
} PNS_IF_START_LED_BLINKING_IND_DATA_T;

typedef struct PNS_IF_START_LED_BLINKING_IND_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_START_LED_BLINKING_IND_DATA_T    tData;
} PNS_IF_START_LED_BLINKING_IND_T;
```

#### Packet Description

Structure PNS_IF_START_LED_BLINKING_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	Status not in use for indications.
	ulCmd	UINT32	0x1F1E	PNS_IF_START_LED_BLINKING_IND-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_START_LED_BLINKING_IND_DATA_T			
	ulFrequency	UINT32		The frequency the LED shall blink with.

Table 88: PNS\_IF\_START\_LED\_BLINKING\_IND\_T - Start LED Blinking Indication

### 7.3.6.2 Start LED Blinking Response

The application shall return this response packet. If blinking with the LED could not be started, the packet shall contain a negative status.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_START_LED_BLINKING_RSP_T;
```

#### Packet Description

Structure PNS_IF_START_LED_BLINKING_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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 below.
	ulCmd	UINT32	0x1F1F	PNS_IF_START_LED_BLINKING_RES-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 89: PNS\_IF\_START\_LED\_BLINKING\_RSP\_T - Start LED Blinking Response



### 7.3.7 Stop LED Blinking Service

The stack informs the application to stop blinking with the appropriate LED.



**Note:**

If the stack is used as LOM and is configured to use LEDs (see section) and if a valid Blinking LED-Name (see section) is given the stack will automatically blink with the LED and this indication is only informative.

#### 7.3.7.1 Stop LED Blinking Indication

The indication packet is sent from the stack to inform the application to stop blinking with the LED.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_STOP_LED_BLINKING_IND_T;
```

#### Packet Description

Structure PNS_IF_STOP_LED_BLINKING_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	Status not in use for indications.
	ulCmd	UINT32	0x1F20	PNS_IF_STOP_LED_BLINKING_IND-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 90: PNS\_IF\_STOP\_LED\_BLINKING\_IND\_T - Stop LED Blinking Indication

### 7.3.7.2 Stop LED Blinking Response

The application shall return this response packet.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_STOP_LED_BLINKING_RSP_T;
```

#### Packet Description

Structure PNS_IF_STOP_LED_BLINKING_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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 below.
	ulCmd	UINT32	0x1F21	PNS_IF_STOP_LED_BLINKING_RES-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 91: PNS\_IF\_STOP\_LED\_BLINKING\_RSP\_T - Stop LED Blinking Response

### 7.3.8 Reset Factory Settings Service

The stack indicates to the application, that a “Reset to Factory Settings” request has been received via DCP. If the application has configured the stack using packets, the application shall now reset some values within the non-volatile memory. The reset values must be restored on power up by means of a Set Configuration Request packet.

The PROFINET specification defines different types of “Reset to Factory”. Each type defines which data the application shall reset. The correct application behavior is shown in the following figure.

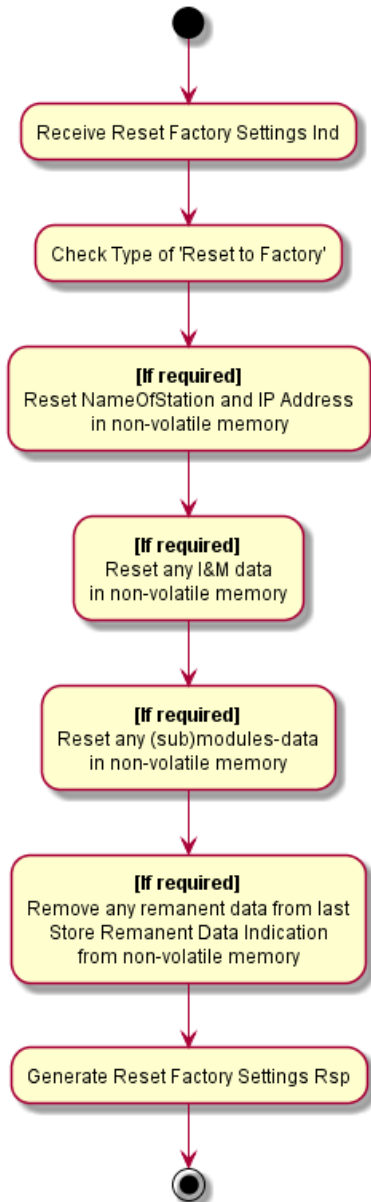


Figure 20: Application behavior on Reset Factory Settings Indication

The application has to use the following reset values as default:

Parameter	Reset Value
NameOfStation	"" (empty string)
IP Address	0.0.0.0
Network Mask	0.0.0.0
Gateway	0.0.0.0
I&M1-I&M3	Set all elements to " " (space character, 0x20).
I&M4	Set all elements to zero (0x00). <b>Exception:</b> The application has implemented a profile which is using I&M4 and the used profile forbids to reset I&M4 data.
Remanent Data From Store Remanent Data Service	Remove
(sub)modules data	If any (sub)module stores own configuration parameters in non-volatile memory, they should be set to their defaults.



#### Notes:

PROFINET specifies that the receipt of a Reset to factory settings Request shall automatically stop any running cyclic communication. This is done automatically by the stack. The stack indicates this with an Abort Indication service to the application.



If the stack is configured using a SYCON.net database then the stack will automatically change the IP parameters and NameOfStation stored in the database to the default values.



If the stack is configured using Set Configuration Service the internally stored parameters will also be changed by the stack in the way that e.g. after a Channellnit the default values will be used.

### 7.3.8.1 Reset Factory Settings Indication

The indication packet sent by the stack.

#### Packet Structure Reference

```
typedef struct
{
    TLR_UINT16 usResetCode;
} PNS_IF_RESET_FACTORY_SETTINGS_IND_DATA_T;

typedef struct
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_RESET_FACTORY_SETTINGS_IND_DATA_T tData;
} PNS_IF_RESET_FACTORY_SETTINGS_IND_T;
```

#### Packet Description

Structure PNS_IF_RESET_FACTORY_SETTINGS_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	2	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indications.
	ulCmd	UINT32	0x1F18	PNS_IF_RESET_FACTORY_SETTINGS_IND-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	0	Routing not in use, set to zero for compatibility reasons
Data	structure PNS_IF_RESET_FACTORY_SETTINGS_IND_DATA_T			
	usResetCode	UINT16		The reset code, see the table.

Table 92: PNS\_IF\_RESET\_FACTORY\_SETTINGS\_IND\_T - Reset Factory Settings Indication

The requested type of reset is indicated to the application in parameter `usResetCode` (it has the same coding as the DCP `BlockQualifier` with option `ControlOption` and suboption `SuboptionResetToFactory`). Parameter `usResetCode` defines which data the application shall reset:

Name	Value	Description
PNS_IF_RESET_FACTORY_SETTINGS_IF_APPLICATION	2	<b>Reset application data in interface</b> Reset data which have been stored permanently in submodules and modules to factory values. - Manufacturer specific record data shall be set to factory values. - All I&M data shall be set also to factory values. <b>The stack does not support this suboption. Reserved for future implementation.</b>
PNS_IF_RESET_FACTORY_SETTINGS_IF_COMMUNICATION	4	<b>Reset communication parameter in interface</b> All parameters active for the interface or the ports and the ARs shall be set to the default value and reset if permanently stored. This mode is intended to set the addressed communication interface of a device into a state which is almost similar to the "out of the box" state. In particular these are: - NameOfStation shall be set to "" (empty string) - IP suite parameter shall be set to 0.0.0.0 - DHCP parameters (if available) shall be reset - All PDev parameters shall be set to factory values (The stack resets these values internally) - Parameters adjusted by SNMP, like sysContact, sysName, and sysLocation from MIB-II (The stack resets these values internally)  <b>Observe that all I&amp;M data shall not be set to factory values!</b> In case, the application takes care of the remanent data (i.e. uses <i>Load Remanent Data Service</i> and <i>Store Remanent Data Service</i> ) but I&M requests are handled internally by the stack (which means that flag <code>PNS_IF_SYSTEM_STACK_HANDLE_I_M_ENABLED</code> is set by the <i>Set Configuration Request</i> ), the application <b>should not remove the remanent data</b> , because the stack preserves I&M data inside of the remanent data. <b>Else the application should reset (remove) remanent data.</b>
PNS_IF_RESET_FACTORY_SETTINGS_IF_ENGEINEERING	6	<b>Reset engineering parameter in interface</b> Reset engineering parameters which have been stored permanently in the IOC or IOD to factory values. - If a node is able to switch its functionality from IOC to IOD and vice versa via engineering system, the factory functionality shall be activated. - An application program loaded by an engineering system shall be reset (or removed). - A configuration loaded by an engineering system shall be reset (or removed). <b>The stack does not support this suboption, reserved for future implementation.</b>
PNS_IF_RESET_FACTORY_SETTINGS_IF_ALL	8	<b>Reset all stored data in interface</b> Reset all stored data in the IOD or IOC to its factory default values. <b>This covers all data related to application data (2), communication (4) and engineering parameters (6).</b> <b>Note:</b> The stack uses this code for previous version of DCP-reset ( <code>SuboptionFactoryReset</code> ) also.

Name	Value	Description
PNS_IF_RESET_FACTORY_SETTINGS_DEVICE_ALL	16	<b>Reset all stored data in device</b> Reset all data stored in the IOD or IOC to its factory values. This service shall reset the communication parameters of all interfaces of the device and should reset all parameters of the device. This mode is intended to set the device into a state which is similar to the “out of the box” state. It includes parameters adjusted by SNMP. <b>The stack does not support this suboption</b> , because it supports only one interface.
PNS_IF_RESET_FACTORY_SETTINGS_DEVICE_RESTORE	18	<b>Reset and restore data in device</b> Reset installed software revisions to factory values. <b>The stack does not support this suboption.</b>

Table 93: Possible values of the reset code

**Note:**

GSDML attribute `ResetToFactoryModes` has following relation to the reset code:

$$\text{ResetToFactoryModes} = \text{usResetCode} / 2$$

For example: `usResetCode = 4`, then `ResetToFactoryModes = 2`.

### 7.3.8.2 Reset Factory Settings Response

The application shall return this packet.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_RESET_FACTORY_SETTINGS_RSP_T;
```

#### Packet Description

Structure <code>PNS_IF_RESET_FACTORY_SETTINGS_RSP_T</code>				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure <code>TLR_PACKET_HEADER_T</code>			
	<code>ulDest</code>	UINT32		Destination queue handle of CMDEV-task process queue
	<code>ulSrc</code>	UINT32		Source queue handle of AP-task process queue
	<code>ulDestId</code>	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	<code>ulSrcId</code>	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	<code>ulLen</code>	UINT32	0	Packet data length in bytes
	<code>ulId</code>	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	<code>ulSta</code>	UINT32		See below.
	<code>ulCmd</code>	UINT32	0x1F19	PNS_IF_RESET_FACTORY_SETTINGS_RES-Command
	<code>ulExt</code>	UINT32	0	Extension not in use, set to zero for compatibility reasons
	<code>ulRout</code>	UINT32	x	Routing, do not touch

Table 94: `PNS_IF_RESET_FACTORY_SETTINGS_RSP_T` - Reset Factory Settings Response

### 7.3.9 APDU Status Changed Service

With this service the stack indicates to the application that the status field of the cyclic frames received by the stack and sent by the PROFINET IO-Controller has changed. The new Status is indicated.

#### 7.3.9.1 APDU Status Changed Indication

This packet indicates the APDU status Change to application.

#### Packet Structure Reference

```
typedef struct PNS_IF_APDU_STATUS_CHANGED_IND_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
    TLR_UINT32 ulStatus;
} PNS_IF_APDU_STATUS_CHANGED_IND_DATA_T;

typedef struct PNS_IF_APDU_STATUS_CHANGED_IND_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_APDU_STATUS_CHANGED_IND_DATA_T    tData;
} PNS_IF_APDU_STATUS_CHANGED_IND_T;
```

#### Packet Description

Structure PNS_IF_APDU_STATUS_CHANGED_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	Status not in use for indications.
	ulCmd	UINT32	0x1F2E	PNS_IF_APDU_STATUS_IND-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_APDU_STATUS_CHANGED_IND_DATA_T			
	hDeviceHandle	UINT32		The device handle
	ulStatus	UINT32		See Table 92: Meaning of bits of APDU status field. A change of the Primary/Backup is not indicated to the application

Table 95 PNS\_IF\_APDU\_STATUS\_CHANGED\_IND\_T - APDU Status Changed Indication



Bit Mask	Description
0x01	Bit is Cleared: IOCR is in state <i>Backup</i>
	Bit is Set: IOCR is in state <i>Primary</i>
0x02	Reserved Bit 1
0x04	Bit is Cleared: Data are invalid
	Bit is Set: Data are valid
0x08	Reserved Bit 2
0x10	Bit is Cleared: The provider station is in <i>stop</i> state
	Bit is Set: The provider station is in <i>run</i> state
0x20	Bit is Cleared: The provider station has a problem
	Bit is Set: The Provider station is fully operational
0x40	Reserved Bit 3
0x80	Reserved Bit 4

Table 96: Meaning of bits of APDU status field

### 7.3.9.2 APDU Status Changed Response

The application shall return this packet.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
} PNS_IF_HANDLE_DATA_T;

typedef struct PNS_IF_HANDLE_PACKET_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_HANDLE_DATA_T         tData;
} PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T          PNS_IF_APDU_STATUS_CHANGED_RSP_T;
```

## Packet Description

Structure PNS_IF_APDU_STATUS_CHANGED_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	The status has to be okay for this service.
	ulCmd	UINT32	0x1F2F	PNS_IF_APDU_STATUS_RES-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	Structure PNS_IF_HANDLE_DATA_T			
	hDeviceHandle	UINT32		Handle to the IO-Device.

Table 97: PNS\_IF\_APDU\_STATUS\_CHANGED\_RSP\_T - APDU Status Changed Response

### 7.3.10 Alarm Indication Service

With this service the stack indicates the receipt of an Alarm PDU from the IO-Controller to the application. This indication is informative for the application only.

The stack will automatically perform all actions needed to handle the alarm.

#### 7.3.10.1 Alarm Indication

This packet informs the application about the receipt of an Alarm PDU from the IO-Controller. The packet contains all information sent by the IO-Controller.

#### Packet Structure Reference

```
typedef struct PNS_IF_ALARM_IND_DATA_Ttag
{
    TLR_UINT32    hDeviceHandle;
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_UINT32    ulModuleId;
    TLR_UINT32    ulSubmodId;
    TLR_UINT16    usAlarmPriority;
    TLR_UINT16    usAlarmType;
    TLR_UINT16    usAlarmSequence;
    TLR_BOOLEAN    fDiagChannelAvailable;
    TLR_BOOLEAN    fDiagGenericAvailable;
    TLR_BOOLEAN    fDiagSubmodAvailable;
    TLR_BOOLEAN    fReserved;
    TLR_BOOLEAN    fArDiagnosisState;
    TLR_UINT16    usUserStructId;
    TLR_UINT16    usAlarmDataLen;
    TLR_UINT8      abAlarmData[PNS_IF_MAX_ALARM_DATA_LEN];
} PNS_IF_ALARM_IND_DATA_T;

typedef struct PNS_IF_ALARM_IND_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_ALARM_IND_DATA_T      tData;
} PNS_IF_ALARM_IND_T;
```

## Packet Description

Structure PNS_IF_ALARM_IND_T					Type: Indication
Area	Variable	Type	Value / Range	Description	
Head	structure TLR_PACKET_HEADER_T				
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue	
	ulSrc	UINT32		Source queue handle of AP-task process queue	
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons	
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.	
	ulLen	UINT32	54 + n	Packet data length in bytes. n is the value of usLenAlarmData.	
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet	
	ulSta	UINT32	0	Status not in use for indication.	
	ulCmd	UINT32	0x1F30	PNS_IF_ALARM_IND-command	
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons	
	ulRout	UINT32	x	Routing, do not touch	
Data	structure PNS_IF_ALARM_IND_DATA_T				
	hDeviceHandle	UINT32		The device handle	
	ulApi	UINT32		The API the alarm belongs to.	
	ulSlot	UINT32		The slot the alarm belongs to.	
	ulSubslot	UINT32		The subslot the alarm belongs to.	
	ulModuleId	UINT32		The ModuleID the alarm belongs to.	
	ulSubmodId	UINT32		The SubmoduleID the alarm belongs to.	
	usAlarmPriority	UINT16		The Alarm priority.	
	usAlarmType	UINT16		The alarm type.	
	usAlarmSequence	UINT16		The alarm sequence number.	
	fDiagChannelAvailable	BOOL32			
	fDiagGenericAvailable	BOOL32			
	fDiagSubmodAvailable	BOOL32			
	fReserved	BOOL32	0	Reserved, will be set to zero.	
	fArDiagnosisState	BOOL32			
	usUserStructId	UINT16		The User Structure Identifier.	
	usAlarmDataLen	UINT16		The length of alarm data.	
	abAlarmData[1024]	UINT8		Alarm data.	

Table 98 PNS\_IF\_ALARM\_IND\_T - Alarm Indication

### 7.3.10.2 Alarm Indication Response

The application shall return this packet.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
} PNS_IF_HANDLE_DATA_T;

typedef struct PNS_IF_HANDLE_PACKET_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_HANDLE_DATA_T        tData;
} PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T          PNS_IF_ALARM_RSP_T;
```

#### Packet Description

Structure PNS_IF_ALARM_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	The status has to be okay for this service.
	ulCmd	UINT32	0x1F31	PNS_IF_ALARM_RES-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_HANDLE_DATA_T			
	hDeviceHandle	UINT32		The device handle

Table 99: PNS\_IF\_ALARM\_RSP\_T - Alarm Indication Response

### 7.3.11 Release Request Indication Service

With this service the stack indicates the receipt of a RPC Release Request to the application. This indication is informative for the application. The stack will accept the release in any case.

#### 7.3.11.1 Release Request Indication

This packet indicates the receipt of a RPC Release Request to application.

#### Packet Structure Reference

```
typedef struct PNS_IF_RELEASE_REQ_IND_DATA_Ttag
{
    TLR_UINT32    hDeviceHandle;
    TLR_UINT16    usSessionKey;
} PNS_IF_RELEASE_REQ_IND_DATA_T;

typedef struct PNS_IF_RELEASE_REQ_IND_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_RELEASE_REQ_IND_DATA_T tData;
} PNS_IF_RELEASE_REQ_IND_T;
```

#### Packet Description

Structure PNS_IF_RELEASE_REQ_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	6	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indication.
	ulCmd	UINT32	0x1FD6	PNS_IF_RELEASE_RECV_IND-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_RELEASE_REQ_IND_DATA_T			
	hDeviceHandle	UINT32		The device handle
	usSessionKey	UINT16		The session key.

Table 100: PNS\_IF\_RELEASE\_REQ\_IND\_T - Release Request Indication

### 7.3.11.2 Release Request Indication Response

The application shall answer to the stack with this packet.

#### Packet Structure Reference

```
typedef struct PNS_IF_RELEASE_REQ_RSP_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
} PNS_IF_RELEASE_REQ_RSP_DATA_T;

typedef struct PNS_IF_RELEASE_REQ_RSP_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_RELEASE_REQ_RSP_DATA_T tData;
} PNS_IF_RELEASE_REQ_RSP_T;
```

#### Packet Description

Structure PNS_IF_RELEASE_REQ_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	The status has to be okay for this service.
	ulCmd	UINT32	0x1FD7	PNS_IF_RELEASE_RECV_RES-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_RELEASE_REQ_RSP_DATA_T			
	hDeviceHandle	UINT32		The device handle

Table 101: PNS\_IF\_RELEASE\_REQ\_RSP\_T - Release Request Indication Response

## 7.3.12 Link Status Changed Service

With this service the stack informs the application about the current Link status. This is informative for the application. Information from any earlier received Link Status Changed Indication is invalid at the time when a new Link Status Changed Indication is received.



### Note:

Depending on the requirements of the user application the stack is able to use different packet commands to indicate a change of the link state. By default the generic command RCX\_LINK\_STATUS\_CHANGE\_IND is used. However this can be changed using Set OEM Parameters Request (PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_7) to force the stack to use command PNS\_IF\_LINK\_STATUS\_CHANGE\_IND instead for compatibility reasons. This command was used by previous stack versions prior to v3.5.x.

### 7.3.12.1 Link Status Changed Indication

This packet indicates the new Link status to the application.

#### Packet Structure Reference

```
typedef struct PNS_IF_LINK_STATUS_DATA_Ttag
{
    TLR_UINT32    ulPort;
    TLR_BOOLEAN   fIsFullDuplex;
    TLR_BOOLEAN   fIsLinkUp;
    TLR_UINT32    ulSpeed;
} PNS_IF_LINK_STATUS_DATA_T;

typedef struct PNS_IF_LINK_STATUS_CHANGED_IND_DATA_Ttag
{
    PNS_IF_LINK_STATUS_DATA_T  atLinkData[2];
} PNS_IF_LINK_STATUS_CHANGED_IND_DATA_T;

typedef struct PNS_IF_LINK_STATUS_CHANGED_IND_Ttag {
    TLR_PACKET_HEADER_T      tHead;
    PNS_IF_LINK_STATUS_CHANGED_IND_DATA_T  tData;
} PNS_IF_LINK_STATUS_CHANGED_IND_T;
```

#### Packet Description

Structure PNS_IF_LINK_STATUS_CHANGED_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	32	Packet data length in bytes
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indication.



Structure PNS_IF_LINK_STATUS_CHANGED_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
	ulCmd	UINT32	0x2F8A or 0x1F70	RCX_LINK_STATUS_CHANGE_IND-command PNS_IF_LINK_STATE_CHANGE_IND-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_LINK_STATUS_CHANGED_IND_DATA_T			
	atLinkData[2]	PNS_IF_LINK_STATUS_DATA_T		Link Status Information for the 2 ports.

Table 102: PNS\_IF\_LINK\_STATUS\_CHANGED\_IND\_T - Link Status Changed Indication

Typically the indication will contain 2 PNS\_IF\_LINK\_STATUS\_DATA\_T-elements.

structure PNS_IF_LINK_STATUS_DATA_T				
Area	Variable	Type	Value / Range	Description
	ulPort	UINT32		The port-number this information belongs to.
	fIsFullDuplex	BOOL32	FALSE (0) TRUE	Is the established link FullDuplex? Only valid if flsLinkUp is set.
	fIsLinkUp	BOOL32	FALSE (0) TRUE	Is the link up for this port?
	ulSpeed	UINT32	10 or 100	If the link is up this field contains the speed of the established link. Possible values are 10 (10 MBit/s) and 100 (100MBit/s). Only valid if flsLinkUp is set.

Table 103: Structure PNS\_IF\_LINK\_STATUS\_DATA\_T

### 7.3.12.2 Link Status Changed Response

The application shall return this packet.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_LINK_STATUS_CHANGED_RSP_T;
```

#### Packet Description

Structure PNS_IF_LINK_STATUS_CHANGED_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	The status has to be okay for this service.
	ulCmd	UINT32	0x2F8B or 0x1F71	RCX_LINK_STATUS_CHANGE_RES-command PNS_IF_LINK_STATE_CHANGE_RES-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 104: PNS\_IF\_LINK\_STATUS\_CHANGED\_RSP\_T - Link Status Changed Response

### 7.3.13 Error Indication Service

With this service the stack informs the user application about an error having occurred. Two different situations are possible. Either the application returned an erroneous packet to the stack (e.g. Read Record Response with too small packet) or a (fatal) error was detected inside the stack.



#### Note:

In case that the stack reports an error it is possible that the value `ulCommand` contains a misleading value and that this command has nothing to do with the error reported in `ulErrorCode`.

#### 7.3.13.1 Error Indication

Using this packet the stack informs the application about an error.

#### Packet Structure Reference

```
typedef struct PNS_IF_USER_ERROR_IND_DATA_Ttag
{
    TLR_UINT32 ulErrorCode;
    TLR_UINT32 ulCommand;
} PNS_IF_USER_ERROR_IND_DATA_T;

typedef struct PNS_IF_USER_ERROR_IND_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_USER_ERROR_IND_DATA_T tData;
} PNS_IF_USER_ERROR_IND_T;
```

#### Packet Description

Structure <code>PNS_IF_USER_ERROR_IND_T</code>				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure <code>TLR_PACKET_HEADER_T</code>			
	<code>ulDest</code>	UINT32		Destination queue handle of CMDEV-task process queue
	<code>ulSrc</code>	UINT32		Source queue handle of AP-task process queue
	<code>ulDestId</code>	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	<code>ulSrcId</code>	UINT32	0 ... $2^{32} - 1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	<code>ulLen</code>	UINT32	8	Packet data length in bytes
	<code>ulId</code>	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
	<code>ulSta</code>	UINT32	0	Status not in use for indication.
	<code>ulCmd</code>	UINT32	0x1FDC	PNS_IF_USER_ERROR_IND-command
	<code>ulExt</code>	UINT32	0	Extension not in use, set to zero for compatibility reasons
	<code>ulRout</code>	UINT32	x	Routing, do not touch
Data	structure <code>PNS_IF_USER_ERROR_IND_DATA_T</code>			
	<code>ulErrorCode</code>	UINT32		The error code.
	<code>ulCommand</code>	UINT32	0 1 ... $2^{32} - 1$	This is an error indication for internal problems inside the stack. The command the application made the error.

Table 105: `PNS_IF_USER_ERROR_IND_T` - Error Indication Service

### 7.3.13.2 Error Indication Response

The application shall return the indication packet as Error Indication Response packet back to the stack.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_USER_ERROR_RSP_T;
```

#### Packet Description

Structure PNS_IF_USER_ERROR_RSP_T				Type: Response
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of CMDEV-task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	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	0	Status not in use for this response.
	ulCmd	UINT32	0x1FDD	PNS_IF_USER_ERROR_RSP-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 106: PNS\_IF\_USER\_ERROR\_RSP\_T - Error Indication Response

## 7.3.14 Read I&M Service

The stack uses this service to obtain I&M information from the user application if needed. I&M0-4 Information records are always stored into the physical (sub)module. For instance, in case of a modular I/O system, removing a (sub)module from one modular I/O block and plugging it into another one will result in the same I&M0-4 data when reading the I&M0-4 data from the new location.



### Note:

In order to fulfill PROFINET conformance needs, the user has to implement at least handling of I&M0, I&M1, I&M2, I&M3 and I&M0 Filter data.

In addition I&M0 shall be readable on every submodule. If I&M1-4 are supported, they shall be readable on every submodule, even if they are only writable on a specific device representant submodule.

### 7.3.14.1 Read I&M Indication

This indication packet is sent by the stack whenever a controller or a supervisor reads out I&M information in order to obtain the requested I&M data.

#### Packet Structure Reference

```
typedef struct PNS_IF_READ_IM_IND_DATA_Ttag
{
    TLR_UINT32    ulApi;
    TLR_UINT16    usSlot;
    TLR_UINT16    usSubslot;
    TLR_UINT8     bIMType;
    TLR_UINT8     abReserved[3];
} PNS_IF_READ_IM_IND_DATA_T;

typedef struct PNS_IF_READ_IM_IND_Ttag
{
    TLR_PACKET_HEADER_T    tHead;
    PNS_IF_READ_IM_IND_DATA_T    tData;
} PNS_IF_READ_IM_IND_T;
```

#### Packet Description

Structure PNS_IF_READ_IM_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of application task process queue
	ulSrc	UINT32		Source Queue-Handle of PNSIF task process queue
	ulDestId	UINT32	0	Destination End Point Identifier
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	12	Packet data length in bytes. n depends on the parameter type contained in the packet.
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32		Status not used for request.
	ulCmd	UINT32	0x1F32	PNS_IF_READ_IM_IND - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons

Structure PNS_IF_READ_IM_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_READ_IM_IND_DATA_T			
	ulApi	UINT32	0-0xFFFFFFFF	The API of the (sub)module to read the I&M data for. Ignore on I&M0 Filter data read.
	usSlot	UINT16	0-0xFFFF	The Slot of the (sub)module to read the I&M data for. Ignore on I&M0 Filter data read.
	usSubslot	UINT16	0-0xFFFF	The Subslot of the (sub)module to read the I&M data for. Ignore on I&M0 Filter data read.
	bIMType	UINT8	0-4, 255	The I&M record to read. (0-4: I&M0-4, 255: I&M0 Filter Data)
	abReserved[3]	UINT8		Reserved for future usage / Padding

Table 107: PNS\_IF\_READ\_IM\_IND\_T – Read I&amp;M Indication

### 7.3.14.2 Read I&M Response

The application shall respond to each Read I&M Indication using the Read I&M Response. The response shall contain the requested I&M data.

#### Packet Structure Reference

```
typedef struct PNS_IF_READ_IM_RES_DATA_Ttag
{
    TLR_UINT32    ulApi;
    TLR_UINT16    usSlot;
    TLR_UINT16    usSubslot;
    TLR_UINT8     bIMType;
    TLR_UINT8     abReserved[3];
    union {
        /** Array of submodules describing I&M state of submodules
         * ( grow Array as required)*/
        PNS_IF_IM0_FILTER_DATA_T  atIM0FilterData[1];
        PNS_IF_IM0_DATA_T         tIM0;
        PNS_IF_IM1_DATA_T         tIM1;
        PNS_IF_IM2_DATA_T         tIM2;
        PNS_IF_IM3_DATA_T         tIM3;
        PNS_IF_IM4_DATA_T         tIM4;
    } tData;
} PNS_IF_READ_IM_RES_DATA_T;

typedef struct PNS_IF_READ_IM_RES_Ttag
{
    TLR_PACKET_HEADER_T    tHead;
    PNS_IF_READ_IM_RES_DATA_T  tData;
} PNS_IF_READ_IM_RES_T;
```

## Packet Description

Structure PNS_IF_READ_IM_RES_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle, do not touch
	ulSrc	UINT32		Source Queue-Handle, do not touch
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, do not touch.
	ulLen	UINT32	12 + n	Packet data length in bytes. N depends on the returned data
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification, untouched
	ulSta	UINT32		Error code. Either TLR_S_OK or TLR_E_FAIL
	ulCmd	UINT32	0x1F33	PNS_IF_READ_IM_RES - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_READ_IM_IND_DATA_T			
	ulApi	UINT32	0-0xFFFFFFFF	Same value as in indication
	usSlot	UINT16	0-0xFFFF	Same value as in indication
	usSubslot	UINT16	0-0xFFFF	Same value as in indication
	bIMType	UINT8	0-4, 255	Same value as in indication
	abReserved[3]	UINT8		Set to zero
	tData	UNION		UNION of different structures. To be filled with the requested I&M information according bIMType and the related information structure. (See below)

Table 108: PNS\_IF\_READ\_IM\_RES\_T – Read I&M Response

Depending on the value of the field `tData.bIMType` the correct substructure of the union has to be filled by the user application. In case of I&M 0-4 the substructures `tIM0-tIM4` have to be used, in case of I&M0 Filter Data the array `atIM0FilterData` has to be filled with all (sub)modules which have discrete I&M data.

```
typedef enum
{
    PNS_IF_IM_TYPE_IM0      = 0,
    PNS_IF_IM_TYPE_IM1      = 1,
    PNS_IF_IM_TYPE_IM2      = 2,
    PNS_IF_IM_TYPE_IM3      = 3,
    PNS_IF_IM_TYPE_IM4      = 4,
    PNS_IF_IM_TYPE_IMOFILTER = 255,
} PNS_IF_IM_TYPE_E;
```

According to requested I&M Type, the following structures shall be used:

```
typedef struct PNS_IF_IM0_DATA_Ttag
{
    TLR_UINT8  abManufacturerSpecific[10];
    TLR_UINT16 usManufacturerId;
    TLR_UINT8  abOrderId[20];
    TLR_UINT8  abSerialNumber[16];
```

```

TLR_UINT16  usHardwareRevision;
struct {
    TLR_UINT8 bPrefix;
    TLR_UINT8 bX;
    TLR_UINT8 bY;
    TLR_UINT8 bZ;
} tSoftwareRevision;
TLR_UINT16  usRevisionCounter;
TLR_UINT16  usProfileId;
TLR_UINT16  usProfileSpecificType;
TLR_UINT16  usIMVersion;
TLR_UINT16  usIMSupported;
} PNS_IF_IM0_DATA_T;

```

structure PNS_IF_IM0_DATA_T				
Area	Variable	Type	Value / Range	Description
	abManufacturerSpecific	UINT8[10]	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
	usManufacturerId	UINT16		The Vendor ID of the device. Usually similar to the specified in Set Configuration Request
	abOrderId	UINT8[20]		The Order ID of the (sub)module. (padded with spaces (0x20))
	abSerialNumber	UINT8[16]		The Serial number of the (sub)module. (padded with spaces)
	usHardwareRevision	UINT16		Hardware revision of the (sub)module
	tSoftwareRevision.bPrefix	UINT8		Character describing the software of the (sub)module. Allowed values: 'V', 'R', 'P', 'U' and 'T'
	tSoftwareRevision.bX	UINT8		Function enhancement. (Major version number) of the (sub)module.
	tSoftwareRevision.bY	UINT8		Bug fix (Minor version number) of the (sub)module
	tSoftwareRevision.bZ	UINT8		Internal Change (Build version number) of the (sub)module
	usRevisionCounter	UINT16		Starting from 0, shall increment on each parameter change.
	usProfileId	UINT16		The profile of the (sub)module.
	usProfileSpecificType	UINT16		Additional value depending on profile of the (sub)module
	usIMVersion	UINT16		The I&M version. (Default value 0x0101)
	usIMSupported	UINT16		Bit list describing the I&M variants supported by the (sub)module: <ul style="list-style-type: none"> <li>0x02 -&gt; I&amp;M1 Supported</li> <li>0x04 -&gt; I&amp;M2 Supported</li> <li>0x08 -&gt; I&amp;M3 Supported</li> <li>0x10 -&gt; I&amp;M4 Supported</li> </ul>

Table 109: PNS\_IF\_IM0\_DATA\_T – Structure of I&M0 Information



```
typedef struct PNS_IF_IM1_DATA_Ttag
{
    TLR_UINT8    abManufacturerSpecific[10];
    TLR_UINT8    abTagFunction[32];
    TLR_UINT8    abTagLocation[22];
}PNS_IF_IM1_DATA_T;
```

structure PNS_IF_IM1_DATA_T				
Area	Variable	Type	Value / Range	Description
	abManufacturerSpecific[10]	UINT8	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
	abTagFunction[32]	UINT8		Function Tag of the (sub)module. (padded with spaces)
	abTagLocation[22]	UINT8		Location Tag of the (sub)module. (padded with spaces)

Table 110: PNS\_IF\_IM1\_DATA\_T – Structure of I&M1 Information

```
typedef struct PNS_IF_IM2_DATA_Ttag
{
    TLR_UINT8    abManufacturerSpecific[10];
    TLR_UINT8    abInstallationDate[16];
    TLR_UINT8    abReserved[38];
}PNS_IF_IM2_DATA_T;
```

structure PNS_IF_IM2_DATA_T				
Area	Variable	Type	Value / Range	Description
	abManufacturerSpecific[10]	UINT8	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
	abInstallationDate[16]	UINT8		Installation Date of the (sub)module. (padded with spaces)
	abReserved[38]	UINT8		Reserved. Set to zero. Not evaluated by stack.

Table 111: PNS\_IF\_IM2\_DATA\_T – Structure of I&M2 Information

```
typedef struct PNS_IF_IM3_DATA_Ttag
{
    TLR_UINT8    abManufacturerSpecific[10];
    TLR_UINT8    abDescriptor[54];
}PNS_IF_IM3_DATA_T;
```

structure PNS_IF_IM3_DATA_T				
Area	Variable	Type	Value / Range	Description
	abManufacturerSpecific[10]	UINT8	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
	abDescriptor[54]	UINT8		Description text of the (sub)module. (padded with spaces)

Table 112: PNS\_IF\_IM3\_DATA\_T – Structure of I&M3 Information

```
typedef struct PNS_IF_IM4_DATA_Ttag
{
    TLR_UINT8    abManufacturerSpecific[10];
    TLR_UINT8    abSignature[54];
}PNS_IF_IM4_DATA_T;
```

structure PNS_IF_IM4_DATA_T				
Area	Variable	Type	Value / Range	Description
	abManufacturerSpecific[10]	UINT8	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
	abSignature[54]	UINT8		Signature generated by engineering system. (padded with spaces, default value: <b>ZERO</b> )

Table 113: PNS\_IF\_IM4\_DATA\_T – Structure of I&M4 Information

I&M0FilterData is a special record identifying which submodules of an IO-Device carry distinct I&M data. I&M0FilterData may also deliver a submodule acting as device representative which means that this submodule's I&M data is valid for the whole IO-Device.

The I&M0FilterData response of application may contain one or more submodules in PNS\_IF\_READ\_IM\_RES\_DATA\_T element atIM0FilterData. The confirmation packet lengths need to be set according to the amount of submodules contained in the packet.

```
typedef enum
{
    PNS_IF_IM0_FILTER_DATA_HAS_IM_DATA = 0x00000001,
    PNS_IF_IM0_FILTER_DATA_MODULE_REF  = 0x00000002,
    PNS_IF_IM0_FILTER_DATA_DEVICE_REF  = 0x00000004,
} PNS_IF_IM0_FILTER_DATA_FLAGS_E;

typedef struct PNS_IF_IM0_FILTER_DATA_Ttag
{
    TLR_UINT32    ulApi;
    TLR_UINT16    usSlot;
    TLR_UINT16    usSubslot;
    TLR_UINT32    ulFlags;
} PNS_IF_IM0_FILTER_DATA_T;
```

structure PNS_IF_IM0_FILTER_DATA_T				
Area	Variable	Type	Value / Range	Description
	ulApi	UINT32		The API of the submodule.
	usSlot	UINT16		The slot of the submodule
	usSubslot	UINT16		The subslot of the submodule
	ulFlags	UINT32		The properties of the submodule. If the submodule has I&M data the flag PNS_IF_IM0_FILTER_DATA_HAS_IM_DATA shall be set. Otherwise the submodule will be ignored. (=The entry can be omitted from the PNS_IF_READ_IM_RES_T) The Flag PNS_IF_IM0_FILTER_DATA_MODULE_REF shall be set if the submodule I&M data represents the module I&M data. (Hint: Use this for physical modules with virtual submodules) PNS_IF_IM0_FILTER_DATA_DEVICE_REF shall be set if the submodule I&M data represents the device I&M data. (Hint: Use this for devices which have no pluggable modules)

Table 114: PNS\_IF\_IM0\_FILTER\_DATA\_T – Structure of I&M0 Filter Information

### 7.3.15 Write I&M Service

The stack uses this service to force the user application to write the given I&M information into the corresponding (sub)module. The I&M information should be stored into the physical (sub)module.



**Note:**

Writing of I&M is only defined for I&M1-4 records.

Writing of I&M1, I&M2 and I&M3 records shall be supported at least for one submodule by each PROFINET IO Device.

Write support for I&M4 record is optional. If write is not supported, the user application shall respond the write indication with error code TLR\_E\_FAIL.

#### 7.3.15.1 Write I&M Indication

This indication is sent by the stack whenever a controller or a supervisor writes I&M information in order to store the given I&M data into the (sub)module.

#### Packet Structure Reference

```
typedef struct PNS_IF_WRITE_IM_IND_DATA_Ttag
{
    TLR_UINT32    ulApi;
    TLR_UINT16    usSlot;
    TLR_UINT16    usSubslot;
    TLR_UINT8     bIMType;
    TLR_UINT8     abReserved[3];
    union {
        PNS_IF_IM1_DATA_T      tIM1;
        PNS_IF_IM2_DATA_T      tIM2;
        PNS_IF_IM3_DATA_T      tIM3;
        PNS_IF_IM4_DATA_T      tIM4;
    } tData;
} PNS_IF_WRITE_IM_IND_DATA_T;

typedef struct PNS_IF_WRITE_IM_IND_Ttag
{
    TLR_PACKET_HEADER_T      tHead;
    PNS_IF_WRITE_IM_IND_DATA_T tData;
} PNS_IF_WRITE_IM_IND_T;
```

#### Packet Description

Structure PNS_IF_WRITE_IM_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of application task process queue
	ulSrc	UINT32		Source Queue-Handle of PNSIF task process queue
	ulDestId	UINT32	0	Destination End Point Identifier
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	76	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		Status not used for request.

Structure PNS_IF_WRITE_IM_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
	ulCmd	UINT32	0x1F34	PNS_IF_WRITE_IM_IND - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_WRITE_IM_IND_DATA_T			
	ulApi	UINT32	0-0xFFFFFFFF	The API of the (sub)module to write the I&M data for.
	usSlot	UINT16	0-0xFFFF	The Slot of the (sub)module to read the I&M data for.
	usSubslot	UINT16	0-0xFFFF	The Subslot of the (sub)module to read the I&M data for.
	bIMType	UINT8	1-4	The I&M record to read. (1-4: I&M1-4)
	abReserved[3]	UINT8		Reserved for future usage / Padding
	tData	UNION		UNION of different structures. Contains the I&M Data to write. Select the substructure according bIMType field. For description of the substructures see section 6.3.14.2.

Table 115: PNS\_IF\_WRITE\_IM\_IND\_T – Write I&amp;M Indication

Depending on the value of the field `tData.bIMType` the correct substructure of the union has to be taken for evaluation at the user application. The I&M data shall be stored in non volatile memory of the (sub)module.

**Note:**

It is recommended to reset I&M1-4 values to defaults on `PNS_IF_RESET_FACTORY_SETTINGS_IND` packet. Except for I&M4 signature the default value is a space filled string. The default signature is a zero filled string.

### 7.3.15.2 Write I&M Response

The application shall respond to each Write I&M Indication using the Write I&M Response. The `ulSta` field of the response header shall be set according success or failure of the write.

#### Packet Structure Reference

```
typedef struct PNS_IF_WRITE_IM_RES_DATA_Ttag
{
    TLR_UINT32    ulApi;
    TLR_UINT16    usSlot;
    TLR_UINT16    usSubslot;
    TLR_UINT8     bIMType;
    TLR_UINT8     abReserved[3];
} PNS_IF_WRITE_IM_RES_DATA_T;

typedef struct PNS_IF_WRITE_IM_RES_Ttag
{
    TLR_PACKET_HEADER_T    tHead;
    PNS_IF_WRITE_IM_RES_DATA_T    tData;
} PNS_IF_WRITE_IM_RES_T;
```

## Packet Description

Structure PNS_IF_WRITE_IM_RES_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle, do not touch
	ulSrc	UINT32		Source Queue-Handle, do not touch
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, do not touch.
	ulLen	UINT32	12	Packet data length in bytes.
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification, untouched
	ulSta	UINT32		Error code. Either TLR_S_OK or TLR_E_FAIL
	ulCmd	UINT32	0x1F35	PNS_IF_WRITE_IM_RES - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_WRITE_IM_IND_DATA_T			
	ulApi	UINT32	0-0xFFFFFFFF	Same value as in indication
	usSlot	UINT16	0-0xFFFF	Same value as in indication
	usSubslot	UINT16	0-0xFFFF	Same value as in indication
	bIMType	UINT8	1-4	Same value as in indication
	abReserved[3]	UINT8		Set to zero

Table 116: PNS\_IF\_WRITE\_IM\_RES\_T – Write I&M Response

### 7.3.16 Parameterization Speedup Support

This service is used to indicate to the user application whether Parameterization Speedup is enabled. This is the case if device shall use fast startup mode (FSU). The FSU mode requires that the application stores all user record data and the associated parameter uuid (this service) into non-volatile memory. On next power up, the application shall restore the user record data from non volatile memory into the submodules before any AR is established. Afterwards, when the AR is being established, the application may compare the stored parameter uuid with the parameter uuid received during last connection establishment to detect if updating the parameters is required. This service allows that the application can parameterize the submodules very early and decreases the time until the device becomes ready for data exchange. If the parameter uuid is zero, the FSU mode is disabled. In this case no parameters should be restored from non-volatile memory.



#### Note:

If the application receives a Parameterization speedup support indication containing the NIL-UUID during connection establishment, the module parameterization shall be done in the regular way without speedup. The same applies if the received UUID is different to the configured one. In this case it the PLC has a different configuration.



#### Note:

If the application does not use any user records, no special action is required on this indication.

#### 7.3.16.1 Parameterization Speedup Support Indication

##### Packet Structure Reference

```
struct FSUUID_Ttag /* UUID data */
{
    TLR_UINT32      ulData1;
    TLR_UINT16      usData2;
    TLR_UINT16      usData3;
    TLR_UINT8       abData4[8];
} ;

typedef struct PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_DATA_Ttag
{
    FSUUID_T tFSUuid;
} PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_DATA_T;

typedef struct PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_Ttag
{
    TLR_PACKET_HEADER_T tHead;
    PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_DATA_T tData;
} PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_T;

typedef TLR_EMPTY_PACKET_T PNS_IF_PARAMET_SPEEDUP_SUPPORTED_RES_T;
```

## Packet Description

Structure <code>PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_T</code>				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure <code>TLR_PACKET_HEADER_T</code>			
	<code>ulDest</code>	UINT32		Destination queue handle of application task process queue
	<code>ulSrc</code>	UINT32		Source Queue-Handle of PNSIF task process queue
	<code>ulDestId</code>	UINT32	0	Destination End Point Identifier
	<code>ulSrcId</code>	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	<code>ulLen</code>	UINT32	16	Packet data length in bytes.
	<code>ulId</code>	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	<code>ulSta</code>	UINT32		Status not used for request.
	<code>ulCmd</code>	UINT32	0x1FF8	<code>PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND</code> - Command
	<code>ulExt</code>	UINT32	0	Extension not in use, set to zero for compatibility reasons
	<code>ulRout</code>	UINT32	x	Routing, do not touch
Data	structure <code>PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_DATA_T</code>			
	<code>tFSUuid</code>	FSUUID_T	0-0xFFFFFFFF	The UUID send to the application. TLR_UINT32 <code>ulData1</code> TLR_UINT16 <code>usData2</code> TLR_UINT16 <code>usData3</code> TLR_UINT8 <code>abData4[8]</code>

Table 117: `PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_T` – Parameterization Speedup Supported Indication

### 7.3.16.2 Parameterization Speedup Supported Response

The application shall respond to each Speedup Supported Indication using the Speedup Supported Response. The `ulSta` field of the response header shall be set according to success or failure of the write access.

## Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T    PNS_IF_PARAMET_SPEEDUP_SUPPORTED_RES_T;
```

## Packet Description

Structure PNS_IF_PARAMET_SPEEDUP_SUPPORTED_RES_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1FF9	PNS_IF_PARAMET_SPEEDUP_SUPPORTED_RSP - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch

Table 118: PNS\_IF\_PARAMET\_SPEEDUP\_SUPPORTED\_RES\_T- Parameterization Speedup Supported Response



## 7.3.17 Event Indication Service

Using the Event Indication Service the stack informs the user application about IO data related events.

This service is only available if the Dual Port Memory Interface is used. It corresponds to the Event Handler Callback service described in section 5.2.2.3 (see more details there as well).

### 7.3.17.1 Event Indication

The following indication packet is sent by the stack.

**Note:**

If an event indication is pending at the host application (no event response returned back to firmware yet) new events are not reported using another event indication but counted. These unreported events will be reported after the event response has been sent to the firmware. This prevents flooding the host application with events.

