



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

**V3.4.x.x**

**Hilscher Gesellschaft für Systemautomation mbH**

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

DOC081102API14EN | Revision 14 | English | 2013-10 | 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 Version V3.4.x.x .....	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.....	12
1.8	Legal Notes .....	13
1.8.1	Copyright.....	13
1.8.2	Important Notes.....	13
1.8.3	Exclusion of Liability .....	14
1.8.4	Export .....	14
<b>2</b>	<b>Fundamentals .....</b>	<b>15</b>
2.1	General Access Mechanisms on netX Systems .....	15
2.2	Accessing the Protocol Stack by Programming the AP Task's Queue.....	16
2.2.1	Getting the Receiver Task Handle of the Process Queue .....	16
2.2.2	Meaning of Source- and Destination-related Parameters.....	16
2.3	Accessing the Protocol Stack via the Dual Port Memory Interface.....	17
2.3.1	Communication via Mailboxes.....	17
2.3.2	Using Source and Destination Variables correctly.....	18
2.3.3	Obtaining useful Information about the Communication Channel.....	21
<b>3</b>	<b>Dual-Port Memory .....</b>	<b>24</b>
3.1	Cyclic Data (Input/Output Data) .....	24
3.1.1	Input Process Data .....	25
3.1.2	Output Process Data .....	25
3.2	Acyclic Data (Mailboxes).....	26
3.2.1	General Structure of Messages or Packets for Non-Cyclic Data Exchange .....	27
3.2.2	Status & Error Codes .....	30
3.2.3	Differences between System and Channel Mailboxes .....	30
3.2.4	Send Mailbox.....	30
3.2.5	Receive Mailbox .....	30
3.2.6	Channel Mailboxes (Details of Send and Receive Mailboxes) .....	31
3.3	Status .....	32
3.3.1	Common Status.....	32
3.3.2	Extended Status .....	38
3.4	Control Block.....	39
3.5	Isochronous Synchronization Event.....	39
<b>4</b>	<b>Getting Started.....</b>	<b>41</b>
4.1	Overview about Essential Functionality .....	41
4.2	Description of behavior of stack under special situations .....	42
4.2.1	Sequence of configuration evaluation .....	42
4.2.2	Configuration Lock .....	42
4.2.3	Setting Parameters by means of DCP .....	43
<b>5</b>	<b>Exchanging cyclic Data .....</b>	<b>44</b>
5.1	General Concepts .....	44
5.2	Exchanging cyclic Data using Callback Interface .....	45
5.2.1	Overview of the Callback Interface .....	45
5.2.2	Callback Functions .....	45
<b>6</b>	<b>Isochronous User Applications .....</b>	<b>51</b>
6.1	Parameter and Values .....	51
6.2	Clock Synchronization.....	53
6.3	Application Cycle and Timings.....	54

6.4	IRT Cycle Counter.....	55
<b>7</b>	<b>Configuring the IO-Device Stack .....</b>	<b>56</b>
7.1	Task Structure of the PROFINET IO Device Stack.....	57
7.2	Configuration of the Memory Area without DPM .....	60
7.2.1	Configuration of the Consumer/Provider Memory Area for non DPM /SHM access .....	60
7.3	Configuration of IOXS States.....	61
7.4	Configuration of the Submodules.....	61
7.5	Commissioning the PROFINET IO Device Stack V3.....	62
7.5.1	Remark on Reconfiguration.....	63
7.6	Set Configuration Service .....	64
7.6.1	Set Configuration Request .....	65
7.6.2	Set Configuration Confirmation .....	75
7.6.3	Behavior when receiving a Set Configuration Command .....	76
7.7	Warmstart Service.....	77
7.7.1	Warmstart Request .....	78
7.7.2	Warmstart Confirmation.....	81
7.8	Get Device Handle Service.....	82
7.8.1	Get Device Handle Request.....	82
7.8.2	Get Device Handle Confirmation .....	83
7.9	Register Application Service .....	84
7.9.1	Register Application Request .....	84
7.9.2	Register Application Confirmation .....	84
7.10	Unregister Application Service.....	85
7.10.1	Unregister Application Request .....	85
7.10.2	Unregister Application Confirmation .....	85
7.11	Register Fatal Error Callback Service.....	86
7.11.1	Register Fatal Error Callback Request .....	86
7.11.2	Register Fatal Error Callback Confirmation .....	88
7.12	Unregister Fatal Error Callback Service.....	89
7.12.1	Unregister Fatal Error Callback Request.....	89
7.12.2	Unregister Fatal Error Callback Confirmation.....	90
7.13	Set Port MAC Address Service.....	91
7.13.1	Set Port MAC Address Request .....	91
7.13.2	Set Port MAC Address Confirmation .....	93
7.14	Set OEM Parameters Service.....	94
7.14.1	Set OEM Parameters Request.....	94
7.14.2	Set OEM Parameters Confirmation .....	97
7.15	Load Remanent Data Service .....	99
7.15.1	Load Remanent Data Request.....	100
7.15.2	Load Remanent Data Confirmation .....	102
7.16	Configuration Delete Service .....	103
7.16.1	Configuration Delete Request .....	103
7.16.2	Configuration Delete Confirmation .....	103
7.17	Set IO-Image Service.....	104
7.17.1	Set IO-Image Request.....	104
7.17.2	Set IO-Image Confirmation.....	106
7.18	Set IOXS Config Service.....	108
7.18.1	Set IOXS Config Request.....	108
7.18.2	Set IOXS Config Confirmation.....	110
7.19	Configure Signal Service.....	111
7.19.1	Configure Signal Request.....	111
7.19.2	Configure Signal Confirmation.....	114
7.19.3	Example: Configure Signal Request packet .....	114
7.20	Usage of Linkable Object Module .....	116
7.20.1	Task Startup Parameters.....	116
7.20.2	Task priorities .....	122
7.20.3	Additional Hardware Resources .....	122
<b>8</b>	<b>Connection Establishment.....</b>	<b>126</b>
8.1	AR Check .....	129
8.1.1	AR Check Indication.....	129
8.1.2	AR Check Response .....	131
8.2	Check Indication Service.....	132
8.2.1	Check Indication.....	132
8.2.2	Check Response .....	134
8.2.3	Module Reconfiguration after a Check Indication .....	137

8.3	Connect Request Done Service.....	138
8.3.1	Connect Request Done Indication.....	138
8.3.2	Connect Request Done Response.....	140
8.4	Parameter End Service.....	141
8.4.1	Parameter End Indication.....	141
8.4.2	Parameter End Response.....	143
8.5	Application Ready Service.....	145
8.5.1	Application Ready Request.....	145
8.5.2	Application Ready Confirmation.....	147
8.6	AR InData Service.....	148
8.6.1	AR InData Indication.....	148
8.6.2	AR InData Response.....	150
8.7	Store Remanent Data Service.....	151
8.7.1	Store Remanent Data Indication.....	151
8.7.2	Store Remanent Data Response.....	153
<b>9</b>	<b>Acyclic Events indicated by the Stack.....</b>	<b>154</b>
9.1	Read Record Service.....	156
9.1.1	Read Record Indication.....	156
9.1.2	Read Record Response.....	158
9.2	Write Record Service.....	169
9.2.1	Write Record Indication.....	169
9.2.2	Write Record Response.....	171
9.3	AR Abort Indication service.....	173
9.3.1	AR Abort Indication Indication.....	173
9.3.2	AR Abort Indication Response.....	175
9.4	Save Station Name Service.....	176
9.4.1	Save Station Name Indication.....	176
9.4.2	Save Station Name Response.....	178
9.5	Save Station Type Service.....	179
9.5.1	Save Station Type Indication.....	179
9.5.2	Save Station Type Response.....	181
9.6	Save IP Address Service.....	182
9.6.1	Save IP Address Indication.....	182
9.6.2	Save IP Address Response.....	184
9.7	Start LED Blinking Service.....	185
9.7.1	Start LED Blinking Indication.....	185
9.7.2	Start LED Blinking Response.....	187
9.8	Stop LED Blinking Service.....	188
9.8.1	Stop LED Blinking Indication.....	188
9.8.2	Stop LED Blinking Response.....	189
9.9	Reset Factory Settings Service.....	190
9.9.1	Reset Factory Settings Indication.....	190
9.9.2	Reset Factory Settings Response.....	192
9.10	APDU Status Changed Service.....	193
9.10.1	APDU Status Changed Indication.....	193
9.10.2	APDU Status Changed Response.....	195
9.11	Alarm Indication Service.....	196
9.11.1	Alarm Indication Indication.....	196
9.11.2	Alarm Indication Response.....	199
9.12	Release Request Indication Service.....	200
9.12.1	Release Request Indication.....	200
9.12.2	Release Request Indication Response.....	202
9.13	Link Status Changed Service.....	204
9.13.1	Link Status Changed Indication.....	204
9.13.2	Link Status Changed Response.....	206
9.14	Error Indication Service.....	207
9.14.1	Error Indication.....	207
9.14.2	Error Indication Response.....	209
9.15	Read I&M Service.....	210
9.15.1	Read I&M Indication.....	210
9.15.2	Read I&M Response.....	211
9.16	Write I&M Service.....	216
9.16.1	Write I&M Indication.....	216
9.16.2	Write I&M Response.....	218
9.17	Parameterization Speedup Support.....	219
9.17.1	Parameterization Speedup Support Indication.....	219