#### Packet Structure Reference

```
typedef enum PNS_IF_IO_EVENT_Etag
{
    PNS_IF_IO_EVENT_RESERVED                = 0x00000000,
    PNS_IF_IO_EVENT_NEW_FRAME                = 0x00000001,
    PNS_IF_IO_EVENT_CONSUMER_UPDATE_REQUIRED = 0x00000002,
    PNS_IF_IO_EVENT_PROVIDER_UPDATE_REQUIRED = 0x00000003,
    PNS_IF_IO_EVENT_FRAME_SENT               = 0x00000004,
    PNS_IF_IO_EVENT_CONSUMER_UPDATE_DONE     = 0x00000005,
    PNS_IF_IO_EVENT_PROVIDER_UPDATE_DONE     = 0x00000006,
    PNS_IF_IO_EVENT_MAX,                    /**< Number of defined events */
} PNS_IF_IO_EVENT_E;

typedef struct
{
    TLR_UINT16  ausEventCnt[PNS_IF_IO_EVENT_MAX];
} PNS_IF_EVENT_IND_DATA_T;

typedef struct
{
    TLR_PACKET_HEADER_T      tHead;
    PNS_IF_EVENT_IND_DATA_T  tData;
} PNS_IF_EVENT_IND_T;
```

#### Packet Description

Structure PNS_IF_EVENT_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of application task process queue
	ulSrc	UINT32		Source Queue-Handle of PNSIF task process queue
	ulDestId	UINT32	0	Destination End Point Identifier
	ulSrcId	UINT32	0 ... $2^{32}-1$	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		Status not used for request.

Structure PNS_IF_EVENT_IND_T				Type: Indication
Area	Variable	Type	Value / Range	Description
	ulCmd	UINT32	0x1FFE	PNS_IF_EVENT_IND- Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	X	Routing, do not touch
Data	structure PNS_IF_EVENT_IND_DATA_T			
	ausEventCnt	UINT16[7]		Array of event counters. For each event the amount if occurrence is counted since the last event indication.

Figure 21. Event Indication

### 7.3.17.2 Event Indication Response

The application shall return this packet as response to an Event Indication.

#### Packet Structure Reference

```
typedef struct
{
    TLR_PACKET_HEADER_T          tHead;
} PNS_IF_EVENT_RSP_T;
```

#### Packet Description

Structure PNS_IF_EVENT_RSP_T				Type: Indication
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of application task process queue
	ulSrc	UINT32		Source Queue-Handle of PNSIF task process queue
	ulDestId	UINT32	0	Destination End Point Identifier
	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		The Status.
	ulCmd	UINT32	0x1FFF	PNS_IF_EVENT_RSP- Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	X	Routing, do not touch

Figure 22. Event Indication Response

## 7.4 Acyclic Events requested by the Application

This section describes all acyclic events and requests that the application can send to the stack.

The application can only use alarm services (see sections 6.4.4 and 6.4.5), if cyclic data exchange is established. The application may use all other services at any time after the stack has been configured.

Table 119 lists all packets of acyclic events that the application can request from the stack.

Packet overview of acyclic events of the PROFINET IO Device stack requested by the application			
No. of section	Packet	Command code	Page
6.4.1	Get Diagnosis Request	0x1FB2	189
	Get Diagnosis Confirmation	0x1FB3	190
6.4.2	Get XMAC (EDD) Diagnosis Request	0x1FE4	194
	Get XMAC (EDD) Diagnosis Confirmation	0x1FE5	194
6.4.3	Process Alarm Request	0x1F52	197
	Process Alarm Confirmation	0x1F53	199
6.4.4	Diagnosis Alarm Request	0x1F4C	200
	Diagnosis Alarm Confirmation	0x1F4D	201
6.4.5	Return of Submodule Alarm Request	0x1F50	203
	Return of Submodule Alarm Confirmation	0x1F51	204
6.4.6	AR Abort Request Service	0x1FD8	206
	AR Abort Request Confirmation	0x1FD9	207
6.4.7	Plug Module Request	0x1F04	208
	Plug Module Confirmation	0x1F05	210
6.4.8	Extended Plug Submodule Request	0x1F08	217
	Extended Plug Submodule Confirmation	0x1F09	219
6.4.9	Pull Module Request	0x1F06	221
	Pull Module Confirmation	0x1F07	222
6.4.10	Pull Submodule Request	0x1F0A	224
	Pull Submodule Confirmation	0x1F0B	225
6.4.11	Get Station Name Request	0x1F8E	227
	Get Station Name Confirmation	0x1F8F	228
6.4.12	Get IP Address Request	0x1FBC	229
	Get IP Address Confirmation	0x1FBD	230
6.4.12.3	Add Channel Diagnosis Request	0x1F46	231
	Add Channel Diagnosis Confirmation	0x1F47	235
6.4.13	Add Extended Channel Diagnosis Request	0x1F54	237
	Add Extended Channel Diagnosis Confirmation	0x1F55	239
6.4.14	Add Generic Channel Diagnosis Request	0x1F58	241
	Add Generic Channel Diagnosis Confirmation	0x1F59	242
6.4.15	Remove Diagnosis Request	0x1FE6	244

Packet overview of acyclic events of the PROFINET IO Device stack requested by the application			
No. of section	Packet	Command code	Page
	Remove Diagnosis Confirmation	0x1FE7	245

Table 119: Packet overview of acyclic events of the PROFINET IO Device stack requested by the application

## 7.4.1 Get Diagnosis Service

With this service the user application can request a collection of diagnostic data concerning the stack.



### Note:

In this service the confirmation packet is larger than the request packet. If the application is programming the stack's AP-Task Queue directly so the application has to provide a buffer which is large enough to hold the confirmation data

### 7.4.1.1 Get Diagnosis Request

#### Packet Structure Reference

```
/* Get Diagnosis Request packet */
typedef TLR_EMPTY_PACKET_T PNS_IF_GET_DIAGNOSIS_REQ_T;
```

#### Packet Description

Structure PNS_IF_GET_DIAGNOSIS_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	0	PNS_IF_GET_DIAGNOSIS_REQ_SIZE - Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not used for requests. Set to zero.
	ulCmd	UINT32	0x1FB2	PNS_IF_GET_DIAGNOSIS_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 120: PNS\_IF\_GET\_DIAGNOSIS\_REQ\_T - Get Diagnosis Request

### 7.4.1.2 Get Diagnosis Confirmation

With this packet the stack provides the collected diagnosis data to the application.

#### Packet Structure Reference

```
/* Get Diagnosis Confirmation packet */
typedef struct PNS_IF_STATUS_Ttag
{
    TLR_UINT32      ulPnsState;
    TLR_UINT32      ulLastRslt;
    TLR_UINT32      ulLinkState;
    TLR_UINT32      ulConfigState;
    TLR_UINT32      ulCommunicationState;
    TLR_UINT32      ulCommunicationError;
    TLR_UINT32      aulLineDelay[2];
} PNS_IF_STATUS_T;

typedef struct PNS_IF_GET_DIAGNOSIS_CNF_Ttag
{
    TLR_PACKET_HEADER_T      tHead;
    PNS_IF_STATUS_T          tData;
} PNS_IF_GET_DIAGNOSIS_CNF_T;
```

#### Packet Description

Structure <b>PNS_IF_GET_DIAGNOSIS_CNF_T</b>				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1FB3	PNS_IF_GET_DIAGNOSIS_CNF - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_STATUS_T			
	ulPnsState	UINT32	Bit mask	State of Protocol Stack. See below.
	ulLastRslt	UINT32	Error code	Last Result
	ulLinkState	UINT32	0-3	Link State. See below.
	ulConfigState	UINT32	0-8	Configuration State. See below.
	ulCommunicationState	UINT32	0-4	Communication State. See below.
	ulCommunicationError	UINT32	Error code	Communication Error. See below.

Structure PNS_IF_GET_DIAGNOSIS_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
	aulLineDelay[2]	UINT32		Structure containing line delay for port 0 and port 1. aulLineDelay [0] stores Line delay for Port 0, aulLineDelay [1] stores Line delay for Port 1.

Table 121: PNS\_IF\_GET\_DIAGNOSIS\_CNF\_T - Get Diagnosis Confirmation

This function will request a diagnostic data block (the status block). The requested data will be delivered by the confirmation message.

The parameters delivered by the confirmation message can have the following values denoting the associated meanings:

■ ulPnsState

This parameter represents the PROFINET IO Device task state. It can have one of the following values:

Bit	Description
D16	Fiber Optic Maintenance Required Record exists for Port 1 <b>Note:</b> Only valid in case of fiber optic hardware.
D15	Fiber Optic Maintenance Demanded Record exists for Port 1 <b>Note:</b> Only valid in case of fiber optic hardware.
D14	Fiber Optic Maintenance Required Record exists for Port 0 <b>Note:</b> Only valid in case of fiber optic hardware.
D13	Fiber Optic Maintenance Demanded Record exists for Port 0 <b>Note:</b> Only valid in case of fiber optic hardware.
D12	A PROFINET Maintenance Demanded Record exists
D11	A PROFINET Maintenance Required Record exists
D10	A PROFINET Diagnosis Record exists
D9	Fatal Error occurred
D8	Configuration is locked
D7	Network Communication is enabled
D6	Network Communication is allowed
D5	Module 0 and Submodule 1 are plugged
D4	Module 0 is plugged
D3	At least one API is present
D2	Reserved
D1	PROFINET Stack is started
D0	Device Information is set

Table 122: Meaning of single Bits in ulPnsState

#### ■ eLastRslt

This parameter denotes the error code of the last encountered error of the RCX/API Task, see the TLR error codes documented in section 8 Status/Error Codes Overview.

#### ■ ulLinkState

This parameter denotes the link state. The following values are supported:

Value	Meaning
0	No information available
1	Physical link works correctly
2	Low speed of physical link
3	No physical link present

Table 123: Values and their corresponding Meanings of `ulLinkState`

#### ■ ulConfigState

This parameter denotes the configuration state. It may have the following values:

Value	Meaning
0	Not configured
1	Configured with DBM Files
2	Error during configuration with DBM Files
3	Configured by application
4	Configuration by application is running
5	Error during configuration by Application
6	Configured with Warmstart-Parameters
7	Configuration with Warmstart-Parameters is running
8	Error during Configuration with Warmstart-Parameters

Table 124: Values and their corresponding Meanings of `ulConfigState`

#### ■ ulCommunicationState

This parameter denotes the communication state. It 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`      0x0000
- OFFLINE      `#define RCX_COMM_STATE_OFFLINE`      0x0001
- STOP      `#define RCX_COMM_STATE_STOP`      0x0002
- IDLE      `#define RCX_COMM_STATE_IDLE`      0x0003
- OPERATE      `#define RCX_COMM_STATE_OPERATE`      0x0004



- ulCommunicationError

This parameter 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\_S\_OK) again. For possible error codes see section *Status/Error Codes Overview*.

- aulLineDelay

This parameter represents the propagation delay for the ports 0 and 1 in nanoseconds.

## 7.4.2 Get XMAC (EDD) Diagnosis Service

Using this service the application can request some statistic information from the integrated switch.



### Note:

The confirmation packet of this service is larger than the request packet. If the application is programming the stacks AP-Task Queue directly the application has to provide a buffer which is large enough to hold the confirmation data.

### 7.4.2.1 Get XMAC (EDD) Diagnosis Request

This request packet allows access to the statistical information of the switch.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_GET_XMAC_DIAGNOSIS_REQ_T;
```

#### Packet Description

Structure <code>PNS_IF_GET_XMAC_DIAGNOSIS_REQ_T</code>				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure <code>TLR_PACKET_HEADER_T</code>			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		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		Status not used for requests. Set to zero.
	ulCmd	UINT32	0x1FE4	<code>PNS_IF_GET_XMAC_DIAGNOSIS_REQ</code>
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 125: `PNS_IF_GET_XMAC_DIAGNOSIS_REQ_T` - Get XMAC (EDD) Diagnosis Request

### 7.4.2.2 Get XMAC (EDD) Diagnosis Confirmation

This packet confirms the Get XMAC (EDD) diagnosis Request and delivers the requested data within the structure `EDD_XMAC_COUNTERS_T`, see Table 123 below.

## Packet Structure Reference

```
typedef struct {
    TLR_UINT32 ulFramesTransmittedOk;
    TLR_UINT32 ulSingleCollisionFrames;
    TLR_UINT32 ulMultipleCollisionFrames;
    TLR_UINT32 ulLateCollisions;
    TLR_UINT32 ulLinkDownDuringTransmission;
    TLR_UINT32 ulUtxUnderflowDuringTransmission;
    TLR_UINT32 ulTxFatalErrors;
    TLR_UINT32 ulFramesReceivedOk;
    TLR_UINT32 ulFrameCheckSequenceErrors;
    TLR_UINT32 ulAlignmentErrors;
    TLR_UINT32 ulFrameTooLongErrors;
    TLR_UINT32 ulRuntFramesReceived;
    TLR_UINT32 ulCollisionFragmentsReceived;
    TLR_UINT32 ulFramesDroppedDueLowResource;
    TLR_UINT32 ulFramesDroppedDueUrxOverflow;
    TLR_UINT32 ulRxFatalErrors;
} EDD_XMAC_COUNTERS_T;

/* confirmation packet */
typedef struct PNS_IF_GET_XMAC_DIAGNOSIS_DATA_Ttag
{
    EDD_XMAC_COUNTERS_T tXMACCounters[2];
} PNS_IF_GET_XMAC_DIAGNOSIS_DATA_T;

typedef struct PNS_IF_GET_XMAC_DIAGNOSIS_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_GET_XMAC_DIAGNOSIS_DATA_T tData;
} PNS_IF_GET_XMAC_DIAGNOSIS_CNF_T;
```

## Packet Description

structure PNS_IF_GET_XMAC_DIAGNOSIS_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	128	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x00001FE5	PNS_IF_GET_XMAC_DIAGNOSIS_CNF - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_GET_XMAC_DIAGNOSIS_DATA_T			
	tXMACCounters[2]	EDD_XMAC_COUNTERS_T		Array containing statistics stored in XMAC counters

Table 126: PNS\_IF\_GET\_XMAC\_DIAGNOSIS\_CNF\_T - Get XMAC (EDD) Diagnosis Confirmation

The structure `EDD_XMAC_COUNTERS_T` contains the following counters collected by the statistical information of the 2-port switch:

<b>structure <code>EDD_XMAC_COUNTERS_T</code></b>			
<b>Variable</b>	<b>Type</b>	<b>Value / Range</b>	<b>Description</b>
<code>ulFramesTransmittedOk</code>	UINT32	0 ... $2^{32} - 1$	Count of frames that are successfully transmitted
<code>ulSingleCollisionFrames</code>	UINT32	0 ... $2^{32} - 1$	Count of frames that are involved into a single collision
<code>ulMultipleCollisionFrames</code>	UINT32	0 ... $2^{32} - 1$	Count of frames that are involved into more than one collisions
<code>ulLateCollisions</code>	UINT32	0 ... $2^{32} - 1$	Later than 512 bit times into the transmitted packet
<code>ulLinkDownDuringTransmission</code>	UINT32	0 ... $2^{32} - 1$	Count of the times that a frame was transmitted during link down
<code>ulUtxUnderflowDuringTransmission</code>	UINT32	0 ... $2^{32} - 1$	UTX FIFO underflow at transmission time
<code>ulTxFatalErrors</code>	UINT32	0 ... $2^{32} - 1$	Wrong TPU error code, should always be zero
<code>ulFramesReceivedOk</code>	UINT32	0 ... $2^{32} - 1$	Count of frames that have successfully been received
<code>ulFrameCheckSequenceErrors</code>	UINT32	0 ... $2^{32} - 1$	Count of frames that are an integral number of octets in length and do not pass the FCS check
<code>ulAlignmentErrors</code>	UINT32	0 ... $2^{32} - 1$	Count of frames that are not an integral number of octets in length and do not pass the FCS check
<code>ulFrameTooLongErrors</code>	UINT32	0 ... $2^{32} - 1$	Count of frames that are received and exceed the maximum permitted frame size
<code>ulRuntFramesReceived</code>	UINT32	0 ... $2^{32} - 1$	Count of frames that have a length between 42..63 bytes and a valid CRC
<code>ulCollisionFragmentsReceived</code>	UINT32	0 ... $2^{32} - 1$	Count of frames that are smaller than 64 bytes and have an invalid CRC
<code>ulFramesDroppedDueLowResource</code>	UINT32	0 ... $2^{32} - 1$	No empty pointer available at indication time
<code>ulFramesDroppedDueUrxOverflow</code>	UINT32	0 ... $2^{32} - 1$	URX FIFO overflow at indication time
<code>ulRxFatalErrors</code>	UINT32	0 ... $2^{32} - 1$	Wrong RPU error code, should always be zero

Table 127: Structure `EDD_XMAC_COUNTERS_T`

### 7.4.3 Process Alarm Service

With this service the application can request the stack to send a process alarm. Process alarms have the high PROFINET priority in this implementation.


**Note:**

If for some reason the IO-Controller does not react to the alarm the application will NOT get a confirmation from the stack as the stack itself is waiting for a reaction of the IO-Controller.


**Note:**

PROFINET only allows one outstanding alarm per priority per time. However the stack is implemented in the way that the user can request sending up to 8 alarms simultaneously. The stack will then queue the outstanding requests and handle them in the order they were reported.



Note: Since GSDML file version V2.32 it is required to use the new keyword “MayIssueProcessAlarm” to indicate that a submodule might generate a process alarm. This keyword defaults to “false” for default Hilscher GSDML files. If this service is used, the GSDML file must be adapted and the keyword must be set to “true” for all affected submodules.

#### 7.4.3.1 Process Alarm Request

This packet causes the stack to send a Process Alarm to the IO-Controller.

#### Packet Structure Reference

```
typedef struct PNS_IF_SEND_PROCESS_ALARM_REQ_DATA_Ttag
{
    TLR_UINT32    ulReserved;
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_UINT32    hAlarmHandle;
    TLR_UINT16    usUserStructId;
    TLR_UINT16    usAlarmDataLen;
    TLR_UINT8     abAlarmData[PNS_IF_MAX_ALARM_DATA_LEN];
} PNS_IF_SEND_PROCESS_ALARM_REQ_DATA_T;

typedef struct PNS_IF_SEND_PROCESS_ALARM_REQ_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_SEND_PROCESS_ALARM_REQ_DATA_T    tData;
} PNS_IF_SEND_PROCESS_ALARM_REQ_T;
```

## Packet Description

Structure PNS_IF_SEND_PROCESS_ALARM_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	24 +n	Packet data length in bytes. n is the value of usLenAlarmData.
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for request.
	ulCmd	UINT32	0x1F52	PNS_IF_SEND_PROCESS_ALARM_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_SEND_PROCESS_ALARM_REQ_DATA_T			
	ulReserved	UINT32		Reserved. Set to zero.
	ulApi	UINT32		The API the alarm belongs to.
	ulSlot	UINT32		The Slot the alarm belongs to.
	ulSubslot	UINT32		The Subslot the alarm belongs to.
	hAlarmHandle	UINT32		A user specific alarm handle. The application is free to choose any value.
	usUserStructId	UINT16		The User Structure Identifier.
	usAlarmDataLen	UINT16	0..1024	The length of the alarm data
	abAlarmData[1024]	UINT8[]		The alarm data.

Table 128: PNS\_IF\_SEND\_PROCESS\_ALARM\_REQ\_T - Process Alarm Request

### 7.4.3.2 Process Alarm Confirmation

This packet is returned to the application by the stack after the controller confirmed the alarm. The reaction of the IO-Controller is reported to the application within the confirmation.

#### Packet Structure Reference

```
typedef struct PNS_IF_SEND_PROCESS_ALARM_CNF_DATA_Ttag
{
    TLR_UINT32    hDeviceHandle;
    TLR_UINT32    hAlarmHandle;
    /* PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2 */
    TLR_UINT32    ulPnio;
} PNS_IF_SEND_PROCESS_ALARM_CNF_DATA_T;

typedef struct PNS_IF_SEND_PROCESS_ALARM_CNF_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T    tHead;
    /** packet data */
    PNS_IF_SEND_PROCESS_ALARM_CNF_DATA_T    tData;
} PNS_IF_SEND_PROCESS_ALARM_CNF_T;
```

#### Packet Description

Structure PNS_IF_SEND_PROCESS_ALARM_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	12	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1F53	PNS_IF_SEND_PROCESS_ALARM_CNF-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_SEND_PROCESS_ALARM_CNF_DATA_T			
	hDeviceHandle	UINT32		The device handle
	hAlarmHandle	UINT32		The user specific alarm handle.
	ulPnio	UINT32		PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See section "PROFINET Status Code".

Table 129: PNS\_IF\_SEND\_PROCESS\_ALARM\_CNF\_T - Process Alarm Confirmation

## 7.4.4 Diagnosis Alarm Service

With this service the application can request the stack to send a diagnosis alarm to the IO-Controller. The application has to use an “add diagnosis” service first (see sections 6.4.12.3, 6.4.13, 6.4.14). The diagnosis handle the stack returned back to the application using those services is needed here to send the alarm.

Diagnosis alarms have the low PROFINET priority in this implementation.



### Note:

If for some reason the IO-Controller does not react to the alarm the application will NOT get a confirmation from the stack as the stack itself is waiting for a reaction of the IO-Controller.



### Note:

PROFINET only allows one outstanding alarm per AR per priority at the same time. However the stack is implemented in the way that diagnostic alarms will be handled with a state per alarm and therefore queue less. The application may issue one diagnostic alarm per diagnosis entry at the same time.

### 7.4.4.1 Diagnosis Alarm Request

This request packet must be used by the application to force the stack to send a diagnosis alarm.

#### Packet Structure Reference

```
typedef struct PNS_IF_SEND_DIAG_ALARM_REQ_DATA_Ttag
{
    TLR_UINT32      ulReserved;
    TLR_UINT32      hAlarmHandle;
    TLR_UINT32      hDiagHandle;
} PNS_IF_SEND_DIAG_ALARM_REQ_DATA_T;

typedef struct PNS_IF_SEND_DIAG_ALARM_REQ_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_SEND_DIAG_ALARM_REQ_DATA_T    tData;
} PNS_IF_SEND_DIAG_ALARM_REQ_T;
```

#### Packet Description

Structure PNS_IF_SEND_DIAG_ALARM_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	12	Packet data length in bytes.
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for request.



Structure PNS_IF_SEND_DIAG_ALARM_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
	ulCmd	UINT32	0x1F4C	PNS_IF_SEND_DIAG_ALARM_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_SEND_DIAG_ALARM_REQ_DATA_T			
	ulReserved	UINT32		Reserved. Set to zero.
	hAlarmHandle	UINT32		A user specific alarm handle. The application is free to choose any value.
	hDiagHandle	UINT32		The handle to the diagnosis record the diagnosis alarm shall be sent for.

Table 130: PNS\_IF\_SEND\_DIAG\_ALARM\_REQ\_T - Diagnosis Alarm Request

#### 7.4.4.2 Diagnosis Alarm Confirmation

This packet is returned to the application by the stack. The reaction of the IO-Controller is reported to the application with this service.

If for some reason the IO-Controller does not respond to the Alarm the IO-Device application will **not** receive this confirmation packet.

#### Packet Structure Reference

```
typedef struct PNS_IF_SEND_DIAG_ALARM_CNF_DATA_Ttag
{
    TLR_UINT32    hDeviceHandle;
    TLR_UINT32    hAlarmHandle;
    /* PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2 */
    TLR_UINT32    ulPnio;
} PNS_IF_SEND_DIAG_ALARM_CNF_DATA_T;

typedef struct PNS_IF_SEND_DIAG_ALARM_CNF_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T            tHead;
    /** packet data */
    PNS_IF_SEND_DIAG_ALARM_CNF_DATA_T    tData;
} PNS_IF_SEND_DIAG_ALARM_CNF_T;
```

## Packet Description

Structure PNS_IF_DIAG_ALARM_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	12	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1F4D	PNS_IF_SEND_DIAG_ALARM_CNF
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_SEND_DIAG_ALARM_CNF_DATA_T			
	hDeviceHandle	UINT32		The device handle.
	hAlarmHandle	UINT32		The user specific alarm handle.
	ulPnio	UINT32		PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See section "PROFINET Status Code".

Table 131: PNS\_IF\_DIAG\_ALARM\_CNF\_T - Diagnosis Alarm Confirmation

## 7.4.5 Return of Submodule Alarm Service

The application has to use this service whenever it changes a provider IOPS of a submodule from BAD to GOOD. This service causes the stack to send a Return of Submodule alarm to the IO-Controller which indicates that submodule provides valid data again.

Return of Submodule alarms use the low PROFINET priority in this implementation.



### Note:

If for some reason the IO-Controller does not react to the alarm the application will NOT get a confirmation from the stack as the stack itself is waiting for a reaction of the IO-Controller.



### Note:

PROFINET only allows one outstanding alarm per priority per time. However the stack is implemented in the way that the user can request sending up to 16 alarms alarm simultaneously. The stack will then queue the outstanding requests and handle them in the order they were reported.

### 7.4.5.1 Return of Submodule Alarm Request

This packet has to be sent to the stack to cause a Return of Submodule alarm.

#### Packet Structure Reference

```
typedef struct PNS_IF_RETURN_OF_SUB_ALARM_REQ_DATA_Ttag
{
    TLR_UINT32    ulReserved;
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_UINT32    hAlarmHandle;
} PNS_IF_RETURN_OF_SUB_ALARM_REQ_DATA_T;

typedef struct PNS_IF_RETURN_OF_SUB_ALARM_REQ_Ttag
{
    TLR_PACKET_HEADER_T    tHead;
    PNS_IF_RETURN_OF_SUB_ALARM_REQ_DATA_T    tData;
} PNS_IF_RETURN_OF_SUB_ALARM_REQ_T;
```

#### Packet Description

Structure PNS_IF_RETURN_OF_SUB_ALARM_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	20	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for request.

Structure PNS_IF_RETURN_OF_SUB_ALARM_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
	ulCmd	UINT32	0x1F50	PNS_IF_RETURN_OF_SUB_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_RETURN_OF_SUB_ALARM_REQ_DATA_T			
	ulReserved	UINT32		Reserved. Set to zero.
	ulApi	UINT32		The API of the submodule.
	ulSlot	UINT32		The Slot of the Submodule.
	ulSubslot	UINT32		The Subslot of the submodule.
	hAlarmHandle	HANDLE		A user specific alarm handle. The application is free to choose any value.

Table 132: PNS\_IF\_RETURN\_OF\_SUB\_ALARM\_REQ\_T - Return of Submodule Alarm Request

### 7.4.5.2 Return of Submodule Alarm Confirmation

This packet will be returned by the stack.

#### Packet Structure Reference

```
typedef struct PNS_IF_RETURN_OF_SUB_ALARM_CNF_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
    TLR_UINT32 hAlarmHandle;
    /* PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2 */
    TLR_UINT32 ulPnio;
} PNS_IF_RETURN_OF_SUB_ALARM_CNF_DATA_T;

typedef struct PNS_IF_RETURN_OF_SUB_ALARM_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_RETURN_OF_SUB_ALARM_CNF_DATA_T    tData;
} PNS_IF_RETURN_OF_SUB_ALARM_CNF_T;
```

## Packet Description

Structure PNS_IF_RETURN_OF_SUB_ALARM_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	12	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1F51	PNS_IF_RETURN_OF_SUB_CNF-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_RETURN_OF_SUB_ALARM_CNF_DATA_T			
	hDeviceHandle	UINT32		The device handle
	hAlarmHandle	UINT32		The user specific alarm handle.
	ulPnio	UINT32		PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See section "PROFINET Status Code"..

Table 133: PNS\_IF\_RETURN\_OF\_SUB\_ALARM\_CNF\_T - Return of Submodule Alarm Confirmation

## 7.4.6 AR Abort Request Service

With this service the application requests the stack to abort an established AR.


**Note:**

If the device handle refers to an SR-AR Set, all SR-ARs of this set will be aborted.

### 7.4.6.1 AR Abort Request

The application has to send this packet to force the stack to abort an established connection.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
} PNS_IF_HANDLE_DATA_T;

typedef struct PNS_IF_HANDLE_PACKET_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_HANDLE_DATA_T         tData;
} PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T          PNS_IF_ABORT_CONNECTION_REQ_T;
```

#### Packet Description

Structure PNS_IF_ABORT_CONNECTION_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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	0	Status not in use for request.
	ulCmd	UINT32	0x1FD8	PNS_IF_ABORT_CONNECTION_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_HANDLE_DATA_T			
	hDeviceHandle	UINT32		The device handle of the AR to abort.

Table 134: PNS\_IF\_ABORT\_CONNECTION\_REQ\_T - AR Abort Request

### 7.4.6.2 AR Abort Request Confirmation

The stack will send this packet back to the application.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
} PNS_IF_HANDLE_DATA_T;

typedef struct PNS_IF_HANDLE_PACKET_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_HANDLE_DATA_T        tData;
} PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T          PNS_IF_ABORT_CONNECTION_CNF_T;
```

#### Packet Description

Structure PNS_IF_ABORT_CONNECTION_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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 below.
	ulCmd	UINT32	0x1FD9	PNS_IF_ABORT_CONNECTION_CNF-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	Structure PNS_IF_HANDLE_DATA_T			
	hDeviceHandle	UINT32		The device handle

Table 135: PNS\_IF\_ABORT\_CONNECTION\_CNF\_T - AR Abort Request Confirmation

## 7.4.7 Plug Module Service

With this service the application can plug additional modules after the `Set_Configuration Req` packet (see section *Set Configuration Service*) was sent. It is also possible to (re)plug a module which has been pulled by the application.

Plugging a module does not lead to any action visible from the outside (e.g. no alarm is generated to an IO-Controller). Only plugging submodules is visible from the outside.

### 7.4.7.1 Plug Module Request

Receiving this packet forces the stack to send a Plug-alarm to the IO-Controller automatically if a connection is established.

#### Packet Structure Reference

```
typedef struct PNS_IF_PLUG_MODULE_REQ_DATA_Ttag
{
    TLR_UINT32 ulReserved;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulModuleId;
    TLR_UINT16 usModuleState; /* module state: 0 = correct module, 1 = substitute module */
} PNS_IF_PLUG_MODULE_REQ_DATA_T;
```



## Packet Description

Structure PNS_IF_PLUG_MODULE_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	18	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for request.
	ulCmd	UINT32	0x1F04	PNS_IF_PLUG_MODULE_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_PLUG_MODULE_REQ_DATA_T			
	ulReserved	UINT32		Reserved. Set to zero.
	ulApi	UINT32		The API of the module.
	ulSlot	UINT32		The Slot to plug the module to.
	ulModuleId	UINT32		The ModuleID of the module to be plugged.
	usModState	UINT16	0..1	Module state. Informative Only.

Table 136: PNS\_IF\_PLUG\_MODULE\_REQ\_T - Plug Module Request

### 7.4.7.2 Plug Module Confirmation

The stack will return this packet to application.

#### Packet Structure Reference

```
typedef struct PNS_IF_PLUG_MODULE_REQ_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulModuleId;
    TLR_UINT16 usModuleState; /* module state: 0 = correct module, 1 = substitute module */
} PNS_IF_PLUG_MODULE_REQ_DATA_T;

typedef PNS_IF_PLUG_MODULE_REQ_T          PNS_IF_PLUG_MODULE_CNF_T;
```

#### Packet Description

Structure PNS_IF_PLUG_MODULE_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	18	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 below.
	ulCmd	UINT32	0x1F05	PNS_IF_PLUG_MODULE_CNF-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_PLUG_MODULE_CNF_DATA_T			
	hDeviceHandle	UINT32		The device handle
	ulApi	UINT32		The API of the module.
	ulSlot	UINT32		The Slot to plug the module to.
	ulModuleId	UINT32		The ModuleID of the module to be plugged.
	usModState	UINT16	0..1	The Module state.

Table 137: PNS\_IF\_PLUG\_MODULE\_CNF\_T - Plug Module Confirmation

### 7.4.8 Plug Submodule Service

With this service the application can plug additional submodules after the stack has been configured and/or while the device is communicating (see section *Set Configuration Service* of this document) was sent. It is also possible to (re)plug a submodule which has been pulled by the application.

If the plugged submodule was expected in a currently active AR, the controller will be notified about this by means of a plug submodule alarm. The controller will now commission the submodule by writing its parameters. The sequence of this is shown in the following figures:

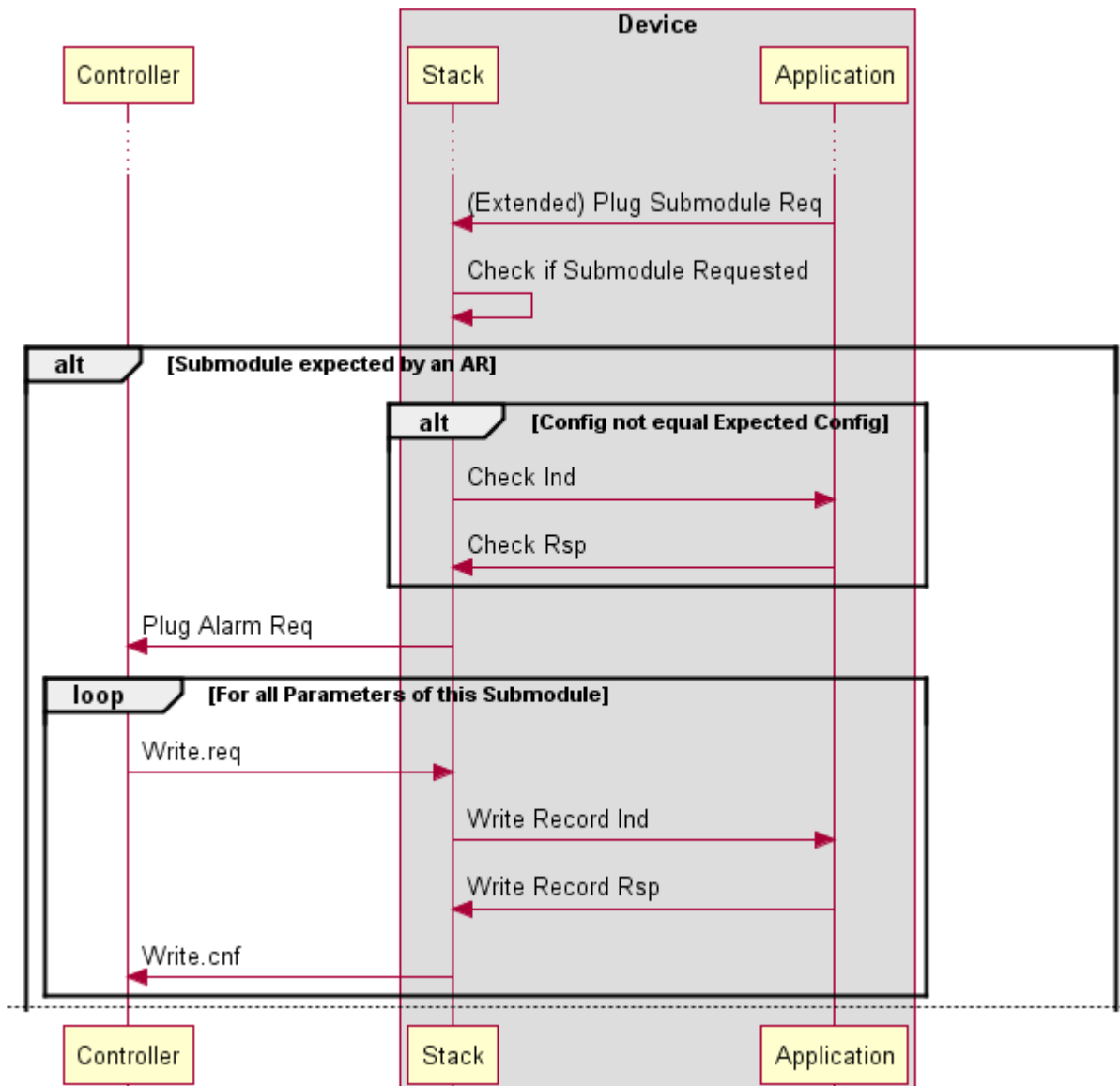


Figure 23: Plug Submodule Service packet sequence

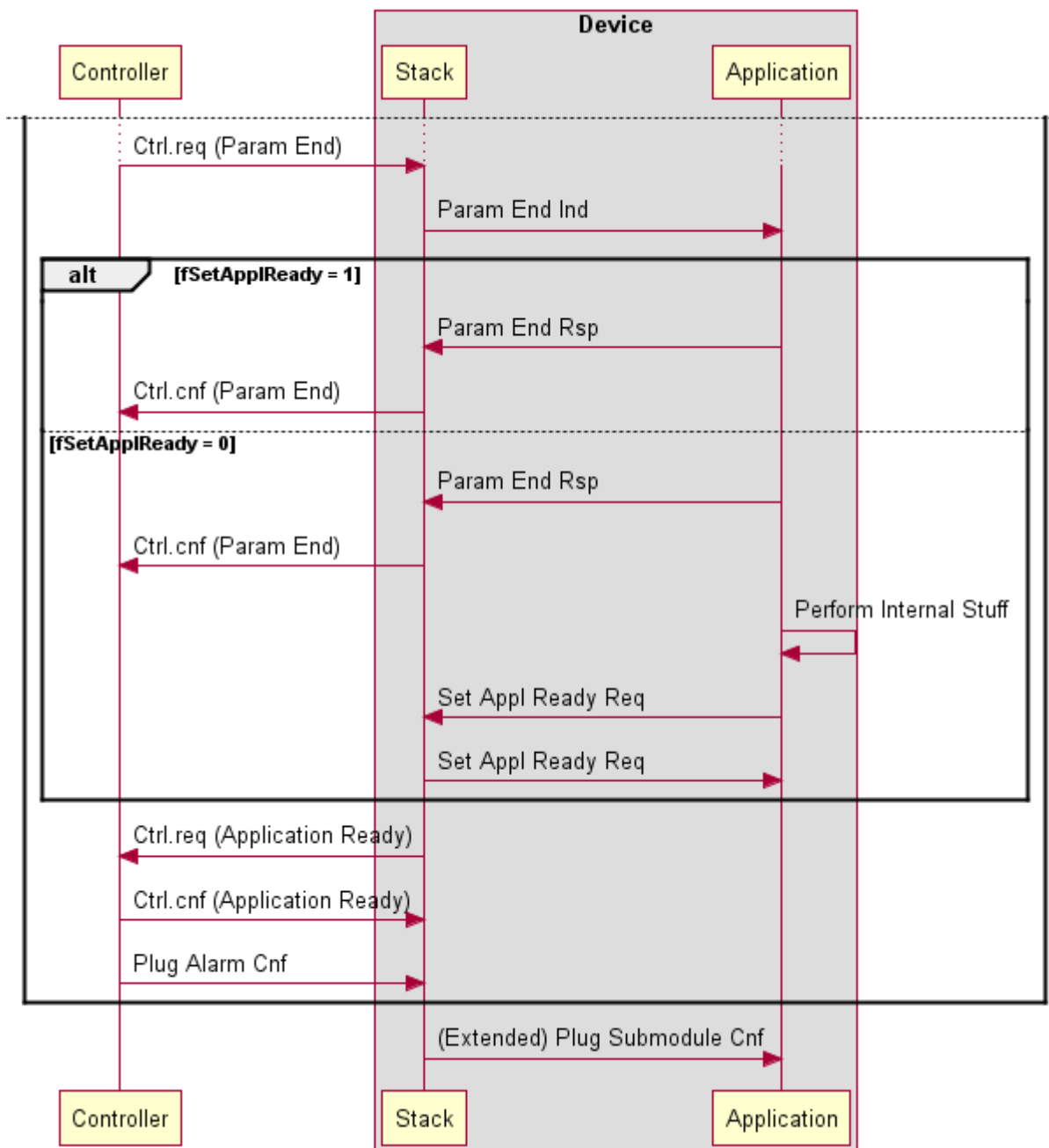


Figure 24: Plug Submodule Service packet sequence (continued)

### 7.4.8.1 Plug Submodule Request



#### Note:

The stack also supports an extended plug submodule request allowing the user to plug modules and submodules using one single request.

#### Packet Structure Reference

```
/* Request packet */
typedef struct PNS_IF_PLUG_SUBMODULE_REQ_DATA_Ttag
{
    TLR_UINT32 ulReserved;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulSubslot;
    TLR_UINT32 ulSubmodId;
    TLR_UINT32 ulProvDataLen;
    TLR_UINT32 ulConsDataLen;
}
```

```

TLR_UINT32 ulDPMOffsetCons;
TLR_UINT32 ulDPMOffsetProv;
TLR_UINT16 usOffsetIOPSPProvider;
TLR_UINT16 usOffsetIOPSCConsumer;
TLR_UINT16 usOffsetIOCSPProvider;
TLR_UINT16 usOffsetIOCSCConsumer;
TLR_UINT32 ulReserved;
TLR_UINT16 usSubmodState;
} PNS_IF_PLUG_SUBMODULE_REQ_DATA_T;

typedef struct PNS_IF_PLUG_SUBMODULE_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_PLUG_SUBMODULE_REQ_DATA_T  tData;
} PNS_IF_PLUG_SUBMODULE_REQ_T;

```

## Packet Description

Structure PNS_IF_PLUG_SUBMODULE_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	50	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for request.
	ulCmd	UINT32	0x1F08	PNS_IF_PLUG_SUBMODULE_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_PLUG_SUBMODULE_REQ_DATA_T			
	ulReserved	UINT32		Reserved. Set to zero.
	ulApi	UINT32		The API of the submodule.
	ulSlot	UINT32		The Slot to plug the submodule to.
	ulSubslot	UINT32		The Subslot to plug the submodule to.
	ulSubmodId	UINT32		The SubmoduleID of the submodule to be plugged.
	ulProvDataLen	UINT32		The provider data length, i.e. the length of the Input data of this submodule. This length describes the data sent by IO-Device and received by IO-Controller.
	ulConsDataLen	UINT32		The consumer data length, i.e. the length of the Output data of this submodule. This length describes the data sent by IO-Controller and received by IO-Device.
	ulDPMOffsetCons	UINT32		Offset in DPM where consumer data for the submodule shall be copied to. If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.

Structure PNS_IF_PLUG_SUBMODULE_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
	ulDPMOffsetProv	UINT32		Offset in DPM where provider data for the submodule shall be taken from. If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.
	usOffsetIOPSPProvider	UINT16		Offset where to put IO provider state for this submodule relative to beginning of IOPS block in DPM output area to
	usOffsetIOPSConsumer	UINT16		Offset where to take IO provider state of this submodule relative to beginning of IOPS block in the DPM input area from
	usOffsetIOCSProvider	UINT16		Offset where to put IO consumer state for this submodule relative to beginning of IOCS block in DPM output area to
	usOffsetIOCSConsumer	UINT16		offset where to take IO consumer state of this submodule relative to beginning of IOCS block in DPM input area from
	ulReserved	UINT32	0	Reserved for future use. Set to zero.
	usSubmodState	UINT16	0..1	The submodule state. See below.

Table 138: PNS\_IF\_PLUG\_SUBMODULE\_REQ\_T - Plug Submodule Request

### 7.4.8.2 Plug Submodule Confirmation

#### Packet Structure Reference

```

/* Confirmation packet */
typedef struct PNS_IF_PLUG_SUBMODULE_REQ_DATA_Ttag
{
    TLR_UINT32 ulReserved;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulSubslot;
    TLR_UINT32 ulSubmodId;
    TLR_UINT32 ulProvDataLen;
    TLR_UINT32 ulConsDataLen;
    TLR_UINT32 ulDPMOffsetCons;
    TLR_UINT32 ulDPMOffsetProv;
    TLR_UINT16 usOffsetIOPSPProvider;
    TLR_UINT16 usOffsetIOPSConsumer;
    TLR_UINT16 usOffsetIOCSProvider;
    TLR_UINT16 usOffsetIOCSConsumer;
    TLR_UINT32 ulReserved;
    TLR_UINT16 usSubmodState;
} PNS_IF_PLUG_SUBMODULE_REQ_DATA_T;

typedef struct PNS_IF_PLUG_SUBMODULE_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_PLUG_SUBMODULE_REQ_DATA_T tData;
} PNS_IF_PLUG_SUBMODULE_REQ_T;

typedef PNS_IF_PLUG_SUBMODULE_REQ_T          PNS_IF_PLUG_SUBMODULE_CNF_T;

```

## Packet Description

Structure PNS_IF_PLUG_SUBMODULE_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	50	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 below.
	ulCmd	UINT32	0x1F09	PNS_IF_PLUG_SUBMODULE_CNF-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_PLUG_SUBMODULE_CNF_DATA_T			
	hDeviceHandle	UINT32		The device handle
	ulApi	UINT32		The API of the module.
	ulSlot	UINT32		The Slot to plug the submodule to.
	ulSubslot	UINT32		The Subslot to plug the submodule to.
	ulSubmodId	UINT32		The SubmoduleID of the submodule to be plugged.
	ulProvDataLen	UINT32		The provider data length, i.e. the length of the Input data of this submodule. This length describes the data sent by IO-Device and received by IO-Controller.
	ulConsDataLen	UINT32		The consumer data length, i.e. the length of the Output data of this submodule. This length describes the data sent by IO-Controller and received by IO-Device.
	ulDPMOffsetCons	UINT32		Offset in DPM where consumer data for the submodule shall be copied to. If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.
	ulDPMOffsetProv	UINT32		Offset in DPM where provider data for the submodule shall be taken from. If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.
	usOffsetIOPSPProvider	UINT16		Offset where to put IO provider state for this submodule relative to beginning of IOPS block in DPM output area to
	usOffsetIOPSPConsumer	UINT16		Offset where to take IO provider state of this submodule relative to beginning of IOPS block in the DPM input area from
	usOffsetIOCSProvider	UINT16		Offset where to put IO consumer state for this submodule relative to beginning of IOCS block in DPM output area to

Structure <code>PNS_IF_PLUG_SUBMODULE_CNF_T</code>				Type: Confirmation
Area	Variable	Type	Value / Range	Description
	<code>ulDPMOffsetIocsOut</code>	UINT32		Offset in DPM where IOCS is taken from.
	<code>ulReserved</code>	UINT32	0	Reserved for future use. Set to zero.
	<code>usSubmodState</code>	UINT16	0..1	The submodule state.

Table 139: `PNS_IF_PLUG_SUBMODULE_CNF_T` - Plug Submodule Confirmation



### 7.4.8.3 Extended Plug Submodule Request

Receiving this packet the stack adds the described submodule and module to its internal configuration in order to use it for data exchange. The module will be added only if necessary. If the submodule was requested by an IO-Controller for data exchange before, the stack will automatically notify the controller by issuing a plug submodule alarm.

#### Packet Structure Reference

```
typedef struct PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_DATA_Ttag
{
    TLR_UINT32 ulReserved;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulSubslot;
    TLR_UINT32 ulSubmodId;
    TLR_UINT32 ulProvDataLen;
    TLR_UINT32 ulConsDataLen;
    TLR_UINT32 ulDPMOffsetCons;
    TLR_UINT32 ulDPMOffsetProv;
    TLR_UINT16 usOffsetIOPSPProvider;
    TLR_UINT16 usOffsetIOPSCConsumer;
    TLR_UINT16 usOffsetIOCSPProvider;
    TLR_UINT16 usOffsetIOCSCConsumer;
    TLR_UINT32 ulReserved;
    TLR_UINT16 usSubmodState;
    TLR_UINT32 ulModuleId;
    TLR_UINT16 usModuleState;
} PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_DATA_T;

typedef struct PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_DATA_T tData;
} PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_T;
```

#### Packet Description

Structure PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_T				Type:
Structure Request				
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	56	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for request.
	ulCmd	UINT32	0x1F08	PNS_IF_PLUG_SUBMODULE_REQ-command (It's the same command as standard plug submodule request)

Structure PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_T				Type:
Structure Request				
Area	Variable	Type	Value / Range	Description
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	0	Routing, set to zero.
Data	structure PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_DATA_T			
	ulReserved	UINT32		Reserved. Set to zero.
	ulApi	UINT32		The API of the submodule.
	ulSlot	UINT32		The Slot to plug the submodule to.
	ulSubslot	UINT32		The Subslot to plug the submodule to.
	ulSubmodId	UINT32		The SubmoduleID of the submodule to be plugged.
	ulProvDataLen	UINT32		The provider data length, i.e. the length of the Input data of this submodule. This length describes the data sent by IO-Device and received by IO-Controller.
	ulConsDataLen	UINT32		The consumer data length, i.e. the length of the Output data of this submodule. This length describes the data sent by IO-Controller and received by IO-Device.
	ulDPMOffsetCons	UINT32		Offset in DPM where consumer data for the submodule shall be copied to. If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.
	ulDPMOffsetProv	UINT32		Offset in DPM where provider data for the submodule shall be taken from. If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.
	usOffsetIOPSProvider	UINT16		Offset where to put IO provider state for this submodule relative to beginning of IOPS block in the DPM output area to
	usOffsetIOPSConsumer	UINT16		Offset where to take IO provider state of this submodule relative to beginning of IOPS block in the DPM input area from
	usOffsetIOCSProvider	UINT16		Offset where to put IO consumer state for this submodule relative to beginning of IOCS block in DPM output area to
	usOffsetIOCSConsumer	UINT16		offset where to take IO consumer state of this submodule relative to beginning of IOCS block in DPM input area from
	ulReserved	UINT32	0	Reserved for future use. Set to zero.
	usSubmodState	UINT16	0..1	The submodule state. See below
	ulModuleId	UINT32	1..0xFFFFFFFF	The module identifier.
	usModuleState	UINT16	0..1	The module state. See section 6.4.7.1

Table 140: PNS\_IF\_PLUG\_SUBMODULE\_EXTENDED\_REQ\_T – Extended Plug Submodule Request

### 7.4.8.4 Extended Plug Submodule Confirmation

This is the confirmation of the extended plug submodule request.

#### Packet Structure Reference

```
typedef PNS_IF_PLUG_SUBMODULE_REQ_T PNS_IF_PLUG_SUBMODULE_CNF_T;
```

#### Packet Description

Structure PNS_IF_PLUG_SUBMODULE_EXTENDED_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Ignore.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	56	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		Result of operation.
	ulCmd	UINT32	0x1F09	PNS_IF_PLUG_SUBMODULE_CNF-command (It's the same command as standard plug submodule confirmation)
	ulExt	UINT32	X	Extension. Ignore
	ulRout	UINT32	X	Routing, Ignore
Data	structure PNS_IF_PLUG_SUBMODULE_EXTENDED_CNF_DATA_T			
	hDeviceHandle	UINT32		The device handle.
	ulApi	UINT32		The API of the submodule.
	ulSlot	UINT32		The Slot the submodule was plugged into.
	ulSubslot	UINT32		The Subslot the submodule was plugged into
	ulSubmodId	UINT32		The SubmoduleID of the submodule.
	ulProvDataLen	UINT32		The provider data length, i.e. the length of the Input data of this submodule.
	ulConsDataLen	UINT32		The consumer data length, i.e. the length of the Output data of this submodule
	ulDPMOffsetCons	UINT32		Offset in DPM where consumer data for the submodule will be copied to.
	ulDPMOffsetProv	UINT32		Offset in DPM where provider data for the submodule will be taken from.
	usOffsetIOPSProvider	UINT16		Offset relative to beginning of IOPS block in the DPM output area. where the IO provider state for this submodule will be put to.
	usOffsetIOPSConsumer	UINT16		Offset relative to beginning of IOPS block in the DPM input area where the IO provider state of this submodule will be taken from.

Structure PNS_IF_PLUG_SUBMODULE_EXTENDED_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
	usOffsetIOCSProvider	UINT16		Offset where to put IO consumer state for this submodule relative to beginning of IOCS block in DPM output area to
	usOffsetIOCSConsumer	UINT16		offset where to take IO consumer state of this submodule relative to beginning of IOCS block in DPM input area from
	ulReserved	UINT32	0	Reserved for future use. Ignore
	usSubmodState	UINT16	0..1	The submodule state
	ulModuleId	UINT32	1..0xFFFFFFFF	The module identifier.
	usModuleState	UINT16	0..1	The module state

Table 141: PNS\_IF\_PLUG\_SUBMODULE\_EXTENDED\_CNF\_T – Extended Plug Submodule Confirmation

### 7.4.9 Pull Module Service

With this service the application can pull modules. This removes the module and its submodules from the configuration. The stack will generate a Pull Module Alarm for each AR which owned any submodule of this module. This sequence is illustrated in the following figure.

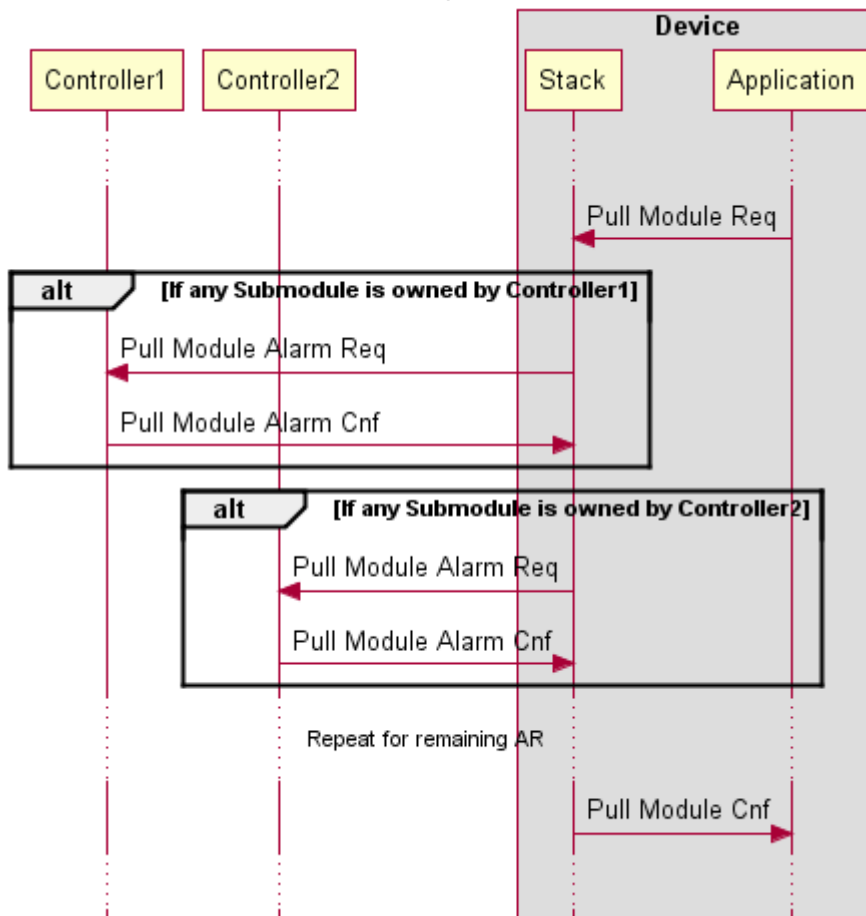


Figure 25: Pull Module Service packet sequence

#### 7.4.9.1 Pull Module Request

Receiving this packet forces the stack to automatically send a Pull-alarm to the IO-Controller if a connection is established.

#### Packet Structure Reference

```

typedef struct PNS_IF_PULL_MODULE_REQ_DATA_Ttag
{
    TLR_UINT32 ulReserved;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
} PNS_IF_PULL_MODULE_REQ_DATA_T;

typedef struct PNS_IF_PULL_MODULE_REQ_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T tHead;
    /** packet data */
    PNS_IF_PULL_MODULE_REQ_DATA_T tData;
} PNS_IF_PULL_MODULE_REQ_T;
  
```

## Packet Description

Structure PNS_IF_PULL_MODULE_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	12	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for request.
	ulCmd	UINT32	0x1F06	PNS_IF_PULL_MODULE_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_PULL_MODULE_REQ_DATA_T			
	ulReserved	UINT32		Reserved. Set to Zero.
	ulApi	UINT32		The API of the module.
	ulSlot	UINT32		The Slot to pull the module from.

Table 142: PNS\_IF\_PULL\_MODULE\_REQ\_T - Pull Module Request

### 7.4.9.2 Pull Module Confirmation

The stack will return this packet to the application.

## Packet Structure Reference

```

/* Confirmation packet */
typedef struct PNS_IF_PULL_MODULE_REQ_DATA_Ttag
{
    TLR_UINT32 ulReserved;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
} PNS_IF_PULL_MODULE_REQ_DATA_T;

typedef struct PNS_IF_PULL_MODULE_REQ_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T                tHead;
    /** packet data */
    PNS_IF_PULL_MODULE_REQ_DATA_T      tData;
} PNS_IF_PULL_MODULE_REQ_T;

typedef PNS_IF_PULL_MODULE_REQ_T      PNS_IF_PULL_MODULE_CNF_T;

```

## Packet Description

Structure PNS_IF_PULL_MODULE_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	12	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1F07	PNS_IF_PULL_MODULE_CNF-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_PULL_MODULE_CNF_DATA_T			
	ulReserved	UINT32		Reserved. Will be set to Zero.
	ulApi	UINT32		The API of the module.
	ulSlot	UINT32		The Slot to pull the module from.

Table 143: PNS\_IF\_PULL\_MODULE\_CNF\_T – Pull Module Confirmation

## 7.4.10 Pull Submodule Service

With this service the application can pull submodules. This will remove the submodule from the stacks configuration. If the submodule is in use by any AR a Pull Alarm will be generated automatically. The sequence is shown in the following figure.

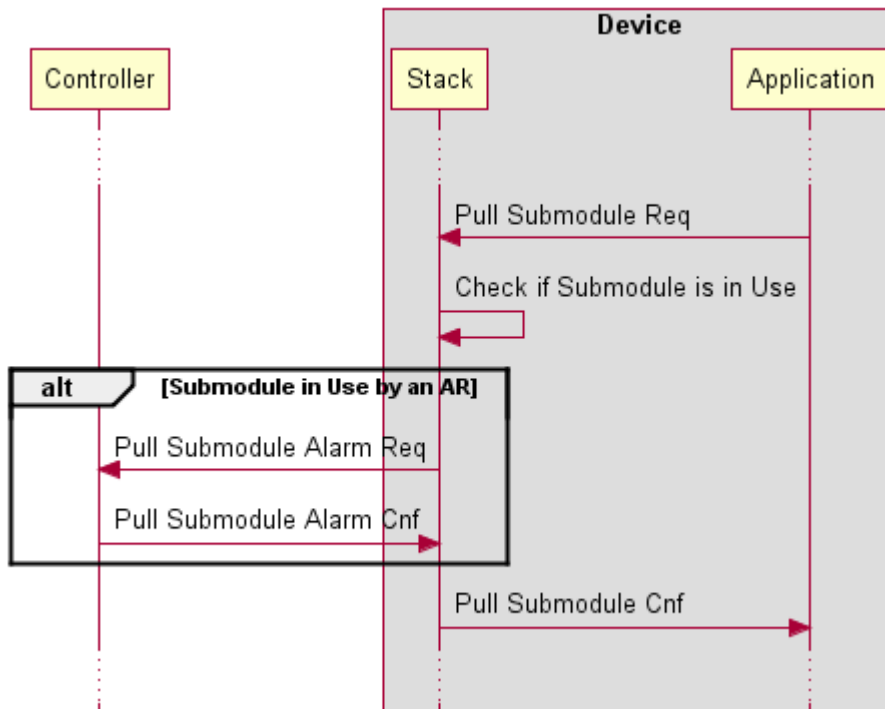


Figure 26: Pull Submodule Service packet sequence

### 7.4.10.1 Pull Submodule Request

#### Packet Structure Reference

```

/* Request packet */
typedef struct PNS_IF_PULL_SUBMODULE_REQ_DATA_Ttag
{
    TLR_UINT32 hDeviceHandle;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulSubslot;
} PNS_IF_PULL_SUBMODULE_REQ_DATA_T;

typedef struct PNS_IF_PULL_SUBMODULE_REQ_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T tHead;
    /** packet data */
    PNS_IF_PULL_SUBMODULE_REQ_DATA_T tData;
} PNS_IF_PULL_SUBMODULE_REQ_T;
  
```



## Packet Description

Structure PNS_IF_PULL_SUBMODULE_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	16	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for request.
	ulCmd	UINT32	0x1F0A	PNS_IF_PULL_SUBMODULE_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_PULL_SUBMODULE_REQ_DATA_T			
	hDeviceHandle	UINT32		Device Handle Was formerly: Reserved. Set to Zero.
	ulApi	UINT32		The API of the submodule.
	ulSlot	UINT32		The Slot to pull the submodule from.
	ulSubslot	UINT32		The subslot to pull the submodule from.

Table 144: PNS\_IF\_PULL\_SUBMODULE\_REQ\_T - Pull Submodule Request

### 7.4.10.2 Pull Submodule Confirmation

The stack will return this packet to the application.

## Packet Structure Reference

```

/* Confirmation packet */
typedef struct PNS_IF_PULL_SUBMODULE_REQ_DATA_Ttag
{
    TLR_UINT32 ulReserved;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulSubslot;
} PNS_IF_PULL_SUBMODULE_REQ_DATA_T;

typedef struct PNS_IF_PULL_SUBMODULE_REQ_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_PULL_SUBMODULE_REQ_DATA_T    tData;
} PNS_IF_PULL_SUBMODULE_REQ_T;
typedef PNS_IF_PULL_SUBMODULE_REQ_T    PNS_IF_PULL_SUBMODULE_CNF

```

## Packet Description

Structure PNS_IF_PULL_SUBMODULE_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	16	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1F0B	PNS_IF_PULL_SUBMODULE_CNF-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_PULL_SUBMODULE_CNF_DATA_T			
	ulReserved	UINT32		Reserved. Will be set to Zero.
	ulApi	UINT32		The API of the submodule.
	ulSlot	UINT32		The Slot to pull the submodule from.
	ulSubslot	UINT32		The subslot to pull the submodule from.

Table 145: PNS\_IF\_PULL\_SUBMODULE\_CNF\_T - Pull Submodule Confirmation

## 7.4.11 Get Station Name Service

With this service the application can request the current *NameOfStation* from the stack.



### Note:

In this service the confirmation packet is larger than the request packet. If the application is running on the netX and DPM is not used the application has to provide a buffer which is big enough.

### 7.4.11.1 Get Station Name Request

The application has to send the following request packet to the stack.

#### Packet Structure Reference

```
/* Request packet */
typedef TLR_EMPTY_PACKET_T                               PNS_IF_GET_STATION_NAME_REQ_T;
```

#### Packet Description

Structure PNS_IF_GET_STATION_NAME_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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	0	Status not in use for requests.
	ulCmd	UINT32	0x1F8E	PNS_IF_GET_STATION_NAME_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 146: PNS\_IF\_GET\_STATION\_NAME\_REQ\_T - Get Station Name Request

### 7.4.11.2 Get Station Name Confirmation

The following confirmation packet containing the current *NameOfStation* will be returned:

#### Packet Structure Reference

```
/* Confirmation packet */
typedef struct PNS_IF_GET_STATION_NAME_CNF_DATA_Ttag
{
    TLR_UINT16    usNameLen;
    TLR_UINT8     abNameOfStation[PONIO_MAX_NAME_OF_STATION];
} PNS_IF_GET_STATION_NAME_CNF_DATA_T;

typedef struct PNS_IF_GET_STATION_NAME_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_GET_STATION_NAME_CNF_DATA_T    tData;
} PNS_IF_GET_STATION_NAME_CNF_T;
```

#### Packet Description

Structure PNS_IF_GET_STATION_NAME_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	242	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status has to be okay for this service.
	ulCmd	UINT32	0x1F8F	PNS_IF_GET_STATION_NAME_CNF-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	Structure PNS_IF_GET_STATION_NAME_CNF_DATA_T			
	usNameLen	UINT16	0..240	Length of the current NameOfStation.
	abNameOfStation[240]	UINT8		The NameOfStation as ASCII byte-array. For the station name, only small characters are allowed

Table 147: PNS\_IF\_GET\_STATION\_NAME\_CNF\_T - Get Station Name Confirmation

## 7.4.12 Get IP Address Service

With this service the application can request the current IP-parameters (IP address, network mask, gateway address) from the stack.



### Note:

In this service the confirmation packet is larger than the request packet. If the application is running on the netX and DPM is not used the application has to provide a buffer which is big enough.

### 7.4.12.1 Get IP Address Request

The application has to send the following request packet to the stack.

#### Packet Structure Reference

```
/* Request packet */
typedef TLR_EMPTY_PACKET_T PNS_IF_GET_IP_ADDR_REQ_T;
```

#### Packet Description

Structure PNS_IF_GET_IP_ADDR_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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	0	Status not in use for requests.
	ulCmd	UINT32	0x1FBC	PNS_IF_GET_IP_ADDR_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 148: PNS\_IF\_GET\_IP\_ADDR\_REQ\_T - Get IP Address Request

### 7.4.12.2 Get IP Address Confirmation

The following confirmation packet containing the IP parameters will be returned:

#### Packet Structure Reference

```
/* Confirmation packet */
typedef struct PNS_IF_GET_IP_ADDR_CNF_DATA_Ttag
{
    TLR_UINT32 ulIpAddr;
    TLR_UINT32 ulNetMask;
    TLR_UINT32 ulGateway;
} PNS_IF_GET_IP_ADDR_CNF_DATA_T;

typedef struct PNS_IF_GET_IP_ADDR_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_GET_IP_ADDR_CNF_DATA_T tData;
} PNS_IF_GET_IP_ADDR_CNF_T;
```

#### Packet Description

Structure PNS_IF_GET_IP_ADDR_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	12	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status has to be okay for this service.
	ulCmd	UINT32	0x1FBD	PNS_IF_GET_IP_ADDR_CNF -command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_GET_IP_ADDR_CNF_DATA_T			
	ulIpAddr	UINT32		The IP address.
	ulNetMask	UINT32		The network mask.
	ulGateway	UINT32		The gateway address.

Table 149: PNS\_IF\_GET\_IP\_ADDR\_CNF\_T - Get IP Address Confirmation

### 7.4.13 Add Channel Diagnosis Service

With this service the user application can add a diagnosis data record to a submodule.

Inside the confirmation packet the stack sends a unique record handle to the application. This handle has to be stored by the application as it is needed to remove the diagnosis record later on. It is also needed if the application wants to send the diagnosis alarm some time later than the record is added.


**Note:**

The stack does not automatically send a diagnosis alarm to the IO-Controller. This has to be initiated by the application using the Diagnosis Alarm Service (see section 6.4.4).

#### 7.4.13.1 Add Channel Diagnosis Request

Using this packet the user application can add diagnosis data to a submodule.

#### Packet Structure Reference

```
/* Request packet */
typedef struct PNS_IF_ADD_CHANNEL_DIAG_Ttag
{
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_UINT32    hDiagHandle;
    TLR_UINT16    usChannelNum;
    TLR_UINT16    usChannelProp;
    TLR_UINT16    usChannelErrType;
} PNS_IF_ADD_CHANNEL_DIAG_T;

typedef struct PNS_IF_ADD_CHANNEL_DIAG_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_ADD_CHANNEL_DIAG_T    tData;
} PNS_IF_ADD_CHANNEL_DIAG_REQ_T;
```

#### Packet Description

Structure PNS_IF_ADD_CHANNEL_DIAG_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	22	PNS_IF_ADD_CHANNEL_DIAG_REQ_T - Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for requests. Set to zero.
	ulCmd	UINT32	0x1F46	PNS_IF_ADD_CHANNEL_DIAG_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons

Structure PNS_IF_ADD_CHANNEL_DIAG_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_ADD_CHANNEL_DIAG_T			
	ulApi	UINT32		The API of the submodule.
	ulSlot	UINT32		The Slot of the submodule.
	ulSubslot	UINT32		The Subslot of the submodule.
	hDiagHandle	UINT32	0	Not used inside this request. Will be used for confirmation by the stack. Set to zero.
	usChannelNum	UINT16		Channel Number for which the diagnosis data shall be added. Supported are Manufacturer specific Channel Numbers in the range of 0x0000-0x7FFF and the Channel number for the submodule itself 0x8000
	usChannelProp	UINT16		See Table 148: Coding of the field usChannelProp
	usChannelErrType	UINT16		See Table 147: Coding of usChannelErrType

Table 150: PNS\_IF\_ADD\_CHANNEL\_DIAG\_REQ\_T - Add Channel Diagnosis Request

### Coding of the field usChannelErrType

Value (Hexadecimal)	Description
0x0000	Reserved
0x0001	Short circuit
0x0002	Undervoltage
0x0003	Overvoltage
0x0004	Overload
0x0005	Overtemperature
0x0006	Line break
0x0007	Upper limit value exceeded
0x0008	Lower limit value exceeded
0x0009	Error
0x000A	Simulation active
0x000B - 0x000E	Reserved
0x000F	Manufacturer specific, recommended for "parameterization missing"
0x0010	Manufacturer specific, recommended for "parameterization fault"
0x0011	Manufacturer specific, recommended for "power supply fault"
0x0012	Manufacturer specific, recommended for "fuse blown / open"
0x0013	Manufacturer specific, recommended for "communication fault"
0x0014	Manufacturer specific, recommended for "ground fault"
0x0015	Manufacturer specific, recommended for "reference point lost"
0x0016	Manufacturer specific, recommended for "process event lost / sampling error"
0x0017	Manufacturer specific, recommended for "threshold warning"
0x0018	Manufacturer specific, recommended for "output disabled"
0x0019	Manufacturer specific, recommended for "safety event"



Value (Hexadecimal)	Description
0x001A	Manufacturer specific, recommended for “external fault”
0x001B – 0x001E	Manufacturer specific
0x001F	Manufacturer specific, recommended for “temporary fault”
0x0020 – 0x00FF	Reserved for common profiles (assigned by PROFIBUS International)
0x0100 – 0x7FFF	Manufacturer specific
0x8000	Data transmission impossible
0x8001	Remote mismatch
0x8002	Media redundancy mismatch
0x8003	Sync mismatch
0x8004	Isochronous mode mismatch
0x8005	Multicast CR mismatch
0x8006	Reserved
0x8007	Fiber optic mismatch
0x8008	Network component function mismatch
0x8009	Time mismatch
0x800A	Dynamic frame packing function mismatch
0x800B	Media redundancy with planned duplication mismatch
0x800C	System redundancy mismatch
0x800D	Multiple interface mismatch
0x800E	Nested diagnosis indication
0x800F – 0x8FFF	Reserved
0x9000 – 0x9FFF	Reserved for profiles
0xA000 – 0xFFFF	Reserved

Table 151: Coding of *usChannelErrType*. Note: values 0x8000 – 0x800E will be triggered by the Stack internally and shall not be set by application

### Coding of the field *usChannelProp*

Bit No	Description
D0 – D7	Data type of this channel (see Table 149: Coding of the field <i>Type</i> in field <i>usChannelProp</i> )
D8	Accumulative. It should be set if the diagnosis is accumulated from several channels
D9 – D10	Maintenance (see Table 150: Coding of the field <i>Maintenance</i> in field <i>usChannelProp</i> )
D11 – D12	Specifier. It will be handled by the Stack and shall not be set by application.
D13 – D15	Direction (see Table 151: Coding of the field <b>Direction</b> in field <i>usChannelProp</i> )

Table 152: Coding of the field *usChannelProp*

### Coding of the field *Type*

Value (Hexadecimal)	Description
0x00	Should be used if ChannelNumber is 0x8000 or If none of the below defined types are appropriate

0x01	1 Bit
0x02	2 Bits
0x03	4 Bits
0x04	8 Bits
0x05	16 Bits
0x06	32 Bits
0x07	64 Bits
0x07 - 0xFF	Reserved

Table 153: Coding of the field *Type* in field *usChannelProp*

### Coding of the field Maintenance

Value (Hexadecimal)	Description
0x00	Diagnosis
0x01	Maintenance Required
0x02	Maintenance Demanded
0x03	Qualified Diagnosis

Table 154: Coding of the field *Maintenance* in field *usChannelProp*

### Coding of the field Direction

Value (Hexadecimal)	Description
0x00	Manufacturer specific
0x01	Input
0x02	Output
0x03	Input / Output
0x04 - 0xFF	Reserved

Table 155: Coding of the field **Direction** in field *usChannelProp*

### 7.4.13.2 Add Channel Diagnosis Confirmation

With this packet the stack informs the application about the success of adding diagnosis data.

#### Packet Structure Reference

```
/* Confirmation packet */
typedef struct PNS_IF_ADD_CHANNEL_DIAG_Ttag
{
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_UINT32    hDiagHandle;
    TLR_UINT16    usChannelNum;
    TLR_UINT16    usChannelProp;
    TLR_UINT16    usChannelErrType;
} PNS_IF_ADD_CHANNEL_DIAG_T;

typedef struct PNS_IF_ADD_CHANNEL_DIAG_REQ_Ttag
{
    TLR_PACKET_HEADER_T    tHead;
    PNS_IF_ADD_CHANNEL_DIAG_T    tData;
} PNS_IF_ADD_CHANNEL_DIAG_REQ_T;

typedef PNS_IF_ADD_CHANNEL_DIAG_REQ_T    PNS_IF_ADD_CHANNEL_DIAG_CNF_T;
```

#### Packet Description

Structure PNS_IF_ADD_CHANNEL_DIAG_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	22	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1F47	PNS_IF_ADD_CHANNEL_DIAG_CNF - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_ADD_CHANNEL_DIAG_T			
	ulApi	UINT32		The API of the submodule.
	ulSlot	UINT32		The Slot of the submodule.
	ulSubslot	UINT32		The Subslot of the submodule.
	hDiagHandle	UINT32		A unique handle representing this diagnostic record inside the stack. The application shall store it as it is has to be used to be able to remove the record later on.

Structure PNS_IF_ADD_CHANNEL_DIAG_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
	usChannelNum	UINT16		Channel Number for which the diagnosis data shall be added. Supported are Manufacturer specific Channel Numbers in the range of 0x0000-0x7FFF and the Channel number for the submodule itself 0x8000
	usChannelProp	UINT16		See Table 148: Coding of the field usChannelProp
	usChannelErrType	UINT16		See Table 147: Coding of usChannelErrType

Table 156: PNS\_IF\_ADD\_CHANNEL\_DIAG\_CNF\_T - Add Channel Diagnosis Confirmation

## 7.4.14 Add Extended Channel Diagnosis Service

With this service the user application can add an extended diagnosis data record to a submodule. Inside the confirmation packet the stack sends a unique record handle to the application. This handle has to be stored by the application as it is needed to remove the diagnosis record later on. It is also needed if the application wants to send the diagnosis alarm some time later than the record is added.



### Note:

The stack does not automatically send a diagnosis alarm to the IO-Controller. This has to be initiated by the application using the Diagnosis Alarm Service (see section 6.4.4).

### 7.4.14.1 Add Extended Channel Diagnosis Request

The user application can add extended diagnosis data to a submodule using this packet.

#### Packet Structure Reference

```
/* Request packet */
typedef struct PNS_IF_ADD_EXTENDED_DIAG_Ttag
{
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_UINT32    hDiagHandle;
    TLR_UINT16    usChannelNum;
    TLR_UINT16    usChannelProp;
    TLR_UINT16    usChannelErrType;
    TLR_UINT16    usReserved;
    TLR_UINT32    ulExtChannelAddValue;
    TLR_UINT16    usExtChannelErrType;
} PNS_IF_ADD_EXTENDED_DIAG_T;

typedef struct PNS_IF_ADD_EXTENDED_DIAG_REQ_Ttag
{
    TLR_PACKET_HEADER_T    tHead;
    PNS_IF_ADD_EXTENDED_DIAG_T    tData;
} PNS_IF_ADD_EXTENDED_DIAG_REQ_T;
```

#### Packet Description

Structure PNS_IF_ADD_EXTENDED_DIAG_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	30	PNS_IF_ADD_EXT_DIAG_REQ_T - Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for requests. Set to zero.

Structure PNS_IF_ADD_EXTENDED_DIAG_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
	ulCmd	UINT32	0x1F54	PNS_IF_ADD_EXTENDED_DIAG_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_ADD_EXTENDED_DIAG_T			
	ulApi	UINT32		The API of the submodule.
	ulSlot	UINT32		The Slot of the submodule.
	ulSubslot	UINT32		The Subslot of the submodule.
	hDiagHandle	UINT32	0	Not used inside this request. Will be used for confirmation by the stack. Set to zero.
	usChannelNum	UINT16		Channel Number for which the diagnosis data shall be added. Supported are Manufacturer specific Channel Numbers in the range of 0x0000-0x7FFF and the Channel number for the submodule itself 0x8000
	usChannelProp	UINT16		See Table <b>148: Coding of the</b> field usChannelProp
	usChannelErrType	UINT16		See Table 147: <b>Coding of usChannelErrType</b>
	usReserved	UINT16		Reserved
	ulExtChannelAddValue	UINT32		Additional Value. Can be used to transfer additional information regarding the diagnosis. (E.g. Temperature) Can be displayed by engineering software using format strings defined in GSDML.
	usExtChannelErrType	UINT16	1 ... 0x7FFF	See Table <b>154: Coding of usExtChannelErrType</b> for Channel Error Type 1 - 0x7FFF

Table 157: PNS\_IF\_ADD\_EXTENDED\_DIAG\_REQ\_T - Add Extended Channel Diagnosis Request

### Coding of the field usExtChannelErrType

The value of this field depends on the field usChannelErrType. The values shall be set according to following tables:

Value (hexadecimal)	Description	Usage
0x0000	Reserved	
0x0001 - 0x7FFF	Manufacturer specific	Alarm / Diagnosis
0x8000	Accumulative info	Alarm / Diagnosis
0x8001 - 0x8FFF	Reserved	
0x9000 - 0x9FFF	Profile specific error codes	Alarm / Diagnosis
0xA000 - 0xFFFF	Reserved	

Table 158: Coding of usExtChannelErrType for Channel Error Type 1 - 0x7FFF

### 7.4.14.2 Add Extended Channel Diagnosis Confirmation

With this packet the stack informs the application about the success of adding extended diagnosis data.

#### Packet Structure Reference

```
/* Confirmation packet */
typedef struct PNS_IF_ADD_EXTENDED_DIAG_Ttag
{
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_UINT32    hDiagHandle;
    TLR_UINT16    usChannelNum;
    TLR_UINT16    usChannelProp;
    TLR_UINT16    usChannelErrType;
    TLR_UINT16    usReserved;
    TLR_UINT32    ulExtChannelAddValue;
    TLR_UINT16    usExtChannelErrType;
} PNS_IF_ADD_EXTENDED_DIAG_T;

typedef struct PNS_IF_ADD_EXTENDED_DIAG_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_ADD_EXTENDED_DIAG_T    tData;
} PNS_IF_ADD_EXTENDED_DIAG_REQ_T;

typedef PNS_IF_ADD_EXTENDED_DIAG_REQ_T          PNS_IF_ADD_EXTENDED_DIAG_CNF_T;
```

#### Packet Description

Structure PNS_IF_ADD_EXTENDED_DIAG_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	30	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1F55	PNS_IF_ADD_EXTENDED_DIAG_CNF - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_ADD_EXTENDED_DIAG_T			
	ulApi	UINT32		The API of the submodule.
	ulSlot	UINT32		The Slot of the submodule.
	ulSubslot	UINT32		The Subslot of the submodule.

Structure PNS_IF_ADD_EXTENDED_DIAG_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
	hDiagHandle	UINT32		A unique handle representing this diagnostic record inside the stack. Application shall store it as it is has to be used to be able to remove the record later on.
	usChannelNum	UINT16		Channel Number for which the diagnosis data shall be added. Supported are Manufacturer specific Channel Numbers in the range of 0x0000-0x7FFF and the Channel number for the submodule itself 0x8000.
	usChannelProp	UINT16		See Table 148: Coding of the field usChannelProp
	usChannelErrType	UINT16		See Table 147: Coding of usChannelErrType
	usReserved	UINT16		Reserved
	ulExtChannelAddValue	UINT32	0	Currently not supported, set to zero.
	usExtChannelErrType	UINT16		See Table 154: Coding of usExtChannelErrType for Channel Error Type 1 - 0x7FFF

Table 159: PNS\_IF\_ADD\_EXTENDED\_DIAG\_CNF\_T - Add Extended Channel Diagnosis Confirmation



## 7.4.15 Add Generic Diagnosis Service

With this service the user application can add a generic diagnosis data record to a submodule.

Inside the confirmation packet the stack sends a unique record handle to the application. This handle has to be stored by the application as it is needed to remove the diagnosis record later on. It is also needed if the application wants to send the diagnosis alarm some time later than the record is added.



### Note:

The stack does not automatically send a diagnosis alarm to the PROFINET IO-Controller. This has to be initiated by the application using the Diagnosis Alarm Service (see section 6.4.4).



### Note:

Usage of this service is not recommended. Generic diagnosis can not be handled automatically by Engineering systems. The PI Diagnosis Guideline recommends not to use this service.

### 7.4.15.1 Add Generic Channel Diagnosis Request

Using this packet the user application can add generic diagnosis data to a submodule.

#### Packet Structure Reference

```
/* Request packet */
typedef struct PNS_IF_ADD_GENERIC_DIAG_REQ_DATA_Ttag
{
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_UINT32    hDiagHandle;
    TLR_UINT16    usChannelNum;
    TLR_UINT16    usChannelProp;
    TLR_UINT16    usUserStructId;
    TLR_UINT16    usReserved;
    TLR_UINT16    usDiagDataLen;
    TLR_UINT8     abDiagData[PNS_IF_MAX_ALARM_DATA_LEN];
} PNS_IF_ADD_GENERIC_DIAG_REQ_DATA_T;

typedef struct
{
    TLR_PACKET_HEADER_T    tHead;
    PNS_IF_ADD_GENERIC_DIAG_REQ_DATA_T    tData;
} PNS_IF_ADD_GENERIC_DIAG_REQ_T;
```

#### Packet Description

Structure PNS_IF_ADD_GENERIC_DIAG_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	26 + n	PNS_IF_ADD_GENERIC_DIAG_REQ - Packet data length in bytes + usDiagDataLen

Structure PNS_IF_ADD_GENERIC_DIAG_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
	ulId	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not used for requests. Set to zero.
	ulCmd	UINT32	0x1F58	PNS_IF_ADD_GENERIC_DIAG_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_ADD_GENERIC_DIAG_REQ_DATA_T			
	ulApi	UINT32		The API of the submodule.
	ulSlot	UINT32		The Slot of the submodule.
	ulSubslot	UINT32		The Subslot of the submodule.
	hDiagHandle	UINT32	0	Not used inside this request. Will be used for confirmation by the stack. Set to zero.
	usChannelNum	UINT16	0x8000	Channel Number for which the diagnosis data shall be added .Supported are Manufacturer specific Channel Numbers in the range of 0x0000-0x7FFF and the Channel number for the submodule itself 0x8000.
	usChannelProp	UINT16		See Table 148: Coding of the field usChannelProp
	usUserStructId	UINT16	0 - 0x7FFF	See Table 157: Coding of the field usUserStructId
	usReserved	UINT16		Reserved
	usDiagDataLen	UINT16	1..1024	Length of Diagnosis Data in bytes
	abDiagData [1024]	UINT8		Diagnosis Data

Table 160: PNS\_IF\_ADD\_GENERIC\_DIAG\_REQ\_T - Add Generic Channel Diagnosis Request

### Coding of the field usUserStructId

Value (Hexadecimal)	Description
0 - 0x7FFF	Manufacturer Specific.

Table 161: Coding of the field usUserStructId

## 7.4.15.2 Add Generic Channel Diagnosis Confirmation

With this packet the stack informs the application about the success of adding generic diagnosis data.

### Packet Structure Reference

```

/* Confirmation packet */
typedef struct PNS_IF_ADD_GENERIC_DIAG_CNF_DATA_Ttag
{
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_UINT32    hDiagHandle;
    TLR_UINT16    usChannelNum;
    TLR_UINT16    usChannelProp;
    TLR_UINT16    usUserStructId;
} PNS_IF_ADD_GENERIC_DIAG_CNF_DATA_T;

```

```
typedef struct
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_ADD_GENERIC_DIAG_CNF_DATA_T tData;
} PNS_IF_ADD_GENERIC_DIAG_CNF_T;
```

## Packet Description

Structure PNS_IF_ADD_GENERIC_DIAG_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32} - 1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	22	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32} - 1$	Packet identification, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1F59	PNS_IF_ADD_GENERIC_DIAG_CNF - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_ADD_GENERIC_DIAG_CNF_DATA_T			
	ulApi	UINT32		The API of the submodule.
	ulSlot	UINT32		The Slot of the submodule.
	ulSubslot	UINT32		The Subslot of the submodule.
	hDiagHandle	UINT32		A unique handle representing this diagnostic record inside the stack. Application shall store it as it is has to be used to be able to remove the record later on.
	usChannelNum	UINT16	0x8000	Channel Number for which the diagnosis data shall be added. Supported are Manufacturer specific Channel Numbers in the range of 0x0000-0x7FFF and the Channel number for the submodule itself 0x8000.
	usChannelProp	UINT16		See Table 148: Coding of the field usChannelProp
	usUserStructId	UINT16		See Table 157: Coding of the field usUserStructId

Table 162: PNS\_IF\_ADD\_GENERIC\_DIAG\_CNF\_T - Add Generic Channel Diagnosis Confirmation

## 7.4.16 Remove Diagnosis Service

With this service the user application can remove previously added diagnosis data from a submodule. This will be reported to the IO-Controller with a “diagnosis disappears” alarm automatically if necessary.

### 7.4.16.1 Remove Diagnosis Request

Using this packet the user application can remove diagnosis data from a submodule.

#### Packet Structure Reference

```
/* Request packet */
typedef struct PNS_IF_REMOVE_DIAG_REQ_DATA_Ttag
{
    TLR_UINT32 hDiagHandle;
} PNS_IF_REMOVE_DIAG_REQ_DATA_T;

typedef struct PNS_IF_REMOVE_DIAG_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_REMOVE_DIAG_REQ_DATA_T tData;
} PNS_IF_REMOVE_DIAG_REQ_T;
```

#### Packet Description

Structure PNS_IF_REMOVE_DIAG_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	4	PNS_IF_REMOVE_DIAG_REQ_T - Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not used for requests. Set to zero.
	ulCmd	UINT32	0xFE6	<a href="#">PNS_IF_REMOVE_DIAG_REQ</a> - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_REMOVE_DIAG_T			
	hDiagHandle	UINT32		The unique diagnosis handle given to application by the stack while adding the diagnosis record.

Table 163: PNS\_IF\_REMOVE\_DIAG\_REQ\_T - Remove Diagnosis Request

### 7.4.16.2 Remove Diagnosis Confirmation

With this packet the stack informs the application about the success of removing diagnosis data.

#### Packet Structure Reference

```
/* Confirmation packet */
typedef PNS_IF_REMOVE_DIAG_REQ_T          PNS_IF_REMOVE_DIAG_CNF_T;
```

#### Packet Description

Structure PNS_IF_REMOVE_DIAG_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1FE7	PNS_IF_REMOVE_DIAG_CNF - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_REMOVE_DIAG_T			
	hDiagHandle	HANDLE		The unique diagnosis handle given to application by the stack while adding the diagnosis record.

Table 164: PNS\_IF\_REMOVE\_DIAG\_CNF\_T - Remove Diagnosis Confirmation

## 7.4.17 Get Submodule Configuration Service

With this service the user application can get the information about all previously configured submodules.

### Packet Structure Reference

```
#define    MAX_SUBMODULE_CNT    95    /* max. count of submodules */

/* request packet */
typedef TLR_EMPTY_PACKET_T    PNS_IF_GET_CONFIGURED_SUBM_REQ_T;

/* confirmation packet */
typedef struct
{
    TLR_UINT32    ulApi;
    TLR_UINT16    usSlot;
    TLR_UINT16    usSubslot;
    TLR_UINT32    ulModuleId;
    TLR_UINT32    ulSubmoduleId;
} IF_CONFIGURED_SUBM_STRUCT_T;

typedef struct
{
    TLR_UINT32                ulSubmCnt;
    PNS_IF_CONFIGURED_SUBM_STRUCT_T    atSubm[MAX_SUBMODULE_CNT];
} IF_GET_CONFIGURED_SUBM_CNF_DATA_T;

typedef struct
{
    TLR_PACKET_HEADER_T                tHead;
    PNS_IF_GET_CONFIGURED_SUBM_CNF_DATA_T    tData;
} IF_GET_CONFIGURED_SUBM_CNF_T;
```

### Packet Description

Structure PNS_IF_GET_CONFIGURED_SUBM_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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, untouched
	ulSta	UINT32		See below.

Structure <code>PNS_IF_GET_CONFIGURED_SUBM_CNF_T</code>				Type: Confirmation
Area	Variable	Type	Value / Range	Description
	<code>ulCmd</code>	UINT32	0x1FB3	<code>PNS_IF_GET_CONFIGURED_SUBM_CNF</code> - Command
	<code>ulExt</code>	UINT32	0	Extension, untouched
	<code>ulRout</code>	UINT32	x	Routing, do not touch
Data	structure <code>PNS_IF_GET_CONFIGURED_SUBM_CNF_T</code>			
	<code>ulSubmCnt</code>	UINT32	0... <code>MAX_SUBMODULE_CNT</code>	Count of configured slaves
	<code>PNS_IF_CONFIGURED_SUBM_STRUCT_T atSubm[<code>MAX_SUBMODULE_CNT</code>]</code>	Structure		Array of <code>PNS_IF_CONFIGURED_SUBM_STRUCT_T</code>

Table 165: `PNS_IF_GET_CONFIGURED_SUBM_CNF_T` - Get Submodule Configuration

The structure `PNS_IF_CONFIGURED_SUBM_STRUCT_T` has following members:

Variable	Type	Range
<code>ulApi</code>	UINT32	0 ... $2^{32} - 1$
<code>usSlot</code>	UINT16	0 ... $2^{16} - 1$
<code>usSubslot</code>	UINT16	0 ... $2^{16} - 1$
<code>ulModuleId</code>	UINT32	0 ... $2^{32} - 1$
<code>ulSubmoduleId</code>	UINT32	0 ... $2^{32} - 1$

Table 166: Elements of `PNS_IF_CONFIGURED_SUBM_STRUCT_T`

To get the current set of configured submodules send the request packet with the command `PNS_IF_GET_CONFIGURED_SUBM_REQ` to the stack, the current configuration will be read and stored in the confirmation structure.

The confirmation packet has a limited size, depending on the maximal DPM packet length. The maximal count of returned submodule configuration data is limited by `MAX_SUBMODULE_CNT` (see Packet Structure Reference). If more submodules are configured, the error code `TLR_E_FAIL` in `ulSta` will be returned.

If no submodule is configured, the error code `TLR_E_PNS_IF_NO_MODULE_CONFIGURED` in `ulSta` will be returned.

## 7.4.18 Set Submodule State Service

With this service the application has the possibility to change the submodule state. This is useful in e.g. gateway applications. Using this service the user application is able to influence the occurrence of a submodule in the ModuleDiffBlock.

**Note:**

This service is only usable for the user application if IOxS is handled by the user application as well. If IOxS is not handled by the application the stack will not allow the usage of this service.

**Note:**

This service is available starting with PROFINET IO Device stack V3.5.16

A possible use case for this service is the following scenario:

A gateway application requires some parameter records to configure the underlying network. The content of the records is expected by the underlying network to work properly. Missing or invalid parameters disallow using some (or all) of the nodes of the underlying network.

To be able to receive the parameter records the application is required to adapt the local PROFINET submodule configuration during Connection establishment by evaluating Check Indication and using the Extended Plug Submodule Request. After the parameters have been received via Write Record Indication and the application received the Parameter End Indication it configures and parameterizes the underlying network. If an error occurs in this phase, it can be indicated to the IO-Controller by setting the submodule state of the erroneous submodules to "ApplicationReady pending". In addition, the application might generate a specific diagnosis to indicate the problem on an additional level. The IOPS of the affected submodules needs to be set to "bad". Afterwards the ApplicationReady shall be sent by application using the Application Ready Request.

**Note:**

The submodule state is bound to the submodule and is only changed by the user application. Thus if the state is set to "ApplicationReady pending" and the AR is terminated and reestablished afterwards, the submodule will be reported with "ApplicationReady pending" by the protocol stack.



Setting the submodule state back to “good” differs in handling depending if the data exchange is active or not.

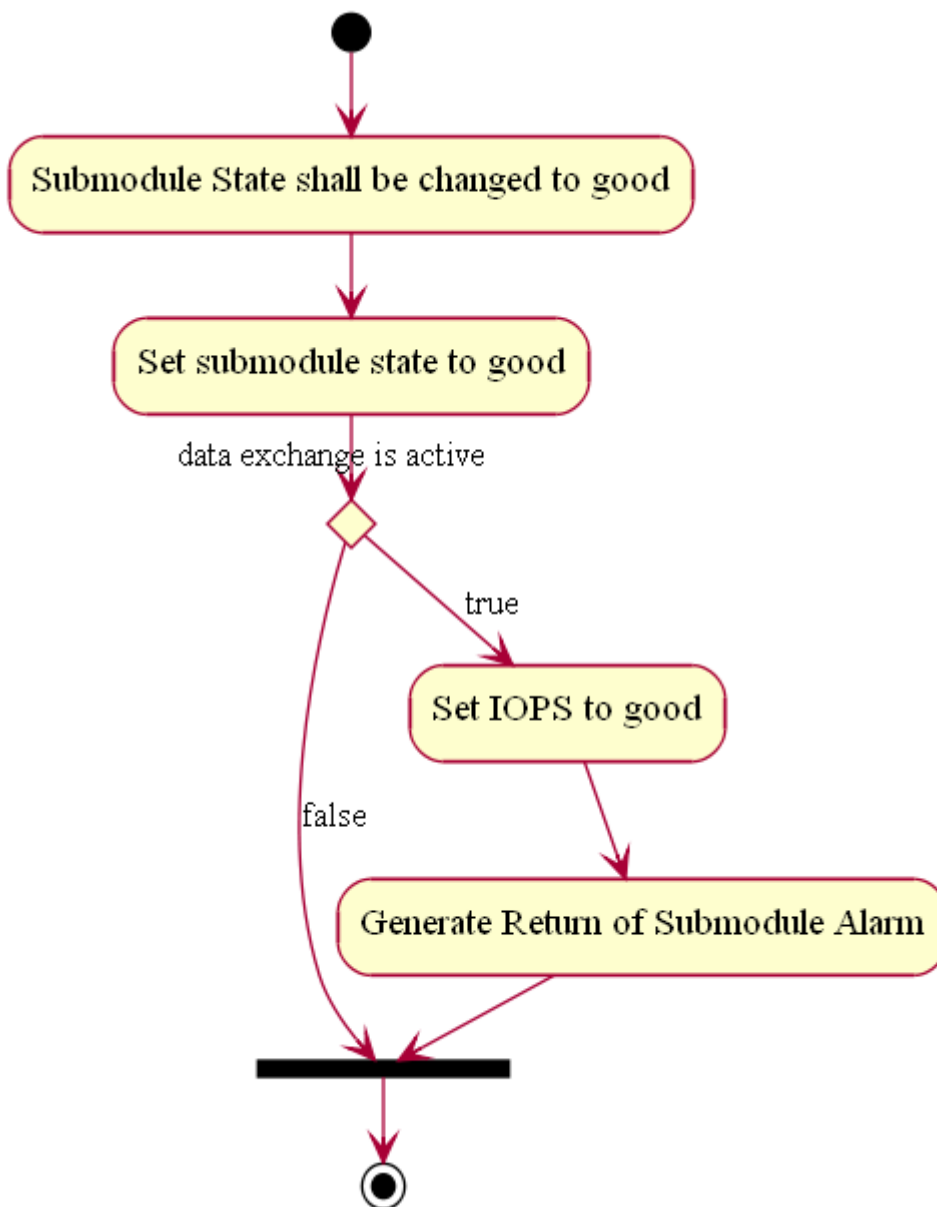


Figure 27: SetSubModuleState from bad to good

#### 7.4.18.1 Set Submodule State Request

To set the Submodule State the following packet shall be used.

Only these states are supported: “ApplicationReady pending”, “Submodule ordained locked” and “okay”.

It is possible to set the state of multiple submodules at the same time to the state “ApplicationReady pending”. However, it is not possible to set multiple submodules at the same time to the state “okay”. Setting the submodule state “okay” needs to be done in a single request for each submodule whose state is now “okay”.

The request packet has a limited size, depending on maximal DPM packet length. The maximal count of configured submodules is limited to 98 (see Packet Structure Reference). If more submodules are configured, the error code `TLR_E_FAIL` in `u/Sta` will be returned.

**Note:**

A submodule in the state “ApplicationReady pending” shall have its IOPS set to “bad”. In consequence if the submodule state changes back to “good” the IOPS needs to be changed to “good” by the application. This requires sending a ReturnOfSubmodule Alarm (see section 7.4.5.1) which must be done by the application as well.

## Packet Structure Reference