9.17.2	Parameterization Speedup Supported Response .....	221
<b>10</b>	<b>Acyclic Events requested by the Application .....</b>	<b>222</b>
10.1	Get Diagnosis Service.....	224
10.1.1	Get Diagnosis Request.....	224
10.1.2	Get Diagnosis Confirmation.....	225
10.2	Get XMAC (EDD) Diagnosis Service .....	229
10.2.1	Get XMAC (EDD) Diagnosis Request .....	229
10.2.2	Get XMAC (EDD) Diagnosis Confirmation .....	230
10.3	Process Alarm Service.....	233
10.3.1	Process Alarm Request.....	233
10.3.2	Process Alarm Confirmation.....	235
10.4	Diagnosis Alarm Service .....	237
10.4.1	Diagnosis Alarm Request.....	237
10.4.2	Diagnosis Alarm Confirmation .....	239
10.5	Return of Submodule Alarm Service.....	241
10.5.1	Return of Submodule Alarm Request.....	241
10.5.2	Return of Submodule Alarm Confirmation .....	243
10.6	AR Abort Request Service .....	244
10.6.1	AR Abort Request Request .....	244
10.6.2	AR Abort Request Confirmation .....	246
10.7	Plug Module Service .....	247
10.7.1	Plug Module Request .....	247
10.7.2	Plug Module Confirmation .....	249
10.8	Plug Submodule Service.....	250
10.8.1	Extended Plug Submodule Request.....	254
10.8.2	Extended Plug Submodule Confirmation.....	257
10.9	Pull Module Service .....	260
10.9.1	Pull Module Request .....	260
10.9.2	Pull Module Confirmation .....	262
10.10	Pull Submodule Service .....	264
10.10.1	Pull Submodule Request.....	264
10.10.2	Pull Submodule Confirmation.....	266
10.11	Set Submodule State Service .....	267
10.11.1	Set Submodule State Request.....	269
10.11.2	Set Submodule State Confirmation.....	271
10.12	Get Station Name Service.....	272
10.12.1	Get Station Name Request .....	272
10.12.2	Get Station Name Confirmation .....	273
10.13	Get Station Type Service .....	274
10.13.1	Get Station Type Request.....	274
10.13.2	Get Station Type Confirmation.....	275
10.14	Get IP Address Service.....	276
10.14.1	Get IP Address Request .....	276
10.14.2	Get IP Address Confirmation .....	277
10.15	Add Channel Diagnosis Service .....	278
10.15.1	Add Channel Diagnosis Request .....	278
10.15.2	Add Channel Diagnosis Confirmation .....	283
10.16	Add Extended Channel Diagnosis Service .....	285
10.16.1	Add Extended Channel Diagnosis Request .....	285
10.16.2	Add Extended Channel Diagnosis Confirmation .....	287
10.17	Add Generic Channel Diagnosis Service.....	289
10.17.1	Add Generic Channel Diagnosis Request.....	289
10.17.2	Add Generic Channel Diagnosis Confirmation.....	291
10.18	Remove Diagnosis Service .....	293
10.18.1	Remove Diagnosis Request.....	293
10.18.2	Remove Diagnosis Confirmation.....	294
<b>11</b>	<b>Status/Error Codes Overview.....</b>	<b>295</b>
11.1	General Errors.....	295
11.2	Status/Error Codes for CMCTL Task .....	307
11.2.1	CMCTL-Task Diagnosis-Codes .....	311
11.3	Status/Error Codes for CM-Dev Task .....	312
11.3.1	CM-Dev-Task Diagnosis-Codes .....	317
11.4	Status/Error Codes for EDD Task.....	318
11.4.1	EDD-Task Diagnosis-Codes.....	318
11.5	Status/Error Codes for ACP Task .....	319

---

11.5.1	ACP-Task Diagnosis-Codes .....	322
11.6	Status/Error Codes for DCP Task .....	323
11.6.1	DCP-Task Diagnosis-Codes .....	326
11.7	Status/Error Codes for MGT Task .....	327
11.7.1	MGT-Task Diagnosis-Codes .....	330
<b>12</b>	<b>Appendix .....</b>	<b>331</b>
12.1	List of Figures .....	331
12.2	List of Tables .....	331
12.3	Contacts .....	335

# 1 Introduction

## 1.1 About this Document

This manual describes the user interface of the PROFINET IO-Device implementation (stack version 3.x) on the netX chip. 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
9	2012-03-02	HH  AM BM	<p>Description for the configuration of IOXS, using the set IO-Image Service and the configuration of the DPM offsets for each submodule.</p> <p>Description for Multiple ARs (shared device, supervisor and supervisor da)</p> <p>Extended description of <i>Parameterization Speedup Support</i> service</p> <p>Updated figure Exchange of Packets between Stack and Application during Connection Establishment</p> <p>Updated textual description in section <i>Connection Establishment</i></p> <p>Add description for I&amp;M0 Filter Data Response</p> <p>Updated Error codes. Removed CR Info Indication.</p> <p>Extended <i>Set OEM Parameters Request</i></p> <p>Added note to IO-data function callback section that callback functions shall not be called in interrupt context</p>
10	2012-04-26	RG  BM	<p>Firmware Version 3.4.36</p> <p>Reference to netX Dual-Port Memory Interface Manual Revision 12</p> <p>Added reference to application note on the integrated Web Server.</p> <p>Extended more details of required application behavior in sections <i>Save IP Address Service</i>, <i>Reset Factory Settings Service</i>, <i>Read I&amp;M Response</i></p> <p>More detailed channel properties for <i>Add Channel Diagnosis Request</i></p>
11	2012-08-17	BM	<p>Added clarification regarding usage of Interface- and PortSubmoduleItems</p> <p>Added new service <i>Set Submodule State Service</i></p> <p>Removed outdated section of Specifications for Stack Version V3.3.x.x</p>
12	2013-01-17	RG	<p>Firmware Version 3.4.38</p> <p>Reference to netX Dual-Port Memory Interface Manual Revision 12</p> <p>Clarified description of status byte of <code>PNS_IF_APDU_STATUS_CHANGED_IND_T</code> packet</p> <p>Some minor corrections</p>
13	2013-09-24	RG	<p>Firmware Version 3.4.42</p> <p>Reference to netX Dual-Port Memory Interface Manual Revision 12</p> <p>Added new section 7.2.1 "Configuration of the Consumer/Provider Memory Area for non DPM /SHM access"</p> <p>Improved description of <code>ulSta</code> variable in <i>Write Record Response</i></p> <p>Added new section describing EthernetInterface functionality</p>
14	2013-10-22	HH	<p>Firmware Version 3.4.144.0</p> <p>Section <i>Technical Data</i>: Ethernet Interface added</p>

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 FastStartUp 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 Version V3.4.x.x

The data below applies to PROFINET IO-Device firmware and stack version 3.4.x.x.

### 1.5.1 Supported Protocols

RTC – Real time Cyclic Protocol

- Class 1 & Class 2 (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	1024 bytes
Maximum number of total cyclic output data	1024 bytes
Acyclic communication	Read/Write Record, max. 1024 Bytes user record data payload for DPM targets. Since V3.4.11.0 up to 32 Kbytes user record data payload for LOM target
Alarm Types	Process Alarm, Diagnostic Alarm, Return of Submodule Alarm, Plug Alarm (implicit), Pull Alarm (implicit)
DCP	
Identification & Maintenance	Since V3.4.12.0 the stacks supports reading and writing of I&M1-4. <sup>1</sup>
Topology recognition	LLDP, SNMP V1, MIB2, physical device
VLAN- and priority tagging	
Context Management by CL-RPC	
Minimum cycle time	1ms for RTC1 & RTC2 250µs for RTC3 with netX100/netx500 1ms for RTC3 with netX50
Baud rate	100 MBit/s
Data transport layer	Ethernet II, IEEE 802.3
Integrated Web Server	supported (for details, see reference #6)

<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.

**Firmware/stack available for netX**

netX 50	yes
netX 100, netX 500	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.

**Ethernet Interface**

Starting with firmware version 3.4.144.0 the loadable firmwares of PROFINET IO Device stack have integrated a new Ethernet Interface component. This supports two modes: NDIS mode and TCP/IP mode. The mode can be set by a tag list entry.

The following restrictions apply for the NDIS mode:

1. Some special frames are not allowed for receiving and sending and are automatically filtered:
  - LLDP (EtherType 0x88CC)
  - MRP (EtherType 0x88E3)
  - PROFINET Sync frames (EtherType 0x8892, FrameID 0x0000 up to FrameID 0x00FF)
  - PROFINET PTCP frames (EtherType 0x8892, FrameID 0xFF00 up to FrameID 0xFFFF)
2. The maximum transfer rate depends on the network load and netX CPU load and thus no guarantee for a specific transfer rate can be given. In a clean environment without PROFINET communication transfer rate up to 50 Mbit/s where reached using netX 100.
3. It is not possible to send frames with a different MAC address than the special NDIS MAC address.

The following restrictions apply for the TCP/IP mode:

1. The maximum transfer rate depends on the network load and netX CPU load and thus no guarantee for a specific transfer rate can be given.
2. Only IP frames and ARP frames can be received.
3. Some IP ports are blocked by the firmware and can thus not be used by the application: 68, 80, 161, 25383, 34964, 49152

4. The port range that is available for the application can be changed by a tag list entry (default is 1024-2048)
5. The IP address available for the application corresponds to the one used by PROFINET IO Device stack. It can not be changed by the application.

### 1.5.3 Limitations

- RT over UDP not supported
- Multicast communication not supported
- Only one device instance is supported
- DHCP is not supported
- IRT “flex” (synchronized RT Class 2) is not supported
- FastStartUp is implemented in the stack. However some additional hardware limitations apply to use it.
- Media Redundancy (except MRP client) is not supported
- Multiple APIs are supported since V3.4.12.0. Previous versions only support one API (API 0).
- Access to the submodule granular status bytes (IOCS) is currently not supported if the application accesses the stack using the Dual-Port Memory interface.
- The amount of configured IO-data influences the minimum cycle time that can be reached.
- Supervisor-AR is not supported. Since V3.4.12.0 Supervisor-DA-Ar (Device Access supported)
- Only 1 Input-CR and 1 Output-CR 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
- If the stack handles I&M data the IMRevisionCounter is not handled correctly according specification
- The stack does not support usage of PDEV submodules (InterfaceSubmodule or PortSubmodule) outside of slot 0. In addition the InterfaceSubmodule is only supported in subslot 0x8000 and the PortSubmodules are only supported in subslots 0x8001 and 0x8002.