```

/* Request packet */
/* the submodule is not yet ready for data exchange */
#define PNS_IF_SET_SUBM_STATE_SUBM_APPL_READY_PENDING (2)
/* the submodule is no longer locked */
#define PNS_IF_SET_SUBM_STATE_SUBM_OKAY (0)

typedef __PACKED_PRE struct PNS_IF_SET_SUBM_STATE_SUBMBLOCK_Ttag
{
    /* the API the submodule belongs to */
    TLR_UINT32 ulApi;
    /* the slot the submodule resides */
    TLR_UINT16 usSlot;
    /* the subslot the submodule resides */
    TLR_UINT16 usSubslot;
    /* the submodule state (see above) */
    TLR_UINT16 usSubmState;
    /* the module state, reserved for future use! */
    TLR_UINT16 usModuleState;
} __PACKED_POST PNS_IF_SET_SUBM_STATE_SUBMBLOCK_T;

typedef __PACKED_PRE struct PNS_IF_SET_SUBM_STATE_DATA_REQ_Ttag
{
    /* amount of submodules contained in this packet */
    TLR_UINT32 ulSubmCnt;
    /* the first of ulSubmCnt submodule datasets */
    PNS_IF_SET_SUBM_STATE_SUBMBLOCK_T atSubm[1];
} __PACKED_POST PNS_IF_SET_SUBM_STATE_DATA_REQ_T;

typedef struct PNS_IF_SET_SUBM_STATE_REQ_Ttag
{
    TLR_PACKET_HEADER_T tHead;
    PNS_IF_SET_SUBM_STATE_DATA_REQ_T tData;
} PNS_IF_SET_SUBM_STATE_REQ_T;

```

## Packet Description

Structure PNS_IF_SET_SUBM_STATE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	16 + n	Packet data length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for request.
	ulCmd	UINT32	0x1F92	PNS_IF_SET_SUBM_STATE_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_SET_SUBM_STATE_REQ_DATA_T			
	ulSubmCnt	UINT32	1..129	The amount of submodules contained in the packet.
	ulApi	UINT32		The API of the first submodule.
	usSlot	UINT16		The Slot of the first submodule.
	usSubslot	UINT16		The subslot of the first submodule.
	usSubmState	UINT16		The new submodule state (see below)
	usModuleState	UINT16		Reserved for future use. Set to 0.

Table 167: PNS\_IF\_SET\_SUBM\_STATE\_REQ-T – Set Submodule State Request

Value	Name	Description
0	PNS_IF_SET_SUBM_STATE_SUBM_OKAY	The submodule state is okay, it is ready for valid data exchange.
2	PNS_IF_SET_SUBM_STATE_SUBM_APPL_READY_PENDING	The submodule is not ready for valid data exchange.

Table 168: Possible Values of usSubmState

### 7.4.18.2 Set Submodule State Confirmation

The stack will return this packet to the application.

#### Packet Structure Reference

```
/* Confirmation packet */
typedef TLR_EMPTY_PACKET_T    PNS_IF_SET_SUBM_STATE_CNF_T;
```

**Packet Description**

Structure PNS_IF_SET_SUBM_STATE_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	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 below.
	ulCmd	UINT32	0x1F93	PNS_IF_SET_SUBM_STATE_CNF-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 169: PNS\_IF\_SET\_SUBM\_STATE\_CNF-T - Set Submodule State Confirmation

## 7.4.19 Get Parameter Service

This service can be used by the application to retrieve runtime parameters from the protocol stack.



### Note:

In this service the confirmation packet is larger than the request packet. If the application is directly programming the stack's AP-Task Queue, the application has to provide a buffer which is large enough to hold the confirmation data.



### Note:

This service is available starting with PROFINET IO Device Stack V3.5.34.0. The parameter type PNS\_IF\_PARAM\_SUBMODULE\_CYCLE is available since Stack version V3.5.47.0



### Note:

The submodule reference parameters are ignored by the Protocol Stack for global parameters.

### 7.4.19.1 Get Parameter Service

#### Packet Structure Reference

```
enum PNS_IF_PARAM_Etag
{
    PNS_IF_PARAM_MRP = 1,
    PNS_IF_PARAM_SUBMODULE_CYCLE = 2,
};

typedef enum PNS_IF_PARAM_Etag PNS_IF_PARAM_E;

typedef struct PNS_IF_PARAM_COMMON_Ttag PNS_IF_PARAM_COMMON_T;

struct PNS_IF_PARAM_COMMON_Ttag
{
    TLR_UINT16    usPrmType;
    TLR_UINT16    usPadding;
    TLR_UINT32    ulApi;
    TLR_UINT16    usSlot;
    TLR_UINT16    usSubslot;
};

typedef struct PNS_IF_GET_PARAM_REQ_Ttag PNS_IF_GET_PARAM_REQ_T;

struct PNS_IF_GET_PARAM_REQ_Ttag
{
    TLR_PACKET_HEADER_T    tHead;
    PNS_IF_PARAM_COMMON_T    tData;
};
```

#### Packet Description

Structure PNS_IF_GET_PARAM_REQ_T				Type: Request
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue

Structure <b>PNS_IF_GET_PARAM_REQ_T</b>				Type: Request
Area	Variable	Type	Value / Range	Description
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32} - 1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	2 or 12	Packet data length in bytes. 2 for generic parameters, 12 for submodule specific parameters
	ulId	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not used for requests. Set to zero.
	ulCmd	UINT32	0x1F64	PNS_IF_GET_PARAMETER_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	0	Routing, set to zero
Data	<b>structure <b>PNS_IF_PARAM_COMMON_T</b></b>			
	usPrmType	UINT16	1 or 2	The parameter to retrieve. See Table 171
	usPadding	UINT16	0	Padding, Set to Zero for future compatibility.
	ulApi	UINT32		Referenced Submodule's Api
	usSlot	UINT16		Referenced Submodule's Slot
	usSubslot	UINT16		Referenced Submodule's Subslot

Table 170: **PNS\_IF\_GET\_PARAMETER\_REQ\_T** - Get Parameter Request

Symbolic Name	Numerical Value	Description
PNS_IF_PARAM_MRP	1	Media Redundancy Parameters (Global Parameter)
PNS_IF_PARAM_SUBMODULE_CYCLE	2	Submodule Process Data Cycle (Output)

Table 171: **PNS\_IF\_PARAM\_E** - Valid parameter options

### 7.4.19.2 Get Parameter Confirmation

This packet will be returned by the stack as response to the Get Parameter Request. It contains the requested information on success.

#### Packet Structure Reference

```
typedef struct PNS_IF_PARAM_COMMON_Ttag PNS_IF_PARAM_COMMON_T;

struct PNS_IF_PARAM_COMMON_Ttag
{
    uint16_t      usPrmType;
};

typedef struct PNS_IF_UUID_Ttag
{
    /** 00:04, uuid data 1, 32Bit */
    TLR_UINT32    ulData1;
    /** 04:02, uuid data 2, 16Bit */
    TLR_UINT16    usData2;
    /** 06:02, uuid data 3, 16Bit */
    TLR_UINT16    usData3;
    /** 08:08, uuid data 4, 8x8Bit */
    TLR_UINT8     abData4[8];
} PNS_IF_UUID_T;

struct PNS_IF_PARAM_MRP_Ttag
{
    uint16_t      usPrmType;
    uint8_t       bState;
    uint8_t       bRole;
    PNS_IF_UUID_T tUUID;
    uint8_t       szDomainName[240];
};

typedef struct PNS_IF_PARAM_SUBMODULE_CYCLE_Ttag PNS_IF_PARAM_SUBMODULE_CYCLE_T;

struct PNS_IF_PARAM_SUBMODULE_CYCLE_Ttag
{
    TLR_UINT16    usPrmType;
    TLR_UINT16    usPadding;
    TLR_UINT32    ulApi;
    TLR_UINT16    usSlot;
    TLR_UINT16    usSubslot;
    TLR_UINT32    ulUpdateInterval;
    TLR_UINT16    usSendClock;
    TLR_UINT16    usReductionRatio;
    TLR_UINT16    usDataHoldFactor;
};

typedef union PNS_IF_PARAM_Ttag PNS_IF_PARAM_T;

union PNS_IF_PARAM_Ttag
{
    PNS_IF_PARAM_COMMON_T tCommon;
    PNS_IF_PARAM_MRP_T    tMrp;
    PNS_IF_PARAM_SUBMODULE_CYCLE_T tSubmoduleCycle;
};

typedef struct PNS_IF_GET_PARAM_CNF_Ttag PNS_IF_GET_PARAM_CNF_T;

struct PNS_IF_GET_PARAM_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;
    PNS_IF_PARAM_T      tData;
};
```

## Packet Description

Structure PNS_IF_GET_PARAM_CNF_T				Type: Confirmation
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of PNS_IF task process queue
	ulSrc	UINT32		Source Queue-Handle of application task process queue
	ulDestId	UINT32	0	Destination End Point Identifier. Not in use, set to zero for compatibility reasons.
	ulSrcId	UINT32	0 ... $2^{32} - 1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	> 2	Packet data length in bytes. Depends on requested information.
	ulId	UINT32	0 ... $2^{32} - 1$	Packet identification, untouched
	ulSta	UINT32		See below.
	ulCmd	UINT32	0x1F65	PNS_IF_GET_PARAM_CNF - Command
	ulExt	UINT32	0	Extension, ignore
	ulRout	UINT32	x	Routing, ignore
Data	union PNS_IF_PARAM_T			
	tCommon	STRUCT		
	usPrmType	UINT16		Parameter Option Value
	tMrp	STRUCT		
	usPrmType	UINT16	1	Mrp Parameter Option Value
	bState	UINT8	0-3	Mrp State. See Table 173
	bRole	UINT8	0-4	Mrp Role of the Device. See Table 174
	tUUID	UUID		Mrp UUID
	szDomainName	UINT8[240]		Mrp Domain Name. String Length is Packet Length minus 20
	tSubmoduleCycle	STRUCT		
	usPrmType	UINT16	2	Submodule Cycle Option Value
	usPadding	UINT16		Ignore for future compatibility
	ulApi	UINT32		Api of the submodule
	usSlot	UINT16		Slot of the submodule
	usSubslot	UINT16		Subslot of the submodule
	ulUpdateInterval	UINT32		Output process data update interval in nanoseconds
	usSendClock	UINT16		PROFINET Send Clock
	usReductionRatio	UINT16		PROFINET Reduction Ratio



Structure <code>PNS_IF_GET_PARAM_CNF_T</code>				Type: Confirmation
Area	Variable	Type	Value / Range	Description
	<code>usDataHoldFactor</code>	UINT16		Output process data watchdog factor: The output process data timeout is <code>usDataHoldFactor</code> x <code>ulUpdateInterval</code> .

Table 172: `PNS_IF_GET_PARAM_CNF_T` - Get Diagnosis Confirmation

The following values are valid for Mrp Parameter:

Symbolic Name	Numerical Value	Description
<code>MRP_STATE_DISABLED</code>	0	MRP was disabled by the Protocol Stack
<code>MRP_STATE_ENABLED_DEFAULT</code>	1	MRP was enabled by the Protocol Stack using default MRP parameters. (No parameter was written by controller / Factory Reset)
<code>MRP_STATE_ENABLED_PRM</code>	2	MRP was enabled on behalf of Controller. (The corresponding parameter was set with MRP enabled)
<code>MRP_STATE_DISABLED_PRM</code>	3	MRP was disabled on behalf of Controller. (The corresponding parameter was et with MRP disabled)

Table 173: `PNS_IF_PARAM_MRP_STATE_E` - Valid values for MRP state.

Symbolic Name	Numerical Value	Description
<code>MRP_ROLE_NONE</code>	0	MRP is disabled
<code>MRP_ROLE_MRP_CLIENT</code>	1	MRP is configured for Client Mode.

Table 174: `PNS_IF_PARAM_MRP_ROLE_E` - Valid values for MRP Role

## 8 Special Topics

### 8.1 Behavior under special situations

This section describes the behavior of the stack under some special situations.

#### 8.1.1 Sequence of configuration evaluation

Three ways of configuration of the PROFINET IO-Device stack are available:

- SYCON.net Database
- netX Configuration Tool (iniBatch Database)
- *Set Configuration Request* packet

Only one type of configuration can be active at a certain time.

These are evaluated at start-up in the following order:

- SYCON.net Database
- iniBatch Database (via netX Configuration Tool)
- Set Configuration Request packet

After a Restart the stack will first search for the SYCON.net Database files (`config.nxd` and `nwid.nxd`). If these are found, all other configuration methods will not be accepted. If no SYCON.net Database exists, but an iniBatch Database exists, its configuration will be used and configuration packets will be not accepted.

If no database is found at all the stack remains unconfigured until the reception of the first configuration packet.

#### 8.1.2 Configuration Lock

If the configuration of the stack is locked as described in Dual Port Memory Interface Manual (reference [3]), the following behavior is implemented in the stack:

- New Set Configuration Requests are not accepted
- Configuration Reload / Channel Init will be rejected

However, PROFINET specific services affecting the configuration are still working as defined by PROFINET specification. This includes setting IP parameters and NameOfStation by means of DCP and writing PDEV-Parameters using records.

#### 8.1.3 Setting Parameters by means of DCP

The PROFINET specification defines the Discovery and basic Configuration Protocol (DCP) to change some basic PROFINET Device Parameters over the bus.

A PROFINET IO-Controller, a PROFINET IO-Supervisor or an Engineering System can easily change the IP parameters or the NameOfStation of a PROFINET Device at any time. These new parameters can either be marked to be used temporarily or marked to be stored remanent.

The receipt of such a request is indicated to the user with the Save Station Name Indication, the Save IP Address Indication or the Reset Factory Settings Indication.

The stack will adapt to these new parameters at runtime. Anyway, as the NameOfStation and the IP parameters are part of the Stacks configuration, the user application must handle these parameters properly, if the stack is configured using the Set Configuration Service:

Configuration Method	Parameter Handling
SYCON.net Database	Parameters are stored by stack into non volatile memory.
netX Configuration Tool (iniBatch Database)	Parameters are stored by stack into non volatile memory.
Set Configuration Request packet	User Application shall store NameOfStation and IP parameters into non-volatile memory. Set Configuration Service fields shall be initialized from non-volatile memory on startup.

Table 175: Handling of basic parameters

## 8.2 Multiple ARs

### 8.2.1 Ownership

As already written above, the PROFINET Device stack supports multiple ARs at the same time. This allows different controllers to access different output submodules and same or different input submodules of one device at the same time. In term of the PROFINET specification this is called Shared Device and Shared Input. Of course, a Shared Output is not defined as it makes no sense to access an output submodule from multiple controllers at the same time. In order to solve the problem of which controller gets the right to write the outputs of a submodule and perform parameterization, the PROFINET specification v2.3 defines a new state machine, the Ownership-State machine (OWNSM). Depending on the submodule specific Ownership, the following access rights are defined:

- **AR is the owner of the submodule:** The submodule puts its input process data to this AR, takes the output process data from this AR and accepts parameter read and writes on this AR.
- **AR is not the owner of the submodule:** The submodule puts its input process data to this AR (Shared Input). The submodule ignores output process data from this AR and rejects parameter reads and writes on this AR.

The ownership itself is assigned according to the rule “*First Come First Serve*” e.g. the first AR will get the ownership of all requested submodules, the second AR will get the ownership of requested submodules not already owned by the first AR. If the first AR is disconnected, the second AR will get the ownership of all requested submodules not yet assigned to this AR (This is called a Release). There is only one exception from this rule: If a Supervisor AR connects, it takes over the ownership of all requested submodules if the owning ARs allow Ownership Takeover (Depends on AR Properties in Connect.req of the AR). Furthermore, if a Supervisor AR is active and was not allowed to take over the ownership of a submodule and the owning AR disconnects, the Supervisor AR will always get the ownership of the submodule (Supervisor AR has higher Priority than “First Come First Serve” Principle).



#### Note:

As already written above, the multi AR feature renders the definition of a “communicating state” ad absurdum. The application usually has no information about which of the submodules is used by any AR. Therefore it is strongly recommended to cyclically update the process data from/to the physical submodules, regardless of any communication state. If the application insists on knowing if a submodule is in data exchange or not, the IOxS status of the submodule should be examined.

## 8.2.2 Possibilities and Limitations for the Feature Shared Device

Starting with PROFINET IO Device version V3.9.0.0, the feature Shared Device is extended to up to 8 IO ARs for netX 51 and netX 100. This feature extension is not supported for netX 50.

---

**Note:** When using PROFINET IRT it is not possible to use Shared Device for IRT.

---

The netX-based PROFINET IO Device only supports exactly 1 IRT IO AR. Additional RT IO ARs are possible.

### Reachable Cycle Time Depending on the Amount of IO ARs

To give the user of the PROFINET IO Device protocol stack a better understanding of reachable cycle times for this feature, extensions of this chapter will give some performance indicators.

As the variety of combinations is very large obviously this chapter can not show every possible use case. Thus only a small subset of possibilities is shown here.

This does neither mean that a not shown combination is not supported nor that it is supported.

Configurations to be used have to be tested by the one combining the netX PROFINET IO Device with his application.

---

**Note:** As each netX chip type has different calculation power this chapter differs the netX chip type.

---

For the following tables it is assumed that two 20 byte input submodules and two 20 byte output submodules are used per IO AR.

Please note, that the amount of submodules influences the reachable cycle time.

netX	Amount of ARs	Smalles possible Cycle Time
netX 100	1-2	1 ms
	3-5	2 ms
	6-8	4 ms
netX 51	1-2	1 ms
	3-4	2 ms
	5-8	4 ms

Table 176: Smalles possible Cycle Time depending using multiple ARs

### GSDML, startup Parameterization and Certification

In any case to pass PROFINET certification it is required that the supported amount of ARs documented in GSDML file matches to the capabilities of the product. Thus the value in GSDML ("NumberOfAR") needs to match the configured amount of ARs of the PROFINET protocol stack.

Loadable firmware offers a tag list entry to modify the amount of ARs supported.

Linkable Object Module offers a stack startup parameter (ulMaxAr).

### 8.2.3 Ethernet MAC Addresses

The PROFINET protocol stack requires up to three MAC addresses for operation: One Interface MAC Address and for each Ethernet port a port MAC address. The MAC addresses are configured via different paths:

- The firmware/protocol stack is provided a single MAC address from the Second Stage Boot Loader on startup of the firmware. This MAC address is used as interface MAC address. The remaining MAC addresses are computed from this MAC address.
- A security memory chip is attached to the netX. The firmware/protocol stack reads the interface MAC address from the security memory. The remaining MAC addresses are computed from this MAC address.
- A Flash label is found in the Flash memory chip attached to the netX (netX51 only). The firmware/protocol stack reads the interface MAC address from the Flash memory. The remaining MAC addresses are computed from this MAC address.
- Finally, before configuring the protocol stack the application can assign custom MAC addresses. This step is mandatory if the firmware was not able to determine the device's MAC address using the methods above. This step can be performed optionally after startup of the firmware to override any other MAC address settings. In order to assign a MAC address the rcX Set\_MAC\_Address\_Service and afterwards the PROFINET\_Set\_Port\_MAC\_Address\_Service must be used in exactly this order.

Figure 28 shows the MAC address determination bootup sequence.

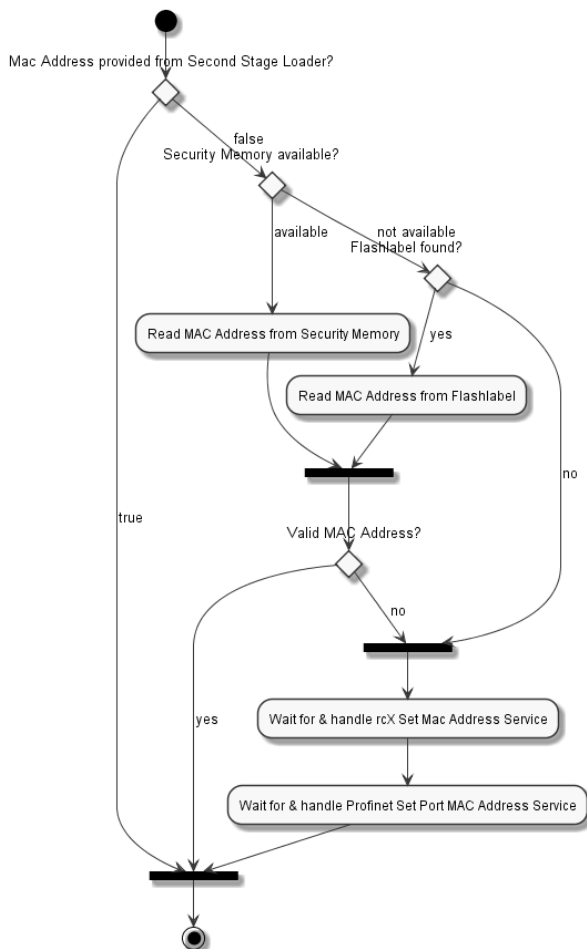


Figure 28: Sequence of MAC Address determination

## 8.3 Usage of Linkable Object Module



### Note:

This section only applies if the stack is used as Linkable Object Module. If the stack is used as Loadable Firmware this section can be ignored.

For more details of the configuration see the example provided on NXLOM CD.

If the stack is used as Linkable Object Module, the user has to create its own configuration file (which among others contains task start-up parameters and hardware resource declarations). The PROFINET stack itself is started by invoking the function `PNS_StackInit()` from the user application.



### Note:

Since PROFINET Stack Version 3.5.0.0 the PROFINET Stack is started by calling the function `PNS_StackInit()`. This function automatically initializes required resources and launches the affected tasks.

### 8.3.1 Config.c

The *config.c* file contains among others the hardware resource declarations and the static task list.

#### 8.3.1.1 Hardware Resources

Besides the standard rcX resources and the user application resources, the following hardware resources should be declared. These are used by the PROFINET Device stack.

##### For netX 100 or netX 500

Resource	Peripheral Type	Peripheral Subtype	Instance
Ethernet Physical Interface	RX_PERIPHERAL_TYPE_PHY		0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XMACRPU	0, 1 and 3*
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XMACTPU	0, 1 and 3*
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XPEC	0, 1 and 3
Fifo units	RX_PERIPHERAL_TYPE_FIFOCHANNEL		0, 1 and 3
Ethernet Driver	RX_PERIPHERAL_TYPE_EDD		0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msync0	0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msync1	1
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msync3	3
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com0	0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com1	1
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com3	3
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_sys_time_ns	0

Table 177: Hardware resources used by the PROFINET Device stack for netX 100/500

See also section *Disable XMAC3* on page 263.

**For netX 50**

Resource	Peripheral Type	Peripheral Subtype	Instance
Ethernet Physical Interface	RX_PERIPHERAL_TYPE_PHY		0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XMACRPU	0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XMACTPU	0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XPEC	0 and 1
Fifo units	RX_PERIPHERAL_TYPE_FIFOCHANNEL		0 and 1
Ethernet Driver	RX_PERIPHERAL_TYPE_EDD		0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msinc0	0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msinc1	1
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com0	0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com1	1
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_systime_ns	0

Table 178: Hardware resources used by the PROFINET Device stack for netX 50

**For netX 51**

Resource	Peripheral Type	Peripheral Subtype	Instance
Ethernet Physical Interface	RX_PERIPHERAL_TYPE_PHY		0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XMACRPU	0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XMACTPU	0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_RPEC	0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_TPEC	0 and 1
Fifo units	RX_PERIPHERAL_TYPE_FIFOCHANNEL		0 and 1
Ethernet Driver	RX_PERIPHERAL_TYPE_EDD		0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msinc0	0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msinc1	1
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com0	0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com1	1
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NX51_vic_irq_status_systime_ns	0

Table 179: Hardware resources used by the PROFINET Device stack for netX 51

**8.3.1.2 Disable XMAC3**

Using netX100 or netX500 it is possible to exclude the XMACRPU3 and XMACTPU3 units from the PROFINET Device stack hardware list (Table 177), but XPEC3 unit is still in use. XMACRPU3 and XMACTPU3 can be used for another purpose.

Without the XMACRPU3 and XMACTPU3 units the PROFINET stack has the following restrictions:

- disabled generation of the external SYNC signals on Sync0 and Sync1 pins that is described in reference [6],
- it is not possible to build synchronous application described in reference [6] because of disabling of synchronization interface,
- certification for “Conformance Class C” is not possible because there is no external SYNC signal to check the synchronization (no sync pin available), but IRT mode is still available.

To exclude the XMACRPU3 and XMACTPU3 units from usage of the PROFINET Device stack do not define their names in the EDD-parameter list RX\_EDD\_PARAMETERS\_T, just initialize to NULL:

```

STATIC RX_EDD_PARAMETERS_T g_atEddParam[] =
{
    ...
    { RX_EDD_PARAM_XPEC_NAME,      "PNS_XPEC",      0 }, /* XPEC0      */
    { RX_EDD_PARAM_XMAC_RPU_NAME,  "PNS_XMACRPU",  0 }, /* XMAC0_RPU */
    { RX_EDD_PARAM_XMAC_TPU_NAME,  "PNS_XMACTPU", 0 }, /* XMAC0_TPU */

    { RX_EDD_PARAM_XPEC_NAME,      "PNS_XPEC",      1 }, /* XPEC1      */
    { RX_EDD_PARAM_XMAC1_RPU_NAME, "PNS_XMACRPU",  1 }, /* XMAC1_TPU */
    { RX_EDD_PARAM_XMAC1_TPU_NAME, "PNS_XMACTPU",  1 }, /* XMAC1_RPU */

    { RX_EDD_PARAM_XPEC3_NAME,      "PNS_XPEC",      3 }, /* XPEC3      */
    { RX_EDD_PARAM_XMAC3_RPU_NAME,  NULL,             3 }, /* don't use XMAC3_RPU for EDD */
    { RX_EDD_PARAM_XMAC3_TPU_NAME,  NULL,             3 }, /* don't use XMAC3_TPU for EDD */
    ...
};

STATIC RX_EDD_SET_T g_atEdd[] = {
    { {PNS_EDD_IDENTIFY_NAME, RX_PERIPHERAL_TYPE_EDD, 0}, /* Ethernet Device Driver */
      0, /* use Edd0 */
      "PROFINET Switch-Cut-Through", /* NIC name */
      RX_EDD_MODE_DEFAULT,
      FALSE,
      g_atEddParam,
      &trEddHalPND
    } };

```

### 8.3.1.3 Systeime Unit

The PROFINET Device stack uses the SysTime unit (namely the SysTime-compare interrupt SRT\_NETX\_VIC\_IRQ\_STAT\_systime\_ns) for internal RT scheduling. The stack reconfigures the SysTime unit differently compared to default rcX, therefore the rcX function delivers following information:

```

SYSTIME_TIMESTAMP_T timestamp;
Drv_SysTimeGetTime(&timestamp);

Timestamp.ulTimeNs - lower 4 bytes of the system time in step of 10 ns
Timestamp.ulTimeS  - higher 4 bytes of the system time in step of 10 ns

```

So the "timestamp" is a 64 bit value of system time in step of 10 ns



### 8.3.1.4 Static Task List

Since PROFINET Stack version 3.5, the static task list should contain the Timer Task, The TCP/IP Task and the User Application Tasks:

```
/* Stack-sizes for the static tasks */
#define TSK_STACK_SIZE_TLR_TIMER      512    /* Memory assignement for the TimerTask */
#define TSK_STACK_SIZE_TCP_TASK      1024    /* Stack Size in multiples of UINTs */
#define TSK_STACK_SIZE_PCK_DEMO      2048    /* Stack Size in multiples of UINTs */

/* Stack for the "TLR TIMER" task /
STATIC UINT auTskStack_Tlr_Timer[TSK_STACK_SIZE_TLR_TIMER];

/* Stack for the "TCP_UDP" task */
STATIC UINT auTskStack_Tcp_Task[TSK_STACK_SIZE_TCP_TASK];

/* Stack for the "Packet API Demo" task */
STATIC UINT auTskStack_Pck_API_Demo[TSK_STACK_SIZE_PCK_DEMO];

RX_STATIC_TASK_T g_atStaticTasks[] = {
    {"TlrTimer",                                /* Identification */
     TSK_PRIO_12, TSK_TOK_12,                  /* Priority Token ID */
     0,                                         /* Instance 0 */
     &auTskStack_Tlr_Timer[0],                 /* Pointer to Stack */
     TSK_STACK_SIZE_TLR_TIMER,                /* Size of Task Stack */
     0,
     RX_TASK_AUTO_START,                      /* Start task automatically */
     (VOID*)TaskEnter_TlrTimer,               /* Task main() Function */
     (VOID*)TaskExit_TlrTimer,               /* Task leave callback */
     (UINT32)&g_tTlrTimerPrm,                 /* Startup Parameter */
     {0,0,0,0,0,0,0,0},                      /* Reserved */
    },
    {"TCP_UDP",                                /* Identification */
     TSK_PRIO_30, TSK_TOK_30,                  /* Priority and Token ID */
     0,                                         /* Instance 0 */
     &auTskStack_Tcp_Task[0],                 /* Pointer to Stack */
     TSK_STACK_SIZE_TCP_TASK,                /* Size of Task Stack */
     0,
     RX_TASK_AUTO_START,                      /* Start task automatically */
     (void FAR*)TaskEnter_TcpipTcpTask,       /* Task main() Function */
     NULL,                                    /* Task leave callback */
     (UINT32)&g_tTcpIpPrm,                    /* Startup Parameter */
     {0,0,0,0,0,0,0,0},                      /* Reserved */
    },
    /* User AP-Task for acyclical and low-priority services */
    {"APP_LOW",                                /* Identification */
     TSK_PRIO_40, TSK_TOK_40,                  /* Priority and Token ID */
     0,                                         /* Instance to 0 */
     & auTskStack_Pck_API_Demo[0],            /* Pointer to Stack */
     TSK_STACK_SIZE_PCK_DEMO,                /* Size of Task Stack */
     0,
     RX_TASK_AUTO_START,                      /* Start task automatically */
     (void FAR*)TaskEnter_UserAp,            /* Task main() Function */
     NULL,                                    /* Task leave callback */
     0,                                       /* Startup Parameter */
     {0,0,0,0,0,0,0,0},                      /* Reserved */
    },
};
```

The startup parameters of the Timer Task and TCP/IP Task shall be set as follows (this depends on the used version as well):

```
STATIC CONST TLR_TIMER_STARTUPPARAMETER_T g_tTlrTimerPrm =
{
    TLR_TASK_TIMER,                          /* ulTaskIdentifier */
    1,                                        /* ulParamVersion */
    160,                                     /* application timer resources */
    2,                                       /* interrupt timer resources */
    2                                        /* retry packet resources */
};
```

```

STATIC CONST TCPIP_TCP_TASK_STARTUPPARAMETER_T g_tTcpIpPrm =
{
    TLR_TASK_TCPUDP,                /* ulTaskIdentifier          */
    TCPIP_STARTUPPARAMETER_VERSION_6, /* ulParamVersion           */
    TCPIP_SRT_QUE_ELEM_CNT_AP_DEFAULT, /* ulQueueElemCntAp         */
    TCPIP_SRT_POOL_ELEM_CNT_DEFAULT, /* ulPoolElemCnt            */
    TCPIP_SRT_FLAG_FAST_START,        /* ulStartFlags              */
    TCPIP_SRT_TCP_CYCLE_EVENT_DEFAULT, /* ulTcpCycleEvent          */
    TCPIP_SRT_QUE_FREE_ELEM_CNT_DEFAULT, /* ulQueueFreeElemCnt      */
    32,                               /* ulSocketMaxCnt           */
    TCPIP_SRT_ARP_CACHE_SIZE_DEFAULT, /* ulArpCacheSize           */
    PNS_EDD_IDENTIFY_NAME,            /* pszEddName                */
    TCPIP_SRT_EDD_QUE_POOL_ELEM_CNT_DEFAULT, /* ulEddQueuePoolElemCnt  */
    TCPIP_SRT_EDD_OUT_BUF_MAX_CNT_DEFAULT, /* ulEddOutBufMaxCnt       */
    NULL,                             /* Pointer to EthIntf Config */
    60,                               /* ARP cache timeout in seconds */
    NULL,
    .ulNetLoadMaxFramesPerTick = TCPIP_SRT_NETLOAD_MAXFRAMESPERTICK_DEFAULT,
    .ulNetLoadMaxPendingARP = TCPIP_SRT_NETLOAD_MAXPENDING_ARP_DEFAULT,
    .ulNetLoadMaxPendingMCastARP = TCPIP_SRT_NETLOAD_MAXPENDING_MCASTARP_DEFAULT,
    .ulNetLoadMaxPendingIP = TCPIP_SRT_NETLOAD_MAXPENDING_IP_DEFAULT,
    .ulNetLoadMaxPendingMCastIP = TCPIP_SRT_NETLOAD_MAXPENDING_MCASTIP_DEFAULT,
};

```

### 8.3.2 PNS\_StackInit()

In order to start the PROFINET Device stack, the user application shall invoke the function `PNS_StackInit()`. The function will setup the resources needed by the stack and launch all required tasks. The function has to be provided with parameters describing the hardware resources:

```
/* PROFINET Device stack - configuration parameters */
PROFINET_IODEVICE_STARTUPPARAMETER_T tPrm;

/* PROFINET Device stack - configured task resources */
PROFINET_IODEVICE_TASK_RESOURCES_T* ptRsc;

eRslt= PNS_StackInit(&tPNSParam, &ptPNSRsc);
```

The configuration structure `PROFINET_IODEVICE_STARTUPPARAMETER_T` of the PROFINET Device stack has following fields:

- **ulParamVersion**

This value needs to be adapted to the stack version used, different versions of parameters will not be accepted by the StackInit function.

- **ullInstance**

Used as a handle variable for the interrupts described above and EDD.

- **ulFlags**

Defines the following options:

Option	Value	Description
PROFINET_IODEVICE_STARTUP_FLAG_USE_IRT	0x0001	Obsolete, set to 0
PROFINET_IODEVICE_STARTUP_FLAG_SYNC_APPL_WITH_FIQ	0x0002	Obsolete, set to 0
PROFINET_IODEVICE_STARTUP_FLAG_SYNC_APPL_WITH_IRQ	0x0004	Obsolete, set to 0
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_HW	0x0008	Obsolete, use values 0x0100 or/and 0x0200 instead
PROFINET_IODEVICE_STARTUP_FLAG_DISALLOW_IO_SUPERVOR_AR	0x0010	Shall be set if IO-Supervisor AR shall not be supported by the stack
PROFINET_IODEVICE_STARTUP_FLAG_USE_LINKLOCAL_IP	0x0020	Set this flag to configure link local IP address (generated from MAC address) instead of zero IP address
---	0x0040	Reserved
---	0x0080	Reserved
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0	0x0100	Shall be set, if hardware has fiber optic on port1
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1	0x0200	Shall be set, if hardware has fiber optic on port2
PROFINET_IODEVICE_STARTUP_FLAG_IM5_SUPPORTED	0x0400	Shall be set to support I&M5 (used for Hilscher's internal purposes)
PROFINET_IODEVICE_STARTUP_FLAG_FODMI_TASK_DISABLED	0x0800	Shall be set to disable the fiber optic service task (FODMI)
PROFINET_IODEVICE_STARTUP_FLAG_DISALLOW_SRAR	0x1000	Shall be set if system redundancy AR shall not be supported by the stack
PROFINET_IODEVICE_STARTUP_FLAG_DISALLOW_IRT	0x2000	Shall be set if RT Class3 (IRT) AR shall not be supported by the stack

■ **ulMinDeviceInterval**

This is a GSDML-file parameter representing the minimum interval time for sending cyclic IO data.

■ **ulMaxAr**

This parameter represents the maximal count of controllers can be connected to the device. See section 8.2 for details.

■ **pszEddName**

The name of the EDD shall be specified in `pszEddName`. This name is used by other tasks, too, and shall always have the same value. The default value is "ETHERNET".

■ **pszPhy0Name**

■ **pszPhy1Name**

Obsolete parameters, not used any more.

■ **pszIrqRTSchedName**

Specify the name of the RT Scheduler Interrupt here. The interrupt with the corresponding name shall be defined in the configuration file, it is the SysTime-compare interrupt `SRT_NETX_VIC_IRQ_STAT_systime_ns`.

■ **pfnFodmiTaskInitFn**

Pointer to the startup function of the fiber optic service task. Shall be initialized to "FODMITask\_Init" if flags

`PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0` or/and  
`PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1` is/are set.

■ **pvFodmiTaskInitPrm**

Pointer to the initialization parameters for fiber optic service task. Shall be initialized with address of the startup parameters for FODMI-Task `FODMI_TASK_INIT_PRM_T` if flags  
`PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0` or/and  
`PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1` is/are set.

■ **tTaskCmdev, tTaskRtc, tTaskRta, tTaskLldp, tTaskMibDB, tTaskPnsif, tTaskRpc, tTaskSnmp, tTaskFodmi**

Configurations of the priorities for corresponding tasks

■ **patPNSModules**

Reserved, set to NULL.

■ **usMauTypePort0, usMauTypePort1**

MAU type of Fiber Optic device, shall be used if hardware has fiber optic (flags  
`PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0` or/and  
`PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1` set). If set to "0" the default MAU type "100BasePXF" is used.

■ **structure tSNMPdatabase**

Used to add custom MIB variables to the SNMP-database. It is a rarely used feature for customer applications inasmuch as the PROFINET stack takes care about all required SNMP-variables. For more information see reference [7].

**.pvCustomGroups**

Pointer to the array of custom MIB groups. It is the same parameter as  
`MIB_DATABASE_STARTUPPARAMETER_V3_T::ptCustomGroups` for SNMP-

database task. Set to "NULL" if an application does not support custom SNMP variables

**.ulCustomGroupsCount**

Number of groups in the array of custom MIB groups. It is the same parameter as `MIB_DATABASE_STARTUPPARAMETER_V3_T::ulCustomGroupsCount` for SNMP-database task. Set to "0" if an application does not support custom SNMP variables

■ **ulActivePROFINETPorts**

Specify the number of Ethernet Ports here. Typically, this value is 2 for all netX based products. Definition of `ulActivePROFINETPorts = 1` means a single port PROFINET device: logical an Ethernet port 1 is the active PROFINET port ; information about port 2 will be not delivered via common services like SNMP and RPC (it is still usable for communication, physical is not deactivated).

Common implementation of a PROFINET IO-device with 2 Ethernet shall set this parameter to 2.

■ **ulActiveLLDPPorts**

This parameter defines amount of active LLDP Ethernet ports. Parameterization `ulActiveLLDPPorts = 1` forces an activity of LLDP service only on the physical Ethernet port 1.

The combination of parameter

`"ulActivePROFINETPorts = 1"` and `"ulActiveLLDPPorts = 2"` is allowed.

## Configuration Example (not valid for a fiber optic device):

```

PROFINET_IODEVICE_STARTUPPARAMETER_T    tPNSParam /* PNS Stack Parameters */
PROFINET_IODEVICE_TASK_RESOURCES_T      *ptPNSRsc /* Pointer to PNS Resources */
TLR_HANDLE hQuePnsIf;                    /* PNS-IF Task Queue */

/* common parameters */
tPnsParam.ulParamVersion      = PROFINET_IODEVICE_STARTUPPARAMETER_VERSION_V6
tPNSParam.ulInstance          = 0;
tPNSParam.ulFlags              = 0;
tPNSParam.ulMinDeviceInterval = 32; /* 1ms */
tPNSParam.ulMaxAr              = 2;
tPNSParam.pszEddName           = "ETHERNET";
tPNSParam.pszPhy0Name          = NULL;
tPNSParam.pszPhy1Name          = NULL;
tPNSParam.pszIrqRTSchedName    = "PNS_RT_IRQ";
tPNSParam.pfnFodmiTaskInitFn   = NULL;
tPNSParam.pvFodmiTaskInitPrm  = NULL;

/* priorities of all tasks, started by PROFINET */
tPNSParam.tTaskCmdev = {NULL, "PNIO_CMDEV",   TSK_PRIO_18},
tPNSParam.tTaskRtc   = {NULL, "PNIO_RTC",     TSK_PRIO_4},
tPNSParam.tTaskRta   = {NULL, "PNIO_RTA",     TSK_PRIO_14},
tPNSParam.tTaskLldp  = {NULL, "LLDP-Task",    TSK_PRIO_13},
tPNSParam.tTaskMibDB = {NULL, "Mib-Database", TSK_PRIO_31},
tPNSParam.tTaskPnsif = {NULL, "PNS_IF",       TSK_PRIO_21},
tPNSParam.tTaskRpc    = {NULL, "RPC",         TSK_PRIO_16},
tPNSParam.tTaskSnmp   = {NULL, "SNMP-Server", TSK_PRIO_32},
tPNSParam.tTaskFodmi  = {NULL, "FODMI",      TSK_PRIO_51},

/* fixed modules */
tPNSParam.patPNSModules      = NULL;

/* MAU types are applied for fiber optic only */
tPNSParam.usMauTypePort0     = 0;
tPNSParam.usMauTypePort1     = 0;

/* no custom SNMP variables */
tPNSParam.tSNMPdatabase.pvCustomGroups = NULL;
tPNSParam.tSNMPdatabase.ulCustomGroupsCount = 0;

/* 2-Ethernet port has a device */
tPNSParam.ulActivePROFINETPorts = 2;

/* LLDP service is active on both Ethernet ports */
tPNSParam.ulActiveLLDPPorts     = 2;

/* now start the stack */
if (TLR_S_OK == PNS_StackInit(&tPNSParam, &ptPNSRsc))
{
    hQuePnsIf = ptPNSRsc->hQuePnsif;
}

```

### 8.3.3 Task Priorities

It is recommended to use the following task priorities as can be seen in the example configuration file:



**Note:**

Any change in the priority order of the tasks may lead to problems which will be difficult to detect. It is recommended that each application task shall have a lower priority than the PNS\_IF-Task

Task Name	Priority	Description
<i>RX_TIMER</i>	<i>TSK_PRIO_DEF_RX_TIMER</i>	<i>rcX Timer interrupt task priority*</i>
PND_HAL_RT	TSK_PRIO_3	PND Switch RT task
PNIO_RTC	TSK_PRIO_4	RTC task
<i>RX_TIMER</i>	<i>TSK_PRIO_5*</i>	<i>rcX Timer interrupt task priority*</i>
PND_HAL_NWC	TSK_PRIO_7	PND Switch NWC task
<b>APP_HI</b>	TSK_PRIO_8	Application task, <b>handles cyclical data</b> (see 8.4.3.2)
PND_HAL_NRT	TSK_PRIO_11	PND Switch NRT task
TlrTimer	TSK_PRIO_12	TLR Timer Task
LLDP-Task	TSK_PRIO_13	LLDP task
PNIO_RTA	TSK_PRIO_14	RTA task
RPC	TSK_PRIO_16	RPC task
PNIO_CMDEV	TSK_PRIO_18	CMDEV task
PNS_IF	TSK_PRIO_21	PROFINET Stack Interface task
TCP_UDP	TSK_PRIO_30	TCP/IP task
Mib-Database	TSK_PRIO_31	SNMP MIB task
SNMP-Server	TSK_PRIO_32	SNMP-Task
<b>APP_LOW</b>	TSK_PRIO_40	Application task, <b>handles acyclical services</b> (see 8.4.3.2)
FODMI	TSK_PRIO_51	Fiber Optic Diagnostic Media Interface

Table 180: Overview about the recommended Task Priorities (\*priority of the *RX\_TIMER* should be set to *TSK\_PRIO\_5* only for Isochronous Applications with fast cycles 250µs, see also [8.6])

### 8.3.4 Fiber optic device

Using Linkable Object Module, it is possible to build a device that uses a fiber optic medium.

#### 8.3.4.1 Fiber optic configuration

As explained above the PROFINET stack starts, if a user application invokes the function `PNS_StackInit()` with configuration parameters. They include a pointer to a configuration structure `FODMI_TASK_INIT_PRM_T` for the fiber optic service task FODMI. This task operates the link LEDs and link activity LEDs for fiber optic ports and reads diagnostic data from the fiber optic transceivers via I2C bus.

Configuration structure `FODMI_TASK_INIT_PRM_T` has following fields:

- `ulCyclicDiagnosis_ms`  
This is a period for sending diagnosis information, evaluated in milliseconds (use 1000ms).
- `pszLEDPort1Link`  
The name of the LED for link (up/down) signaling of the port 1.
- `pszLEDPort1Act`  
The name of the LED for link activity signaling of the port 1.
- `pszLEDPort2Link`  
The name of the LED for link (up/down) signaling of the port 2.
- `pszLEDPort2Act`  
The name of the LED for link activity signaling of the port 2.
- `ulCyclicLEDActState_ms`  
This is a period for checking the activity of the fiber optic ports, evaluated in milliseconds (use 250ms).
- **bSDA1PinIdx** (used only for netX50)  
MMIO pin number on the netX for I2C serial data line (SDA) to connect the fiber optic transceiver for the port 1.
- **bSDA2PinIdx** (used only for netX50)  
MMIO pin number on the netX for I2C serial data line (SDA) to connect the fiber optic transceiver for the port 2.
- **bSCLPinIdx** (used only for netX50)  
MMIO pin number on the netX for I2C serial clock line (SCL) commonly used for fiber optic transceivers for both ports.
- **ulPintype** (used only for netX500 and netX100)  
Pin to switch between fiber optic transceivers on port1 and port 2.

Type of the *switch-pin*. Allowed values:

```

TAG_FIBER_OPTIC_IF_DMI_PINTYPE_NONE = 0 /* not used */
TAG_FIBER_OPTIC_IF_DMI_PINTYPE_GPIO = 1 /* GPIO */
TAG_FIBER_OPTIC_IF_DMI_PINTYPE_PIO = 2 /* PIO or HIFPIO */

```

(HIFPIO used if `uPin` greater than 32)



- **uPin** (used only for netX500 and netX100)  
Number of the *switch-pin*.  
Allowed values:  
0 – 15 if ulPintype=TAG\_FIBER\_OPTIC\_IF\_DMI\_PINTYPE\_GPIO  
0 – 84 if ulPintype=TAG\_FIBER\_OPTIC\_IF\_DMI\_PINTYPE\_PIO
- **fPinInvert** (used only for netX500 and netX100)  
Set to 1 to invert the signal of the *switch-pin*. Possible values 0 or 1.
- **pszPnsIfQueName** (*not used*)  
Set to NULL.
- **pszPioGpioHifPioPinIdentifyName** (used only for netX500 and netX100)  
Name of GPIO, PIO or HIFPIO used for *switch-pin* to identify (depends on ulPintype)

**Note:**

- Because of differences in the netX chips the fields bSDA1PinIdx, bSDA2PinIdx, bSCLPinIdx are evaluated only for netX50 but the fields ulPintype, uPin, fPinInvert – for netX500 and netX100.
- **The netX50** has only one I2C channel and a flexible MMIO matrix that allows to connect the fiber optic transceivers (I2C bus) to any MMIO pin. The MMIO matrix is used to switch the I2C bus (exactly pin I2C\_SDA) between two fiber optic transceivers (the clock pin I2C\_SCL is common for both transceivers). The FODMI task does configure of MMIO martix and switch between SDA pins on its own according to the fields bSDA1PinIdx, bSDA2PinIdx and bSCLPinIdx.  
  
Be sure to select the fiber optic type for the PHY conection in the “IO Config Register”.
- **The netX500 and netX100** have only one I2C channel and fixed pins for I2C bus. Therefore the hardware design (schematic) has to implement a switch of the I2C bus between two fiber optic transceivers. The *switch-pin* will control this switch, it can be configured on any CPIO, PIO or HIFPIO using fields ulPintype and uPin.  
  
Be sure to select the fiber optic type for the PHY conection in the “IO Config Register”.
- **The netX51** has two I2C channels and a flexible MMIO matrix that allows to connect the fiber optic transceivers to any MMIO pin. For each fiber optic transceiver is used own I2C channel.  
  
The FODMI task does not configure MMIO matrix, please configure MMIO matrix for all used I2C channels (ports) before starting the stack.  
  
Select position A or B for fiber optic PHY0 or PHY1 in the “IO Config Register”, see also the pinning table of the netX51.

Please refer to the “Design-In Guide” of used netX chip for layout and wiring and to the configuration examples shown below

For a fiber optic device all parameters for the FODMI task should be set, flags – which port is for fiber optic and MAU types for these ports.

The following examples show the fiber optic specific configuration for different netX chips and different number of fiber optic ports.

**Example 1:** chip type – **netX50**, fiber optic ports – port 1 and port 2, MAU types – default for both ports, I2C bus connected to the MMIO matrix, MMIO pin number 21 used for I2C-clock wire, MMIO pin number 2 for I2C-SDA wire of the port 1 and MMIO pin number 3 for I2C-SDA wire of the port 2

```
int main(void)
{
    ...
    ...
    /* before the operating system starts */
    volatile unsigned long* pulAccessKey =
        (volatile unsigned long*) NETX_IO_CFG_ACCESS_KEY;
    volatile unsigned long* pulIOConfig = (volatile unsigned long*)NETX_IO_CFG;

    unsigned long val = *pulIOConfig;

    /* allow internal connection of the i2c module to MMIO matrix */
    Val |= MSK_NETX_IO_CFG_sel_i2c_mmio;

    /* Select the fiber optic type for the PHY0/1 connection */
    val |= ( MSK_NETX_IO_CFG_sel_fo0 | MSK_NETX_IO_CFG_sel_fo1 );

    /* unlock the netX access-key mechanism and set the values */
    *pulAccessKey = *pulAccessKey;
    *pulIOConfig = val;

    ...
    ...
}

FODMI_TASK_INIT_PRM_T g_tFodmiInitPrm =
{
    .ulCyclicDiagnosis_ms          = 1000,
    .pszLEDPort1Link              = "POF_PORT0LINK",
    .pszLEDPort1Act               = "POF_PORT0ACT",
    .pszLEDPort2Link              = "POF_PORT1LINK",
    .pszLEDPort2Act               = "POF_PORT1ACT",
    .ulCyclicLEDActState_ms       = 250,
    .bSDA1PinIdx                  = 2, /* MMIO2 is SDA for port 1 */
    .bSDA2PinIdx                  = 3, /* MMIO3 is SDA for port 2 */
    .bSCLPinIdx                   = 21, /* MMIO21 is SCL for clock */
    .ulPintype                    = TAG_FIBER_OPTIC_IF_DMI_PINTYPE_NONE,
    .uPin                         = 0, /* not used */
    .fPinInvert                   = 0, /* not used */
    .pszPnsIfQueueName            = NULL,
    .pszPioGpioHifPioPinIdentifyName = NULL, /* not used */
};

...
...
PROFINET_IODEVICE_STARTUPPARAMETER_T tPNSTParam /* PNS Stack Parameters */

tPNSTParam.ulFlags                = PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0 |
    PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1;

...
tPNSTParam.pfnFodmiTaskInitFn = (FN_FODMI_TASK_INIT) FODMITask_Init;
tPNSTParam.pvFodmiTaskInitPrm = &g_tFodmiInitPrm;

...
tPNSTParam.usMauTypePort0        = 0; /* default MAU type: polymer optical fiber */
tPNSTParam.usMauTypePort1        = 0; /* default MAU type: polymer optical fiber */
...
/* now start the stack */
```

```
if (TLR_S_OK == PNS_StackInit(&tPNSParam, &ptPNSRsc))
{
    hQuePnsIf = ptPNSRsc->hQuePnsif;
}
```

All used LEDs should be configured in config.c (RX\_LED\_SET\_T). The FODMI task uses their names to identify them in runtime. It applies to the `pszPioGpioHifPioPinIdentifyName` also.

**Example 2:** chip type – **netX100**, fiber optic ports – port 1 and port 2, MAU types – “100BaseFXFD” for both ports. GPIO number 7 used to switch between fiber optic transceivers

```
int main(void)
{
    ...
    ...
    /* before the operating system starts */
    volatile unsigned long* pulAccessKey =
        (volatile unsigned long*) NETX_IO_CFG_ACCESS_KEY;
    volatile unsigned long* pulIOConfig = (volatile unsigned long*)NETX_IO_CFG;

    unsigned long val = *pulIOConfig;

    /* Select the fiber optic type for the PHY0/1 connection */
    val |= ( MSK_NETX_IO_CFG_sel_fo0 | MSK_NETX_IO_CFG_sel_fo1 );

    /* unlock the netX access-key mechanism and set the values */
    *pulAccessKey = *pulAccessKey;
    *pulIOConfig = val;

    ...
    ...
}

FODMI_TASK_INIT_PRM_T g_tFodmiInitPrm =
{
    .ulCyclicDiagnosis_ms          = 1000,
    .pszLEDPort1Link               = "POF_PORT0LINK",
    .pszLEDPort1Act                = "POF_PORT0ACT",
    .pszLEDPort2Link               = "POF_PORT1LINK",
    .pszLEDPort2Act                = "POF_PORT1ACT",
    .ulCyclicLEDActState_ms        = 250,
    .bSDA1PinIdx                   = 0,      /* not used */
    .bSDA2PinIdx                   = 0,      /* not used */
    .bSCLPinIdx                    = 0,      /* not used */
    .ulPintype                     = TAG_FIBER_OPTIC_IF_DMI_PINTYPE_GPIO,
    .uPin                          = 7,      /* GPIO 7 */
    .fPinInvert                    = 0,      /* not invert */
    .pszPnsIfQueueName             = NULL,
    .pszPioGpioHifPioPinIdentifyName = "GPIOFODMI",
};

...
...

PROFINET_IODEVICE_STARTUPPARAMETER_T tPNSParam; /* PNS Stack Parameters */

tPNSParam.ulFlags = PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0 |
    PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1;

...
tPNSParam.pfnFodmiTaskInitFn = (FN_FODMI_TASK_INIT) FODMITask_Init;
tPNSParam.pvFodmiTaskInitPrm = &g_tFodmiInitPrm;

...
PNSParam.usMauTypePort0 = 0x12; /* MAU type: glas optical fiber */
tPNSParam.usMauTypePort1 = 0x12; /* MAU type: glas optical fiber */
.....
```

**Example 3:** chip type – **netX50**, fiber optic port – port 1, MAU type – “100BaseFXFD” for port 1, I2C bus connected to the MMIO matrix, MMIO pin number 21 used for I2C-clock wire and MMIO pin number 2 for I2C-SDA wire of the port 1. The second port is connected with twisted pair

```
int main(void)
{
    ...
    ...
    /* before the operating system starts */
    volatile unsigned long* pulAccessKey =
        (volatile unsigned long*) NETX_IO_CFG_ACCESS_KEY;
    volatile unsigned long* pulIOConfig = (volatile unsigned long*)NETX_IO_CFG;

    unsigned long val = *pulIOConfig;

    /* allow internal connection of the i2c module to MMIO matrix */
    Val |= MSK_NETX_IO_CFG_sel_i2c_mmio;

    /* Select the fiber optic type for the PHY0 connection */
    val |= ( MSK_NETX_IO_CFG_sel_fo0);

    /* unlock the netX access-key mechanism and set the values */
    *pulAccessKey = *pulAccessKey;
    *pulIOConfig = val;

    ...
    ...
}

FODMI_TASK_INIT_PRM_T g_tFodmiInitPrm =
{
    .ulCyclicDiagnosis_ms          = 1000,
    .pszLEDPort1Link              = "POF_PORT0LINK",
    .pszLEDPort1Act               = "POF_PORT0ACT",
    .pszLEDPort2Link              = "", /* not used */
    .pszLEDPort2Act               = "", /* not used */
    .ulCyclicLEDActState_ms       = 250,
    .bSDA1PinIdx                  = 2, /* MMIO2 is SDA for port 1 */
    .bSDA2PinIdx                  = 0, /* not used */
    .bSCLPinIdx                   = 21, /* MMIO21 is SCL for clock */
    .ulPintype                    = TAG_FIBER_OPTIC_IF_DMI_PINTYPE_NONE,
    .uPin                         = 0, /* not used */
    .fPinInvert                   = 0, /* not used */
    .pszPnsIfQueueName            = NULL,
    .pszPioGpioHifPioPinIdentifyName = NULL, /* not used */};
    ...
    ...

PROFINET_IODEVICE_STARTUPPARAMETER_T tPNSTParam; /* PNS Stack Parameters */

tPNSTParam.ulFlags                = PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0;
    ...
tPNSTParam.pfnFodmiTaskInitFn = (FN_FODMI_TASK_INIT) FODMITask_Init;
tPNSTParam.pvFodmiTaskInitPrm= &g_tFodmiInitPrm;
    ...
tPNSTParam.usMauTypePort0        = 0x12; /* MAU type: glas optical fiber */
tPNSTParam.usMauTypePort1        = 0; /* not used */
    ...
}
```

**Example 4:** chip type – **netX51**, fiber optic ports – port 1 and port 2, MAU types – “100BaseFXFD” for both ports. I2C buses are connected to the MMIO matrix. FODMI task does use the I2C0 bus to service the port 1 using MMIO pin number 2 for I2C0\_SCL wire and MMIO pin number 3 for I2C0\_SDA wire, and does use the I2C1 bus to service the port 2 using MMIO pin number 9 for I2C1\_SCL wire and MMIO pin number 8 for I2C1\_SDA

```
int main(void)
{
    unsigned long val;
    volatile unsigned long* pulAccessKey =
        (volatile unsigned long*) Adr_NX51_asic_ctrl_access_key;
    volatile unsigned long* pulIOConfig =
        (volatile unsigned long*) Adr_NX51_io_config;
    volatile unsigned long* pulMMIOConfig =
        (volatile unsigned long*) NX51_NETX_MMIO_CTRL_AREA;

    ...
    ...
    /* before the operating system starts */
    ...
    ...
    /* configure MMIO matrix for both I2C channels used by FODMI task (2 fiber
       optic ports) */

    /* unlock the netX access-key mechanism and set the values */
    *pulAccessKey = *pulAccessKey;
    /* set MMIO pin number 2 for I2C0_SCL wire */
    pulMMIOConfig[2] = MMIO_CONFIG_I2C0_SCL_MMIO;

    *pulAccessKey = *pulAccessKey;
    pulMMIOConfig[3] = MMIO_CONFIG_I2C0_SDA_MMIO;

    *pulAccessKey = *pulAccessKey;
    pulMMIOConfig[9] = MMIO_CONFIG_I2C1_SCL;

    *pulAccessKey = *pulAccessKey;
    pulMMIOConfig[8] = MMIO_CONFIG_I2C1_SDA;

    /* prevent activation of fiber optics on both positions A and B */
    val = *pulIOConfig;
    val &= ~(MSK_NX51_io_config_sel_fo0_a | MSK_NX51_io_config_sel_fo0_b |
            MSK_NX51_io_config_sel_fo1_a | MSK_NX51_io_config_sel_fo1_b);

    /* Select position A for connection of the fiber optic PHY0 and PHY1 */
    val |= (MSK_NX51_io_config_sel_fo0_a | MSK_NX51_io_config_sel_fo1_a);

    /* allow internal connection of the i2c module to MMIO matrix */
    Val |= MSK_NETX_IO_CFG_sel_i2c_mmio;

    /* unlock the netX access-key mechanism and set the values */
    *pulAccessKey = *pulAccessKey;
    *pulIOConfig = val;
    ...
    ...
}

FODMI_TASK_INIT_PRM_T g_tFodmiInitPrm =
{
    .ulCyclicDiagnosis_ms          = 1000,
    .pszLEDPort1Link               = "POF_PORT0LINK",
    .pszLEDPort1Act                = "POF_PORT0ACT",
```

```

.pszLEDPort2Link          = "POF_PORT1LINK",
.pszLEDPort2Act           = "POF_PORT1ACT",
.ulCyclicLEDActState_ms   = 250,
.bSDA1PinIdx              = 0,      /* not used */
.bSDA2PinIdx              = 0,      /* not used */
.bSCLPinIdx               = 0,      /* not used */
.ulPintype                = 0,      /* not used */
.uPin                     = 0,      /* not used */
.fPinInvert               = 0,      /* not used */
.pszPnsIfQueueName        = NULL,
.pszPioGpioHifPioPinIdentifyName = NULL, /* not used */
};
...
...
PROFINET_IODEVICE_STARTUPPARAMETER_T  tPNSTParam /* PNS Stack Parameters */

tPNSTParam.ulFlags                = PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0 |
                                   PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1;
...
tPNSTParam.pfnFodmiTaskInitFn = (FN_FODMI_TASK_INIT) FODMITask_Init;
tPNSTParam.pvFodmiTaskInitPrm= &g_tFodmiInitPrm;
...
tPNSTParam.usMauTypePort0        = 0x12; /* MAU type: glas optical fiber */
tPNSTParam.usMauTypePort1        = 0x12; /* MAU type: glas optical fiber */
...
/* now start the stack */
if (TLR_S_OK == PNS_StackInit(&tPNSTParam, &ptPNSRsc))
{
    hQuePnsIf = ptPNSRsc->hQuePnsif;
}

```

### 8.3.4.2 Medium Attachment Unit for Fiber Optic

The fiber optic device can differentiate between following types of the Medium Attachment Units (MAU):

MAU Type	Value	Description
100BaseFXFD	0x12	Used transceiver for glas optical fiber (GOF)
100BasePXF	0x36	Used transceiver for polymer optical fiber (POF)

Table 181: MAU types for fiber optic ports

### 8.3.5 PROFINET Netload Requirements

The PROFINET Certification requires at least to pass the **Netload Class I** tests since V2.31 (mandatory). If the device does not pass these tests the Conformance Test will not be passed and the device does not get a certificate. The application and firmware must be designed as follows in order to be able to pass the tests:

- The application shall be split into at least two tasks. One task (e.g. "APP\_HI") is entirely dedicated to the handling of the process data while the acyclic services are handled by the other task(s) (e.g. "APP\_LOW"). The process data task must be configured with a priority between the "PND\_HAL\_NWC" task and "RTA" task. Therefore this task must be designed carefully in order to avoid unintentional consumption of CPU power. If an application can process the cyclical-data really fast (e.g. one or two checks, apply to outputs and fetch inputs) the "APP\_HI" can be omitted and whole processing performed in the "pfmEventHandler" on "PNS\_IF\_IO\_EVENT\_CONSUMER (PROVIDER)\_UPDATE\_DONE" (see section *Set IO-Image Request* on page 94 for details). The acyclic application task shall have a lower priority than the protocol stack (see 8.3.3 for recommended task priorities).

For netX51 a special memory setup is required to pass the Netload Test Class III:

- In order to improve execution speed the netX51 has internal SRAM memory used by the protocol stack and application. The linker script shall be designed that important parts of the protocol stack and the process data handling of the application is relocated into the SRAM areas. The startup code must be adapted as well to perform the necessary relocations from SDRAM to SRAM at startup. An example startup script and linker script can be found in the corresponding LOM example.
- To improve execution speed of separate tasks it is recommended to allocate the stacks of "TCP\_UDP" task, PROFINET tasks (by priority: "PND\_HAL\_RT", "PNIO\_RTC", "PND\_HAL\_NWC", "PND\_HAL\_NRT", "PNIO\_RTA", "PNIO\_CMDEV") and Application's tasks also in the internal SRAM memory. E.g. the following part of linker script shows how to allocate the stacks of "TCP\_UDP" and "Packet API Demo" tasks in the internal memory (see chapter 8.3.1.4 for definition of the task stacks):

```
MEMORY
{
    .....
    /* itcm & dtcm */
    ITCM(rx): ORIGIN = 0x00000080, LENGTH = 128K + 128K + 64K - 0x80 /* INTRAM0/1/2 */
    DTCM(rw): ORIGIN = 0x04050000, LENGTH = 64K + 64K + 32K + 32K /* INTRAM3/4/5/6 */
    .....
}

.itcm :
{
    .....
}>ITCM AT>SDRAM

.dtcn :
{
    .....
    /* stack of the "TLR TIMER" task*/
    *Config_netX51.o(.bss.auTskStack_Tlr_Timer)

    /* stack of the "Packet API Demo" task*/
    *Config_netX51.o(.bss.auTskStack_Pck_API_Demo)

    /* stack of the "TCP_UDP" task*/
    *Config_netX51.o(.bss.auTskStack_Tcp_Task)

    .../* or stack of ALL static tasks declared in the "Config_netX51.c" /
    *Config_netX51.o(.bss*)
```

```
.....

/* For example the stacks of some PROFINET tasks: */

/*stacks of the "HAL_RTC" task */
*Hal_EddPROFINETDevice_common.o(.bss.aulHal_RTC_TaskStack*)
/*stacks of the "HAL_NWC" task */
*Hal_EddPROFINETDevice_common.o(.bss.aulHal_NWC_TaskStack*)
/*stacks of the "HAL_NRT" task */
*Hal_EddPROFINETDevice_common.o(.bss.aulHal_NRT_TaskStack*)

/*stack of the "PNIO_RTC" task */
*PNIORTC_Init.o(.bss.g_RTC_Task_Stack*)

/*stack of the "PNIO_RTA" task */
*PNIORTA_Init.o(.bss.s_auiRTATask_TaskStack*)

/*stack of the "PNIO_CMDEV" task */
*PNIOCMDEV_init.o(.bss.s_auiCMDEVTask_TaskStack*)

.....
}>DTCM AT>SDRAM
```



## 8.4 PROFINET Certification

Before any PROFINET device appears on the market it has to pass PROFINET certification tests to prove the conformance according to one of the defined PROFINET conformance classes (A, B or C).

To achieve conformance class A (without SNMP) or B (with SNMP) the PROFINET Real Time (RT) protocol behavior and the behavior on network load have to be tested on the device.

If a PROFINET device shall conform to conformance class C, it has to fulfill some additional requirements including PROFINET Isochronous Real Time (IRT). This chapter explains common requirements to a PROFINET device for all types of the certification tests.

### 8.4.1 RT Tests (Conformance class A, B and C)

#### 8.4.1.1 Description

The common PROFINET RT certification tests have to be passed by all IO Device implementations. These test cases cover the following functionality

- basic state machine behavior
- different parts of protocol (coding)
- reaction of device on erroneous configurations and situations
- acyclic services
- cyclic data and data-status (IOXS)

#### 8.4.1.2 General Requirements for RT Tests

In order to execute all test cases the examiner in the certification laboratory should have a possibility to trigger alarms (diagnosis or process alarms) on a device. For example additional push button causes an alarm, “magic” sequence of push buttons, deliberate shorting of outputs and so on. This is required if the device generates alarm / diagnosis at runtime. If no alarm and no diagnosis are generated at runtime, these tests can be skipped.

It is recommended (but not required) to have a possibility to change input values on a device. For example set some inputs of an IO-Device to “HI-level”, change measurement values if it is a sensor and so on.

A small informal paper describing how to trigger alarms and change IO Data needs to be given to the certification laboratory together with the device.

### 8.4.1.3 Common checks before Certification (GSDML)

The device has to be consistent with its own GSDML file. The “DeviceID” and “VendorID” in the GSDML shall be the same that was used in Set Configuration Service for fields “PNS\_IF\_DEVICE\_PARAMETER\_T::ulVendorId” and “PNS\_IF\_DEVICE\_PARAMETER\_T::ulDeviceId”, or used in data base (if the stack configured with data base), or used in tag list (if according tag was enabled in firmware). The vendor name shall also be correct:

```
<ProfileBody>
  <DeviceIdentity DeviceID="0x0103" VendorID="0x011E"> ←
    <InfoText TextId="DeviceDescription_InfoText"/>
    <VendorName Value="Hilscher Gesellschaft für Systemautomation mbH"/> ←
  </DeviceIdentity>
  <DeviceFunction>
    <Family MainFamily="I/O" ProductFamily="PNS"/>
  </DeviceFunction>
  <ApplicationProcess>
    <DeviceAccessPointList>
      <DeviceAccessPointItem AddressAssignment="DCP;LOCAL" DNS-CompatibleName="cifxrepns"
        <ModuleInfo>
          <Name TextId="CIFX RE/PNS V3.5.18 - V3.5.x"/>
          <InfoText TextId="DIM 20_InfoText"/>
          <VendorName Value="Hilscher Gesellschaft für Systemautomation mbH"/> ←
          <OrderNumber Value="1250.100"/> ←
          <HardwareRelease Value="2"/>
          <SoftwareRelease Value="3.5.x"/>
        </ModuleInfo>
      </DeviceAccessPointItem>
    </DeviceAccessPointList>
  </ApplicationProcess>
</ProfileBody>
```

Figure 29: Vendor and device identification in the GSDML file

In addition it is required that the OrderNumber, HardwareRelease and SoftwareRelease from the GSDML file match the configuration of the PROFINET IO-Device. The values are contained in PNS\_IF\_DEVICE\_PARAMETER\_T.

If a new product is created based on Hilscher GmbH reference GSDML files, it is highly recommended to remove all non-matching DAPs from the new GSDML file. Otherwise it may lead to confusion why a new product already has several DAPs inside the GSDML file.

### 8.4.1.4 Basic Application Behavior

If the application takes care of storing device NameOfStation and IP parameters it has to store these parameters according to the following rules:

IP-parameters (see section *Save IP Address Service* on page 148):

- If the flag `bRemanent` is not set the application shall set the permanently stored IP parameters to 0.0.0.0 and use IP 0.0.0.0 after the next PowerUp cycle.
- If the flag `bRemanent` is set the application shall store the indicated IP parameters permanent and use them after the next PowerUp cycle

Device NameOfStation (see *Save Station Name Service* on page 145):

- If the flag `bRemanent` is not set the application shall delete the permanently stored Station Name and use empty Station Name after the next PowerUp cycle.
- If the flag `bRemanent` is set the application shall store Station Name permanently and use it after the next PowerUp cycle.

On Reset Factory Settings Indication the application shall delete the permanently stored Station Name, set the permanently stored IP parameters to 0.0.0.0 and use them after the next PowerUp cycle (see section *Reset Factory Settings Service* on page 155).

## 8.4.2 IRT Tests (Conformance class C only)

### 8.4.2.1 Description

The common PROFINET IRT certification tests have to be passed by all IO Device implementations. These test cases cover the following functionality

- basic IRT-related state machine behavior
- synchronization related tests
- different parts of protocol (coding)
- reaction of the device on erroneous configurations and situations
- cyclic data and data status (IOXS)

### 8.4.2.2 General Requirements for IRT Tests

There are no special additional requirements besides those from the RT tests.

### 8.4.2.3 Hardware Requirements for IRT Tests

To evaluate the synchronization capability of the device it is required to offer an external synchronization pin to the examiner in the certification laboratory. The synchronization jitter of this pin needs to be verified. This pin shall be present for the device used during certification test. If this pin is not available at the devices used in the field, this is accepted.

### 8.4.2.4 Software Requirements for IRT Tests

Regarding application handling nothing special needs to be taken in account. So for certification no additional software requirements exist compared to the ones defined for RT tests.

### 8.4.2.5 GSDML Requirements for IRT Tests

The GSDML file needs to contain the required IRT related keywords. Please refer to Hilscher reference GSDML files for details. The following listing will just briefly name the required keywords, there may be more keywords required by GSDML checker tool.

- InterfaceSubmoduleItem
  - SupportedRT\_Classes="RT\_CLASS\_1;RT\_CLASS\_3"
- RT\_Class3Properties
  - ForwardingMode="Relative"
  - MaxBridgeDelay="5500"
  - MaxNumberIR\_FrameData="256"
  - StartupMode="Advanced;Legacy"
- SynchronisationMode
  - MaxLocalJitter="50"
  - SupportedRole="SyncSlave"
  - SupportedSyncProtocols="PTCP"
  - T\_PLL\_MAX="1000"
- RT\_Class3TimingProperties
  - ReductionRatio="1 2 4 8 16"
  - SendClock="8 16 32 64 128"

- PortSubmoduleItem
  - MaxPortRxDelay="340"
  - MaxPortTxDelay="92"

## 8.4.3 Network Load Tests

### 8.4.3.1 Description

Starting with certification for PROFINET specification V2.3 (expressed in GSDML file of the IO-Device by using PNIO\_Version="V2.31") it is mandatory to execute special network load tests during certification.

These tests verify the correct behavior of the PROFINET IO device during different network loads.

The following classes of network load are defined:

- network load class I
- network load class II
- network load class III

The device has to pass at least the tests for "network load class I" to fulfill the certification requirements.

### 8.4.3.2 Requirements to the Application

The main point in the application is the optimization. The following recommendations will help to achieve at least network load class I:

- Separate cyclic and acyclic parts of application;
- Arrange the tasks priorities according to the recommendations in Table 180;
- Allocate all important parts of the protocol stack and the process data handling of the application in to internal SRAM memory (netX51);
- Allocate the stacks of tasks also in internal SRAM memory (netX51).

For more information to the optimization see chapter *PROFINET Netload Requirements* on page 279 and refer to the LOM example.

## 8.4.4 How to handle I&M Data



### Note:

In order to fulfill PROFINET conformance needs, any PROFINET IO device should be able to handle the I&M0, I&M1, I&M2, I&M3 and I&M0 Filter data.

### 8.4.4.1 Overview

Identification & Maintenance (I&M) is an integral part of each PROFINET Device implementation. It provides standardized information about a device and its parts. The I&M Information is accessible through PROFINET Record Objects and is always bound to a submodule belonging to the item to be described. An item means here the PROFINET Device itself or a part of this device e.g. a pluggable module for modular devices. Submodules can provide own I&M objects or share the I&M objects of other submodules. The I&M objects can be grouped into three kinds of information as described as follows.

#### I&M0

I&M0 is a read only information which describes the associated item. The following fields are defined:

Field	Description	Usage Hints
Vendor ID	The PNO vendor ID of the associated item. E.g. the vendor of the device or the vendor of a pluggable module/submodule.	This is the vendor ID of the manufacturer of the item and not the vendor ID of the PROFINET protocol vendor. Do not use the Hilscher vendor ID.
Order ID	The order ID of the associated item.	Order ID as defined by the manufacturer of the item. Must be equal to any order ID markings on the item itself.
Serial Number	The serial number of the associated item.	Must be an unique serial number associated with the item. Must be equal to any serial number markings on the item itself.
Hardware Revision	The revision of the hardware of the item.	Must be equal to any hardware revision markings on the item itself.
Software Revision	The software revision of the item.	This is the software version of the whole item including the PROFINET Protocol implementation and the Application. This version is managed by the manufacturer of the item. It must be changed whenever a part of the software within the item (including the PROFINET Protocol implementation if it is part of the item) changes. This is not the version number of the PROFINET Protocol implementation. Do not use the Hilscher Version of the PROFINET Protocol implementation.
Revision Counter	Counts the changes of I&M1 to I&M4 objects	-
Profile ID	The Profile of the item if applicable.-	-
Profile Specific Type		
Supported I&M objects	Bitmask defining which I&M objects are supported by this item.	-

#### I&M1 to I&M4

The I&M1 to I&M4 objects provide a non-volatile storage for PROFINET engineering related information. This information is typically generated by the engineering software and stored within the objects at engineering time. The information must be stored by the device in non-volatile

memory. The objects must be stored physically within the associated item. This means in particular, if a plugable module is removed from one backplane and plugged into another backplane, it must deliver the same I&M1 to I&M4 information as stored before.

## I&M5

Finally, the I&M5 record provides information about the PROFINET protocol implementation itself. It is quite similar to I&M0 but describes the PROFINET protocol implementation instead. Thus it is handled by the PROFINET Protocol implementation itself.

## I&M0 Filter

Additionally to these submodule specific I&M objects, each PROFINET device must support the global I&M Filter Data object, the so called "I&M0FilterData". This is a read-only object which is used to determine which submodules are associated with dedicated I&M objects.

### 8.4.4.2 Structure and access paths of I&M objects

The following figure shows the structural organization of I&M Records within a device and the access paths:

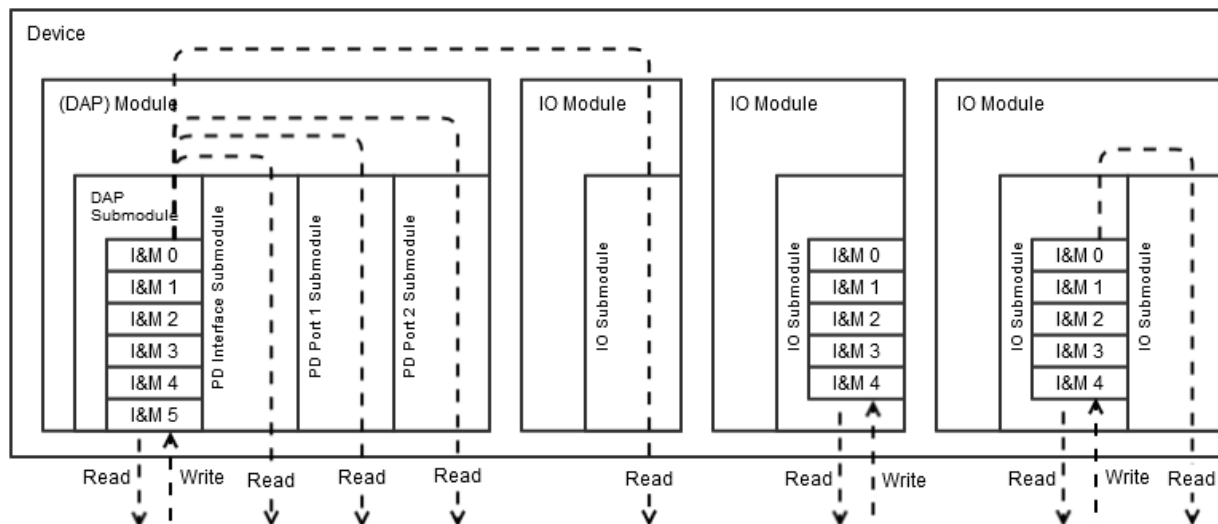


Figure 30: Structural organization of I&M Records within a Device and the Access Paths

According to I&M a submodule can be characterized as follows:

- **Module Representative:** The submodule is a representative for the module. This means that the submodule provides the I&M information of the superordinate module.
- **Device Representative:** The submodule is a representative for the device. In this case the submodule provides the I&M information of the superordinate device. Exactly one submodule of the device must have this property. Typical the DAP submodule is chosen for this purpose.
- **Default:** The submodule only represents itself or does not have any I&M information at all. In the latter case only I&M Read Access is allowed and the Read will deliver the I&M information of the Module Representative or the Device Representative.

When reading the I&M objects of a submodule the following order is used to deliver the requested data:

1. If the submodule provides an own I&M0 object, the I&M read access will deliver the submodule's I&M objects

2. If the submodule has no own I&M0 object and the superordinate module has a module representative, the module representative's I&M objects will be delivered.
3. If the submodule has no own I&M0 object and the superordinate module has no representative, the device representative's I&M objects will be delivered.

In contrast to this, writing I&M1 to I&M4 objects is only possible on the submodules associated with the I&M objects itself.

Finally, the I&M0 Filter Data object contains the information which submodules are associated with their own I&M data, which submodules represent their superordinate module and which submodule represents the whole device. This object is global and a read-only object and it is not associated with any submodule.

#### 8.4.4.3 Usage of I&M with Hilscher PROFINET Protocol

The PROFINET Device protocol implementation provided by Hilscher supports two modes of operation with I&M. For simple applications the PROFINET protocol implementation provides I&M0 to I&M5 objects within the DAP submodule. In this case, the I&M is fully handled by the PROFINET Protocol implementation without any interaction with the application. If a more complex I&M structure is required, the PROFINET Protocol implementation can forward I&M accesses to the Application. In this case all I&M objects must be handled by the application. The desired mode of operation is configured using the Set Configuration Service.

If the PROFINET protocol implementation is configured to handle the I&M internally, the following parameters will be used within the I&M0 object:

Field	Source of value
Vendor ID	Vendor ID in Set Configuration Service or Configuration Database. Can be overwritten by tag list if configuration database is used.
Order ID	Order ID in Set Configuration Service or configuration database.
Serial Number	Defaults to the serial number from Security Memory / Flash device label. This is the serial number of the Hilscher communication module if applicable. It shall be changed to the serial number of the manufacturers device using the Set OEM Parameters service.
Hardware Revision	Hardware revision in Set Configuration Service. If a Configuration Database is used the hardware revision from Security Memory / Flash device label is used. This is the hardware revision of the Hilscher communication module if applicable.
Software Revision	Software revision in Set Configuration Service. If a Configuration Database is used the PROFINET Protocol implementations software revision will be used.
Revision Counter	Internally stored and incremented on each change of I&M1 to I&M4.
Profile ID	Default is '0x00' (Manufacturer specific). Can be changed to a desired value using Set OEM Parameters service.
Profile Specific Type	Default is '0x05' (Generic Device). Can be changed to a desired value using Set OEM Parameters service.
Supported I&M objects	Defaults to I&M0 to I&M5. Can be changed to a desired value using Set OEM Parameters service.

If the application requested to handle the I&M objects, all I&M accesses will be forwarded to the application using the I&M Read and I&M Write service. For details on the data structures of the I&M services refer to the protocol API. The application must store the data of I&M write services permanently, so that the I&M1 to I&M4 objects and the I&M0 revision counter can be recovered after a power cycle. The I&M0 Filter data structure must be filled by the application with a list of the submodules which are associated with dedicated I&M data. Exactly one of these submodules must be characterized as device representative. Characterization as a module representative is optional and only sensible if the module has more than one submodule. The protocol stack will then internally build up to three lists out of the information provided by the application.

## 8.5 Second DPM channel – Ethernet Interface

This section is valid for loadable firmware only.

From version 3.8.0.0 of the PROFINET stack, loadable firmware for netX 51 and netX 100 offers a second dual-port memory channel (channel1) for devices with a 64 KB large dual-port memory only.

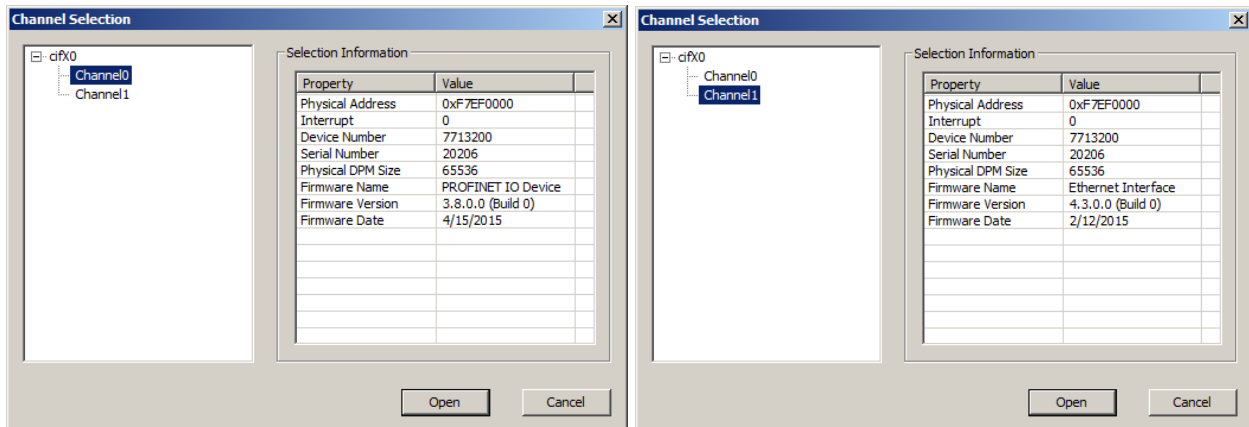


Figure 31: PROFINET Stack is accessible on the DPM Channel0 and Ethernet Interface – on the Channel1

The Ethernet interface stack provides an API for:

- Send/receive raw Ethernet frames to/from PROFINET interface MAC address (OEM mode)
- Reusing the API of internal TCP/IP stack running with PROFINET
- Send/receive raw Ethernet frames using a dedicated MAC address (NDIS mode)

Depending on the PROFINET loadable firmware some of this APIs may not be available.

For more information about Ethernet Interface stack refer to reference [7].



### Note:

Loadable firmware for netX 51 does not support OEM mode!

Loadable firmware for netX 51 shows the second DPM channel only, if the NDIS mode is activated.



### Note:

Loadable firmware for netX 50 does not support NDIS mode!



### Note:

First, the NDIS mode needs to be activated in loadable firmware using the **Tag List Editor** software!



### Note:

The “cifX Device Driver Setup” delivers a driver for Windows that allows to use the Ethernet interface stack running NDIS mode as usual Ethernet adapter and appears as “Virtual cifX Ethernet Adapter”.



## Activation of the NDIS mode

By default the NDIS mode for the Ethernet interface is deactivated in loadable firmware. First, use the Tag List Editor software to activate the NDIS mode.

- load firmware (1)
- go to the “Ethernet NDIS Support” tag (2)
- set check box “Enable NDIS Support” (3)
- save the loadable firmware with activated NDIS support (4).

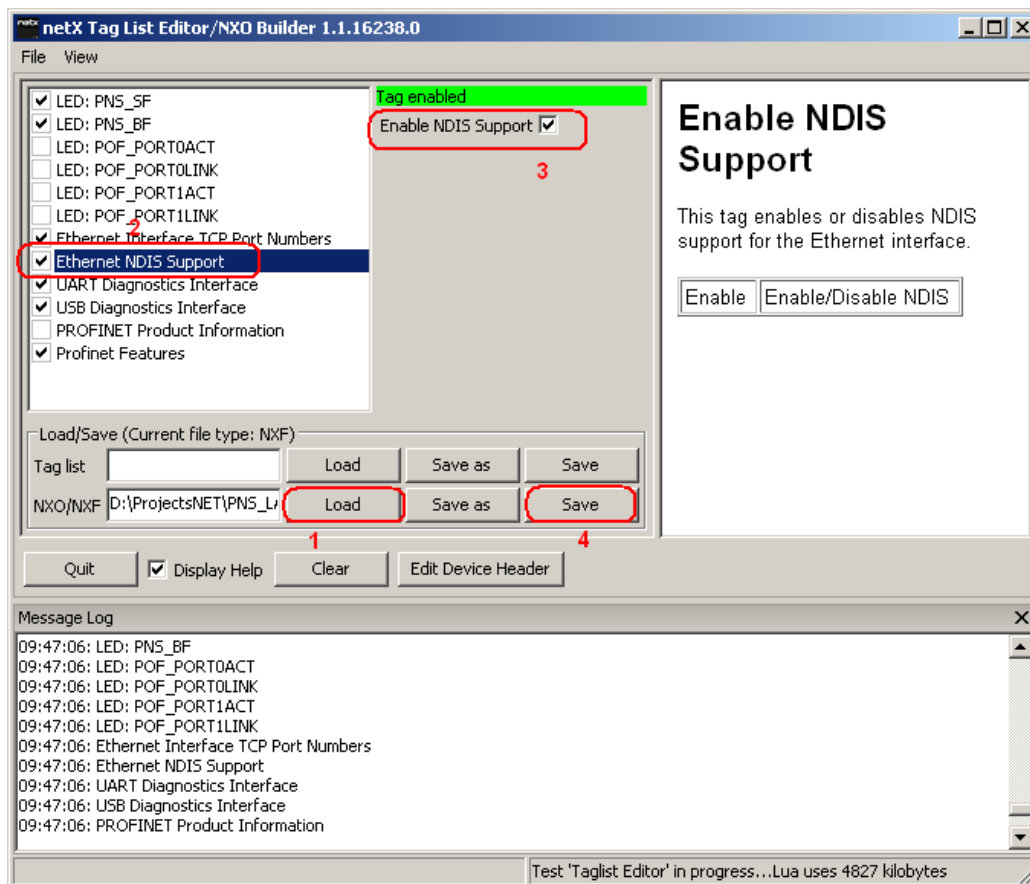


Figure 32: PROFINET Stack is accessible on the DPM channel0 and Ethernet Interface on the channel1



### Important note:

Using the “Ethernet Interface” in NDIS mode has a negative effect on netload environment! It may be a drawback in achieving of the desired network load class in the PROFINET IO-Device certification.

## 8.6 Isochronous Application

The aim of this chapter is to support the development of the isochronous PROFINET devices based on the netX chip with the current version of the stack.



---

Detailed description of the isochronous mode is given in [6].

---

The most important ability of the isochronous PROFINET device is to deliver the process data (latch inputs and set outputs) exactly to the configured (planned) time in nanosecond accuracy. These strict timing-requirements have to be solved by software structure: all the tasks in OS that are responsible for cyclic data exchange should have the highest priority.

In case of NXLOM see recommendations to the task priorities described above in chapter 8.3.3

In case of LFW an application has possibility to activate the DPM watchdog to supervise the protocol stack and vice versa (see 7.1.5.1 parameter `ulWdgTime`). The DPM watchdog mechanism is implemented in rcX OS in context of the `RX_TIMER` task. **If a developed application will support isochronous mode the `RX_TIMER` task has to be necessarily switched to lower priority.** Attention, it has impact on watchdog functionality:



---

**The DPM watchdog supervising can reach timeout in case of high network load with PROFINET cyclic (RTC) frames (case of network load tests in certification) without any failure of the PROFINET Stack or OS.**

---

So it is not recommended to activate the DPM watchdog in case of isochronous application!

In case of isochronous application use "Tag List Editor" to switch the `RX_TIMER` task to lower priority in the "Interrupt: `RX_TIMER`" tag (see figure below).

To limit the possible side-effects the priority of the `RX_TIMER` task will be set forcibly always to `TSK_PRIO_5` (according to chapter 8.3.3) independently of the priority "`TSK_PRIO_**`" selection!

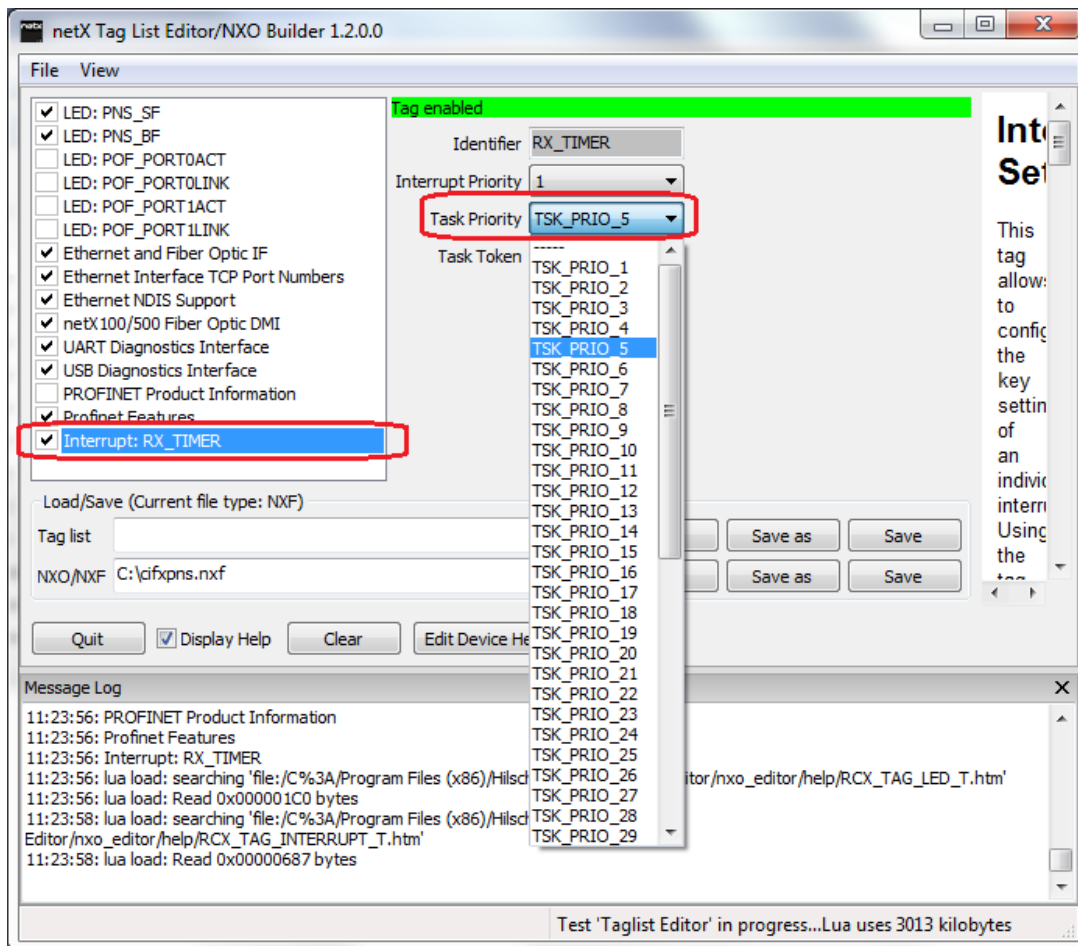


Figure 33: Priority setup of the `RX_TIMER` task in the loadable PROFINET firmware for isochronous application

## 8.7 PROFINET Status Code

The PROFINET status code is used by the PROFINET protocol to indicate success or failure. This chapter shall give a short introduction to this topic, as the status helps to solve problems and is used in some services as well (The according field is often named `ulpnio`). These services include:

- Read Record Service
- Write Record Service
- AR Abort Indication service
- AR Abort Request Service

The status code is an unsigned 32 bit integer value which can be structured into four fields as shown in Figure 21: Structure of the PROFINET status code.

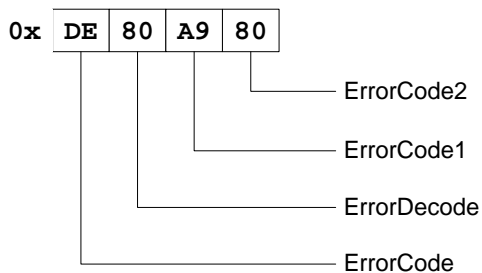


Figure 34: Structure of the PROFINET Status Code

The fields define a hierarchy on the error codes. The ordering is as follows: `ErrorCode`, `ErrorDecode`, `ErrorCode1` and `ErrorCode2`. The meaning of lower order fields depends on the values of the higher order fields. The special value `0x00000000` means no error or success.

### 8.7.1 The ErrorCode Field

This field defines the domain, in which the error occurred. The following table shows the valid values:

Value	Meaning/Use	Description
0x20 – 0x3F	Manufacturer specific: for Log Book	To be used when the application generates a log book entry. Meaning of the values is manufacturer specific (Defined by manufacturer of the device.)
0x81	PNIO: for all errors not covered elsewhere	Used for all errors not covered by the other domains
0xCF	RTA error: used in RTA error PDUs	Used in alarm error telegrams (Low Level)
0xDA	AlarmAck: used in RTA data PDUs	Used in alarm data telegrams (High Level)
0xDB	IODConnectRes: RPC Connect Response	Used to indicate errors occurred when handling a Connect request.
0xDC	IODReleaseRes: RPC Release Response	Used to indicate errors occurred when handling a Release request.
0xDD	IODControlRes: RPC Control Response	Used to indicate errors occurred when handling a Control request.
0xDE	IODReadRes: RPC Read Response	Used to indicate errors occurred when handling a Read request.
0xDF	IODWriteRes: RPC Write Response	Used to indicate error occurred when handling a Write response

Table 182: Coding of PNIO Status ErrorCode (Excluding reserved Values)

### 8.7.2 The ErrorDecode Field

This field defines the context, under which the error occurred.

Value	Meaning/Use	Description
0x80	PNIORW: Application errors of the services Read and Write	Used in the context of Read and Write services handled by either the PROFINET Device stack or by the Application
0x81	PNIO: Other Services	Used in context with any other services.
0x82	Manufacturer Specific: LogBook Entries	Used in context with LogBook entries generated by the stack or by the application.

Table 183: Coding of PNIO status ErrorDecode (Excluding reserved Values)

### 8.7.3 The ErrorCode1 and ErrorCode2 Fields

The ErrorCode1 and ErrorCode2 fields finally specify the error. The meaning of the both fields depends on the value of the ErrorDecode Field.

### 8.7.3.1 ErrorCode1 and ErrorCode2 for ErrorDecode = PNORW

If the field ErrorDecode is set to the value PNORW, the field ErrorCode2 shall be encoded user specific and the ErrorCode1 field is split into an ErrorClass Nibble and an ErrorDecode Nibble as shown in Table 167: Coding of ErrorCode1 for ErrorDecode = PNORW.


**Note:**

As the ErrorCode2 field can be freely chosen in this case, the application may use it to provide more detailed information about the error (e.g. why writing the record into the module was not possible, which value of the parameter was wrong...)

ErrorClass (Bit 7 – 4)	Meaning	ErrorDecode (Bit 3- 0)	Meaning
0xA	Application	0x0	Read Error
		0x1	Write Error
		0x2	Module Failure
		0x7	Busy
		0x8	Version Conflict
		0x9	Feature Not Supported
		0xA-0xF	User Specific
0xB	Access	0x0	Invalid Index
		0x1	Write Length Error
		0x2	Invalid Slot/Subslot
		0x3	Type Conflict
		0x4	Invalid Area/Api
		0x5	State Conflict
		0x6	Access Denied
		0x7	Invalid Range
		0x8	Invalid Parameter
		0x9	Invalid Type
		0xA	Backup
		0xB-0xF	User specific
0xC	Resource	<u>0x0</u>	Read Constrain Conflict
		0x1	Write Constrain Conflict
		0x2	Resource Busy
		0x3	Resource Unavailable
		0x8-0xF	User specific
0xD – 0xF	User Specific	0x0-0xF	User specific

Table 184: Coding of ErrorCode1 for ErrorDecode = PNORW. (Excluding reserved Values)

### 8.7.4 ErrorCode1 and ErrorCode2 for ErrorDecode = PNIO

The meaning of ErrorCode1 and ErrorCode2 for ErrorDecode = PNIO is shown in the following table. This kind of PROFINET status codes should only be used in conjunction with the AR Abort Request Service (See end of table).

Coding of ErrorCode1 and ErrorCode 2 for ErrorDecode = PNIO:

ErrCode1	Description	ErrCode2	Description / Use
0x01	Connect Parameter Error. Faulty ARBlockReq	0x00-0x0D	Error in one of the Block Parameters
0x02	Connect Parameter Error. Faulty IOCRBlockReq	0x00-0x1C	Error in one of the Block Parameters
0x03	Connect Parameter Error. Faulty ExpectedSubmoduleBlockReq	0x00-0x10	Error in one of the Block Parameters
0x04	Connect Parameter Error. Faulty AlarmCRBlockReq	0x00-0x0F	Error in one of the Block Parameters
0x05	Connect Parameter Error. Faulty PrmServerBlockReq	0x00-0x08	Error in one of the Block Parameters
0x06	Connect Parameter Error. Faulty MCRBlockReq	0x00-0x08	Error in one of the Block Parameters
0x07	Connect Parameter Error. Faulty ARRPCBlockReq	0x00-0x04	Error in one of the Block Parameters
0x08	Read Write Record Parameter Error. Faulty Record	0x00-0x0C	Error in one of the Block Parameters
0x09	Connect Parameter Error Faulty IRInfoBlock	0x00-0x05	Error in one of the Block Parameters
0x0A	Connect Parameter Error Faulty SRInfoBlock	0x00-0x05	Error in one of the Block Parameters
0x0B	Connect Parameter Error Faulty ARFSUBlock	0x00-0x05	Error in one of the Block Parameters
0x14	IODControl Parameter Error. Faulty ControlBlockConnect	0x00-0x09	Error in one of the Block Parameters
0x15	IODControl Parameter Error. Faulty ControlBlockPlug	0x00-0x09	Error in one of the Block Parameters
0x16	IOXControl Parameter Error. Faulty ControlBlock after a connection establishment	0x00-0x07	Error in one of the Block Parameters
0x17	IOXControl Parameter Error. Faulty ControlBlock a plug alarm	0x00-0x07	Error in one of the Block Parameters
0x18	IODControl Parameter Error Faulty ControlBlockPrmBegin	0x00-0x09	Error in one of the Block Parameters
0x19	IODControl Parameter Error Faulty SubmoduleListBlock	0x00-0x07	Error in one of the Block Parameters
0x28	Release Parameter Error. Faulty ReleaseBlock	0x00-0x07	Error in one of the Block Parameters
0x32	Response Parameter Error. Faulty ARBlockRes	0x00-0x08	Error in one of the Block Parameters
0x33	Response Parameter Error. Faulty IOCRBlockRes	0x00-0x06	Error in one of the Block Parameters
0x34	Response Parameter Error. Faulty AlarmCRBlockRes	0x00-0x06	Error in one of the Block Parameters
0x35	Response Parameter Error. Faulty ModuleDiffBlock	0x00-0x0D	Error in one of the Block Parameters

ErrCode1	Description	ErrCode2	Description / Use
0x36	Response Parameter Error. Faulty ARPCBlockRes	0x00-0x04	Error in one of the Block Parameters
0x37	Response Parameter Error Faulty ARServerBlockRes	0x00-0x05	Error in one of the Block Parameters
0x3C	AlarmAck Error Codes	0x00	Alarm Type Not Supported
		0x01	Wrong Submodule State
0x3D	CMDEV	0x00	State conflict
		0x01	Resource
		0x02-0xFF	State machine specific
0x3E	CMCTL	0x00	State conflict
		0x01	Timeout
		0x02	No data send
0x3F	CTLDINA	0x00	No DCP active
		0x01	DNS Unknown_RealStationName
		0x02	DCP No_RealStationName
		0x03	DCP Multiple_RealStationName
		0x04	DCP No_StationName
		0x05	No_IP_Addr
		0x06	DCP_Set_Error
0x40	CMRPC	0x00	ArgsLength invalid
		0x01	Unknown Blocks
		0x02	IOCR Missing
		0x03	Wrong AlarmCRBlock count
		0x04	Out of AR Resources
		0x05	AR UUID Unknown
		0x06	State conflict
		0x07	Out of Provider, Consumer or Alarm Resources
		0x08	Out of memory
		0x09	PDev already owned
		0x0A	ARset State conflict during connection establishment
		0x0B	ARset Parameter conflict during connection establishment
0x41	ALPMI	0x00	Invalid state
		0x01	Wrong ACK-PDU
0x42	ALPMR	0x00	Invalid state
		0x01	Wrong Notification PDU
0x43	LMPM	0x00-0xFF	Ethernet Switch Errors
0x44	MAC	0x00-0xFF	
0x45	RPC	0x01	CLRPC_ERR_REJECTED: EPM or Server rejected the call.
		0x02	CLRPC_ERR_FAULTED: Server had fault while executing the call



ErrCode1	Description	ErrCode2	Description / Use
		0x03	CLRPC_ERR_TIMEOUT: EPM or Server did not respond
		0x04	CLRPC_ERR_IN_ARGS; Broadcast or maybe "ndr_data" too large
		0x05	CLRPC_ERR_OUT_ARGS: Server sent back more than "alloc_len"
		0x06	CLRPC_ERR_DECODE: Result of EPM Lookup could not be decoded
		0x07	CLRPC_ERR_PNIO_OUT_ARGS: Out-args not "PN IO signature", too short or inconsistent
		0x08	CLRPC_ERR_PNIO_APP_TIMEOUT: RPC call was terminated after RPC application timeout
0x46	APMR	0x00	Invalid state
		0x01	LMPM signaled an error
0x47	APMS	0x00	Invalid state
		0x01	LMPM signaled an error
		0x02	Timeout
0x48	CPM	0x00	Invalid state
0x49	PPM	0x00	Invalid state
0x4A	DCPUCS	0x00	Invalid state
		0x01	LMPM signaled an error
		0x02	Timeout
0x4B	DCPUCR	0x00	Invalid state
		0x01	LMPM signaled an error
0x4C	DCPMCS	0x00	Invalid state
		0x01	LMPM signaled an error
0x4D	DCPMCR	0x00	Invalid state
		0x01	LMPM signaled an error
0x4E	FSPM	0x00-0xFF	FAL Service Protocol Machine error
0x64	CTLISM	0x00	Invalid state
		0x01	CTLISM signaled error
0x65	CTLRDI	0x00	Invalid state
		0x01	CTLRDI signaled an error
0x66	CTLRDR	0x00	Invalid state
		0x01	CTLRDR signaled an error
0x67	CTLWRI	0x00	Invalid state
		0x01	CTLWRI signaled an error
0x68	CTLWRR	0x00	Invalid State
		0x01	CTLWRR signaled an error
0x69	CTLIO	0x00	Invalid state
		0x01	CTLIO signaled an error
0x6A	CTLSU	0x00	Invalid state
		0x01	AR add provider or consumer failed

ErrCode1	Description	ErrCode2	Description / Use
		0x02	AR alarm-open failed
		0x03	AR alarm-ack-send
		0x04	AR alarm-send
		0x05	AR alarm-ind
0x6B	CTLRPC	0x00	Invalid state
		0x01	CTLRPC signaled an error
0x6C	CTLPBE	0x00	Invalid state
		0x01	CTLPBE signaled an error
0xC8	CMSM	0x00	Invalid state
		0x01	CMSM signaled an error
0xCA	CMRDR	0x00	Invalid state
		0x01	CMRDR signaled an error
0xCC	CMWRR	0x00	Invalid state
		0x01	AR is not in state Primary. (Write not allowed)
		0x02	CMWRR signaled an error
0xCD	CMIO	0x00	Invalid state
		0x01	CMIO signaled an error
0xCE	CMSU	0x00	Invalid state
		0x01	AR add provider or consumer failed
		0x02	AR alarm-open failed
		0x03	AR alarm-send
		0x04	AR alarm-ack-send
		0x05	AR alarm-ind
0xD0	CMINA	0x00	Invalid state
		0x01	CMINA signaled an error
0xD1	CMPBE	0x00	Invalid state
		0x01	CMPBE signaled an error
0xD2	CMDMC	0x00	Invalid state
		0x01	CMDMC signaled an error
0xFD	Used by RTA for protocol error (RTA_ERR_CLS_PROTOCOL)	0x00	Reserved
		0x01	error within the coordination of sequence numbers (RTA_ERR_CODE_SEQ)
		0x02	instance closed (RTA_ERR_ABORT)
		0x03	AR out of memory (RTA_ERR_ABORT)
		0x04	AR add provider or consumer failed (RTA_ERR_ABORT)
		0x05	AR consumer DHT / WDT expired (RTA_ERR_ABORT)
		0x06	AR cmi timeout (RTA_ERR_ABORT)
		0x07	AR alarm-open failed (RTA_ERR_ABORT)
		0x08	AR alarm-send.cnf(-) (RTA_ERR_ABORT)
		0x09	AR alarm-ack- send.cnf(-) (RTA_ERR_ABORT)

ErrCode1	Description	ErrCode2	Description / Use
		0x0A	AR alarm data too long (RTA_ERR_ABORT)
		0x0B	AR alarm.ind(err) (RTA_ERR_ABORT)
		0x0C	AR rpc-client call.cnf(-) (RTA_ERR_ABORT)
		0x0D	AR abort.req (RTA_ERR_ABORT)
		0x0E	AR re-run aborts existing (RTA_ERR_ABORT)
		0x0F	AR release.ind received (RTA_ERR_ABORT)
		0x10	AR device deactivated (RTA_ERR_ABORT)
		0x11	AR removed (RTA_ERR_ABORT)
		0x12	AR protocol violation (RTA_ERR_ABORT)
		0x13	AR name resolution error (RTA_ERR_ABORT)
		0x14	AR RPC-Bind error (RTA_ERR_ABORT)
		0x15	AR RPC-Connect error (RTA_ERR_ABORT)
		0x16	AR RPC-Read error (RTA_ERR_ABORT)
		0x17	AR RPC-Write error (RTA_ERR_ABORT)
		0x18	AR RPC-Control error (RTA_ERR_ABORT)
		0x19	AR forbidden pull or plug after check.rsp and before in- data.ind (RTA_ERR_ABORT)
		0x1A	AR AP removed (RTA_ERR_ABORT)
		0x1B	AR link down (RTA_ERR_ABORT)
		0x1C	AR could not register multicast-mac address (RTA_ERR_ABORT)
		0x1D	not synchronized (cannot start companion-ar) (RTA_ERR_ABORT)
		0x1E	wrong topology (cannot start companion-ar) (RTA_ERR_ABORT)
		0x1F	dcp, station-name changed (RTA_ERR_ABORT)
		0x20	dcp, reset to factory-settings (RTA_ERR_ABORT)
		0x21	cannot start companion-AR because a 0x8ipp submodule in the first AR... (RTA_ERR_ABORT)
		0x22	no irdata record yet (RTA_ERR_ABORT)
		0x23	PDEV (RTA_ERROR_ABORT)
		0x24	PDEV, no port offers required speed / duplexity (RTA_ERR_ABORT)
		0x25	IP-suite [of the IOC] changed by means of DCP set(IPPParameter) or local engineering (RTA_ERR_ABORT)
		0x26	IOCARSR RDHT expired (RTA_ERROR_ABORT)
		0xC9	AR removed by reason of watchdog timeout in the Application task (RTA_ERROR_ABORT)

ErrCode1	Description	ErrCode2	Description / Use
		0xCA	AR removed by reason of pool underflow in the Application task (RTA_ERROR_ABORT)
		0xCB	AR removed by reason of unsuccessful packet sending (Queue) inside OS in the Application task (RTA_ERROR_ABORT)
		0xCC	AR removed by reason of unsuccessful memory allocation in the Application task (RTA_ERROR_ABORT)
0xFF	User specific	0x00-0xFE	User specific
		0xFF	Recommended for "User abort" without further detail

Table 185: Coding of ErrorCode1 for ErrorDecode = PNIO. (Excluding reserved Values)

### 8.7.5 ErrorCode1 and ErrorCode2 for ErrorDecode is Manufacturer Specific

If ErrorDecode is set to Manufacturer Specific, the values of the fields ErrorCode1 and ErrorCode2 could be freely chosen.

## 9 Status/Error Codes Overview

### 9.1 General Errors

#### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC0300001	TLR_E_PNS_IF_COMMAND_INVALID Invalid command.
0xC0300002	TLR_E_PNS_IF_OS_INIT_FAILED Initialization of PNS Operating system adaptation failed.
0xC0300003	TLR_E_PNS_IF_SET_INIT_IP_FAILED Initialization of PNS IP address failed.
0xC0300004	TLR_E_PNS_IF_PNIO_SETUP_FAILED PROFINET IO-Device Setup failed.
0xC0300005	TLR_E_PNS_IF_DEVICE_INFO_ALREADY_SET Device information set already.
0xC0300006	TLR_E_PNS_IF_SET_DEVICE_INFO_FAILED Setting of device information failed.
0xC0300007	TLR_E_PNS_IF_NO_DEVICE_SETUP PROFINET IO-Device stack is not initialized. Send PNS_IF_SET_DEVICEINFO_REQ before PNS_IF_OPEN_DEVICE_REQ.
0xC0300008	TLR_E_PNS_IF_DEVICE_OPEN_FAILED Opening a device instance failed.
0xC0300009	TLR_E_PNS_IF_NO_DEVICE_INSTANCE No device instance open.
0xC030000A	TLR_E_PNS_IF_PLUG_MODULE_FAILED Plugging a module failed.
0xC030000B	TLR_E_PNS_IF_PLUG_SUBMODULE_FAILED Plugging a submodule failed.
0xC030000C	TLR_E_PNS_IF_DEVICE_START_FAILED Start of PROFINET IO-Device failed.
0xC030000D	TLR_E_PNS_IF_EDD_ENABLE_FAILED Start of network communication failed.
0xC030000E	TLR_E_PNS_IF_ALLOC_MNGMNT_BUFFER_FAILED Allocation of a device instance management buffer failed.
0xC030000F	TLR_E_PNS_IF_DEVICE_HANDLE_NULL Given device handle is NULL.
0xC0300010	TLR_E_PNS_IF_SET_APPL_READY_FAILED Command PNS_IF_SET_APPL_READY_REQ failed.
0xC0300011	TLR_E_PNS_IF_SET_DEVSTATE_FAILED Command PNS_IF_SET_DEVSTATE_REQ failed.
0xC0300012	TLR_E_PNS_IF_PULL_SUBMODULE_FAILED Pulling the submodule failed.
0xC0300013	TLR_E_PNS_IF_PULL_MODULE_FAILED Pulling the module failed.
0xC0300014	TLR_E_PNS_IF_WRONG_DEST_ID Destination ID in command invalid.

Hexadecimal Value	Definition Description
0xC0300015	TLR_E_PNS_IF_DEVICE_HANDLE_INVALID Device Handle in command invalid.
0xC0300016	TLR_E_PNS_IF_CALLBACK_TIMEOUT PNS stack callback timeout.
0xC0300017	TLR_E_PNS_IF_PACKET_POOL_EMPTY PNS_IF packet pool empty.
0xC0300018	TLR_E_PNS_IF_ADD_API_FAILED Command PNS_IF_ADD_API_REQ failed.
0xC0300019	TLR_E_PNS_IF_SET_SUB_STATE_FAILED Setting submodule state failed.
0xC030001A	TLR_E_PNS_IF_NO_NW_DBM_ERROR No network configuration DBM-file.
0xC030001B	TLR_E_PNS_IF_NW_SETUP_TABLE_ERROR Error during reading the "SETUP" table of the network configuration DBM-file .
0xC030001C	TLR_E_PNS_IF_CFG_SETUP_TABLE_ERROR Error during reading the "SETUP" table of the config.xxx DBM-file .
0xC030001D	TLR_E_PNS_IF_NO_CFG_DBM_ERROR No config.xxx DBM-file.
0xC030001E	TLR_E_PNS_IF_DBM_DATASET_ERROR Error getting dataset pointer.
0xC030001F	TLR_E_PNS_IF_SETUPEX_TABLE_ERROR Error getting dataset pointer( <i>SETUP_EX</i> table).
0xC0300020	TLR_E_PNS_IF_AP_TABLE_ERROR Error getting either dataset pointer or number of datasets( <i>AP</i> table).
0xC0300021	TLR_E_PNS_IF_MODULES_TABLE_ERROR Error getting either dataset pointer or number of datasets( <i>MODULE</i> table).
0xC0300022	TLR_E_PNS_IF_SUBMODULES_TABLE_ERROR Error getting either dataset pointer or number of datasets( <i>SUBMODULE</i> table).
0xC0300023	TLR_E_PNS_IF_PNIO_SETUP_ERROR Error setting up PNIO configuration( <i>PNIO_setup()</i> ).
0xC0300024	TLR_E_PNS_IF_MODULES_GET_REC Error getting record of " <i>MODULES</i> " linked table.
0xC0300025	TLR_E_PNS_IF_SUBMODULES_GET_REC Error getting record of " <i>SUBMODULES</i> " linked table.
0xC0300026	TLR_E_PNS_IF_PNIOD_MODULE_ID_TABLE_ERROR Error accessing " <i>PNIOD_MODULE_ID</i> " table or table record error.
0xC0300027	TLR_E_PNS_IF_SIGNALS_TABLE_ERROR Error accessing " <i>SIGNALS</i> " table or table record error.
0xC0300028	TLR_E_PNS_IF_MODULES_IO_TABLE_ERROR Error accessing " <i>MODULES_IO</i> " table or table record error.
0xC0300029	TLR_E_PNS_IF_CHANNEL_SETTING_TABLE_ERROR Error accessing " <i>CHANNEL_SETTING</i> " table or table record error.
0xC030002A	TLR_E_PNS_IF_WRITE_DBM Error writing DBM-file.
0xC030002B	TLR_E_PNS_IF_DPM_CONFIG No basic DPM configuration.
0xC030002C	TLR_E_PNS_IF_WATCHDOG Application did not trigger the watchdog.
0xC030002D	TLR_E_PNS_IF_SIGNALS_SUBMODULES Data length in " <i>SIGNALS</i> " table does not correspond to that in " <i>SUBMODULES</i> " table.

Hexadecimal Value	Definition Description
0xC030002E	TLR_E_PNS_IF_READ_DPM_SUBAREA Failed to read DPM subarea.
0xC030002F	TLR_E_PNS_IF_MOD_0_SUB_1 Error configuring Module 0 Submodule 1.
0xC0300030	TLR_E_PNS_IF_SIGNALS_LENGTH Length of I/O signals is bigger then the size of DPM subarea.
0xC0300031	TLR_E_PNS_IF_SUB_TRANSFER_DIRECTION A submodule cannot have input and outputs at the same time.
0xC0300032	TLR_E_PNS_IF_FORMAT_PNVOLUME Error while formatting PNVOLUME.
0xC0300033	TLR_E_PNS_IF_MOUNT_PNVOLUME Error while mounting PNVOLUME.
0xC0300034	TLR_E_PNS_IF_INIT_REMOTE Error during initialization of the remote resources of the stack.
0xC0300035	TLR_E_PNS_IF_WARMSTART_CONFIG_REDUNDANT Warmstart parameters are redundant. The stack was configured with DBM or packets.
0xC0300036	TLR_E_PNS_IF_WARMSTART_PARAMETER Incorrect warmstart parameter(s).
0xC0300037	TLR_E_PNS_IF_SET_APPL_STATE_READY PNIO_set_appl_state_ready() returns error.
0xC0300038	TLR_E_PNS_IF_SET_DEV_STATE PNIO_set_dev_state() returns error.
0xC0300039	TLR_E_PNS_IF_PROCESS_ALARM_SEND PNIO_process_alarm_send() returns error.
0xC030003A	TLR_E_PNS_IF_RET_OF_SUB_ALARM_SEND PNIO_ret_of_sub_alarm_send() returns error.
0xC030003B	TLR_E_PNS_IF_DIAG_ALARM_SEND PNIO_diag_alarm_send() returns error.
0xC030003C	TLR_E_PNS_IF_DIAG_GENERIC_ADD PNIO_diag_generic_add() returns error.
0xC030003D	TLR_E_PNS_IF_DIAG_GENERIC_REMOVE PNIO_diag_generic_remove() returns error.
0xC030003E	TLR_E_PNS_IF_DIAG_CHANNEL_ADD PNIO_diag_channel_add() returns error.
0xC030003F	TLR_E_PNS_IF_DIAG_CHANNEL_REMOVE PNIO_diag_channel_remove() returns error.
0xC0300040	TLR_E_PNS_IF_EXT_DIAG_CHANNEL_ADD PNIO_ext_diag_channel_add() returns error.
0xC0300041	TLR_E_PNS_IF_EXT_DIAG_CHANNEL_REMOVE PNIO_ext_diag_channel_remove() returns error.
0xC0300042	TLR_E_PNS_IF_STATION_NAME_LEN Parameter station name length is incorrect.
0xC0300043	TLR_E_PNS_IF_STATION_NAME Parameter station name is incorrect.
0xC0300044	TLR_E_PNS_IF_STATION_TYPE_LEN Parameter station type length is incorrect.
0xC0300045	TLR_E_PNS_IF_DEVICE_TYPE Parameter device type is incorrect.
0xC0300046	TLR_E_PNS_IF_ORDER_ID Parameter order id is incorrect.

Hexadecimal Value	Definition Description
0xC0300047	TLR_E_PNS_IF_INPUT_STATUS Parameter input data status bytes length is incorrect.
0xC0300048	TLR_E_PNS_IF_OUTPUT_STATUS Parameter output data status bytes length is incorrect.
0xC0300049	TLR_E_PNS_IF_WATCHDOG_PARAMETER Parameter watchdog timing is incorrect(must be >= 10).
0xC030004A	TLR_E_PNS_IF_OUT_UPDATE Parameter output data update timing is incorrect.
0xC030004B	TLR_E_PNS_IF_IN_UPDATE Parameter input data update timing is incorrect.
0xC030004C	TLR_E_PNS_IF_IN_SIZE Parameter input memory area size is incorrect.
0xC030004D	TLR_E_PNS_IF_OUT_SIZE Parameter output memory area size is incorrect.
0xC030004E	TLR_E_PNS_IF_GLOBAL_RESOURCES Unable to allocate memory for global access to local resources.
0xC030004F	TLR_E_PNS_IF_DYNAMIC_CFG_PCK Unable to allocate memory for dynamic configuration packet.
0xC0300050	TLR_E_PNS_IF_DEVICE_STOP Unable to stop device.
0xC0300051	TLR_E_PNS_IF_DEVICE_ID Parameter device id is incorrect.
0xC0300052	TLR_E_PNS_IF_VENDOR_ID Parameter vendor id is incorrect.
0xC0300053	TLR_E_PNS_IF_SYS_START Parameter system start is incorrect.
0xC0300054	TLR_E_PNS_IF_DYN_CFG_IO_LENGTH The length of IO data expected by the controller exceeds the limit specified in warmstart parameters.
0xC0300055	TLR_E_PNS_IF_DYN_CFG_MOD_NUM The count of the IO modules expected by the controller exceeds the supported by the stack count.
0xC0300056	TLR_E_PNS_IF_ACCESS_LOCAL_RSC No global access to local resources.
0xC0300057	TLR_E_PNS_IF_PULL_PLUG Plugging and pulling modules during creation of communication is not allowed.
0xC0300058	TLR_E_PNS_IF_AR_NUM Maximum number of ARs is 1.
0xC0300059	TLR_E_PNS_IF_API_NUM Only API = 0 is supported.
0xC030005A	TLR_E_PNS_IF_ALREADY_OPEN Device is already opened.
0xC030005B	TLR_E_PNS_IF_API_ADDED Application is already added.
0xC030005C	TLR_E_PNS_IF_CONFIG_MODE Configuration modes should not be mixed( DBM-files,application,warmstart message ).
0xC030005D	TLR_E_PNS_IF_UNK_LED_MODE Unknown LED mode.
0xC030005E	TLR_E_PNS_IF_PHYSICAL_LINK Physical link rate is less then 100 MBit.



Hexadecimal Value	Definition Description
0xC030005F	TLR_E_PNS_IF_MAX_SLOT_SUBSLOT Number of slots or subslots too big.
0xC0300060	TLR_E_PNS_IF_AR_REASON_MEM AR error. Out of memory.
0xC0300061	TLR_E_PNS_IF_AR_REASON_FRAME AR error. Add provider or consumer failed.
0xC0300062	TLR_E_PNS_IF_AR_REASON_MISS AR error. Consumer missing.
0xC0300063	TLR_E_PNS_IF_AR_REASON_TIMER AR error. CMI timeout.
0xC0300064	TLR_E_PNS_IF_AR_REASON_ALARM AR error. Alarm open failed.
0xC0300065	TLR_E_PNS_IF_AR_REASON_ALSND AR error. Alarm send confirmation failed.
0xC0300066	TLR_E_PNS_IF_AR_REASON_ALACK AR error. Alarm acknowledge send confirmation failed.
0xC0300067	TLR_E_PNS_IF_AR_REASON_ALLEN AR error. Alarm data too long.
0xC0300068	TLR_E_PNS_IF_AR_REASON_ASRT AR error. Alarm indication error.
0xC0300069	TLR_E_PNS_IF_AR_REASON_RPC AR error. RPC client call confirmation failed.
0xC030006A	TLR_E_PNS_IF_AR_REASON_ABORT AR error. Abort request.
0xC030006B	TLR_E_PNS_IF_AR_REASON_RERUN AR error. Re-Run.
0xC030006C	TLR_E_PNS_IF_AR_REASON_REL AR error. Release indication received.
0xC030006D	TLR_E_PNS_IF_AR_REASON_PAS AR error. Device deactivated.
0xC030006E	TLR_E_PNS_IF_AR_REASON_RMV AR error. Device/AR removed.
0xC030006F	TLR_E_PNS_IF_AR_REASON_PROT AR error. Protocol violation.
0xC0300070	TLR_E_PNS_IF_AR_REASON_NARE AR error. NARE error.
0xC0300071	TLR_E_PNS_IF_AR_REASON_BIND AR error. RPC-Bind error.
0xC0300072	TLR_E_PNS_IF_AR_REASON_CONNECT AR error. RPC-Connect error.
0xC0300073	TLR_E_PNS_IF_AR_REASON_READ AR error. RPC-Read error.