## 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

Table 2: Terms, Abbreviations and Definitions

## 1.7 References to Documents

This document based on the following specification respectively documents:

1	Task Layer Reference Model; Rev. 4; Hilscher GmbH; 2006
2	PROFINET IO; Application Layer Service Definition, Application Layer Protocol Specification, Version 2.2, October 2007, Order No.: 2.722, PROFIBUS International
3	Operating System Manual – Realtime Communication System for netX – Kernel API Function Reference; Rev. 6; Hilscher GmbH; 2007
4	DPM Interface Manual for netX based Products; Hilscher Gesellschaft für Systemautomation mbH; 2006-2008, Revision 12
5	TCP/IP Protocol Interface Manual; Rev. 8; Hilscher GmbH; 2009
6	Application Note: Functions of the Integrated WebServer, Hilscher GmbH, Revision 4, English, 2012

Table 3: References

## 1.8 Legal Notes

### 1.8.1 Copyright

© 2006-2013 Hilscher Gesellschaft für Systemautomation mbH

All rights reserved.

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

### 1.8.2 Important Notes

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

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

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

### 1.8.3 Exclusion of Liability

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

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

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

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

### 1.8.4 Export

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

## 2 Fundamentals

### 2.1 General Access Mechanisms on netX Systems

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

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

The picture below visualizes these three ways:

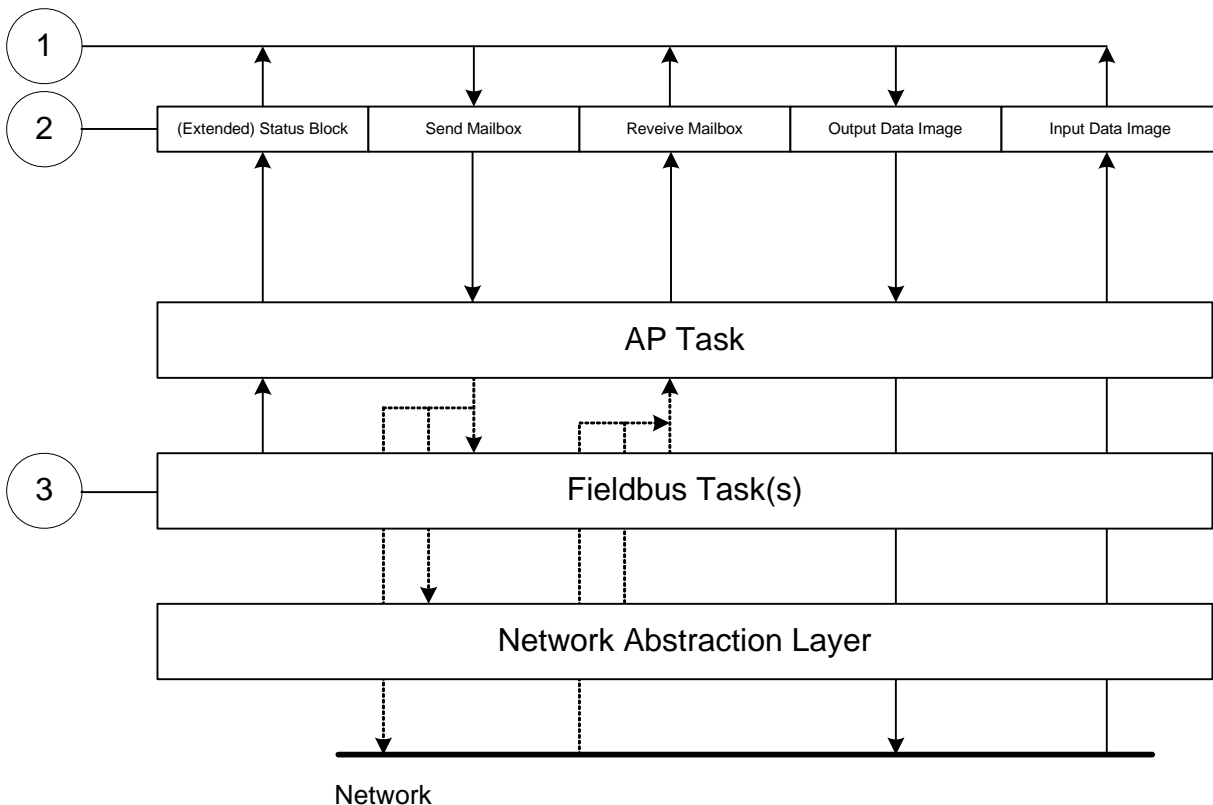


Figure 1: The 3 different Ways to access a Protocol Stack running on a netX System

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

In general, programming the AP task or the stack has to be performed according to the rules explained in the Hilscher Task Layer Reference Manual. There you can also find more information about the variables discussed in the following.

### 2.2.1 Getting the Receiver Task Handle of the Process Queue

To get the handle of the process queue of the `PNS_IF` –Task, the Macro `TLR_QUE_IDENTIFY()` needs to be used. It is described in detail within section 10.1.9.3 of the Hilscher Task Layer Reference Model Manual. This macro delivers a pointer to the handle of the intended queue to be accessed (which is returned within the third parameter, `phQue`), if you provide it with the name of the queue (and an instance of your own task). The correct ASCII-queue names for accessing the `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

The returned handle 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.

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

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

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

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

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

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



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

This chapter defines the application interface of the PROFINET IO RT/IRT Device protocol stack.

### 2.3.1 Communication via Mailboxes

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

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

For more details about acyclic data transfer via mailboxes see section 3.2. [Acyclic Data \(Mailboxes\)](#) in this context, is described in detail in section 3.2.1 “[General Structure of Messages or Packets for Non-Cyclic Data Exchange](#)” while the possible codes that may appear are listed in section 3.2.2. “[Status & Error Codes](#)”.

However, this section concentrates on correct addressing the mailboxes.

## 2.3.2 Using Source and Destination Variables correctly

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

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

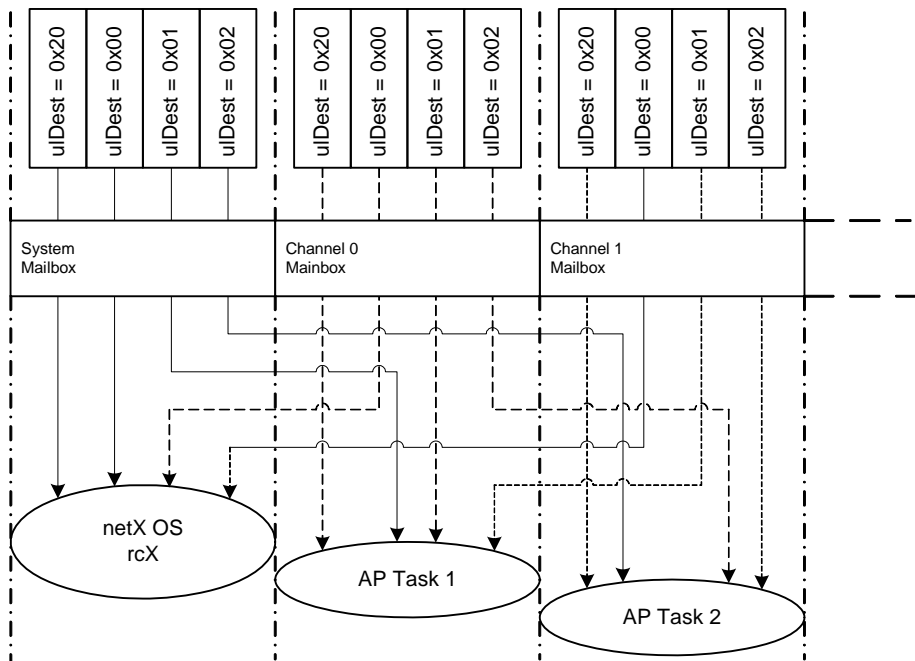


Figure 2: Use of `ulDest` in Channel and System Mailbox

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

<code>ulDest</code>	Description
0x00000000	Packet is passed to the <code>netX</code> operating system <code>rcX</code>
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 6: Meaning of Destination-Parameter `ulDest`.Parameters.

The figure and the table above both show the use of the destination identifier `ulDest`.

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

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

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

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

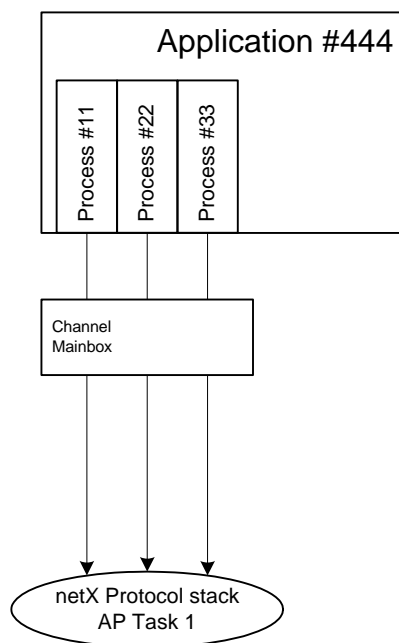


Figure 3: Using `ulSrc` and `ulSrcId`

**Example:**

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

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

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

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

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

### 2.3.2.3 How to Route rcX Packets

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

### 2.3.3 Obtaining useful Information about the Communication Channel

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

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

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

Proceed as follows:

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

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

Table 8: 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 9: 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 <code>define RCX_CHANNEL_TYPE_COMMUNICATION = 0x05</code> )
0x0051	UINT8	ID (Channel Number, Port Number)
0x0052	UINT8	Size / Position Of Handshake Cells
0x0053	UINT8	Total Number Of Blocks Of This Channel
0x0054	UINT32	Size Of Channel In Bytes
0x0058	UINT8[8]	Reserved (set to zero)

Table 10: 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".

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 11: Communication Channel Addresses in Dual-Port-Memory

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

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

The information is delivered as follows:

Size of Channel in Bytes

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

Table 12: 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.

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".

## 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 netX IO Synchronization manual). The supported modes are shown in the following table.

In/Out-Handshake Mode	In/Out-Source	Description
RCX_IO_MODE_BUFF_HST_CTRL	0	Buffered, unsynchronized mode (default)
RCX_IO_MODE_BUFF_HST_CTRL	1	Synchronized mode. Sync on send/receive process data. (Not supported for stack version V3.3.x and V3.4.x, reserved for future versions)

Table 13: Supported process data image synchronization modes

The behavior for the different modes is as follows:

- **Buffered, unsynchronized mode:** 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.
- **Synchronized mode (Sync on send/receive process data):** If the host passes access to the input block to the netX, the netX will first wait until new process data has been received. Then, the input block will be updated from the internal receive buffer and access to the input block is passed back to the host. When the host passes access to the output block to the netX, the netX will prepare an internal send buffer using the data from output block.



Afterwards, the netX waits until the process data has been transmitted before passing access to the output block back to the host. Therefore the process data exchange is synchronized to the process data update interval.

### 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 the *netX Dual-Port Memory Manual*).

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

Table 14: 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 the *netX DPM Manual*).

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

Table 15: Output Data Image

## 3.2 Acyclic Data (Mailboxes)

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

### Send Mailbox

Packet transfer from host system to firmware

### Receive Mailbox

Packet transfer from firmware to host system

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

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



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

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

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

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

Table 16: General Structure of Packets for non-cyclic Data Exchange.

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

#### Destination Queue 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 & Error Codes

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

### 3.2.3 Differences between System and Channel Mailboxes

The mailbox system on netX provides a non-cyclic data transfer channel for 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 <b>from</b> the network or <b>from</b> the protocol stack

Table 17: Channel Mailboxes.

#### Channel Mailboxes Structure

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

### 3.3 Status

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

#### 3.3.1 Common Status

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

##### 3.3.1.1 All Implementations

The structure outlined below is common to all protocol stacks.

##### Common Status Structure Definition

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



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

Table 18: Common Status Structure Definition

### Common Status Block Structure Reference

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

## Common Status Block Structure Reference

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

### Communication Change of State (All Implementations)

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

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

Table 19: Communication State of Change

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

Bit	Definition / Description
0	<p>Ready (RCX_COMM_COS_READY)</p> <p>0 - ...</p> <p>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.</p>
1	<p>Running (RCX_COMM_COS_RUN)</p> <p>0 - ...</p> <p>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.</p>
2	<p>Bus On (RCX_COMM_COS_BUS_ON)</p> <p>0 - ...</p> <p>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.</p>
3	<p>Configuration Locked (RCX_COMM_COS_CONFIG_LOCKED)</p> <p>0 - ...</p> <p>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 39).</p>
4	<p>Configuration New (RCX_COMM_COS_CONFIG_NEW)</p> <p>0 - ...</p> <p>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.</p>
5	<p>Restart Required (RCX_COMM_COS_RESTART_REQUIRED)</p> <p>0 - ...</p> <p>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.</p>
6	<p>Restart Required Enable (RCX_COMM_COS_RESTART_REQUIRED_ENABLE)</p> <p>0 - ...</p> <p>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)).</p>
7 ... 31	Reserved, set to 0

Table 20: Meaning of Communication Change of State Flags

**Communication State (All Implementations)**

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

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

**Communication Channel Error (All Implementations)**

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

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

**Runtime Failures**

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

**Initialization Failures**

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

**Configuration Failures**

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

**Network Failures**

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

**Version (All Implementations)**

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

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

**Watchdog Timeout (All Implementations)**

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

**Host Watchdog (All Implementations)**

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

**Error Count (All Implementations)**

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

**Error Log Indicator (All Implementations)**

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

**3.3.1.2 Master Implementation**

In addition to the common status block as outlined in the previous section, a master firmware maintains the additional structures for the administration of all slaves which are connected to the master. These are not discussed here as they are not relevant for the slave.

**3.3.1.3 Slave Implementation**

The slave firmware uses only the common structure as outlined in section 3.2.5.1. of the *netX DPM Interface Manual*. This is true for all protocol stacks.

### 3.3.2 Extended Status

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



**Note:** 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 the *netX Dual-Port-Memory Manual*).

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

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

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

Table 21: Communication Control Block

#### Communication Control Block Structure

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

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

### 3.5 Isochronous Synchronization Event

For isochronous applications the host is required to synchronize to the masters application cycle. In case of loadable firmwares the synchronization event “Start of Cycle” can be signaled from the netX to the host using different ways:

- **Dedicated Synchronization Pin:** The netX provides a dedicated synchronization pin. The Start of bus cycle is indicated with a falling edge. This method provides the lowest achievable synchronization jitter. This option is available for firmwares since V3.4.0.0
- **DPM Synchronization handshake event:** According to netX IO Synchronization manual the DPM provides a separate handshake cell to use for synchronization events. The Start of Cycle is indicated by toggling of corresponding handshake bit. This feature is available in a

limited implementation for firmwares since firmware version V3.4.9.0. The feature is always enabled and the synchronization event is signaled without checking the acknowledge by the host. Future Firmware version will provide an enhanced implementation with error checking and configuration via RCX\_SET\_HANDSHAKE\_CONFIGURATION\_REQ packet. The following table shows the possible configuration values to use with **future** versions of the firmware

Sync-Handshake Mode	Sync-Source	Description
RCX_SYNC_MODE_OFF	-	No synchronization event enabled
RCX_SYNC_MODE_DEV_CTRL	RCX_SYNC_SOURCE_1	Synchronization event on start of cycle. (Not supported for stack version V3.3.x and V3.4.x, reserved for future versions)

Table 22: Configuration Values for Synchronization



## 4 Getting Started

### 4.1 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
Get Device Handle	7.8.1	Get Device Handle Request
Acyclic data transfer (Read Indication)	9.1.1	Read Record Indication
Acyclic data transfer (Write Indication)	9.2.1	Write Record Indication
Register Application	7.9.1	Register Application Service
Save IP	9.6.1	Save IP Address Indication
Check Indication	8.2.1	Check Indication
AR InData Indication	8.6.1	AR InData Indication
Error Indication	9.14.1	Error Indication
Get Diagnosis	10.1	Get Diagnosis Service
Diagnostic Alarm	10.4.1	Diagnosis Alarm Request
Process Alarm	10.3.1	Process Alarm Request

Table 23: Overview about essential Functionality

For more information how to configure and setup the protocol stack, see chapter *Configuring the IO-Device Stack*, and especially section *Commissioning the PROFINET IO Device Stack V3* on page 62 might be very useful.

## 4.2 Description of behavior of stack under special situations

This section describes the behavior of the stack under some special situations.

### 4.2.1 Sequence of configuration evaluation

Four ways of configuration of the PROFINET IO-Device stack are possible:

- SYCON.net Database
- netX Configuration Tool (iniBatch Database)
- *Set Configuration Request* packet
- Warmstart 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)
- Warmstart Request packet
- 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 the stack is unconfigured until the receipt of the first configuration packet. In this case no packet is privileged. The first configuration packet will win. If a Set Configuration Request packet is received by the stack any *Warmstart Request* packet received later will be rejected. If the first configuration packet is a *Warmstart Request* packet a *Set Configuration Request* received later will be rejected.

### 4.2.2 Configuration Lock

If the configuration of the stack is locked as described in Dual Port Memory Interface Manual [4], the following behavior is implemented in the stack:

- new Set Configuration Requests or Warmstart Requests are not accepted

However the IP parameters and NameOfStation can be changed by means of DCP (see section “*Setting Parameters by means of DCP*”). This new values will be stored inside the stack. ConfigLock will be ignored for this service.

### 4.2.3 Setting Parameters by means of DCP

The PROFINET specification offers a service to change some IO-Device parameters over the bus. The Discovery and basic Configuration Protocol (DCP) is specified and supported by this stack.

An IO-Controller or IO-Supervisor can easily change the IP parameters or the NameOfStation of the running IO-Device stack. These new parameters can be marked to be used temporarily or remanent.

The receipt of such a request is indicated to the user with the Save Station Name Service and the Save IP Address Service. If the new values are marked as remanent the stack will automatically adapt its internal copy of the parameters.

If the new parameters are marked as temporary the local copy of the parameters will not be changed. After the next Channellnit the “old” values will be used again.

If the stack is configured with a database file which is stored in a flash volume the stack will automatically store the new values in the database file if the parameter is requested to be permanently stored.

## 5 Exchanging cyclic Data

This section describes how the user application can get access to the cyclic IO-data which is exchanged with the IO-Controller. The PROFINET IO-Device stack provides different ways to interchange this data. Depending on the used stack version and 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 its own host processor, 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 exist two possibilities to access the cyclic i/o-data. The first of them is the *Shared Memory Interface* which is basically an emulation of the *Dual Port Memory Interface* for applications running local on the netX chip. To avoid the overhead due to this emulation the user has also the opportunity to directly access the I/O data. For stack versions up to version 3.2.x the Triple Buffer Interface described in former versions of this document has to be used. This API has been superseded by the callback interface described in section 5.2 since stack version 3.3.x.

### 5.1 General Concepts

PROFINET uses frames with a special Ethertype which are sent periodically between device and controller to exchange the cyclic data. These frames contain the IO 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 IO-data and its IO-states. Therefore for every submodule used by the IO-Controller there is not only the IO-data exchanged but additionally also two IO-states. In terms of the Ethernet frame structure the IO-Provider state resides directly behind the IO-data itself and contains information if the IO-data is good and may be evaluated or not. The IO-Consumer state is sent in the opposite direction in a completely different frame and contains information if the IO-data could be handled by the consumer.