0xC0300074	TLR_E_PNS_IF_AR_REASON_WRITE AR error. RPC-Write error.
0xC0300075	TLR_E_PNS_IF_AR_REASON_CONTROL AR error. RPC-Control error.
0xC0300076	TLR_E_PNS_IF_AR_REASON_UNKNOWN AR error. Unknown.
0xC0300077	TLR_E_PNS_IF_INIT_WATCHDOG Watchdog initialization failed.

Hexadecimal Value	Definition Description
0xC0300078	TLR_E_PNS_IF_NO_PHYSICAL_LINK The Device is not connected to a network.
0xC0300079	TLR_DPM_CYCLIC_IO_RW Failed to copy from DPM or to DPM the cyclic IO data.
0xC030007A	TLR_E_PNS_IF_SUBMODULE Submodule number is wrong.
0xC030007B	TLR_E_PNS_IF_MODULE Module number is wrong.
0xC030007C	TLR_E_PNS_IF_NO_AR The AR was closed or the AR handle is not valid.
0xC030007D	TLR_E_PNS_IF_WRITE_REC_RES_TIMEOUT Timeout while waiting for response to write_record_indication.
0xC030007E	TLR_E_PNS_IF_UNREGISTERED_SENDER The sender of the request is not registered with request PNS_IF_REGISTER_AP_REQ.
0xC030007F	TLR_E_PNS_IF_RECORD_HANDLE_INVALID Unknown record handle.
0xC0300080	TLR_E_PNS_IF_REGISTER_AP Another instance is registered at the moment.
0xC0300081	TLR_E_PNS_IF_UNREGISTER_AP One instance cannot unregister another one.
0xC0300082	TLR_E_PNS_IF_CONFIG_DIFFER The Must-configuration differs from the Is-configuration.
0xC0300083	TLR_E_PNS_IF_NO_COMMUNICATION No communication processing.
0xC0300084	TLR_E_PNS_IF_BAD_PARAMETER At least one parameter in a packet was wrong or/and did not meet the requirements.
0xC0300085	TLR_E_PNS_IF_AREA_OVERFLOW Input or Output data requires more space than available.
0xC0300086	TLR_E_PNS_IF_WRM_PCK_SAVE Saving Warmstart Configuration for later use was not successful.
0xC0300087	TLR_E_PNS_IF_AR_REASON_PULLPLUG AR error. Pull and Plug are forbidden after check.rsp and before in-data.ind.
0xC0300088	TLR_E_PNS_IF_AR_REASON_AP_RMV AR error. AP has been removed.
0xC0300089	TLR_E_PNS_IF_AR_REASON_LNK_DWN AR error. Link "down".
0xC030008A	TLR_E_PNS_IF_AR_REASON_MMAC AR error. Could not register multicast-MAC.
0xC030008B	TLR_E_PNS_IF_AR_REASON_SYNC AR error. Not synchronized (Cannot start companion-AR).
0xC030008C	TLR_E_PNS_IF_AR_REASON_TOPO AR error. Wrong topology (Cannot start companion-AR).
0xC030008D	TLR_E_PNS_IF_AR_REASON_DCP_NAME AR error. DCP. Station Name changed.
0xC030008E	TLR_E_PNS_IF_AR_REASON_DCP_RESET AR error. DCP. Reset to factory-settings.
0xC030008F	TLR_E_PNS_IF_AR_REASON_PRM AR error. Cannot start companion-AR because a 0x8ipp submodule in the first AR /has appl-ready-pending/ is locked/ is wrong or pulled/ .

Hexadecimal Value	Definition Description
0xC0300090	TLR_E_PNS_IF_PACKET_MNGMNT Packet management error.
0xC03000B1	TLR_E_PNS_IF_RESET_FACTORY_IND A module was already plugged to the slot.
0xC03000B2	TLR_E_PNS_IF_MODULE_ALREADY_PLUGGED Failed to init the OS adaptation layer.
0xC03000B3	TLR_E_PNS_IF_OSINIT Failed to init the TCPIP adaptation layer.
0xC03000B4	TLR_E_PNS_IF_OSSOCKINIT Failed to init the TCPIP adaptation layer.
0xC03000B5	TLR_E_PNS_IF_INVALID_NETMASK Invalid subnetwork mask.
0xC03000B6	TLR_E_PNS_IF_INVALID_IP_ADDR Invalid IP address.
0xC03000B7	TLR_E_PNS_IF_STA_STARTUP_PARAMETER Erroneous Task start-up parameters.
0xC03000B8	TLR_E_PNS_IF_INIT_LOCAL Failed to initialize the Task local resources.
0xC03000B9	TLR_E_PNS_IF_APP_CONFIG_INCOMPLETE The configuration per packets is incomplete.
0xC03000BA	TLR_E_PNS_IF_INIT_EDD EDD Initialization failed.
0xC03000BB	TLR_E_PNS_IF_DPM_NOT_ENABLED DPM is not enabled.
0xC03000BC	TLR_E_PNS_IF_READ_LINK_STATUS Reading Link Status failed.
0xC03000BD	TLR_E_PNS_IF_INVALID_GATEWAY Invalid gateway address (not reachable with configured net mask).
0xC0300100	TLR_E_PNS_IF_PACKET_SEND_FAILED Error while sending a packet to another task.
0xC0300101	TLR_E_PNS_IF_RESOURCE_OUT_OF_MEMORY Insufficient memory to handle the request.
0xC0300102	TLR_E_PNS_IF_NO_APPLICATION_REGISTERED No application to send the indication to is registered.
0xC0300103	TLR_E_PNS_IF_INVALID_SOURCE_ID The host-application returned a packet with invalid (changed) SourceID.
0xC0300104	TLR_E_PNS_IF_PACKET_BUFFER_FULL The buffer used to store packets exchanged between host-application and stack is full.
0xC0300105	TLR_E_PNS_IF_PULL_NO_MODULE Pulling the (sub)module failed because no module is plugged into the slot specified.
0xC0300106	TLR_E_PNS_IF_PULL_NO_SUBMODULE Pulling the submodule failed because no submodule is plugged into the subslot specified.
0xC0300107	TLR_E_PNS_IF_PACKET_BUFFER_RESTORE_ERROR The packet buffer storing packets exchanged between host-application and stack returned an invalid packet.
0xC0300108	TLR_E_PNS_IF_DIAG_NO_MODULE Diagnosis data not accepted because no module is plugged into the slot specified.
0xC0300109	TLR_E_PNS_IF_DIAG_NO_SUBMODULE Diagnosis data not accepted because no submodule is plugged into the subslot specified.

Hexadecimal Value	Definition Description
0xC030010A	TLR_E_PNS_IF_CYCLIC_EXCHANGE_ACTIVE The services requested is not available while cyclic communication is running.
0xC030010B	TLR_E_PNS_IF_FATAL_ERROR_CLB_ALREADY_REGISTERED This fatal error callback function could not be registered because there is already a function registered.
0xC030010C	TLR_E_PNS_IF_ERROR_STACK_WARMSTART_CONFIGURATION The stack did not accept the warmstart parameters.
0xC030010D	TLR_E_PNS_IF_ERROR_STACK_MODULE_CONFIGURATION The stack did not accept the module configuration packet.
0xC030010E	TLR_E_PNS_IF_CHECK_IND_FOR_UNEXPECTED_MODULE The stack sent a Check Indication for an unexpected module.
0xC030010F	TLR_E_PNS_IF_CHECK_IND_FOR_UNEXPECTED_SUBMODULE The stack sent a Check Indication for an unexpected submodule.
0xC0300110	TLR_E_PNS_DIAG_BUFFER_FULL No more diagnosis records can be added to the stack because the maximum amount is already reached.
0xC0300111	TLR_E_PNS_IF_CHECK_IND_FOR_UNEXPECTED_API The stack sent a Check Indication for an unexpected API.
0xC0300112	TLR_E_PNS_IF_DPM_ACCESS_WITH_INVALID_OFFSET The DPM shall be accessed with an invalid data offset.
0xC0300113	TLR_E_PNS_IF_DUPLICATE_INPUT_CR_INFO The stack indicated to CR Info Indications with type input.
0xC0300114	TLR_E_PNS_IF_DUPLICATE_OUTPUT_CR_INFO The stack indicated to CR Info Indications with type output.
0xC0300115	TLR_E_PNS_IF_FAULTY_CR_INFO_IND_RECEIVED The stack indicated a faulty CR Info Indications.
0xC0300116	TLR_E_PNS_IF_CONFIG_RELOAD_RUNNING The request cannot be executed because configuration reload respectively <i>Channellnit</i> is running.
0xC0300117	TLR_E_PNS_IF_NO_MAC_ADDRESS_SET There is no valid chassis MAC address set Without MAC address the stack will not work.
0xC0300118	TLR_E_PNS_IF_SET_PORT_MAC_NOT_POSSIBLE The Port MAC addresses have to be set before sending Set-Configuration Request to the stack.
0xC030011A	TLR_E_PNS_IF_INVALID_MODULE_CONFIGURATION Evaluating the module configuration failed.
0xC030011B	TLR_E_PNS_IF_CONF_IO_LEN_TO_BIG The sum of IO-data length exceeds the maximum allowed value.
0xC030011C	TLR_E_PNS_IF_NO_MODULE_CONFIGURED The module configuration does not contain at least one module.
0xC030011D	TLR_E_PNS_IF_INVALID_SW_REV_PREFIX The value of bSwRevisionPrefix is invalid.
0xC030011E	TLR_E_PNS_IF_RESERVED_VALUE_NOT_ZERO The value of usReserved it not zero.
0xC030011F	TLR_E_PNS_IF_IDENTIFY_CMDEV_QUEUE_FAILED Identifying the stack message queue CMDEV failed.
0xC0300120	TLR_E_PNS_IF_CREATE_SYNC_QUEUE_FAILED Creating the sync message queue failed.
0xC0300121	TLR_E_PNS_IF_CREATE_ALARM_LOW_QUEUE_FAILED Creating the low alarm message queue failed.

Hexadecimal Value	Definition Description
0xC0300122	TLR_E_PNS_IF_CREATE_ALARM_HIGH_QUEUE_FAILED Creating the high alarm message queue failed.
0xC0300123	TLR_E_PNS_IF_CFG_PACKET_TO_SMALL While evaluating SetConfiguration packet the packet length was found smaller than amount of configured modules needs.
0xC0300124	TLR_E_PNS_IF_FATAL_ERROR_OCCURRED A fatal error occurred prior to this request. Therefore this request cannot be fulfilled.
0xC0300125	TLR_E_PNS_IF_SUBMODULE_NOT_IN_CYCLIC_EXCHANGE The request could not be executed because the submodule is not in cyclic data exchange.
0xC0300126	TLR_E_PNS_IF_SERVICE_NOT_AVAILABLE_THROUGH_DPM This service is not available through DPM.
0xC0300127	TLR_E_PNS_IF_INVALID_PARAMETER_VERSION The version of parameters is invalid (most likely too old).
0xC0300128	TLR_E_PNS_IF_DATABASE_USAGE_IS_FORBIDDEN The usage of database is forbidden by task's startup parameters.
0xC0300129	TLR_E_PNS_IF_RECORD_LENGTH_TOO_BIG The amount of record data is too big.
0xC030012A	TLR_E_PNS_IF_IDENTIFY_LLDP_QUEUE_FAILED Identifying the stack message queue LLDP failed.
0xC030012BL	TLR_E_PNS_IF_INVALID_TOTAL_PACKET_LENGTH SetConfiguration Requests total packet length is invalid.
0xC030012CL	TLR_E_PNS_IF_APPLICATION_TIMEOUT The application needed to much time to respond to an indication.
0xC030012DL	TLR_E_PNS_IF_PACKET_BUFFER_INVALID_PACKET The packet buffer storing packets exchanged between host-application and stack returned a faulty packet.
0xC030012EL	TLR_E_PNS_IF_NO_IO_IMAGE_CONFIGURATION_AVAILABLE The request cannot be handled until a valid IO Image configuration is available.
0xC030012FL	TLR_E_PNS_IF_IO_IMAGE_ALREADY_CONFIGURED A valid IO Image configuration is already available.
0xC0300130L	TLR_E_PNS_IF_INVALID_PDEV_SUBSLOT A submodule may only be plugged into a PDEV-subslot which does not exceed the number of supported interfaces and port numbers.
0xC0300131L	TLR_E_PNS_IF_NO_DAP_PRESENT The module configuration does not contain a the Device Access Point DAP-submodule in slot 0 subslot 1.
0xC0300123	TLR_E_PNS_IF_PLUG_SUBMOD_OUTPUT_SIZE_EXCEEDED Plugging the submodule would exceed the maximum output data length. (ulCompleteOutputSize of set configuration service)
0xC0300133	TLR_E_PNS_IF_PLUG_SUBMOD_INPUT_SIZE_EXCEEDED Plugging the submodule would exceed the maximum input data length. (ulCompleteInputSize of Set configuration service)
0xC0300134	TLR_E_PNS_IF_PLUG_SUBMOD_NO_MODULE_ATTACHED_TO_ADD_TO No module attached to add the submodule to.
0xC0300135	TLR_E_PNS_IF_PLUG_SUBMOD_ALREADY_PLUGGED_THIS_SUBMOD Submodule already plugged.
0xC0300136	TLR_E_PNS_IF_SETIOXS_INVALID_PROV_IMAGE Invalid IOXS provider image.
0xC0300137	TLR_E_PNS_IF_SETIOXS_INVALID_CONS_IMAGE Invalid IOXS consumer image.

Hexadecimal Value	Definition Description
0xC0300138	TLR_E_PNS_IF_INVALID_IOPS_MODE Invalid IOPS mode.
0xC0300139	TLR_E_PNS_IF_INVALID_IOCS_MODE Invalid IOCS mode.
0xC030013A	TLR_E_PNS_IF_INVALID_API Invalid API.
0xC030013B	TLR_E_PNS_IF_INVALID_SLOT Invalid slot.
0xC030013C	TLR_E_PNS_IF_INVALID_SUBSLOT Invalid subslot.
0xC030013D	TLR_E_PNS_IF_INVALID_CHANNEL_NUMBER Invalid channel number.
0xC030013E	TLR_E_PNS_IF_INVALID_CHANNEL_PROPERTIES Invalid channel properties.
0xC030013F	TLR_E_PNS_IF_CHANNEL_ERRORTYPE_NOT_ALLOWED Invalid channel errortype not allowed.
0xC0300140	TLR_E_PNS_IF_EXT_CHANNEL_ERRORTYPE_NOT_ALLOWED Invalid channel EXT errortype not allowed.
0xC0300141	TLR_E_PNS_IF_INVALID_USER_STRUCT_IDENTIFIER Invalid user struct identifier.
0xC0300142	TLR_E_PNS_IF_INVALID_SUBMODULE Invalid submodule.
0xC0300143	TLR_E_PNS_IF_INVALID_IM_TYPE Invalid IM type.
0xC0300144	TLR_E_PNS_IF_IDENTIFY_FODMI_QUEUE_FAILED Failed to identify the FODMI Queue.
0xC0300145	TLR_E_PNS_IF_DPM_MAILBOX_OVERFLOW The DPM Receive Mailbox Queue run out of space. Most likely the host did not fetch the packets.
0xC0300A03	TLR_E_PNS_IF_CM_AR_REASON_MEM AR Out of memory.
0xC0300A04	TLR_E_PNS_IF_CM_AR_REASON_FRAME AR add provider or consumer failed.
0xC0300A05	TLR_E_PNS_IF_CM_AR_REASON_MISS AR consumer DHT/WDT expired.
0xC0300A06	TLR_E_PNS_IF_CM_AR_REASON_TIMER AR cmi timeout.
0xC0300A07	TLR_E_PNS_IF_CM_AR_REASON_ALARM AR alarm-open failed.
0xC0300A08	TLR_E_PNS_IF_CM_AR_REASON_ALSND AR alarm-send.cnf(-).
0xC0300A09	TLR_E_PNS_IF_CM_AR_REASON_ALACK AR alarm-ack-send.cnf(-).
0xC0300A0A	TLR_E_PNS_IF_CM_AR_REASON_ALLEN AR alarm data too long.
0xC0300A0B	TLR_E_PNS_IF_CM_AR_REASON_ASRT AR alarm.ind(err).
0xC0300A0C	TLR_E_PNS_IF_CM_AR_REASON_RPC AR rpc-client call.cnf(-).

Hexadecimal Value	Definition Description
0xC0300A0D	TLR_E_PNS_IF_CM_AR_REASON_ABORT AR abort.req.
0xC0300A0E	TLR_E_PNS_IF_CM_AR_REASON_RERUN AR re-run aborts existing AR.
0xC0300A0F	TLR_E_PNS_IF_CM_AR_REASON_REL AR release.ind received.
0xC0300A10	TLR_E_PNS_IF_CM_AR_REASON_PAS AR device deactivated.
0xC0300A11	TLR_E_PNS_IF_CM_AR_REASON_RMV AR removed.
0xC0300A12	TLR_E_PNS_IF_CM_AR_REASON_PROT AR protocol violation.
0xC0300A13	TLR_E_PNS_IF_CM_AR_REASON_NARE AR name resolution error.
0xC0300A14	TLR_E_PNS_IF_CM_AR_REASON_BIND AR RPC-Bind error.
0xC0300A15	TLR_E_PNS_IF_CM_AR_REASON_CONNECT AR RPC-Connect error.
0xC0300A16	TLR_E_PNS_IF_CM_AR_REASON_READ AR RPC-Read error.
0xC0300A17	TLR_E_PNS_IF_CM_AR_REASON_WRITE AR RPC-Write error.
0xC0300A18	TLR_E_PNS_IF_CM_AR_REASON_CONTROL AR RPC-Control error.
0xC0300A19	TLR_E_PNS_IF_CM_AR_REASON_PULLPLUG AR forbidden pull or plug after check.rsp and before in-data.ind.
0xC0300A1A	TLR_E_PNS_IF_CM_AR_REASON_AP_RMV AR AP removed.
0xC0300A1B	TLR_E_PNS_IF_CM_AR_REASON_LNK_DWN AR link down.
0xC0300A1C	TLR_E_PNS_IF_CM_AR_REASON_MMAC AR could not register multicast-MAC address.
0xC0300A1D	TLR_E_PNS_IF_CM_AR_REASON_SYNC Not synchronized (cannot start companion-ar).
0xC0300A1E	TLR_E_PNS_IF_CM_AR_REASON_TOPO Wrong topology (cannot start companion-ar).
0xC0300A1F	TLR_E_PNS_IF_CM_AR_REASON_DCP_NAME DCP, station-name changed.
0xC0300A20	TLR_E_PNS_IF_CM_AR_REASON_DCP_RESET DCP, reset to factory-settings.
0xC0300A21	TLR_E_PNS_IF_CM_AR_REASON_PRM 0x8ipp submodule in the first AR has either an appl-ready-pending (erroneous parameterization) or is locked (no parameterization) or is wrong or pulled (no parameterization).
0xC0300A22	TLR_E_PNS_IF_CM_AR_REASON_IRDATA No irdata record yet.
0xC0300A23	TLR_E_PNS_IF_CM_AR_REASON_PDEV Ownership of PDEV.
0xC0300A24	TLR_E_PNS_IF_IDENTIFY_FODMI_QUEUE_FAILED Identifying the stack message queue FODMI failed.

Hexadecimal Value	Definition Description
0xC0910001	TLR_E_IO_SIGNAL_COMMAND_INVALID Invalid command received.
0xC0910002	TLR_E_IO_SIGNAL_INVALID_SIGNAL_DIRECTION The value of signal direction is invalid.
0xC0910003	TLR_E_IO_SIGNAL_INVALID_SIGNAL_AMOUNT The value of signal amount is invalid.
0xC0910004	TLR_E_IO_SIGNAL_INVALID_SIGNAL_TYPE The value of signal type is invalid.
0xC0910005	TLR_E_IO_SIGNAL_UNSUPPORTED_SIGNAL_TYPE The value of signal type is unsupported.

Table 186: Status/Error Codes Overview

## 9.2 Status/Error Codes for CMCTL Task

### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC00A0001	TLR_E_PNIO_CMCTL_COMMAND_INVALID Received invalid command in CMCTL task.
0xC00A0002	TLR_E_PNIO_STATUS Generic error code. See packets data-status code for details.
0xC00A0010	TLR_E_PNIO_CMCTL_INIT_PARAM_INVALID Invalid parameter in CMCTL_ResourceInit().
0xC00A0011	TLR_E_PNIO_CMCTL_RESOURCE_LIMIT_EXCEEDED No more CMCTL protocol machines possible.
0xC00A0012	TLR_E_PNIO_CMCTL_RESOURCE_OUT_OF_MEMORY Insufficient memory for this request to CMCTL.
0xC00A0013	TLR_E_PNIO_CMCTL_CLOSED This CMCTL protocol machine was closed.
0xC00A0014	TLR_E_PNIO_CMCTL_STATE_CONFLICT This request can not be served in current CMCTL state.
0xC00A0015	TLR_E_PNIO_CMCTL_CONFIG_PENDING The state of CMCTL's management resource is pending.
0xC00A0016	TLR_E_PNIO_CMCTL_CONFIG_STATE_INVALID The state of CMCTL's management resource is invalid.
0xC00A0017	TLR_E_PNIO_CMCTL_PACKET_OUT_OF_MEMORY Insufficient memory to create a packet in CMCTL task.
0xC00A0018	TLR_E_PNIO_CMCTL_PACKET_SEND_FAILED Error while sending a packet to another task in CMCTL.
0xC00A0019	TLR_E_PNIO_CMCTL_CONN_REQ_LEN_INVALID The length of the Connect-Packet in CMCTL_Connect_req() is invalid.
0xC00A001A	TLR_E_PNIO_CMCTL_NAME_LEN_INVALID The length of the name for IO-Device does not match to the name in CMCTL_Connect_req().
0xC00A001B	TLR_E_PNIO_CMCTL_BLKNUM_UNEXPECTED The Connect-Confirmation contains an incorrect amount of blocks.



Hexadecimal Value	Definition Description
0xC00A001C	TLR_E_PNIO_CMCTL_BLKNUM_UNEXPECTED_MEMORY_FAULT The Connect-Confirmation contains an incorrect amount of blocks but may be received correctly in RPC-layer. CMCTL protocol-machine has not reserved enough memory for the whole confirmation.
0xC00A001D	TLR_E_PNIO_CMCTL_INVALID_FRAMEID_RECEIVED The Connect-Response from IO-Device specified an invalid FrameID to use for IO-Controllers OutputCR.
0xC00A001E	TLR_E_PNIO_CMCTL_EMPTY_POOL_DETECTED The packet pool of CMCTL is empty.
0xC00A0020	TLR_E_PNIO_CMCTL_BLKTYPE_UNEXPECTED The connect-confirmation contains an unexpected block.
0xC00A0021	TLR_E_PNIO_CMCTL_BLKTYPE_UNEXPECTED_INIT CMCTL_Connect_req() expected an INIT-block that is missing.
0xC00A0022	TLR_E_PNIO_CMCTL_BLKTYPE_UNEXPECTED_IODW_REQ CMCTL_RMWrite_req() expected a WriteReq-block that is missing.
0xC00A0023	TLR_E_PNIO_CMCTL_BLKTYPE_UNEXPECTED_IODW_DATA CMCTL_RMWrite_req() expected a WriteData-block that is missing.
0xC00A0030	TLR_E_PNIO_CMCTL_BLKLEN_INVALID_INIT INIT-block length for CMCTL_Connect_req() is invalid.
0xC00A0031	TLR_E_PNIO_CMCTL_BLKLEN_INVALID_IODW_REQ WriteReq-block's length for CMCTL_RMWrite_req() is invalid.
0xC00A0032	TLR_E_PNIO_CMCTL_BLKLEN_INVALID_IODW_DATA WriteData-block's length for CMCTL_RMWrite_req() is invalid.
0xC00A0040	TLR_E_PNIO_CMCTL_INVALID_PM_INDEX The index of CMCTL protocol-machine is invalid.
0xC00A0041	TLR_E_PNIO_CMCTL_INVALID_PM The CMCTL protocol-machine corresponding to index is invalid.
0xC00A0042	TLR_E_PNIO_CMCTL_INVALID_CMCTL_HANDLE The handle to CMCTL protocol-machine is invalid.
0xC00A0050	TLR_E_PNIO_CMCTL_DEVICE_NOT_RESPONDING The IO-Device which shall be connected does not answer.
0xC00A0051	TLR_E_PNIO_CMCTL_DUPLICATE_DEVICE_NAME_DETECTED More than one IO-Device with the specified NameOfStation exists; a connection can not be established.
0xC00A0052	TLR_E_PNIO_CMCTL_DEVICE_IP_ADDRESS_ALREADY_IN_USE The IP-address the controller shall use for the IO-Device is already in use by another network device; a connection can not be established.
0xC00A0060	TLR_E_PNIO_CMCTL_RPC_CONNECT_FAILED The Connect-Response of IO-Device contained an error code; a connection could not be established.
0xC00A0061	TLR_E_PNIO_CMCTL_RPC_WRITE_PARAM_FAILED The Write_Param-Response of IO-Device contained an error code; a connection could not be established.
0xC00A0062	TLR_E_PNIO_CMCTL_RPC_WRITE_FAILED The Write-Response of IO-Device contained an error code.
0xC00A0063	TLR_E_PNIO_CMCTL_RPC_READ_FAILED The Read-Response of IO-Device contained an error code.
0xC00A0064	TLR_E_PNIO_CMCTL_TCP_IP_SHUTDOWN The TCP/IP-Stack closed a socket needed for communication.
0xC00A0065	TLR_E_PNIO_CMCTL_RPC_RESPONSE_TOO_SHORT The RPC-Response received does not have the required minimum length.

Hexadecimal Value	Definition Description
0xC00A0070	TLR_E_PNIO_CMCTL_AR_BLOCKTYPE The expected configuration block for AR in CMCTL_RMConnect_req_LoadAr() is missing.
0xC00A0071	TLR_E_PNIO_CMCTL_AR_BLOCKLEN The expected configuration block for AR in CMCTL_RMConnect_req_LoadAr() has an invalid length.
0xC00A0072	TLR_E_PNIO_CMCTL_AR_TYPE The configuration block for AR in CMCTL_RMConnect_req_LoadAr() has an invalid type.
0xC00A0073	TLR_E_PNIO_CMCTL_AR_UUID The configuration block for AR in CMCTL_RMConnect_req_LoadAr() has an invalid UUID.
0xC00A0074	TLR_E_PNIO_CMCTL_AR_PROPERTY The configuration block for AR in CMCTL_RMConnect_req_LoadAr() has an invalid network properties value.
0xC00A0075	TLR_E_PNIO_CMCTL_AR_REF_UNEXPECTED The AR-Reference for CMCTL protocol-machine is invalid.
0xC00A0076	TLR_E_PNIO_CMCTL_AR_UUID_COMP_FAILED The UUID inside IO-Device's Connect-Confirmation is incorrect.
0xC00A0077	TLR_E_PNIO_CMCTL_AR_KEY_COMP_FAILED The session-key inside IO-Device's Connect-Confirmation is incorrect.
0xC00A0078	TLR_E_PNIO_CMCTL_AR_MAC_COMP_FAILED The MAC-address of IO-Device is incorrect.
0xC00A0080	TLR_E_PNIO_CMCTL_ALCR_BLOCKTYPE The expected configuration block for Alarm-CR in CMCTL_RMConnect_req_LoadAlcr() is missing.
0xC00A0081	TLR_E_PNIO_CMCTL_ALCR_BLOCKLEN The expected configuration block for Alarm-CR in CMCTL_RMConnect_req_LoadAlcr() has an invalid length.
0xC00A0082	TLR_E_PNIO_CMCTL_ALCR_TYPE The configuration block for Alarm-CR in CMCTL_RMConnect_req_LoadAlcr() has an invalid type.
0xC00A0083	TLR_E_PNIO_CMCTL_ALCR_PROPERTY The configuration block for Alarm-CR in CMCTL_RMConnect_req_LoadAlcr() has an invalid network properties value.
0xC00A0084	TLR_E_PNIO_CMCTL_ALCR_RTA_FACTOR The configuration block for Alarm-CR in CMCTL_RMConnect_req_LoadAlcr() has an invalid RTA-factor.
0xC00A0085	TLR_E_PNIO_CMCTL_ALCR_RTA_RETRY The configuration block for Alarm-CR in CMCTL_RMConnect_req_LoadAlcr() has an invalid value for RTA-retry.
0xC00A0090	TLR_E_PNIO_CMCTL_IOCRR_BLOCKLEN The expected configuration block for IOCRR in CMCTL_RMConnect_req_Loadiocrr() has an invalid length.
0xC00A0091	TLR_E_PNIO_CMCTL_IOCRR_TYPE_UNSUPPORTED The type of IOCRR is unsupported.
0xC00A0092	TLR_E_PNIO_CMCTL_IOCRR_TYPE_UNKNOWN The type of IOCRR is unknown.
0xC00A0093	TLR_E_PNIO_CMCTL_IOCRR_RTCCLASS_UNSUPPORTED The RTC-class is unsupported.
0xC00A0094	TLR_E_PNIO_CMCTL_IOCRR_RTCCLASS_UNKNOWN The RTC-class is unknown.

Hexadecimal Value	Definition Description
0xC00A0095	TLR_E_PNIO_CMCTL_IOCRR_IFTYPE_UNSUPPORTED The expected configuration block for IOCRR in CMCTL_RMConnect_req_Loadlocr() has an unsupported interface-type.
0xC00A0096	TLR_E_PNIO_CMCTL_IOCRR_SCSYNC_UNSUPPORTED The expected configuration block for IOCRR in CMCTL_RMConnect_req_Loadlocr() has an unsupported value for SendClock.
0xC00A0097	TLR_E_PNIO_CMCTL_IOCRR_ADDRESS_UNSUPPORTED The expected configuration block for IOCRR in CMCTL_RMConnect_req_Loadlocr() has an unsupported Address-Resolution.
0xC00A0098	TLR_E_PNIO_CMCTL_IOCRR_REDUNDANCY_UNSUPPORTED The expected configuration block for IOCRR in CMCTL_RMConnect_req_Loadlocr() has an unsupported Media-Redundancy.
0xC00A0099	TLR_E_PNIO_CMCTL_IOCRR_REFERENCE No IOCRR could be found or created.
0xC00A009A	TLR_E_PNIO_CMCTL_IOCRR_OBJECT_IOD The expected configuration block for IOCRR in CMCTL_RMConnect_req_Loadlocr() does not contain any IO-Data.
0xC00A009B	TLR_E_PNIO_CMCTL_IOCRR_OBJECT_IOS The expected configuration block for IOCRR in CMCTL_RMConnect_req_Loadlocr() does not contain any IO-Status.
0xC00A009C	TLR_E_PNIO_CMCTL_IOCRR_API The expected configuration block for IOCRR in CMCTL_RMConnect_req_Loadlocr() does not contain any API.
0xC00A00A0	TLR_E_PNIO_CMCTL_EXPS_BLOCKLEN The expected configuration block for Expected-Submodules in CMCTL_RMConnect_req_LoadExps() has an invalid length.
0xC00A00A1	TLR_E_PNIO_CMCTL_EXPS_API The expected configuration block for Expected-Submodules in CMCTL_RMConnect_req_LoadExps() does not contain any API.
0xC00A00A2	TLR_E_PNIO_CMCTL_EXPS_SUBMODULE The expected configuration block for Expected-Submodules in CMCTL_RMConnect_req_LoadExps() does not contain any submodules.
0xC00A00A3	TLR_E_PNIO_CMCTL_EXPS_DATADESCRIPTION The expected configuration block for Expected-Submodules in CMCTL_RMConnect_req_LoadExps() does not contain the expected amount of data-descriptions.
0xC00C00AA	TLR_E_PNIO_CMCTL_ACYCLIC_REQ_FAILED_REMOTE The acyclic service failed. The IO-Device answered with an error code which is contained in confirmation packet.
0xC00C00AB	TLR_E_PNIO_CMCTL_ACYCLIC_REQ_FAILED_RPC The acyclic service failed. The RPC-layer detected an error which is contained in confirmation packet.
0xC00C00AC	TLR_E_PNIO_CMCTL_ACYCLIC_REQ_FAILED_INTERNAL The acyclic service failed. An internal error occurred.

Table 187: Status/Error Codes for CMCTL Task

## 9.2.1 CMCTL-Task Diagnosis-Codes

### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC00AF000	TLR_DIAG_E_CMCTL_TASK_RESOURCE_INIT_FAILED Initializing CMCTL's task-resources failed.
0xC00AF001	TLR_DIAG_E_CMCTL_TASK_CREATE_QUE_FAILED Failed to create message-queue for CMCTL.
0xC00AF002	TLR_DIAG_E_CMCTL_TASK_CREATE_SYNC_QUE_FAILED Failed to create synchronous message-queue for CMCTL.
0xC00AF003	TLR_DIAG_E_CMCTL_TASK_RPC_INIT_FAILED Failed to initialize CMCTL's local RPC-resources.
0xC00AF004	TLR_DIAG_E_CMCTL_TASK_IDENT_ACP_QUE_FAILED Failed to get handle to ACP message-queue in CMCTL.
0xC00AF005	TLR_DIAG_E_CMCTL_TASK_IDENT_MGT_QUE_FAILED Failed to get handle to MGT message-queue in CMCTL.
0xC00AF006	TLR_DIAG_E_CMCTL_TASK_IDENT_RPC_QUE_FAILED Failed to get handle to RPC message-queue in CMCTL.
0xC00AF007	TLR_DIAG_E_CMCTL_TASK_IDENT_TCP_QUE_FAILED Failed to get handle to TCP/IP message-queue in CMCTL.

Table 188: CMCTL -Task Diagnosis-Codes

## 9.3 Status/Error Codes for CM-Dev Task

### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC00B0001	TLR_E_PNIO_CMDEV_COMMAND_INVALID Received invalid command in CMDEV task.
0xC00B0010	TLR_E_PNIO_CMDEV_INIT_PARAM_INVALID Invalid parameter in CMDEV_ResourceInit().
0xC00B0011	TLR_E_PNIO_CMDEV_RESOURCE_LIMIT_EXCEEDED No more CMDEV protocol machines possible.
0xC00B0012	TLR_E_PNIO_CMDEV_RESOURCE_OUT_OF_MEMORY Insufficient memory for this request to CMDEV.
0xC00B0013	TLR_E_PNIO_CMDEV_CLOSED This CMDEV protocol machine was closed.
0xC00B0014	TLR_E_PNIO_CMDEV_STATE_CONFLICT This request cannot be served in current CMDEV state.
0xC00B0015	TLR_E_PNIO_CMDEV_CONFIG_PENDING The state of CMDEV's management resource is pending.
0xC00B0016	TLR_E_PNIO_CMDEV_CONFIG_STATE_INVALID The state of CMDEV's management resource is invalid.
0xC00B0017	TLR_E_PNIO_CMDEV_PACKET_OUT_OF_MEMORY Insufficient memory to create a packet in CMDEV task.
0xC00B0018	TLR_E_PNIO_CMDEV_PACKET_SEND_FAILED Error while sending a packet to another task in CMDEV.
0xC00B0019	TLR_E_PNIO_CMDEV_CONN_REQ_LEN_INVALID The length of the Connect-Packet in CMDEV_Connect_req() is invalid.
0xC00B001A	TLR_E_PNIO_CMDEV_NAME_LEN_INVALID The length of the name for IO-Device does not match to the name in CMDEV_Connect_req().
0xC00B001B	TLR_E_PNIO_CMDEV_BLKNUM_UNEXPECTED The Connect-Confirmation contains an incorrect amount of blocks.
0xC00B001C	TLR_E_PNIO_CMDEV_BLKNUM_UNEXPECTED_MEMORY_FAULT The Connect-Confirmation contains an incorrect amount of blocks but may be received correctly in RPC-layer. CMDEV protocol-machine has not reserved enough memory for the whole confirmation.
0xC00B001D	TLR_E_PNIO_CMDEV_INVALID_FRAMEID_RECEIVED The Connect-Response from IO-Device specified an invalid FrameID to use for IO-Controllers OutputCR.
0xC00B001F	TLR_E_PNIO_CMDEV_EMPTY_POOL_DETECTED The packet pool of CMDEV is empty.
0xC00B0020	TLR_E_PNIO_CMDEV_PACKET_WRONG_DEVICEHANDLE
0xC00B0021	TLR_E_PNIO_CMDEV_POINTER_INVALID
0xC00B0022	TLR_E_PNIO_CMDEV_FUNCTION_RETURN_FAILURE
0xC00B0023	TLR_E_PNIO_CMDEV_WAIT_FOR_PACKET_FAILED

Hexadecimal Value	Definition Description
0xC00B0024	TLR_E_PNIO_CMDEV_ALPMI_ACTIVATE_FAILED
0xC00B0025	TLR_E_PNIO_CMDEV_BUILD_CONNECT_RSP_FAILED
0xC00B0026	TLR_E_PNIO_CMDEV_AP_ENTRY_NOT_FOUND
0xC00B0027	TLR_E_PNIO_CMDEV_TIMER_CREATE_FAILED
0xC00B0028	TLR_E_PNIO_CMDEV_ERROR_SEQUENCE
0xC00B0029	TLR_E_PNIO_CMDEV_INVALID_PLUG_REQUEST_PCK
0xC00B002A	TLR_E_PNIO_CMDEV_INVALID_PULL_REQUEST_PCK
0xC00B002B	TLR_E_PNIO_CMDEV_PLUG_SLOT_NOT_EXPECTED
0xC00B002C	TLR_E_PNIO_CMDEV_PLUG_SUBSLOT_NOT_EXPECTED
0xC00B002D	TLR_E_PNIO_CMDEV_RPC_PACKET_INVALID
0xC00B002E	TLR_E_PNIO_CMDEV_ALPMI_INIT_FAILED Initializing the ALPMI state machine failed.
0xC00B002F	TLR_E_PNIO_CMDEV_CHANGE_BUS_STATE_FAILED Changing the internal Bus state failed.
0xC00B0040	TLR_E_PNIO_CMDEV_INVALID_PM_INDEX The index of CMDEV protocol-machine is invalid.
0xC00B0041	TLR_E_PNIO_CMDEV_INVALID_PM The CMDEV protocol-machine corresponding to index is invalid.
0xC00B0042	TLR_E_PNIO_CMDEV_INVALID_CMDEV_HANDLE The handle to CMDEV protocol-machine is invalid.
0xC00B0043	TLR_E_PNIO_CMDEV_SUBMODULE_NOT_IN_CYCLIC_DATA_EXCHANGE The request can not be handled because the submodule is not contained in cyclic data exchange.
0xC00B0050	TLR_E_PNIO_CMDEV_DEVICE_NOT_RESPONDING The IO-Device which shall be connected does not answer.
0xC00B0051	TLR_E_PNIO_CMDEV_DUPLICATE_DEVICE_NAME_DETECTED More than one IO-Device with the specified NameOfStation exists; a connection can not be established.
0xC00B0052	TLR_E_PNIO_CMDEV_DEVICE_IP_ADDRESS_ALREADY_IN_USE The IP-address the controller shall use for the IO-Device is already in use by another network device; a connection cannot be established.
0xC00B0053	TLR_E_PNIO_CMDEV_TOO_MUCH_ALARM_DATA The packet contains too much alarm data.
0xC00B0060	TLR_E_PNIO_CMDEV_RPC_CONNECT_FAILED The Connect-Response of IO-Device contained an error code; a connection could not be established.
0xC00B0061	TLR_E_PNIO_CMDEV_RPC_WRITE_PARAM_FAILED The Write_Param-Response of IO-Device contained an error code; a connection could not be established.
0xC00B0062	TLR_E_PNIO_CMDEV_RPC_WRITE_FAILED The Write-Response of IO-Device contained an error code.

Hexadecimal Value	Definition Description
0xC00B0063	TLR_E_PNIO_CMDEV_RPC_READ_FAILED The Read-Response of IO-Device contained an error code.
0xC00B0064	TLR_E_PNIO_CMDEV_TCP_IP_SHUTDOWN The TCP/IP-Stack closed a socket needed for communication.
0xC00B0070	TLR_E_PNIO_CMDEV_AR_BLOCKTYPE The expected configuration block for AR in CMDEV_RMConnect_req_LoadAr() is missing.
0xC00B0071	TLR_E_PNIO_CMDEV_AR_BLOCKLEN The expected configuration block for AR in CMDEV_RMConnect_req_LoadAr() has an invalid length.
0xC00B0072	TLR_E_PNIO_CMDEV_AR_TYPE The configuration block for AR in CMDEV_RMConnect_req_LoadAr() has an invalid type.
0xC00B0073	TLR_E_PNIO_CMDEV_AR_UUID The configuration block for AR in CMDEV_RMConnect_req_LoadAr() has an invalid UUID.
0xC00B0074	TLR_E_PNIO_CMDEV_AR_PROPERTY The configuration block for AR in CMDEV_RMConnect_req_LoadAr() has an invalid network properties value.
0xC00B0075	TLR_E_PNIO_CMDEV_AR_REF_UNEXPECTED The AR-Reference for CMDEV protocol-machine is invalid.
0xC00B0076	TLR_E_PNIO_CMDEV_AR_UUID_COMP_FAILED The UUID inside IO-Device's Connect-Confirmation is incorrect.
0xC00B0077	TLR_E_PNIO_CMDEV_AR_KEY_COMP_FAILED The session-key inside IO-Device's Connect-Confirmation is incorrect.
0xC00B0078	TLR_E_PNIO_CMDEV_AR_MAC_COMP_FAILED The MAC-address of IO-Device is incorrect.
0xC00B0080	TLR_E_PNIO_CMDEV_INSERT_MODULE_ERROR
0xC00B0081	TLR_E_PNIO_CMDEV_INSERT_SUBMODULE_ERROR
0xC00B0082	TLR_E_PNIO_CMDEV_MAX_API_LIMIT_EXCEEDED
0xC00B0083	TLR_E_PNIO_CMDEV_API_ALREADY_ADDED
0xC00B0084	TLR_E_PNIO_CMDEV_SLOT_OUT_OF_RANGE
0xC00B0085	TLR_E_PNIO_CMDEV_SUBSLOT_OUT_OF_RANGE
0xC00B0086	TLR_E_PNIO_CMDEV_SUBSLOT_ALREADY_EXISTS
0xC00B0087	TLR_E_PNIO_CMDEV_PACKET_WRONG_API
0xC00B0088	TLR_E_PNIO_CMDEV_PACKET_WRONG_SLOT
0xC00B0089	TLR_E_PNIO_CMDEV_PACKET_WRONG_SUBSLOT
0xC00B008A	TLR_E_PNIO_CMDEV_SLOT_ENTRY_NOT_FOUND
0xC00B008B	TLR_E_PNIO_CMDEV_SLOT_ALREADY_EXISTS

Hexadecimal Value	Definition Description
0xC00B008C	TLR_E_PNIO_CMDEV_SUBSLOT_ENTRY_NOT_FOUND
0xC00B008D	TLR_E_PNIO_CMDEV_FILTERED A CheckIndication shall not be forwarded to the user according to configuration.
0xC00B0090	TLR_E_PNIO_CMDEV_IOCRR_BLOCKLEN The expected configuration block for IOCRR in CMDEV_RMConnect_req_Loadlocr() has an invalid length.
0xC00B0091	TLR_E_PNIO_CMDEV_IOCRR_TYPE_UNSUPPORTED The type of IOCRR is unsupported.
0xC00B0092	TLR_E_PNIO_CMDEV_IOCRR_TYPE_UNKNOWN The type of IOCRR is unknown.
0xC00B0093	TLR_E_PNIO_CMDEV_IOCRR_RTCCLASS_UNSUPPORTED The RTC-class is unsupported.
0xC00B0094	TLR_E_PNIO_CMDEV_IOCRR_RTCCLASS_UNKNOWN The RTC-class is unknown.
0xC00B0095	TLR_E_PNIO_CMDEV_IOCRR_IFTYPE_UNSUPPORTED The expected configuration block for IOCRR in CMDEV_RMConnect_req_Loadlocr() has an unsupported interface-type.
0xC00B0096	TLR_E_PNIO_CMDEV_IOCRR_SCSYNC_UNSUPPORTED The expected configuration block for IOCRR in CMDEV_RMConnect_req_Loadlocr() has an unsupported value for SendClock.
0xC00B0097	TLR_E_PNIO_CMDEV_IOCRR_ADDRESS_UNSUPPORTED The expected configuration block for IOCRR in CMDEV_RMConnect_req_Loadlocr() has an unsupported Address-Resolution.
0xC00B0098	TLR_E_PNIO_CMDEV_IOCRR_REDUNDANCY_UNSUPPORTED The expected configuration block for IOCRR in CMDEV_RMConnect_req_Loadlocr() has an unsupported Media-Redundancy.
0xC00B0099	TLR_E_PNIO_CMDEV_IOCRR_REFERENCE No IOCRR could be found or created.
0xC00B009A	TLR_E_PNIO_CMDEV_IOCRR_OBJECT_IOD The expected configuration block for IOCRR in CMDEV_RMConnect_req_Loadlocr() does not contain any IO-Data.
0xC00B009B	TLR_E_PNIO_CMDEV_IOCRR_OBJECT_IOS The expected configuration block for IOCRR in CMDEV_RMConnect_req_Loadlocr() does not contain any IO-Status.
0xC00B009C	TLR_E_PNIO_CMDEV_IOCRR_API The expected configuration block for IOCRR in CMDEV_RMConnect_req_Loadlocr() does not contain any API.
0xC00B0100	TLR_E_PNIO_CMDEV_FRAME_ID_COUNT_INVALID
0xC00B0101	TLR_E_PNIO_CMDEV_FRAME_ID_OUT_OF_RANGE
0xC00B0102	TLR_E_PNIO_CMDEV_RT_CLASS_NOT_SUPPORTED
0xC00B0103	TLR_E_PNIO_CMDEV_INSERT_AR_ERROR
0xC00B0104	TLR_E_PNIO_CMDEV_MAX_AR_LIMIT_EXCEEDED
0xC00B0105	TLR_E_PNIO_CMDEV_AR_INVALID
0xC00B0106	TLR_E_PNIO_CMDEV_IOCRR_INVALID



Hexadecimal Value	Definition Description
0xC00B0107	TLR_E_PNIO_CMDEV_TYPE_LEN_INVALID
0xC00B0108	TLR_E_PNIO_CMDEV_INVALID_CTRL_REQUEST_BLOCK
0xC00B0109	TLR_E_PNIO_CMDEV_MODULECONFIG_PACKET_INVALID

*Table 189: Status/Error Codes for CM-Dev Task*

### 9.3.1 CM-Dev-Task Diagnosis-Codes

#### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC00BF000	TLR_DIAG_E_CMDEV_TASK_RESOURCE_INIT_FAILED Initializing CMDEV's task-resources failed.
0xC00BF001	TLR_DIAG_E_CMDEV_TASK_CREATE_QUE_FAILED Failed to create message-queue for CMDEV.
0xC00BF002	TLR_DIAG_E_CMDEV_TASK_CREATE_SYNC_QUE_FAILED Failed to create synchronous message-queue for CMDEV.
0xC00BF003	TLR_DIAG_E_CMDEV_TASK_RPC_INIT_FAILED Failed to initialize CMDEV's local RPC-resources.
0xC00BF004	TLR_DIAG_E_CMDEV_TASK_IDENT_ACP_QUE_FALIED Failed to get handle to ACP message-queue in CMDEV.
0xC00BF005	TLR_DIAG_E_CMDEV_TASK_IDENT_MGT_QUE_FALIED Failed to get handle to MGT message-queue in CMDEV.
0xC00BF006	TLR_DIAG_E_CMDEV_TASK_IDENT_RPC_QUE_FALIED Failed to get handle to RPC message-queue in CMDEV.
0xC00BF007	TLR_DIAG_E_CMDEV_TASK_IDENT_TCP_QUE_FALIED Failed to get handle to TCP/IP message-queue in CMDEV.
0xC00BF008	TLR_DIAG_E_CMDEV_TASK_IDENT_DCP_QUE_FALIED Failed to get handle to DCP message-queue in CMDEV .
0xC00BF009	TLR_DIAG_E_CMDEV_TASK_IDENT_PNSIF_QUE_FALIED Failed to get handle to PNSIF message-queue in CMDEV.

Table 190: CM-Dev-Task Diagnosis-Codes

## 9.4 Status/Error Codes for EDD Task

### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC00E0001	TLR_E_PNIO_EDD_PROCESS_END Return value of EDD_Scheduler_PreProcess().
0xC00E0002	TLR_E_PNIO_EDD_PARAM_INVALID_EDD Invalid parameter for EDD_Scheduler_Start_req().

Table 191: Status/Error Codes for EDD Task

### 9.4.1 EDD-Task Diagnosis-Codes

#### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC00EF001	TLR_E_PNIO_EDD_COMMAND_INVALID Received invalid command in EDD task.
0xC00EF010	TLR_DIAG_E_EDD_TASK_INIT_LOCAL_FAILED Failed to initialize EDD's local resources.

Table 192: EDD-Task Diagnosis-Codes

## 9.5 Status/Error Codes for ACP Task

### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC0110010	TLR_E_PNIO_ACP_PHASE_OUT_OF_MEMORY Insufficient memory to initialize ACP-phase.
0xC0110011	TLR_E_PNIO_ACP_PHASE_REDUCTION_RATIO Invalid reduction-ratio (uiMaxRatio) in ACP_PhaseInit().
0xC0110012	TLR_E_PNIO_ACP_PHASE_SEND_CLOCK_FACTOR Invalid sendClock-factor (uiScFact) in ACP_PhaseInit().
0xC0110013	TLR_E_PNIO_ACP_PHASE_FRAME_RESOURCES Invalid parameter (uiMaxFrame) in ACP_PhaseInit().
0xC0110014	TLR_E_PNIO_ACP_PACKET_SEND_FAILED Error sending a packet to another task in ACP task.
0xC0110015	TLR_E_PNIO_ACP_RESOURCE_OUT_OF_MEMORY Insufficient memory in ACP task.
0xC0110016	TLR_E_PNIO_ACP_DRV_EDD_IOCTL_ERROR
0xC0110017	TLR_E_PNIO_SYNC_LOAD_IRT_DATA_ERROR
0xC0110018	TLR_E_PNIO_ACP_EMPTY_POOL_DETECTED The packet pool of ACP is empty.
0xC0110020	TLR_E_PNIO_ALARM_PARAM_INVALID_INIT Invalid parameter "uiMaxAlpm" in Alarm_ResourceInit().
0xC0110021	TLR_E_PNIO_ALARM_RESOURCE_OUT_OF_MEMORY Insufficient memory in Alarm_ResourceInit().
0xC0110030	TLR_E_PNIO_ALPMR_PRIORITY_INVALID Invalid alarm priority in request packet of ALPMR_AlarmAck_req().
0xC0110031	TLR_E_PNIO_ALPMR_RESOURCE_LIMIT_EXCEEDED The requested number of ALPMR protocol machines exceeds the highest possible number in ALPMR_Init_req().
0xC0110032	TLR_E_PNIO_ALPMR_RESOURCE_OUT_OF_MEMORY Insufficient memory in ALPMR_Init_req().
0xC0110033	TLR_E_PNIO_ALPMR_HANDLE_INVALID The ALPMR protocol-machine corresponding to the index in request packet is invalid.
0xC0110034	TLR_E_PNIO_ALPMR_STATE_INVALID The ALPMR protocol-machine state is invalid for the current request.
0xC0110035	TLR_E_PNIO_ALPMR_PACKET_SEND_FAILED Sending an Alarm-Indication-packet to another task failed in ALPMR.
0xC0110036	TLR_E_PNIO_ALPMR_PACKET_OUT_OF_MEMORY Creating an Alarm-Indication-packet to be send to another task failed due to insufficient memory.
0xC0110037	TLR_E_PNIO_ALPMR_RESOURCE_INDEX_INVALID The index of ALPMR's protocol machine is invalid.
0xC0110040	TLR_E_PNIO_APMR_PARAM_INVALID_INIT The parameter uiMaxApmr (maximum number of parallel APMR protocol-machines) in APMR_ResourceInit() is invalid.
0xC0110041	TLR_E_PNIO_APMR_RESOURCE_OUT_OF_MEMORY Insufficient memory in APMR_ResourceInit() to create the APMR protocol machines.

Hexadecimal Value	Definition Description
0xC0110042	TLR_E_PNIO_APMR_HANDLE_INVALID The APMR protocol machine or its index is invalid.
0xC0110043	TLR_E_PNIO_APMR_STATE_INVALID The state of APMR protocol machine is invalid for current request.
0xC0110044	TLR_E_PNIO_APMR_FRAME_SEND_FAILED Sending an ACK or NAK in response to a received Alarm-PDU failed.
0xC0110050	TLR_E_PNIO_APMS_PARAM_INVALID_INIT The parameter uiMaxApms (maximum number of parallel APMS protocol-machines) in APMS_ResourceInit() is invalid.
0xC0110051	TLR_E_PNIO_APMS_RESOURCE_OUT_OF_MEMORY Insufficient memory in APMS_ResourceInit() to create the APMS protocol machines.
0xC0110052	TLR_E_PNIO_APMS_HANDLE_INVALID The APMS protocol machine or its index is invalid.
0xC0110053	TLR_E_PNIO_APMS_STATE_INVALID The state of APMS protocol machine is invalid for current request.
0xC0110054	TLR_E_PNIO_APMS_FRAME_OUT_OF_MEMORY APMS was not able to get an Edd_FrameBuffer for sending a packet.
0xC0110055	TLR_E_PNIO_APMS_FRAME_SEND_FAILED An error occurred while APMS was trying to send an Edd_Frame.
0xC0110056	TLR_E_PNIO_APMS_TIMER_CREATE_FAILED APMS_Activate_req() was not able to create a TLR-Timer.
0xC0110057	TLR_E_PNIO_APMS_TIMER_OUT_OF_MEMORY Insufficient memory for APMS_Send_req_Data() to allocate a timer-indication packet.
0xC0110058	TLR_E_PNIO_APMS_INDEX_INVALID
0xC0110060	TLR_E_PNIO_CPM_PARAM_INVALID_INIT The parameter uiMaxCpmRtc1 and/or uiMaxCpmRtc2 of CPM_ResourceInit() is invalid.
0xC0110061	TLR_E_PNIO_CPM_PARAM_INVALID_CLASS The requested RTC-class is invalid in CPM_Init_req().
0xC0110062	TLR_E_PNIO_CPM_RESOURCE_LIMIT_EXCEEDED The requested amount of CPM protocol machines is higher than the highest possible value.
0xC0110063	TLR_E_PNIO_CPM_RESOURCE_OUT_OF_MEMORY Insufficient memory for current request in CPM.
0xC0110064	TLR_E_PNIO_CPM_HANDLE_INVALID The handle to CPM protocol machine is invalid.
0xC0110065	TLR_E_PNIO_CPM_STATE_INVALID The state of CPM protocol machine is incorrect for current request.
0xC0110066	TLR_E_PNIO_CPM_PHASE_LIMIT_EXCEEDED Invalid phase found in Init-request-packet in CPM_Init_req() or in ACP_PhaseCpmAdd_req() or ACP_PhaseCpmRemove_req().
0xC0110067	TLR_E_PNIO_CPM_SEND_CLOCK_LIMIT_EXCEEDED The SendClock-factor in Init-request-packet to CPM does not match the one in ACP_Tasks' resources.
0xC0110069	TLR_E_PNIO_CPM_DATALEN_LIMIT_EXCEEDED Packet size to receive is too big. Error is detected in CPM_Init_req().
0xC011006A	TLR_E_PNIO_CPM_PACKET_SEND_FAILED Error while sending a packet to another task in CPM.
0xC0110080	TLR_E_PNIO_PPM_PARAM_INVALID_INIT The parameter "uiMaxPPMRtc1" and/or "uiMaxPPMRtc2" of PPM_ResourceInit() is invalid.

Hexadecimal Value	Definition Description
0xC0110081	TLR_E_PNIO_PPM_PARAM_INVALID_CLASS The requested RTC-class is invalid in PPM_Init_req().
0xC0110082	TLR_E_PNIO_PPM_RESOURCE_LIMIT_EXCEEDED The requested amount of PPM protocol machines is higher than the highest possible value.
0xC0110083	TLR_E_PNIO_PPM_RESOURCE_OUT_OF_MEMORY Insufficient memory for current request in PPM.
0xC0110084	TLR_E_PNIO_PPM_HANDLE_INVALID The handle to PPM protocol machine is invalid.
0xC0110085	TLR_E_PNIO_PPM_STATE_INVALID The state of PPM protocol machine is incorrect for current request.
0xC0110086	TLR_E_PNIO_PPM_PHASE_LIMIT_EXCEEDED Invalid phase found in Init-request-packet in PPM_Init_req() or in ACP_PhasePPMAdd_req() or ACP_PhasePPMRemove_req().
0xC0110087	TLR_E_PNIO_PPM_SEND_CLOCK_LIMIT_EXCEEDED The SendClock-factor in PPMs Init-request-packet does not match the one in ACP_Tasks' resources.
0xC0110089	TLR_E_PNIO_PPM_DATALEN_LIMIT_EXCEEDED Packet size to send is too big. Error is detected in PPM_Init_req().
0xC011008A	TLR_E_PNIO_PPM_RESOURCE_CLASS_INVALID
0xC0110090	TLR_E_PNIO_ALPMI_PRIORITY_INVALID Invalid alarm priority in request packet of ALPMI_AlarmAck_req().
0xC0110091	TLR_E_PNIO_ALPMI_RESOURCE_LIMIT_EXCEEDED The requested number of ALPMI protocol machines exceeds the highest possible number in ALPMI_Init_req().
0xC0110092	TLR_E_PNIO_ALPMI_RESOURCE_OUT_OF_MEMORY Insufficient memory in ALPMI_Init_req().
0xC0110093	TLR_E_PNIO_ALPMI_HANDLE_INVALID The ALPMI protocol-machine corresponding to the index in request packet is invalid.
0xC0110094	TLR_E_PNIO_ALPMI_STATE_INVALID The ALPMI protocol-machine state is invalid for the current request.
0xC0110095	TLR_E_PNIO_ALPMI_PACKET_SEND_FAILED Sending an Alarm-Indication-packet to another task failed in ALPMI.
0xC0110096	TLR_E_PNIO_ALPMI_PACKET_OUT_OF_MEMORY Creating an Alarm-Indication-packet to be send to another task failed due to insufficient memory.

0xC0110097	TLR_E_PNIO_ALPMI_RESOURCE_INDEX_INVALID The index of ALPIR's protocol machine is invalid.
------------	--

Table 193: Status/Error Codes for ACP Task

## 9.5.1 ACP-Task Diagnosis-Codes

### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC011F001	TLR_E_PNIO_ACP_COMMAND_INVALID Received invalid command in ACP task.
0xC011F010	TLR_DIAG_E_ACP_TASK_ACP_PHASE_INIT_FAILED Failed to initialize ACP Phase.
0xC011F011	TLR_DIAG_E_ACP_TASK_ALARM_INIT_FAILED Failed to initialize Alarm-machines.
0xC011F012	TLR_DIAG_E_ACP_TASK_APMR_INIT_FAILED Failed to initialize APMR.