## 5.2 Exchanging cyclic Data using Callback Interface

---

**Note:**

The following section applies to stack version 3.3.x and above only. Furthermore this section only applies if the user is NOT using the Dualport Memory Interface or the Shared Memory Interface. (as described in section 2.3 of this document)

---

### 5.2.1 Overview of the Callback Interface

The PROFINET IO-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:

6. 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.
7. 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 104. In return this service will provide the user application with function pointers of the callback functions to call for data exchange.
8. 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 48 whenever the outgoing data has changed.
  - To get the newest incoming consumer data, call the *UpdateConsumerImage* described in section *UpdateConsumerImage* on page 47 either periodically or after the stack has signaled a new frame by calling the user application's event callback.

### 5.2.2 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 application to stack. This callback is used by the stack to inform the application about cyclic events.

**Note:**

The callback functions of the callback interface 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:**

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.

### 5.2.2.1 UpdateConsumerImage Callback


**Note:**

The `UpdateConsumerImage` 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 its 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 callback is running, the data within consumer image is updated and hence not consistent. Therefore the user's application must not access the consumer image until the function returns. The `UpdateConsumerImage` callback is defined by the following type declaration:

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

The four parameters of the callback are described as follows:

Parameter	Meaning
<code>hUserParam</code>	This parameter is a handle obtained from the stack by using the <code>PNS_IF_SET_IOIMAGE_REQ</code> .
<code>uiTimeout</code>	This parameter is currently unused. Set to zero for compatibility with future versions of the stack.
<code>uiReserved1</code>	Reserved for future usage. Set to zero.
<code>uiReserved2</code>	Reserved for future usage. Set to zero.

Table 24: Parameters of `UpdateConsumerImage` Callback

The callback uses the following return codes:

Return code	Meaning
<code>0x00000000</code>	<code>TLR_S_OK</code> : No Error.
<code>0xC0000009</code>	<code>TLR_E_INVALID_PARAMETER</code> : <code>hUserParam</code> is invalid; The input data image is invalid (usually the <code>PNS_IF_SET_IOIMAGE_REQ</code> request was not used);
<code>0xC0000002</code>	<code>TLR_E_UNEXPECTED</code> : Locking of semaphore failed. This is a fatal error.

Table 25: Return Codes of `UpdateConsumerImage` Callback

### 5.2.2.2 UpdateProviderImage Callback

**Note:**

The `UpdateProviderImage` 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 callback is running, the data within the provider image is accessed by the stack. Therefore the user's application must not change the data within the provider image until the function returns. The `UpdateProviderImage` callback is defined by the following type declaration:

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

The four parameters of the callback are described as follows:

Parameter	Meaning
<code>hUserParam</code>	This parameter is a handle obtained from the stack by using the <code>PNS_IF_SET_IOIMAGE_REQ</code> .
<code>uiTimeout</code>	This parameter is currently unused. Set to zero for compatibility with future versions of the stack.
<code>uiReserved1</code>	Reserved for future usage. Set to zero.
<code>uiReserved2</code>	Reserved for future usage. Set to zero.

Table 26: Parameters of `UpdateProviderImage` Callback

The callback uses the following return codes:

Return code	Meaning
<code>0x00000000</code>	<code>TLR_S_OK</code> : No Error.
<code>0xC0000009</code>	<code>TLR_E_INVALID_PARAMETER</code> : <code>hUserParam</code> is invalid; The output data image is invalid (usually the <code>PNS_IF_SET_IOIMAGE_REQ</code> request was not used);
<code>0xC0000002</code>	<code>TLR_E_UNEXPECTED</code> : Locking of semaphore failed. This is a fatal error.

Table 27: Return Codes of `UpdateProviderImage` Callback



### 5.2.2.3 UpdateExtStaBlock Callback


**Note:**

The `UpdateExtStaBlock` 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 can be used by the user's application to fill an *Extended status block* structure as defined in *Dual Port Memory Interface manual*. For a description of the structure of this block please refer to this manual. Basically the extended status block of the PROFINET IO-Device stack contains information about the data state blocks in consumer and provider data image if these blocks have been enabled by user's application request. The `UpdateExtStaBlock` callback is defined by the following type declaration:

```
typedef TLR_RESULT (*PNS_IF_UPDATE_EXTSTA_BLOCK_CLB_T)(
    TLR_HANDLE hUserParam,
    PNS_IF_EXTENDED_STATUS_BLOCK_T* ptExtSta,
    TLR_UINT uiOffsetInput,
    TLR_UINT uiOffsetOutput
);
```

The four parameters of the callback are described as follows:

Parameter	Meaning
<code>hUserParam</code>	This parameter is a handle obtained from the stack by using the <code>PNS_IF_SET_IOIMAGE_REQ</code> .
<code>ptExtSta</code>	Points to an extended status block structure.
<code>uiOffsetCons</code>	This value is an offset which will be added to all offsets given in the extended status block referring to the consumer block. This parameter is used by the stack's application interface task when updating the DPM extended status block. Set to zero.
<code>uiOffsetProv</code>	This value is an offset which will be added to all offsets given in the extended status block referring to the provider block. This parameter is used by the stack's application interface task when updating the DPM extended status block. Set to zero.

Table 28: Parameters of the Callback

The callback uses the following return codes:

Return code	Meaning
<code>0x00000000</code>	<code>TLR_S_OK</code> : No Error.

Table 29: Return Codes of the Callback

#### 5.2.2.4 EventHandler Callback

This callback has to be provided by the user's application to the stack. It will be called by the stack upon certain events. As the callback will be called from stacks task context, the user must consider locking of shared resources. Currently the following events are indicated by the stack:

Eventnumber	Meaning
0x00000000	PNS_IF_IO_EVENT_RESERVED: Reserved
0x00000001	PNS_IF_IO_EVENT_NEW_FRAME: A new frame has arrived. The user's application should call the UpdateConsumerImage callback to obtain the new data.
0x00000002	PNS_IF_IO_EVENT_CONSUMER_UPDATE_REQUIRED: The Data within the consumer 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: (Not available for stack versions V3.3.x and V3.4.x) The stack has send the data from last call to UpdateProviderImage callback function.

Table 30: Events indicated by the Callback

The EventHandler 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 bit field indicates the type of pending events. See table above.

Table 31: Parameters of the Callback

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 Isochronous User Applications

An isochronous user application describes an operation mode where the application on device side operates synchronized with the application at controller side. This may be required for instance for drives and positional encoders. A precondition for this scenario is the usage of PROFINET IRT High Performance at the bus side.

### 6.1 Parameter and Values

For isochronous applications many timing parameters are defined. The following listing shall give a short description of each defined value. See the IEC61158-5-10 standard specification for more information.

- **Time Data Cycle(T\_DC):** The time factor for the application data cycle. This is the application cycle the isochronous application shall use. The Time Data Cycle is specified in multiples of T\_DC\_BASE (Time Data Cycle = T\_DC \* T\_DC\_BASE \* 31.25 microseconds). The value is calculated by the engineering system and provided to the device using SynchronousModeData record write (See below).
- **Time IO Input (T\_IO\_Input):** The time factor to fetch the inputs for the devices. This value indicates the time before application cycle start the inputs shall be fetched. The Time IO Input is specified in multiples of T\_IO\_BASE (Time IO Input = T\_IO\_Input \* T\_IO\_BASE \* 1 nanosecond). The value is calculated by the engineering system and provided to the device using SynchronousModeData record write (See below).
- **Time IO Output (T\_IO\_Output):** The time factor to set the outputs for the device. This value indicates the time after application cycle start the outputs shall be set to the last transferred values. The Time IO Output is specified in multiples of T\_IO\_BASE (Time IO Output = T\_IO\_Output \* T\_IO\_Base \* 1 nanoseconds). The value is calculated by the engineering system and provided to the device using SynchronousModeData record write (See below).
- **Time IO Input Valid (T\_IO\_InputValid):** The time factor the new input data shall be available to convey. This value indicates the time before cycle start the input data shall be available for sending on the bus. The Time IO Input Valid is specified in multiples of T\_IO\_BASE (Time IO Input Valid = T\_IO\_InputValid \* T\_IO\_BASE \* 1 nanosecond). The value is calculated by the engineering system and provided to the device using SynchronousModeData record write (See below).
- **Time IO Output Valid (T\_IO\_OutputValid):** The time factor the new output data is available for processing. This value indicates the time after cycle start the output data has been transferred to the device. The Time IO Output Valid is specified in multiples of T\_IO\_BASE (Time IO Output Valid = T\_IO\_OutputValid \* T\_IO\_BASE \* 1 nanosecond). The value is calculated by the engineering system and provided to the device using SynchronousModeData record write (See below).
- **Controller Application Cycle Factor (CACF):** The time factor of the controller applications cycle. The value indicates the controllers application cycle time. The Controller Application Cycle Factor is specified in multiples of T\_DC (Controller Application Cycle time = CACF \* T\_DC). The value is calculated by the engineering system and provided to the device using SynchronousModeData record write (See below).
- **Time Data Cycle Base (T\_DC\_Base):** The time factor for the devices application cycle granularity. The value indicates the granularity of the devices application cycle time. The Time Data Cycle Base is specified in multiples of 31.25 microseconds (Time Data Cycle

Base =  $T\_DC\_Base * 31.25$  microseconds). The value is defined by the manufacturer of the device and has to be specified in the GSDML Device Description.

- **Time IO Base (T\_IO\_Base):** The time factor for the granularity for the devices input / output delay time. The value indicates the granularity the device input or output delay can be adjusted. The Time IO Base is specified in multiples of 1 nanosecond (Time IO Base =  $T\_IO\_Base * 1$  nanosecond). The value is defined by the manufacturer of the device and has to be specified in the GSDML Device Description.
- **Time Data Cycle Minimum (T\_DC\_Min):** The time factor for the smallest possible data cycle. The value indicates the minimum device data cycle (application cycle). The Time Data Cycle Minimum is specified in multiples of  $T\_DC\_Base$  (Time Data Cycle Minimum =  $T\_DC\_Min * T\_DC\_Base * 31.25$  microseconds). The value is defined by the manufacturer of the device and has to be specified in the GSDML Device Description.
- **Time Data Cycle Maximum (T\_DC\_Max):** The time factor for the largest possible data cycle. The value indicates the maximum device data cycle (application cycle). The Time Data Cycle Maximum is specified in multiples of  $T\_DC\_Base$  (Time Data Cycle Maximum =  $T\_DC\_Max * T\_DC\_Base * 31.25$  microseconds). The value is defined by the manufacturer of the device and has to be specified in the GSDML Device Description.
- **Time IO Input Minimum (T\_IO\_InputMin):** The time factor for the smallest possible input refresh delay. The value indicates the minimum time devices needs to copy or preprocess the input data for sending. The Time IO Input Minimum is specified in multiples of  $T\_IO\_Base$  (Time IO Input Minimum =  $T\_IO\_InputMin * T\_IO\_Base * 31.25$  microseconds). The value is defined by the manufacturer of the device and has to be specified in the GSDML Device Description.
- **Time IO Output Maximum (T\_IO\_OutputMin):** The time factor for the smallest possible output refresh delay. The value indicates the minimum time the device requires to copy or preprocess the received output data. The Time IO Output Minimum is specified in multiples of  $T\_IO\_Base$  (Time IO Output Minimum =  $T\_IO\_OutputMin * T\_IO\_Base * 31.25$  microseconds). The value is defined by the manufacturer of the device and has to be specified in the GSDML Device Description.

## 6.2 Clock Synchronization

When using PROFINET IRT a distributed clock mechanism ensures that all devices from the same synchronization domain can synchronize with a jitter of less than one microsecond to the same synchronization event (start of bus cycle). The netX provides different ways to pass this event to the user application:

- **Dedicated Synchronization Pin:** A dedicated pin of the netX controller (XC\_TRIGGER0) provides a signal with falling edge on each bust cycle start. This signal may be used by a host to either synchronize its clock with an own PLL or to use it as event to update the process data. This method provides the lowest achievable jitter for loadable firmwares.
- **Synchronization Handshake Event:** The synchronization event may also reported using a special synchronization handshake event (see section 3.5). Due to internal handling, this method increases the jitter several microseconds but can be used for existing hardware.
- **Synchronization Callback:** If the application runs on the netX itself it is possible to register a callback function to be called on the event. The callback has to be registered using special function call (not via Packet API).

Depending on the controllers configuration the synchronization event may appear in intervals of 0.25, 0.5, 1.0, 2.0 and 4.0 milliseconds.

## 6.3 Application Cycle and Timings

Due to the architecture of the PROFINET system, the controller's isochronous application may execute with an application cycle interval different to the bus cycle. The applications interval is in general an integral multiple of the synchronization cycle interval. Furthermore it is even possible to have multiple isochronous applications each running with its on cycle time and each driving one submodule of the same device. Therefore it is necessary to inform the device's application about the application cycles for each submodule. For this the controller will write a special record (IsochronousModeData Index 0x8030) for each submodule running in isochronous mode. The data of this record will be passed to the user application via standard PNS\_IF\_WRITE\_RECORD\_REQ packet to the application. The application has to evaluate the record data by itself. The structure is as follows (all fields are encoded using **Big-Endian** byte order):

```
typedef struct
{
    /* block header */
    TLR_UINT16    usType;
    TLR_UINT16    usLen;
    TLR_UINT8     bMajor;
    TLR_UINT8     bMinor;
    /* padding */
    TLR_UINT16    usPadding;
    /* isochronous mode data */
    TLR_UINT16    usSlotNumber;
    TLR_UINT16    usSubslotNumber;

    TLR_UINT16    usControllerAppCycle;
    TLR_UINT16    usTimeDataCycle;
    TLR_UINT32    ulTimeIOInput;
    TLR_UINT32    ulTimeIOOutput;
    TLR_UINT32    ulTimeIOInputValid;
    TLR_UINT32    ulTimeIOOutputValid;
} PNIO_PDU_ISOCHRONOUSMODEDATA_T;
```

Field	Type	Value/ Range	Description
usType	UINT16	0x0204	The type of the PDU structure. The actual value shall be checked against the expected value
usLen	UINT16	28	The length of the PDU structure (remaining bytes). The actual value shall be checked against the expected value
bMajor	UINT8	1	Major version of the PDU structure. The actual value shall be checked against the expected value
bMinor	UINT8	0	Minor version of the PDU structure. The actual value shall be checked against the expected value
usPadding	UINT16		Padding bytes to ensure DWord alignment
usSlotNumber	UINT16		Slot number. Should be equal to slot number from PNS_IF_WRITE_RECORD_DATA_T structure.
usSubslotNumber	UINT16		Slot number. Should be equal to slot number from PNS_IF_WRITE_RECORD_DATA_T structure.
usControllerAppCycle	UINT16	0x1 – 0x0400	Controllers application cycle interval in multiples of T_DC.
usTimeDataCycle	UINT16	0x1 – 0x0400	Isochronous data cycle (TimeDC, T_DC) in units of 31.25 microseconds.
ulTimeIOInput	UINT32	0 – 3200000 0	The amount of time before the start of data cycle the inputs shall be read (T_IO_Input) in units of T_IO_BASE (GSDML).

Field	Type	Value/ Range	Description
ulTimeIOOutput	UINT32	0 – 3200000 0	The amount of time after the start of data cycle the outputs shall be written (T_IO_Output) in units of T_IO_BASE (GSDML).
ulTimeIOInputValid	UINT32	0 – 3200000 0	The time before the start of cycle the input data shall available for sending (T_IO_InputValid) in units of T_IO_BASE (GSDML).
ulTimeIOOutputValid	UINT32	0 – 3200000 0	The time after the start of cycle the output data is available for processing (T_IO_OutputValid) in units of T_IO_BASE (GSDML).

Table 32: Structure *PNIO\_PDU\_ISOCHRONOUSMODEDATA\_T*

## 6.4 IRT Cycle Counter

The isochronous application on host (i.e. device) may execute with a cycle interval different to the bus cycle. The application can determine the actual bus cycle reading the IRT cycle counter value from input DPM area with offset *usIRTCycleCounterOffset*. The application must define this offset (before IRT mode) with PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_6 of the Set OEM Parameters Service (as described in 7.14.1.3). IRT cycle counter value is specified in multiples of T\_IO\_Base.

## 7 Configuring the IO-Device Stack

This chapter explains how the PROFINET IO Device firmware stack is structured internally and how to configure it on start-up. Configuration is done by sending one single packet from the application to the stack. The `Set_Configuration` Packet (see section *Set Configuration Service* on page 64) has to be sent for this purpose. Finally, the usage of linkable object modules will be discussed.

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

Overview over the Configuration Packets of the PROFINET IO Device Stack			
Section	Packet	Command code	Page
7.6	Set Configuration Request	0x1FE2	65
	Set Configuration Confirmation	0x1FE3	75
7.7	Warmstart Request	0x1F6A	78
	Warmstart Confirmation	0x1F6B	81
7.13	Set Port MAC Address Request	0x1FE0	91
	Set Port MAC Address Confirmation	0x1FE1	93
7.14	Set OEM Parameters Request	0x1FE8	94
	Set OEM Parameters Confirmation	0x1FE9	97
7.15	Load Remanent Data Request	0x1FEC	100
	Load Remanent Data Confirmation	0x1FED	102
7.16	Configuration Delete Request	0x2F14	103
	Configuration Delete Confirmation	0x2F15	103
7.17	Set IO-Image Request	0x1FF0	104
	Set IO-Image Confirmation	0x1FF1	106
7.18	Set IOXS Config Request	0x1FF2	108
	Set IOXS Config Confirmation	0x1FF3	110

Table 33: Overview over the Configuration Packets of the PROFINET IO Device Stack



## 7.1 Task Structure of the PROFINET IO Device Stack

The figure below displays the internal structure of the tasks which together represent the PROFINET IO Device Stack V3.3:

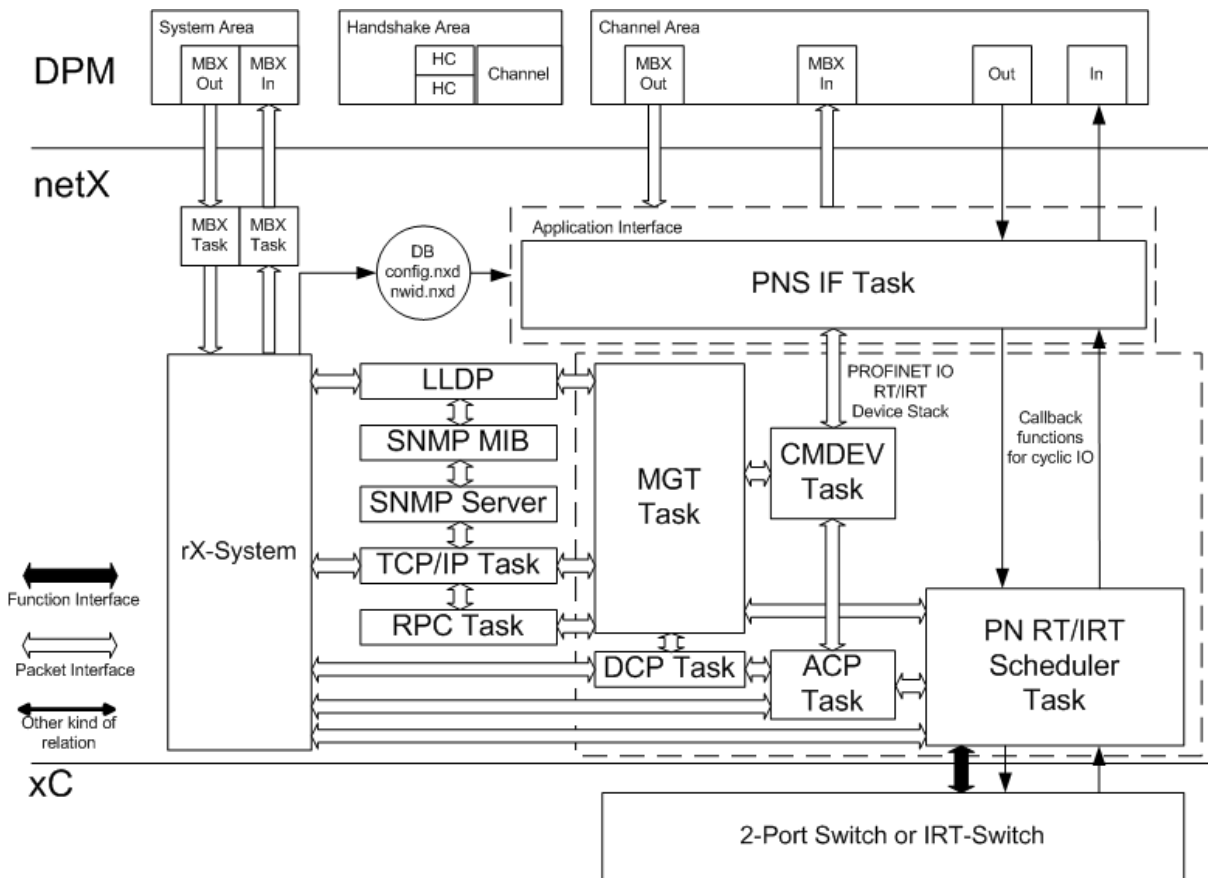


Figure 4: Task Structure of the PROFINET IO Device Stack V3.3

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

The user application only accesses the tasks located in the highest layer namely the PNS\_IF task which represent the application interface of the PROFINET IO Device stack.

- The ACP task, CMDEV task, MGT task, DCP task and RT/IRT Scheduler task represent the core of the PROFINET IO Device Stack.
- The RPC task and the TCP/IP task are required to perform connection-less RPC using the UDP protocol.
- The SNMP task and the TCP/IP task are required to handle the SNMP protocol.
- The LLDP task is required to handle the LLDP protocol.

Handling of Ethernet frames is required for the ACP, DCP, RT/IRT Scheduler, LLDP and TCP/IP tasks and performed using appropriate rcX functionality. In detail, the various tasks have the following functionality and responsibilities:

**ACP task**

The ACP task provides the following functionality:

- It processes all acyclic real-time telegrams (Incoming alarms from connected PROFINET IO Devices and diagnosis).
- All cyclic telegrams are received and handled by this task.
- All DCP telegrams are received by this task. They are directed to the DCP task without any change and processed there, however.
- Additionally, the current link status (link working or broken) is known by this task. The ACP task informs the PNS\_IF task (and therefore the user) about the current link status.
- It implements the PN IO state machines APMS, APMR, ALPMR, DelayRequestor, DelayResponder, PPM and CPM.

**CMDEV task**

The CMDEV task implements the CMDEV state machine defined in the PROFINET IO specification. It coordinates and controls all lower level tasks. It is responsible for the connection establishment and for the start-up of the consumer and provider protocol state machines.

**DCP task**

The DCP task implements all DCP (Discovery and Basic Configuration Protocol) related state machines setting up the core of the PROFINET IO Device stack.

**RT/IRT Scheduler task**

The RT/IRT Scheduler task is responsible for the following activities:

- Handling of all incoming cyclic telegrams and producing all outgoing cyclic telegrams.
- Control of sending lists.
- Providing the LMPM according to the PROFINET IO specification.

**LLDP task**

The LLDP task implements the required functionality to handle LLDP according to the PROFINET IO and LLDP specification.

**MGT task**

The MGT task is responsible for resource administration, configuration and providing all DCP services.

It implements the RMPM state machine according to PROFINET IO specification.

**PNS\_IF task**

The PNS\_IF task provides the interface to the user application and the control of the stack. In detail, it is responsible for the following:

- Process data exchange (to the user)
- handling everything related to PROFINET diagnosis
- Configuration and control of the underlying tasks
- Handling of the Dual-Port-Memory

**RPC task**

The RPC task is required for connection-less RPC

It provides:

- RPC client, RPC server and RPC Endpoint Mapper functionality
- The CLRPC protocol machine for connection-less RPC.

Connection-less RPC is required for

- Connection establishment
- UDP-based acyclic services such as Read Record/Write Record requests

As connection-less RPC uses the UDP protocol the RPC task also requires access to the TCP/IP task.

**SNMP task**

The SNMP task implements functions according to handle SNMP requests. The MIB (Management Information Base) is located here.

**TCP/IP task**

The TCP/IP task coordinates the PROFINET IO Device stack with the underlying TCP/IP stack. It provides services required by the RPC task.

## 7.2 Configuration of the Memory Area without DPM

If no DPM is used, the memory area has to be configured using the Set IO-Image Service. When a DPM is used, the area is still configured and the configuration can be found at the *netX Dual-Port Memory Manual*.

At this example (see *Figure 6: Memory area configuration*) the consumer data are placed in the memory from address 0x80070000 (with a width of 0x500 bytes) on until address 0x800704FF. The provider data will start at address 0x80070550 and end (after a width of 0x500 bytes) at address 0x800705FF.

### 7.2.1 Configuration of the Consumer/Provider Memory Area for non DPM /SHM access

If neither the Dual-Port-Memory nor the Shared Memory API is used to access the PROFINET Device stack, the application has to provide the stack with Consumer- and Provider- Memory areas. For this the application has to allocate the required amount of memory and pass pointers to these memories to the stack by means of the *Set IO-Image Service*, see page 104.

In the example shown in *Figure 5: Memory Area Configuration* the consumer data are placed in the memory from address 0x80070000 (with a width of 0x500 bytes) until address 0x800704FF. The provider data will start at address 0x80070550 and end (after a width of 0x500 bytes) at address 0x800705FF.