0xC011F013	TLR_DIAG_E_ACP_TASK_APMS_INIT_FAILED Fails to initialize APMS.
0xC011F014	TLR_DIAG_E_ACP_TASK_CPM_INIT_FAILED Failed to initialize CPM.
0xC011F015	TLR_DIAG_E_ACP_TASK_PPM_INIT_FAILED Failed to initialize PPM.
0xC011F016	TLR_DIAG_E_ACP_TASK_CREATE_QUE_FAILED Failed to create message-queue for ACP.
0xC011F017	TLR_DIAG_E_ACP_TASK_IDENT_EDD_FAILED Failed to identify Drv_EDD.
0xC011F018	TLR_DIAG_E_ACP_TASK_IDENT_EDD_QUE_FAILED Failed to get handle to EDD message-queue.
0xC011F019	TLR_DIAG_E_ACP_TASK_IDENT_DCP_QUE_FAILED Failed to get handle to DCP message-queue.
0xC011F01A	TLR_DIAG_E_ACP_TASK_IDENT_CMDEV_QUE_FAILED Failed to get handle to CMDEV message-queue.

Table 194: ACP-Task Diagnosis-Codes

## 9.6 Status/Error Codes for DCP Task

### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC012000A	TLR_E_PNIO_DCP_PARAM_INVALID_EDD Invalid parameter in Start-Edd-packet for DCP_StartEDD_req().
0xC0120010	TLR_E_PNIO_DCPMCR_INIT_PARAM_INVALID Invalid parameter (uiMaxMcr) in DCPMCR_ResourceInit().
0xC0120011	TLR_E_PNIO_DCPMCR_INIT_OUT_OF_MEMORY Insufficient memory to initialize DCPMCR protocol machines in DCPMCR_ResourceInit().
0xC0120012	TLR_E_PNIO_DCPMCR_RESOURCE_LIMIT_EXCEEDED The index of DCPMCR's protocol machine is invalid.
0xC0120013	TLR_E_PNIO_DCPMCR_RESOURCE_OUT_OF_MEMORY Insufficient memory for request in DCPMCR_Activate_req().
0xC0120014	TLR_E_PNIO_DCPMCR_RESOURCE_STATE_INVALID The state of DCPMCR protocol machine is incorrect for current request.
0xC0120015	TLR_E_PNIO_DCPMCR_RESOURCE_HANDLE_INVALID The handle to DCPMCR protocol machine is invalid.
0xC0120016	TLR_E_PNIO_DCPMCR_TIMER_CREATE_FAILED DCPMCR_Activate_req() was unable to create a TLR-timer.
0xC0120017	TLR_E_PNIO_DCPMCR_TIMER_OUT_OF_MEMORY Insufficient memory for DCPMCR_Identify_ind() to allocate a timer-indication packet.
0xC0120018	TLR_E_PNIO_DCPMCR_PACKET_OUT_OF_MEMORY Insufficient memory to allocate a packet to be send to another task in DCPMCR.
0xC0120019	TLR_E_PNIO_DCPMCR_PACKET_SEND_FAILED Error while sending a packet to another task in DCPMCR.
0xC012001A	TLR_E_PNIO_DCPMCR_FRAME_OUT_OF_MEMORY DCPMCR was not able to get an Edd_FrameBuffer for sending a packet.
0xC012001B	TLR_E_PNIO_DCPMCR_FRAME_SEND_FAILED An error occurred while DCPMCR was trying to send an Edd_Frame.
0xC012001C	TLR_E_PNIO_DCPMCR_WAIT_ACK DCPMCR could not be closed because it is still waiting for an ACK.
0xC012001D	TLR_E_PNIO_DCPMCR_TASK_RES_ADDRESS DCPMCR: Invalid parameter (task resources block address) while handling DCP Identify indication.
0xC012001E	TLR_E_PNIO_DCPMCR_EDD_FRAME_ADDRESS DCPMCR: Invalid parameter (EDD frame address) while handling DCP Identify indication.
0xC012001F	TLR_E_PNIO_DCPMCR_MCR_ADDRESS DCPMCR: Invalid parameter (DCPMCR state machine address) while handling DCP Identify indication.
0xC0120020	TLR_E_PNIO_DCPMCR_RMPM_ADDRESS DCPMCR: Invalid parameter (RMPM state machine address) while handling DCP Identify indication.
0xC0120021	TLR_E_PNIO_DCP_EMPTY_POOL_DETECTED The packet pool of DCP is empty.
0xC0120100	TLR_E_PNIO_DCPMCS_INIT_PARAM_INVALID Invalid parameter (uiMaxMcs) in DCPMCS_ResourceInit().



Hexadecimal Value	Definition Description
0xC0120101	TLR_E_PNIO_DCPMCS_INIT_OUT_OF_MEMORY Insufficient memory to initialize DCPMCS protocol machines in DCPMCS_ResourceInit().
0xC0120102	TLR_E_PNIO_DCPMCS_RESOURCE_LIMIT_EXCEEDED There are too many outstanding DCPMCS requests. New requests will not be accepted.
0xC0120103	TLR_E_PNIO_DCPMCS_RESOURCE_OUT_OF_MEMORY Insufficient memory for request in DCPMCS_Activate_req().
0xC0120104	TLR_E_PNIO_DCPMCS_RESOURCE_STATE_INVALID The state of DCPMCS protocol machine is incorrect for current request.
0xC0120105	TLR_E_PNIO_DCPMCS_RESOURCE_HANDLE_INVALID The handle to DCPMCS protocol machine is invalid.
0xC0120106	TLR_E_PNIO_DCPMCS_TIMER_CREATE_FAILED DCPMCS_Activate_req() was unable to create a TLR-timer.
0xC0120107	TLR_E_PNIO_DCPMCS_TIMER_OUT_OF_MEMORY Insufficient memory for DCPMCS_Identify_req() to allocate a timer-indication packet.
0xC0120108	TLR_E_PNIO_DCPMCS_PACKET_OUT_OF_MEMORY Insufficient memory to allocate a packet to be send to another task in DCPMCS.
0xC0120109	TLR_E_PNIO_DCPMCS_PACKET_SEND_FAILED Error while sending a packet to another task in DCPMCS.
0xC012010A	TLR_E_PNIO_DCPMCS_FRAME_OUT_OF_MEMORY DCPMCS was not able to get an Edd_FrameBuffer for sending a packet.
0xC012010B	TLR_E_PNIO_DCPMCS_FRAME_SEND_FAILED An error occurred while DCPMCS was trying to send an Edd_Frame.
0xC0120200	TLR_E_PNIO_DCPUCR_INIT_PARAM_INVALID Invalid parameter (uiMaxUcr) in DCPUCR_ResourceInit().
0xC0120201	TLR_E_PNIO_DCPUCR_INIT_OUT_OF_MEMORY Insufficient memory to initialize DCPUCR protocol machines in DCPUCR_ResourceInit().
0xC0120202	TLR_E_PNIO_DCPUCR_RESOURCE_LIMIT_EXCEEDED The index of DCPUCR's protocol machine is invalid.
0xC0120203	TLR_E_PNIO_DCPUCR_RESOURCE_OUT_OF_MEMORY Insufficient memory for request in DCPUCR_Activate_req().
0xC0120204	TLR_E_PNIO_DCPUCR_RESOURCE_STATE_INVALID The state of DCPUCR protocol machine is incorrect for current request.
0xC0120205	TLR_E_PNIO_DCPUCR_RESOURCE_HANDLE_INVALID The handle to DCPUCR protocol machine is invalid.
0xC0120206	TLR_E_PNIO_DCPUCR_TIMER_CREATE_FAILED DCPUCR_Activate_req() was unable to create a TLR-timer.
0xC0120207	TLR_E_PNIO_DCPUCR_TIMER_OUT_OF_MEMORY Insufficient memory to allocate a timer-indication packet.
0xC0120208	TLR_E_PNIO_DCPUCR_PACKET_OUT_OF_MEMORY Insufficient memory to allocate a packet to be send to another task in DCPUCR.
0xC0120209	TLR_E_PNIO_DCPUCR_PACKET_SEND_FAILED Error while sending a packet to another task in DCPUCR.
0xC012020A	TLR_E_PNIO_DCPUCR_FRAME_OUT_OF_MEMORY DCPUCR was not able to get an Edd_FrameBuffer for sending a packet.
0xC012020B	TLR_E_PNIO_DCPUCR_FRAME_SEND_FAILED An error occurred while DCPUCR was trying to send an Edd_Frame.
0xC012020C	TLR_E_PNIO_DCPUCR_SERVICE_INVALID The DCP-command of received response does not match the outstanding request in DCPUCR.

Hexadecimal Value	Definition Description
0xC012020D	TLR_E_PNIO_DCPUCR_WAIT_ACK DCPUCR could not be closed because it is still waiting for an ACK.
0xC0120300	TLR_E_PNIO_DCPUCS_INIT_PARAM_INVALID Invalid parameter (uiMaxUcs) in DCPUCS_ResourceInit().
0xC0120301	TLR_E_PNIO_DCPUCS_INIT_OUT_OF_MEMORY Insufficient memory to initialize DCPUCS protocol machines in DCPUCS_ResourceInit().
0xC0120302	TLR_E_PNIO_DCPUCS_RESOURCE_LIMIT_EXCEEDED There are too many outstanding DCPUCS requests. New requests will not be accepted.
0xC0120303	TLR_E_PNIO_DCPUCS_RESOURCE_OUT_OF_MEMORY Insufficient memory for request in DCPUCS_Activate_req().
0xC0120304	TLR_E_PNIO_DCPUCS_RESOURCE_STATE_INVALID The state of DCPUCS protocol machine is incorrect for current request.
0xC0120305	TLR_E_PNIO_DCPUCS_RESOURCE_HANDLE_INVALID The handle to DCPUCS protocol machine is invalid.
0xC0120306	TLR_E_PNIO_DCPUCS_TIMER_CREATE_FAILED DCPUCS_Activate_req() was unable to create a TLR-timer.
0xC0120307	TLR_E_PNIO_DCPUCS_TIMER_OUT_OF_MEMORY Insufficient memory for DCPUCS_DataSend_req() to allocate a timer-indication packet.
0xC0120308	TLR_E_PNIO_DCPUCS_PACKET_OUT_OF_MEMORY Insufficient memory to allocate a packet to be send to another task in DCPUCS.
0xC0120309	TLR_E_PNIO_DCPUCS_PACKET_SEND_FAILED Error while sending a packet to another task in DCPUCS.
0xC012030A	TLR_E_PNIO_DCPUCS_FRAME_OUT_OF_MEMORY DCPUCS was not able to get an Edd_FrameBuffer for sending a packet.
0xC012030B	TLR_E_PNIO_DCPUCS_FRAME_SEND_FAILED An error occurred while DCPUCS was trying to send an Edd_Frame.
0xC012030C	TLR_E_PNIO_DCPUCS_FRAME_TIMEOUT DCPUCS did not get a response to an Edd_Frame send .
0xC0120320	TLR_E_PNIO_DCPUCS_DCP_OPTION_UNSUPPORTED The DCP option to set is not supported by IO-Device.
0xC0120321	TLR_E_PNIO_DCPUCS_DCP_SUBOPTION_UNSUPPORTED The DCP suboption to set is not supported by IO-Device.
0xC0120022	TLR_E_PNIO_DCPUCS_DCP_SUBOPTION_NOT_SET The DCP suboption to set was not set inside IO-Device.
0xC0120023	TLR_E_PNIO_DCPUCS_DCP_RESOURCE_ERROR An internal resource error occurred in IO-Device while performing a DCP request.
0xC0120024	TLR_E_PNIO_DCPUCS_DCP_SET_IMPOSSIBLE_LOCAL_REASON The DCP (sub)option could not be set inside IO-Device for IO-Device internal reasons.
0xC0120025	TLR_E_PNIO_DCPUCS_DCP_SET_IMPOSSIBLE_WHILE_OPERATION The DCP (sub)option could not be set inside IO-Device because IO-Device is in operation.

Table 195: Status/Error Codes for DCP Task

## 9.6.1 DCP-Task Diagnosis-Codes

### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC012F001	TLR_E_PNIO_DCP_COMMAND_INVALID Received invalid command in DCP task.
0xC012F010	TLR_DIAG_E_DCP_TASK_UCS_RESOURCE_INIT_FAILED Failed to initialize DCPUCS.
0xC012F011	TLR_DIAG_E_DCP_TASK_UCR_RESOURCE_INIT_FAILED Failed to initialize DCPUCR.
0xC012F012	TLR_DIAG_E_DCP_TASK_MCS_RESOURCE_INIT_FAILED Failed to initialize DCPMCS.
0xC012F013	TLR_DIAG_E_DCP_TASK_MCR_RESOURCE_INIT_FAILED Failed to initialize DCPMCR.
0xC012F014	TLR_DIAG_E_DCP_TASK_CREATE_QUE_FAILED Failed to create message-queue for DCP task.

Table 196: DCP-Task Diagnosis-Codes

## 9.7 Status/Error Codes for MGT Task

### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC0130001	TLR_E_PNIO_MGT_PACKET_SEND_FAILED ACP_EDDStartDCP_req() was unable to send request packet to DCP-Task.
0xC0130002	TLR_E_PNIO_MGT_WAIT_FOR_PACKET_FAILED
0xC0130003	TLR_E_PNIO_MGT_CMDEV_HANDLE_INVALID
0xC0130004	TLR_E_PNIO_MGT_MAPPER_REGISTER_ERROR
0xC0130005	TLR_E_PNIO_MGT_SERVER_REGISTER_ERROR
0xC0130006	TLR_E_PNIO_MGT_OBJECT_REGISTER_ERROR
0xC0130007	TLR_E_PNIO_MGT_CLIENT_REGISTER_ERROR
0xC0130008	TLR_E_PNIO_MGT_OPCODE_UNKNOWN
0xC0130009	TLR_E_PNIO_MGT_RPCCLIENT_HANDLE_INVALID
0xC013000A	TLR_E_PNIO_MGT_OBJECT_UUID_NOT_FOUND
0xC013000B	TLR_E_PNIO_MGT_ARUUID_NOT_FOUND
0xC013000C	TLR_E_PNIO_MGT_INVALID_PORT_NUMBER
0xC013000D	TLR_E_PNIO_MGT_DRV_EDD_IOCTL_ERROR
0xC013000E	TLR_E_PNIO_MGT_INVALID_SESSION_KEY
0xC013000F	TLR_E_PNIO_MGT_TARGET_UUID_NOT_NIL
0xC0130010	TLR_E_PNIO_NRPM_PARAM_INVALID_INIT Invalid parameter (uiMaxNrpm) in NRPM_ResourceInit().
0xC0130011	TLR_E_PNIO_NRPM_HANDLE_INVALID The handle to NRPM protocol machine is invalid.
0xC0130012	TLR_E_PNIO_NRPM_STATE_INVALID The state of NRPM protocol machine is invalid.
0xC0130013	TLR_E_PNIO_NRPM_IDENTIFY_FLAG_INVALID The identify-flag in NRPM_Init_req() is invalid.
0xC0130014	TLR_E_PNIO_NRPM_RESOURCE_LIMIT_EXCEEDED The requested number of NRPM protocol machines exceeds the highest possible number in NRPM_Init_req().
0xC0130015	TLR_E_PNIO_NRPM_RESOURCE_OUT_OF_MEMORY Insufficient memory in NRPM_Init_req().
0xC0130016	TLR_E_PNIO_NRPM_PACKET_SEND_FAILED Error while sending a packet to another task in NRPM.

Hexadecimal Value	Definition Description
0xC0130017	TLR_E_PNIO_NRPM_PACKET_OUT_OF_MEMORY Insufficient memory to allocate a packet in NRPM.
0xC0130018	TLR_E_PNIO_NRPM_DCP_TYPE_INVALID Received request with invalid type of DCP request in NRPM.
0xC0130019	TLR_E_PNIO_NRPM_NAME_OF_STATION_INVALID The requested NameOfStation is invalid. Either it has an invalid length or it contains invalid characters.
0xC013001A	TLR_E_PNIO_NRPM_DCP_SET_ERROR The requested DCP Set operation failed.
0xC013001B	TLR_E_PNIO_NRPM_DEVICE_IP_ADDRESS_ALREADY_IN_USE The IP-address the controller shall set for the IO-Device is already in use by another network device.
0xC01300F0	TLR_E_PNIO_MGT_EMPTY_POOL_DETECTED The packet pool of MGT is empty.
0xC01300F1	TLR_E_PNIO_MGT_INVALID_DEV_INDEX The index of the device is invalid.
0xC0130101	TLR_E_PNIO_RMPM_HANDLE_INVALID The handle to RMPM is invalid.
0xC0130102	TLR_E_PNIO_RMPM_STATE_INVALID The state of RMPM is invalid for current request.
0xC0130103	TLR_E_PNIO_RMPM_STATE_CLOSING The state of RMPM is closed
0xC0130104	TLR_E_PNIO_RMPM_RESOURCE_LIMIT_EXCEEDED The number of RMPM state-machines is too high.
0xC0130105	TLR_E_PNIO_RMPM_RESOURCE_OUT_OF_MEMORY Insufficient memory to fulfill the current request in RMPM.
0xC0130106	TLR_E_PNIO_RMPM_PACKET_SEND_FAILED Error while sending a packet to another task in RMPM.
0xC0130107	TLR_E_PNIO_RMPM_PACKET_OUT_OF_MEMORY Insufficient memory to allocate a packet in RMPM.
0xC0130108	TLR_E_PNIO_RMPM_ROLE_UNSUPPORTED The parameter "role" is unsupported in RMPM_Init_req_ParameterRole().
0xC0130109	TLR_E_PNIO_RMPM_ROLE_UNKNOWN The parameter "role" is unknown in RMPM_Init_req_ParameterRole() .
0xC013010A	TLR_E_PNIO_RMPM_ROLE_IN_USE The parameter "role" is already in use in RMPM_Init_req_ParameterRole() .
0xC013010B	TLR_E_PNIO_RMPM_CONFIG_SEQUENCE Incorrect sequence of configuration in RMPM_ConfigSet_req().
0xC013010C	TLR_E_PNIO_RMPM_CONFIG_INVALID_VENDOR_ID Incorrect configuration of Vendor-ID in RMPM_ConfigSet_req().
0xC013010D	TLR_E_PNIO_RMPM_CONFIG_INVALID_NAME Incorrect name of station in RMPM_ConfigSet_req().
0xC013010E	TLR_E_PNIO_RMPM_CONFIG_INVALID_TYPE Incorrect name of type in RMPM_ConfigSet_req().
0xC0130110	TLR_E_PNIO_RMPM_DUPLICATE_NAME_OF_STATION The NameOfStation of IO-Controller is in use by another network device.
0xC0130111	TLR_E_PNIO_RMPM_DUPLICATE_IP The IP-address the IO-Controller shall use is in use by another network device.
0xC0130112	TLR_E_PNIO_RMPM_RPC_PACKET_INVALID The packet length of an RPC-packet received is invalid (most likely too short).

Hexadecimal Value	Definition Description
0xC0130113	TLR_E_PNIO_RMPM_DCP_PACKET_INVALID The packet length of an DCP-packet received is invalid (most likely too short).
0xC0130120	TLR_E_PNIO_RMPM_INVALID_IP_ADDRESS The IP address is invalid.
0xC0130121	TLR_E_PNIO_RMPM_INVALID_NETMASK The network mask is invalid.
0xC0130122	TLR_E_PNIO_RMPM_INVALID_GATEWAY The gateway address is invalid.
0xC0130200	TLR_E_PNIO_NRMC_PARAM_INVALID_INIT
0xC0130201	TLR_E_PNIO_NRMC_HANDLE_INVALID The handle to NRMC is invalid.
0xC0130202	TLR_E_PNIO_NRMC_STATE_INVALID The state of NRMC is invalid for current request.
0xC0130203	TLR_E_PNIO_NRMC_IDENTIFY_FLAG_INVALID
0xC0130204	TLR_E_PNIO_NRMC_RESOURCE_LIMIT_EXCEEDED The number of NRMC state-machines is too high.
0xC0130205	TLR_E_PNIO_NRMC_RESOURCE_OUT_OF_MEMORY Insufficient memory to fulfill the current request in NRMC.
0xC0130206	TLR_E_PNIO_NRMC_PACKET_SEND_FAILED Error while sending a packet to another task in NRMC.
0xC0130207	TLR_E_PNIO_NRMC_PACKET_OUT_OF_MEMORY Insufficient memory to allocate a packet in NRMC.
0xC0130208	TLR_E_PNIO_NRMC_DCP_TYPE_INVALID

Table 197: Status/Error Codes for MGT Task

## 9.7.1 MGT-Task Diagnosis-Codes

### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC013F001	TLR_E_PNIO_MGT_COMMAND_INVALID Received invalid command in MGT task.
0xC013F010	TLR_DIAG_E_MGT_TASK_RMPM_RESOURCE_INIT_FAILED Failed to initialize RMPM.
0xC013F011	TLR_DIAG_E_MGT_TASK_NRPM_RESOURCE_INIT_FAILED Failed to initialize NRPM.
0xC013F012	TLR_DIAG_E_MGT_TASK_CREATE_QUE_FAILED Failed to create message-queue for MGT task.
0xC013F013	TLR_DIAG_E_MGT_TASK_IDENT_TCPUDP_QUE_FAILED Failed to get handle to TCP/IP task in MGT task.
0xC013F014	TLR_DIAG_E_MGT_TASK_IDENT_DCP_QUE_FAILED Failed to get handle to DCP task in MGT task.
0xC013F015	TLR_DIAG_E_MGT_TASK_IDENT_EDD_FAILED Failed to identify Drv_Edd imp MGT task.
0xC013F016	TLR_DIAG_E_MGT_TASK_IDENT_RPC_QUE_FAILED Failed to get handle to RPC task in MGT task.

Table 198: MGT-Task Diagnosis-Codes

## 9.8 Status/Error Codes for FODMI-Task

0xC0960001	TLR_E_FODMI_TASK_COMMAND_INVALID Command not valid.
0xC0960002	TLR_DIAG_E_FODMI_TASK_INIT_LOCAL_CREATE_QUE_FAILED Failure at create que in init local.
0xC0960003	TLR_DIAG_E_FODMI_TASK_INIT_REMOTE_IDENT_APPLICATION_QUE_FAILED Failure identifie the application queue.
0xC0960004	TLR_DIAG_E_FODMI_TASK_INIT_EDD_FAILED Failure to identify the application queue.

Table 199: Status/Error Codes Overview

## 9.9 Status/Error Codes for RPC-Task

### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC02E0001	TLR_E_RPC_TASK_COMMAND_INVALID Received packet with invalid command.
0xC02E0100	TLR_E_RPC_STATUS Generic RPC-error code. See PROFINET-status code for details.
0xC02E0101	TLR_E_RPC_CONNECT_OUT_OF_MEMORY There was not enough memory allocated to receive the whole IO-Device's Connect-Response PDU. Most likely it contains a very large ModuleDiff-Block.

Hexadecimal Value	Definition Description
0xC02E0102	TLR_E_RPC_FATAL_ERROR_CLB_ALREADY_REGISTERED The fatal error callback function is already registered.
0xC02E0200	TLR_E_CLRPC_PACKET_SEND_FAILED Error while sending internal message to another task.
0xC02E0201	TLR_E_CLRPC_TIMER_OUT_OF_MEMORY Creating a TLR-Timer-packet in RPC task failed due to insufficient memory.
0xC02E0202	TLR_E_CLRPC_REF_COUNTER_INVALID The reference counter value is invalid.
0xC02E0203	TLR_E_CLRPC_INVALID_PORT_HANDLE The port handle is invalid.
0xC02E0204	TLR_E_CLRPC_TIMER_ALREADY_ACTIVE The soft timer is already active (expected inactive).
0xC02E0300	TLR_E_CLRPC_MAPPER_INIT_FAILED The parameter "uiMaxReg" (maximum amount of RPC-mapper registrations) is invalid in CLRPC_EPMap_Initialize().
0xC02E0301	TLR_E_CLRPC_MAPPER_RESOURCE_LIMIT_EXCEEDED The requested Endpoint-Mapper index is invalid.
0xC02E0302	TLR_E_CLRPC_MAPPER_RESOURCE_OUT_OF_MEMORY Insufficient memory for this request.
0xC02E0303	TLR_E_CLRPC_MAPPER_STATUS_INVALID The state of Endpoint-Mapper is invalid for this request.
0xC02E0304	TLR_E_CLRPC_MAPPER_STATUS_CLOSING The Endpoint-Mapper is waiting for close-confirmation and therefore its status is invalid for this request.
0xC02E0305	TLR_E_CLRPC_MAPPER_STATUS_UNKNOWN The status of Endpoint-Mapper is unknown.
0xC02E0306	TLR_E_CLRPC_MAPPER_STATUS_CONFLICT The status of Endpoint-Mapper is not "Ready" and therefore request CLRPC_EPMap_Deregister_req() is invalid.
0xC02E0307	TLR_E_CLRPC_MAPPER_PARAMETER_FAILED Invalid parameter in CLRPC_EPMap_Register_req_Compare().
0xC02E0308	TLR_E_CLRPC_MAPPER_SERVER_REGISTERED CLRPC_EPMap_Deregister_req() is not allowed because at least one RPC-Server is registered to this Endpoint-Mapper.
0xC02E0400	TLR_E_CLRPC_SERVER_INIT_FAILED An error occurred in CLRPC_Server_Initialize().
0xC02E0401	TLR_E_CLRPC_SERVER_RESOURCE_LIMIT_EXCEEDED The maximum number of registered RPC-Servers is exceeded or the maximum number of outstanding requests is exceeded.
0xC02E0402	TLR_E_CLRPC_SERVER_TIMER_CREATE_FAILED Creating TLR-Timer for RPC-Server failed.
0xC02E0403	TLR_E_CLRPC_SERVER_NO_SERVER_REGISTERED There is no RPC-Server registered that could be deregistered (CLRPC_ServerDeregister_req()).
0xC02E0404	TLR_E_CLRPC_SERVER_RESOURCE_OUT_OF_MEMORY Insufficient memory to create an instance of RPC-Server.
0xC02E0405	TLR_E_CLRPC_SERVER_MAPPER_HANDLE_INVALID The handle to Endpoint-Mapper in CLRPC_ServerRegister_req() is invalid.
0xC02E0406	TLR_E_CLRPC_SERVER_MAPPER_STATUS_INVALID The status of Endpoint-Mapper in CLRPC_ServerRegister_req() is invalid.



Hexadecimal Value	Definition Description
0xC02E0407	TLR_E_CLRPC_SERVER_HANDLE_INVALID The handle to RPC-Server instance is invalid.
0xC02E0408	TLR_E_CLRPC_SERVER_OBJECT_REGISTERED There is at least one object registered to RPC-Server instance. CLRPC_ServerDeregister_req() cannot proceed.
0xC02E0409	TLR_E_CLRPC_SERVER_PARAM_RECV_INVALID Invalid parameter "uiMaxRecv" in request-packet in CLRPC_ServerRegister_req().
0xC02E040A	TLR_E_CLRPC_SERVER_PARAM_SEND_INVALID Invalid parameter "uiMaxSend" in request-packet in CLRPC_ServerRegister_req().
0xC02E040B	TLR_E_CLRPC_SERVER_ELEMENT_INVALID Invalid RPC-Server element "ptElem". Internal RPC-Error.
0xC02E040C	TLR_E_CLRPC_SERVER_REQUEST_CANCELED This RPC-Request was cancelled.
0xC02E040D	TLR_E_CLRPC_SERVER_STATE_INVALID The state of RPC server is invalid for this request.
0xC02E040E	TLR_E_CLRPC_SERVER_ACTIVITY_ALREADY_INITIALIZED The activity has already been initialized.
0xC02E040F	TLR_E_CLRPC_SERVER_RECEIVED_INVALID_RSP_PACKET The RPC server received an invalid (unexpected) response packet.
0xC02E0500	TLR_E_CLRPC_OBJECT_RESOURCE_OUT_OF_MEMORY Insufficient memory to create an RPC-Object instance in CLRPC_ObjectRegister_req().
0xC02E0501	TLR_E_CLRPC_OBJECT_SERVER_HANDLE_INVALID The handle to RPC-Server instance in CLRPC_ObjectRegister_req() is invalid.
0xC02E0502	TLR_E_CLRPC_OBJECT_SERVER_STATUS_INVALID The status of RPC-Server instance in CLRPC_ObjectRegister_req() is invalid.
0xC02E0503	TLR_E_CLRPC_OBJECT_HANDLE_INVALID The handle to RPC-Object instance in CLRPC_ObjectDeregister_req() is invalid.
0xC02E0600	TLR_E_CLRPC_CLIENT_INIT_FAILED One of the parameters "uiMaxReg" or "uiMaxReq" in CLRPC_Client_Initialize() is invalid.
0xC02E0601	TLR_E_CLRPC_CLIENT_RESOURCE_LIMIT_EXCEEDED The maximum number of parallel RPC-Client instances in reached in CLRPC_ClientRegister_req()
0xC02E0602	TLR_E_CLRPC_CLIENT_TIMER_CREATE_FAILED Creating the TLR-Timer for RPC-Client instance in CLRPC_ClientRegister_req() failed.
0xC02E0603	TLR_E_CLRPC_CLIENT_RESOURCE_OUT_OF_MEMORY Insufficient memory for this request.
0xC02E0604	TLR_E_CLRPC_CLIENT_MAPPER_STATUS_INVALID The state of RPC Client is invalid for this request.
0xC02E0605	TLR_E_CLRPC_CLIENT_HANDLE_INVALID The handle to RPC-Client instance is invalid.
0xC02E0606	TLR_E_CLRPC_CLIENT_REQUEST_LIMIT_EXCEEDED The maximum amount of outstanding RPC-Requests for this RPC-Clients instance is reached.
0xC02E0607	TLR_E_CLRPC_CLIENT_OPCODE_SEQUENCE RPC-Client instances can only connect to an IO-Device if there are no outstanding RPC-Requests. Currently at least one RPC-Request is outstanding.
0xC02E0608	TLR_E_CLRPC_CLIENT_DEREGISTERED The RPC-Client instance you tried to use is going to deregister right now. Aborting your Request !
0xC02E0609	TLR_E_CLRPC_CLIENT_ELEMENT_INVALID Invalid RPC-Client instance element "ptElem". Internal RPC-Error.

Hexadecimal Value	Definition Description
0xC02E060A	TLR_E_CLRPC_CLIENT_LONG_TIMEOUT_HIT The LONG timeout TLR-timer for an outstanding RPC-Request hit. Used internally in RPC only.
0xC02E060B	TLR_E_CLRPC_CLIENT_RESPONSE_SEQUENCE_NUMBER Invalid sequence number in RPC-Message received by RPC-Client instance.
0xC02E060C	TLR_E_CLRPC_CLIENT_CANCEL_TIMED_OUT Canceling a running request timed out. This RPC Client will no longer be usable.
0xC02E060D	TLR_E_CLRPC_CLIENT_NO_REQUEST_PACKET The RPC Client did not have a packet to return.
0xC02E060E	TLR_E_CLRPC_CLIENT_RECV_REQ_WITH_UNEXPECTED_FLAG The RPC Client received a request with an unexpected Flag value.
0xC02E060F	TLR_E_CLRPC_CLIENT_ABORTED_BY_UNBIND_REQ The request was aborted because the RPC client was unbind.
0xC02E0610	TLR_E_CLRPC_MAX_ACTIVITY_RESEND_RETRY_REACHED The maximum resend number was reached by the activity.

Table 200: Status/Error Codes for RPC-Task

## 9.9.1 RPC -Task Diagnosis-Codes

### Packet Status/Error

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC02E0010	TLR_DIAG_E_RPC_TASK_CLIENT_RESOURCE_INIT_FAILED Initiating CLRPC-Client failed. (CLRPC_Client_Initialize())
0xC02E0011	TLR_DIAG_E_RPC_TASK_SERVER_RESOURCE_INIT_FAILED Initiating CLRPC-Server failed (CLRPC_Server_Initialize()).
0xC02E0012	TLR_DIAG_E_RPC_TASK_EPMAP_RESOURCE_INIT_FAILED Initiating CLRPC-Endpoint-Mapper failed (CLRPC_Mapper_Initialize()).
0xC02E0013	TLR_DIAG_E_RPC_TASK_INIT_LOCAL_CREATE_QUE_FAILED Creating message queue failed.
0xC02E0014	TLR_DIAG_E_RPC_TASK_INIT_REMOTE_IDENT_EDD_FAILED Identifying Drv_EDD failed.
0xC02E0015	TLR_DIAG_E_RPC_TASK_INIT_REMOTE_GET_MAC_FAILED Getting the MAC address failed.
0xC02E0016	TLR_DIAG_E_RPC_TASK_INIT_REMOTE_IDENT_TCPUDP_QUE_FAILED Getting queue handle to TCPIP-Task failed.

Table 201: RPC-Task Diagnosis-Codes

# 10 Appendix

## 10.1 List of Tables

Table 1: List of Revisions .....	7
Table 2: Terms, Abbreviations and Definitions .....	12
Table 3: References to Documents .....	13
Table 4: Names of Queues in PROFINET Firmware .....	18
Table 5: Meaning of Destination-Parameter ulDest.Parameters .....	20
Table 6: Example for correct Use of Source- and Destination-related parameters. ....	21
Table 7: Hardware Assembly Options for different xC Ports .....	22
Table 8: Communication Channel Addresses in Dual-Port-Memory .....	23
Table 9: Communication Channel-related Information .....	23
Table 10: Supported process data image synchronization modes .....	26
Table 11: Input Data Image .....	27
Table 12: Output Data Image .....	27
Table 13: General Structure of Packets for non-cyclic Data Exchange. ....	29
Table 14: Channel Mailboxes .....	32
Table 15: Packet Header ulExt field for fragmented transfers .....	33
Table 16: Common Status Structure Definition .....	37
Table 17: Communication State of Change .....	38
Table 18: Meaning of Communication Change of State Flags .....	39
Table 19: Communication Control Block .....	43
Table 20: Naming convention of Input/Output Data .....	47
Table 21: Overview about essential Functionality .....	47
Table 22: PROFINET Device Stack Events .....	48
Table 23: Parameters of UpdateConsumerImage Callback .....	53
Table 24: Return Codes of UpdateConsumerImage Callback .....	53
Table 25: Parameters of UpdateProviderImage Callback .....	54
Table 26: Return Codes of UpdateProviderImage Callback .....	54
Table 27: Communication State (V3.10 and later) .....	56
Table 28: Communication State (V3.9 and earlier) .....	56
Table 29: Overview over the Configuration Packets of the PROFINET IO Device IRT Stack .....	57
Table 30: PNS_IF_SET_CONFIGURATION_REQ_T - Set Configuration Request .....	64
Table 31: Structure tDeviceParameters .....	66
Table 32: System flags to use for configuration of the Stack .....	68
Table 33: Structure PNS_IF_API_STRUCT_T .....	68
Table 34: Structure PNS_IF_SUBMODULE_STRUCT_T .....	69
Table 35: PNS_IF_SET_CONFIGURATION_CNF_T - Set Configuration Confirmation .....	71
Table 36: PNS_REG_FATAL_ERROR_CALLBACK_REQ_T - Register Fatal Error Callback Request .....	77
Table 37: PNS_REG_FATAL_ERROR_CALLBACK_CNF_T - Register Fatal Error Callback Confirmation .....	78
Table 38: PNS_UNREG_FATAL_ERROR_CALLBACK_REQ_T - Unregister Fatal Error Callback Request .....	79
Table 39: PNS_UNREG_FATAL_ERROR_CALLBACK_CNF_T - Unregister Fatal Error Callback Confirmation .....	80
Table 40: PNS_IF_SET_PORT_MAC_REQ_T - Set Port MAC Address Request .....	82
Table 41: PNS_IF_SET_PORT_MAC_CNF_T - Set Port MAC Address Confirmation .....	83
Table 42: PNS_IF_SET_OEM_PARAMETERS_REQ_T - Set OEM Parameters Request .....	86
Table 43: PNS_IF_SET_OEM_PARAMETERS_TYPE_1_T - Set OEM Parameters for ulParamType = 1 .....	86
Table 44: PNS_IF_SET_OEM_PARAMETERS_TYPE_2_T - Set OEM Parameters for ulParamType = 2 .....	86
Table 45: PNS_IF_SET_OEM_PARAMETERS_TYPE_3_T - Set OEM Parameters for ulParamType = 3 .....	87
Table 46: PNS_IF_SET_OEM_PARAMETERS_TYPE_4_T - Set OEM Parameters for ulParamType = 4 .....	87
Table 47: PNS_IF_SET_OEM_PARAMETERS_TYPE_5_T - Set OEM Parameters for ulParamType = 5 .....	87
Table 48: Coding of ulIMFlag .....	88
Table 49: PNS_IF_SET_OEM_PARAMETERS_TYPE_6_T - Set OEM Parameters for ulParamType = 6 .....	88
Table 50: PNS_IF_SET_OEM_PARAMETERS_TYPE_7_T - Set OEM Parameters for ulParamType = 7 .....	88
Table 51: PNS_IF_SET_OEM_PARAMETERS_CNF_T - Set OEM Parameters Confirmation .....	89
Table 52: PNS_IF_LOAD_REMANENT_DATA_REQ_T - Load Remanent Data Request .....	91
Table 53: PNS_IF_LOAD_REMANENT_DATA_CNF_T - Load Remanent Data Confirmation .....	92
Table 54: PNS_IF_SET_IOIMAGE_REQ_T - Set IO-Image Request .....	95
Table 55: PNS_IF_SET_IOIMAGE_CNF_T - Set IO-Image Confirmation .....	97
Table 56: PNS_IF_SET_IOXS_CONFIG_REQ_T - Set IOXS Config Request .....	99
Table 57: PNS_IF_SET_IOXS_CONFIG_CNF_T - Set IOXS Config Confirmation .....	100
Table 58: Structure IO_SIGNALS_CONFIGURE_SIGNAL_REQ_T .....	103
Table 59: IO_SIGNALS_CONFIGURE_SIGNAL_CNF_T - Configure Signal Confirmation .....	104
Table 60: Overview over the Connection Establishment Packets of the PROFINET IO Device IRT Stack .....	111
Table 61: PNS_IF_AR_CHECK_IND_T - AR Check Indication .....	113
Table 62: PNS_IF_AR_CHECK_RSP_T - AR Check Response .....	114

Table 63: PNS_IF_CHECK_IND_T - Check Indication .....	118
Table 64: PNS_IF_CHECK_RSP_T - Check Response .....	120
Table 65: Field usModuleState.....	120
Table 66: Field usSubmodState.....	120
Table 67: PNS_IF_CONNECTREQ_DONE_IND_T - Connect Request Done Indication.....	121
Table 68: PNS_IF_CONNECTREQ_DONE_RSP_T - Connect Request Done Response.....	122
Table 69: PNS_IF_PARAM_END_IND_T - Parameter End Indication .....	124
Table 70: PNS_IF_PARAM_END_RSP_T - Parameter End Response.....	125
Table 71: PNS_IF_APPL_READY_IND_T - Application Ready.....	127
Table 72: PNS_IF_APPL_READY_CNF_T - Application Ready Confirmation.....	128
Table 73: PNS_IF_AR_IN_DATA_IND_T - AR InData Indication .....	129
Table 74: PNS_IF_AR_IN_DATA_RSP_T - AR InData Response .....	130
Table 75: PNS_IF_STORE_REMANENT_DATA_IND_T - Store Remanent Data Indication.....	132
Table 76: PNS_IF_STORE_REMANENT_DATA_RES_T - Store Remanent Data Response.....	132
Table 77: Packet overview of acyclic events indicated by the PROFINET IO Device stack.....	133
Table 78: PNS_IF_READ_RECORD_IND_T - Read Record Indication .....	135
Table 79: PNS_IF_READ_RECORD_RSP_T - Read Record Response .....	137
Table 80: PNS_IF_WRITE_RECORD_IND_T - Write Record Indication .....	139
Table 81: PNS_IF_WRITE_RECORD_RSP_T - Write Record Response .....	141
Table 82: PNS_IF_AR_ABORT_IND_IND_T - AR Abort Indication.....	143
Table 83: PNS_IF_AR_ABORT_IND_RSP_T - AR Abort Indication Response.....	144
Table 84: PNS_IF_SAVE_STATION_NAME_IND_T - Save Station Name Indication.....	146
Table 85: PNS_IF_SAVE_STATION_NAME_RSP_T - Save Station Name Response.....	147
Table 86: PNS_IF_SAVE_IP_ADDRESS_IND_T - Save IP Address Indication .....	149
Table 87: PNS_IF_SAVE_IP_ADDRESS_RSP_T - Save IP Address Response .....	150
Table 88: PNS_IF_START_LED_BLINKING_IND_T - Start LED Blinking Indication .....	151
Table 89: PNS_IF_START_LED_BLINKING_RSP_T - Start LED Blinking Response .....	152
Table 90: PNS_IF_STOP_LED_BLINKING_IND_T - Stop LED Blinking Indication .....	153
Table 91: PNS_IF_STOP_LED_BLINKING_RSP_T - Stop LED Blinking Response.....	154
Table 92: PNS_IF_RESET_FACTORY_SETTINGS_IND_T - Reset Factory Settings Indication.....	157
Table 93: Possible values of the reset code.....	159
Table 94: PNS_IF_RESET_FACTORY_SETTINGS_RSP_T - Reset Factory Settings Response.....	159
Table 95: PNS_IF_APDU_STATUS_CHANGED_IND_T - APDU Status Changed Indication.....	160
Table 96: Meaning of bits of APDU status field.....	161
Table 97: PNS_IF_APDU_STATUS_CHANGED_RSP_T - APDU Status Changed Response.....	162
Table 98: PNS_IF_ALARM_IND_T - Alarm Indication.....	164
Table 99: PNS_IF_ALARM_RSP_T - Alarm Indication Response.....	165
Table 100: PNS_IF_RELEASE_REQ_IND_T - Release Request Indication.....	166
Table 101: PNS_IF_RELEASE_REQ_RSP_T - Release Request Indication Response.....	167
Table 102: PNS_IF_LINK_STATUS_CHANGED_IND_T - Link Status Changed Indication.....	169
Table 103: Structure PNS_IF_LINK_STATUS_DATA_T .....	169
Table 104: PNS_IF_LINK_STATUS_CHANGED_RSP_T - Link Status Changed Response .....	170
Table 105: PNS_IF_USER_ERROR_IND_T - Error Indication Service.....	171
Table 106: PNS_IF_USER_ERROR_RSP_T - Error Indication Response .....	172
Table 107: PNS_IF_READ_IM_IND_T - Read I&M Indication .....	174
Table 108: PNS_IF_READ_IM_RES_T - Read I&M Response .....	175
Table 109: PNS_IF_IM0_DATA_T - Structure of I&M0 Information .....	176
Table 110: PNS_IF_IM1_DATA_T - Structure of I&M1 Information .....	177
Table 111: PNS_IF_IM2_DATA_T - Structure of I&M2 Information .....	177
Table 112: PNS_IF_IM3_DATA_T - Structure of I&M3 Information .....	177
Table 113: PNS_IF_IM4_DATA_T - Structure of I&M4 Information .....	178
Table 114: PNS_IF_IM0_FILTER_DATA_T - Structure of I&M0 Filter Information .....	178
Table 115: PNS_IF_WRITE_IM_IND_T - Write I&M Indication .....	180
Table 116: PNS_IF_WRITE_IM_RES_T - Write I&M Response .....	181
Table 117: PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_T - Parameterization Speedup Supported Indication.....	183
Table 118: PNS_IF_PARAMET_SPEEDUP_SUPPORTED_RES_T - Parameterization Speedup Supported Response.....	184
Table 119: Packet overview of acyclic events of the PROFINET IO Device stack requested by the application .....	188
Table 120: PNS_IF_GET_DIAGNOSIS_REQ_T - Get Diagnosis Request.....	189
Table 121: PNS_IF_GET_DIAGNOSIS_CNF_T - Get Diagnosis Confirmation.....	191
Table 122: Meaning of single Bits in ulPnsState .....	191
Table 123: Values and their corresponding Meanings of ulLinkState.....	192
Table 124: Values and their corresponding Meanings of ulConfigState .....	192
Table 125: PNS_IF_GET_XMAC_DIAGNOSIS_REQ_T - Get XMAC (EDD) Diagnosis Request.....	194
Table 126: PNS_IF_GET_XMAC_DIAGNOSIS_CNF_T - Get XMAC (EDD) Diagnosis Confirmation.....	195
Table 127: Structure EDD_XMAC_COUNTERS_T .....	196

Table 128: PNS_IF_SEND_PROCESS_ALARM_REQ_T - Process Alarm Request .....	198
Table 129: PNS_IF_SEND_PROCESS_ALARM_CNF_T - Process Alarm Confirmation .....	199
Table 130: PNS_IF_SEND_DIAG_ALARM_REQ_T - Diagnosis Alarm Request .....	201
Table 131: PNS_IF_DIAG_ALARM_CNF_T - Diagnosis Alarm Confirmation .....	202
Table 132: PNS_IF_RETURN_OF_SUB_ALARM_REQ_T - Return of Submodule Alarm Request .....	204
Table 133: PNS_IF_RETURN_OF_SUB_ALARM_CNF_T - Return of Submodule Alarm Confirmation .....	205
Table 134: PNS_IF_ABORT_CONNECTION_REQ_T - AR Abort Request .....	206
Table 135: PNS_IF_ABORT_CONNECTION_CNF_T - AR Abort Request Confirmation .....	207
Table 136: PNS_IF_PLUG_MODULE_REQ_T - Plug Module Request .....	209
Table 137: PNS_IF_PLUG_MODULE_CNF_T - Plug Module Confirmation .....	210
Table 138: PNS_IF_PLUG_SUBMODULE_REQ_T - Plug Submodule Request .....	214
Table 139: PNS_IF_PLUG_SUBMODULE_CNF_T - Plug Submodule Confirmation .....	216
Table 140: PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_T - Extended Plug Submodule Request .....	218
Table 141: PNS_IF_PLUG_SUBMODULE_EXTENDED_CNF_T - Extended Plug Submodule Confirmation .....	220
Table 142: PNS_IF_PULL_MODULE_REQ_T - Pull Module Request .....	222
Table 143: PNS_IF_PULL_MODULE_CNF_T - Pull Module Confirmation .....	223
Table 144: PNS_IF_PULL_SUBMODULE_REQ_T - Pull Submodule Request .....	225
Table 145: PNS_IF_PULL_SUBMODULE_CNF_T - Pull Submodule Confirmation .....	226
Table 146: PNS_IF_GET_STATION_NAME_REQ_T - Get Station Name Request .....	227
Table 147: PNS_IF_GET_STATION_NAME_CNF_T - Get Station Name Confirmation .....	228
Table 148: PNS_IF_GET_IP_ADDR_REQ_T - Get IP Address Request .....	229
Table 149: PNS_IF_GET_IP_ADDR_CNF_T - Get IP Address Confirmation .....	230
Table 150: PNS_IF_ADD_CHANNEL_DIAG_REQ_T - Add Channel Diagnosis Request .....	232
Table 151: Coding of usChannelErrType. Note: values 0x8000 – 0x800E will be triggered by the Stack internally and shall not be set by application .....	233
Table 152: Coding of the field usChannelProp .....	233
Table 153: Coding of the field Type in field usChannelProp .....	234
Table 154: Coding of the field Maintenance in field usChannelProp .....	234
Table 155: Coding of the field <b>Direction</b> in field usChannelProp .....	234
Table 156: PNS_IF_ADD_CHANNEL_DIAG_CNF_T - Add Channel Diagnosis Confirmation .....	236
Table 157: PNS_IF_ADD_EXTENDED_DIAG_REQ_T - Add Extended Channel Diagnosis Request .....	238
Table 158: Coding of usExtChannelErrType for Channel Error Type 1 - 0x7FFF .....	238
Table 159: PNS_IF_ADD_EXTENDED_DIAG_CNF_T - Add Extended Channel Diagnosis Confirmation .....	240
Table 160: PNS_IF_ADD_GENERIC_DIAG_REQ_T - Add Generic Channel Diagnosis Request .....	242
Table 161: Coding of the field usUserStructId .....	242
Table 162: PNS_IF_ADD_GENERIC_DIAG_CNF_T - Add Generic Channel Diagnosis Confirmation .....	243
Table 163: PNS_IF_REMOVE_DIAG_REQ_T - Remove Diagnosis Request .....	244
Table 164: PNS_IF_REMOVE_DIAG_CNF_T - Remove Diagnosis Confirmation .....	245
Table 165: PNS_IF_GET_CONFIGURED_SUBM_CNF_T - Get Submodule Configuration .....	247
Table 166: Elements of PNS_IF_CONFIGURED_SUBM_STRUCT_T .....	247
Table 167: PNS_IF_SET_SUBM_STATE_REQ_T - Set Submodule State Request .....	251
Table 168: Possible Values of usSubmState .....	251
Table 169: PNS_IF_SET_SUBM_STATE_CNF_T - Set Submodule State Confirmation .....	252
Table 170: PNS_IF_GET_PARAMETER_REQ_T - Get Parameter Request .....	254
Table 171: PNS_IF_PARAM_E - Valid parameter options .....	254
Table 172: PNS_IF_GET_PARAM_CNF_T - Get Diagnosis Confirmation .....	257
Table 173: PNS_IF_PARAM_MRP_STATE_E - Valid values for MRP state .....	257
Table 174: PNS_IF_PARAM_MRP_ROLE_E - Valid values for MRP Role .....	257
Table 175: Handling of basic parameters .....	259
Table 176: Smallest possible Cycle Time depending using multiple ARs .....	260
Table 177: Hardware resources used by the PROFINET Device stack for netX 100/500 .....	262
Table 178: Hardware resources used by the PROFINET Device stack for netX 50 .....	263
Table 179: Hardware resources used by the PROFINET Device stack for netX 51 .....	263
Table 180: Overview about the recommended Task Priorities (*priority of the RX_TIMER should be set to TSK_PRIO_5 only for Isochronous Applications with fast cycles 250µs, see also [8.6]) .....	271
Table 181: MAU types for fiber optic ports .....	278
Table 182: Coding of PNIO Status ErrorCode (Excluding reserved Values) .....	293
Table 183: Coding of PNIO status ErrorDecode (Excluding reserved Values) .....	293
Table 184: Coding of ErrorCode1 for ErrorDecode = PNIO. (Excluding reserved Values) .....	294
Table 185: Coding of ErrorCode1 for ErrorDecode = PNIO. (Excluding reserved Values) .....	300
Table 186: Status/Error Codes Overview .....	312
Table 187: Status/Error Codes for CMCTL Task .....	315
Table 188: CMCTL -Task Diagnosis-Codes .....	316
Table 189: Status/Error Codes for CM-Dev Task .....	321
Table 190: CM-Dev-Task Diagnosis-Codes .....	322
Table 191: Status/Error Codes for EDD Task .....	323
Table 192: EDD-Task Diagnosis-Codes .....	323

Table 193: Status/Error Codes for ACP Task.....	327
Table 194: ACP-Task Diagnosis-Codes.....	327
Table 195: Status/Error Codes for DCP Task .....	330
Table 196: DCP-Task Diagnosis-Codes.....	331
Table 197: Status/Error Codes for MGT Task.....	334
Table 198: MGT-Task Diagnosis-Codes .....	335
Table 199: Status/Error Codes Overview.....	335
Table 200: Status/Error Codes for RPC-Task .....	338
Table 201: RPC-Task Diagnosis-Codes.....	339

## 10.2 List of Figures

Figure 1: The 3 different Ways to access a Protocol Stack running on a netX System.....	17
Figure 2: Use of ulDest in Channel and System Mailbox.....	19
Figure 3: Using ulSrc and ulSrcId .....	20
Figure 4: Packet Types .....	24
Figure 5: Splitting a large packet into fragments (figure shows 3 fragments) .....	34
Figure 6: Sequence of fragmented indication and response .....	36
Figure 7: Task Structure of the PROFINET IO Device Stack V3.7 .....	44
Figure 8: Provider Process Data Structure.....	59
Figure 9: Consumer Process Data Structure.....	60
Figure 10: Configuring the PROFINET IO Device Stack .....	62
Figure 11: Register Application Service Packet sequence.....	73
Figure 12: Initial Configuration by DCP without topology information at controller side. ....	106
Figure 13: Initial Configuration by DCP using topology information at controller side. ....	107
Figure 14: Sequence between Controller, Stack and Application during Connection Establishment .....	108
Figure 15: Sequence between Controller, Stack and Application during Connection Establishment (continued) .....	109
Figure 16: Check Service Packet sequence for non adapting applications.....	115
Figure 17: Check Service Packet sequence for adapting applications.....	116
Figure 18: Desired Application behavior on Save Station Name Indication .....	145
Figure 19: Desired application behavior on Save IP Address Indication .....	148
Figure 20: Application behavior on Reset Factory Settings Indication .....	155
Figure 21: Event Indication.....	186
Figure 22: Event Indication Response .....	186
Figure 23: Plug Submodule Service packet sequence.....	211
Figure 24: Plug Submodule Service packet sequence (continued) .....	212
Figure 25: Pull Module Service packet sequence .....	221
Figure 26: Pull Submodule Service packet sequence .....	224
Figure 27: SetSubModuleState from bad to good .....	249
Figure 28: Sequence of MAC Address determination .....	261
Figure 29: Vendor and device identification in the GSDML file .....	282
Figure 30: Structural organization of I&M Records within a Device and the Access Paths.....	286
Figure 31: PROFINET Stack is accessible on the DPM Channel0 and Ethernet Interface – on the Channel1 .....	288
Figure 32: PROFINET Stack is accessible on the DPM channel0 and Ethernet Interfaceon the channel1.....	289
Figure 33: Priority setup of the RX_TIMER task in the loadable PROFINET firmware for isochronous application.....	291
Figure 34: Structure of the PROFINET Status Code.....	292

## 10.3 Contacts

### Headquarters

#### Germany

Hilscher Gesellschaft für  
Systemautomation mbH  
Rheinstrasse 15  
65795 Hattersheim  
Phone: +49 (0) 6190 9907-0  
Fax: +49 (0) 6190 9907-50  
E-Mail: [info@hilscher.com](mailto:info@hilscher.com)

#### Support

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

### Subsidiaries

#### China

Hilscher Systemautomation (Shanghai) Co. Ltd.  
200010 Shanghai  
Phone: +86 (0) 21-6355-5161  
E-Mail: [info@hilscher.cn](mailto:info@hilscher.cn)

#### Support

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

#### France

Hilscher France S.a.r.l.  
69500 Bron  
Phone: +33 (0) 4 72 37 98 40  
E-Mail: [info@hilscher.fr](mailto:info@hilscher.fr)

#### Support

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

#### India

Hilscher India Pvt. Ltd.  
Pune, Delhi, Mumbai  
Phone: +91 8888 750 777  
E-Mail: [info@hilscher.in](mailto:info@hilscher.in)

#### Italy

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

#### Support

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

#### Japan

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

#### Support

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

#### Korea

Hilscher Korea Inc.  
Seongnam, Gyeonggi, 463-400  
Phone: +82 (0) 31-789-3715  
E-Mail: [info@hilscher.kr](mailto:info@hilscher.kr)

#### Switzerland

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

#### Support

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

#### USA

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

#### Support

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