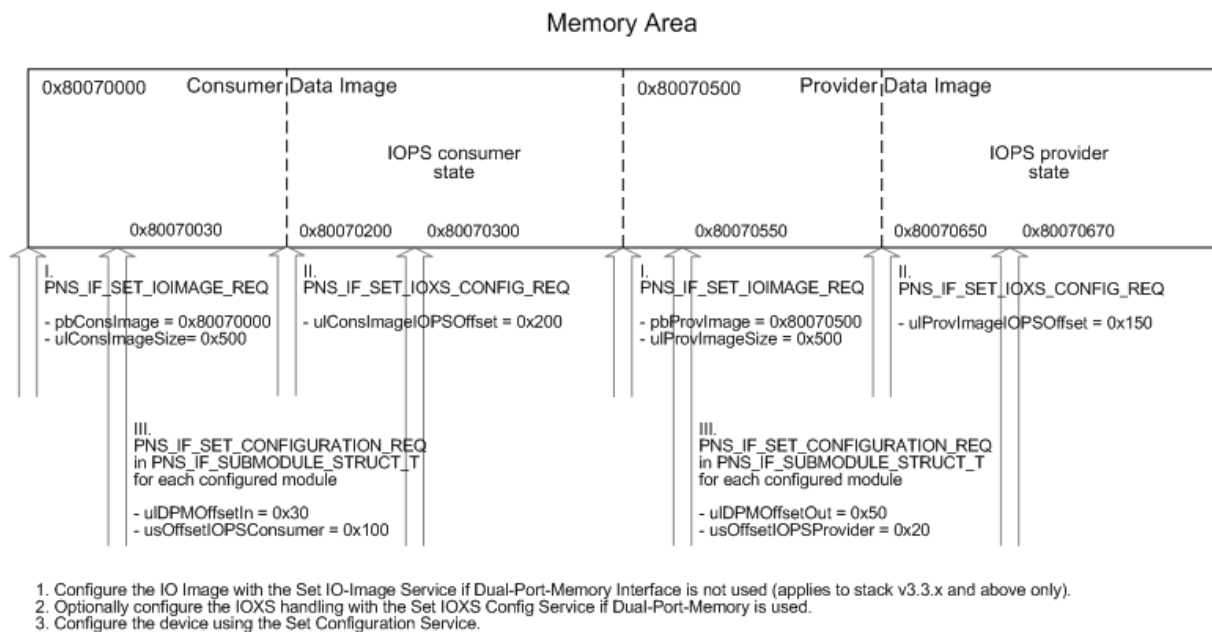


Figure 5: Memory Area Configuration

## 7.3 Configuration of IOXS States

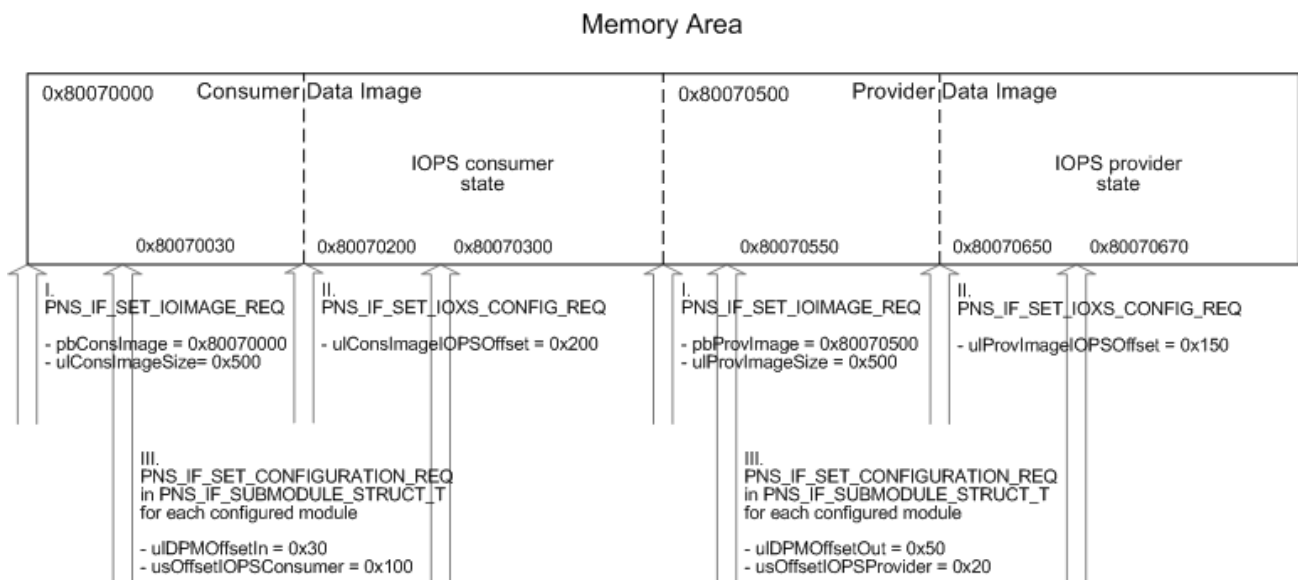
The configuration of the IOXS states is optional. When enabling this function, the application is responsible to serve the appropriate provider/consumer states. This service enables reading/writing data states from/to the consumer/provider data image (see subchapter Set IOXS Config Service) and can be used with and without DPM. The stack will copy the provider/consumer data states to the area configured relative to the previously configured memory area or the memory area given by DPM (see *netX Dual-Port Memory Manual*). In the example (see Figure 6: Memory area configuration) the consumer data state block will start at 0x80070200 (with an offset of 0x200) and the provider data state block starts at 0x80070650 (with an offset of 0x150).

## 7.4 Configuration of the Submodules

This option will just be evaluated if the IOXS Configuration Service has been done 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 (see subchapter Set Configuration Request) for each. In the example (see Figure 6: Memory area configuration) a consumer submodule has been configured with an offset of 0x30. The i/o-data will start at 0x80070030. The related consumer state was configured with an offset of 0x100 and will start, for this submodule, at 0x80070300.

Also the provider state is configured generic at the example figure. The i/o-data of the state block start with an offset of 0x50 at 0x80070550 and the provider state is placed at 0x80070670 (with an offset of 0x20).



1. Configure the IO Image with the Set IO-Image Service if Dual-Port-Memory Interface is not used (applies to stack v3.3.x and above only).
2. Optionally configure the IOXS handling with the Set IOXS Config Service if Dual-Port-Memory is used.
3. Configure the device using the Set Configuration Service.

Figure 6: Memory area configuration

## 7.5 Commissioning the PROFINET IO Device Stack V3

In order to configure a PROFINET IO Device using the PROFINET IO Device Stack V3 correctly, proceed as follows:

- If necessary, assign a MAC Address to the device. This is described in the netX Dual-Port Memory Manual. It should be necessary to assign a MAC address, if no local MAC address exists for the PROFINET IO Device to be commissioned.
- Optionally assign a Port MAC Address using the *Set Port MAC Address Service*, if the LLDP protocol should explicitly use other MAC addresses than the default ones.
- Optionally restore the remanent data using the *Load Remanent Data Service* if remanent data is not handled by the stack but by the application.
- Optionally set the OEM parameters using the *Set OEM Parameters Service* if the Hilscher default values are insufficient.
- Configure the IO Image with the *Set IO-Image Service* if Dual-Port-Memory Interface is not used (applies to stack v3.3.x and above only).
- Optionally configure the IOXS handling with the *Set IOXS Config Service* if Dual-Port-Memory Interface is not used (applies to stack v3.3.x and above only).
- 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, it is configured completely. It will indicate the configuration to the user by setting the bits "Configuration new" and "Restart required" in the communication COS register in DPM.
- Register the application using the *Register Application Service* in order to receive indications from the PROFINET IO Device stack.
- Perform the Channel Initialization (see DPM manual) 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.
- Get a device handle using the *Get Device Handle Service*. You need a device handle afterwards to be able to access various services

For a graphical representation of this sequence see below:



Figure 7: Commissioning the PROFINET IO Device Stack V3

### 7.5.1 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 Initialization Request (Channellnit)* (see DPM manual). Sending the new configuration without the *Channel Initialization Request* will not have an effect on any running communication. The new parameters will simply be stored. Sending the *Configuration Reload Request* will stop any communication and the new parameters will be taken over by the stack.

## 7.6 Set Configuration Service

This packet has to be sent by the application. The `Set_Configuration Req` packet shall be used by application to set basic configuration information to the stack.

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 10.9 and 10.10) and Plug services (see sections 10.7 and 10.8).



---

**Note:**

It is required that the application also registers itself at the stack in the way that the stack knows where to send indications to. This has to be done using the service described in section "*Register Application Service*".

---



---

**Note:**

It is required that the application also registers itself at the stack in the way that the stack knows where to send indications to. This has to be done using the service described in section "*Register Application Service*".

---



---

**Note:**

4 KB of free memory space need to be available for processing the set configuration packet.

---



## 7.6.1 Set Configuration Request

This request has to be sent by the application to the protocol stack. As the configuration of modules and submodules can (almost) be freely 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 it may be required for the application the fragment the request packet. This is due to the fact that the mailbox size is limited. How the fragmented service shall be handled by the application and the stack is described in [4].

structure PNS_IF_SET_CONFIGURATION_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		Packet data length in bytes. This is a value depending on the amount of submodules.
	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	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.

structure PNS_IF_SET_CONFIGURATION_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
	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 (Applicable for Stack versions 3.4. and above only)

Table 34: 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.. 1024 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..1024 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.
	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
	ulNetMask	UINT32	Valid network mask, default: 0.0.0.0	Network mask
	ulGateway	UINT32	Valid gateway address, default: 0.0.0.0	Gateway address
	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 number of user diagnosis records the stack shall be able to handle in parallel.  : 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 35: Structure *tDeviceParameters*

Coding of the field `ulSystemFlags`

```
#define      PNS_IF_SYSTEM_START_AUTO_START
              0x0000
```

Use this value to allow direct access to the network for the stack.

```
#define      PNS_IF_SYSTEM_START_APPL_CONTROLLED
              0x0001
```

Use this value to disallow direct access to the network for the stack. The application explicitly has to allow access to the network later.

For more information concerning this topic see section 4.4.1 “*Controlled or Automatic Start*” of the *netX DPM Interface Manual*.

```
#define      PNS_IF_SYSTEM_DATA_STA_DISABLED
              0x0000
```

This option is currently not supported.

```
#define      PNS_IF_SYSTEM_DATA_STA_ENABLED
              0x0002
```

This option is currently not supported. For each input and output date the following status information (in Byte) is memorized in the dual-port memory..

```
#define      PNS_IF_SYSTEM_LONG_DATA_STA_DISABLED
              0x0000
```

Use this value to enable 1 Byte mode. This option is currently not supported.

```
#define      PNS_IF_SYSTEM_LONG_DATA_STA_ENABLED
              0x0004
```

Use this value to enable 4 Byte mode. This option is currently not supported.

```
#define      PNS_IF_SYSTEM_BYTEORDER_BIG_ENDIAN
              0x0000
```

Use this value to use standard big endian byte order for cyclic process data image.

```
#define      PNS_IF_SYSTEM_BYTEORDER_LITTLE_ENDIAN
              0x0008
```

Use this value to enable conversion to little endian byte order for cyclic process data image



**Note:** The `PNS_IF_SYSTEM_BYTEORDER_LITTLE_ENDIAN` flag is available for stack version V3.4.0 and above. Using this flag will increase the CPU usage on netX chip and may have bad impact on minimum reachable cycle time.

```
#define      PNS_IF_SYSTEM_STACK_HANDLE_I_M_DISABLED
              0x0000
```

Handling of I&M Requests by the stack is disabled. Requests are transferred to the user.

```
#define      PNS_IF_SYSTEM_STACK_HANDLE_I_M_ENABLED
            0x0100
```

Handling of I&M Requests by the stack is enabled.



**Note:**

For stack versions previous than V3.4.12.0 the stack only handles the minimum required I&M functionality. Therefore only I&M 0 is handled. If this is not sufficient, the user has to implement the complete handling of I&M on its own. From stack version V3.4.12.0 and greater the stack supports reading of I&M0-4 and writing of I&M1-4 or, in case the user wants to handle I&M, the stacks performs decoding of I&M requests and forwards them to users application using of PNS\_IF\_READ\_IM\_IND and PNS\_IF\_WRITE\_IM\_IND packets.

```
#define PNS_IF_SYSTEM_ARDY_WOUT_APPL_REG_DISABLED
            0x0000
```

The stack will not send ApplicationReady on its own.

```
#define PNS_IF_SYSTEM_ARDY_WOUT_APPL_REG_ENABLED
            0x0200
```

The stack is automatically sending ApplicationReady (after receipt of ParameterEnd) if no application is registered. If an application is registered this flag is ignored and the standard handling will be applied.

---

**ATTENTION:** Use this flag carefully. Sending ApplicationReady to the IO-Controller when no application is available can be dangerous.

---

```
#define PNS_IF_SYSTEM_USE_OEM_PARAMETERS_DISABLE
            0x0000
```

The stack will use well-defined Hilscher default values for some I&M elements (e.g. serial number).

```
#define PNS_IF_SYSTEM_USE_OEM_PARAMETERS_ENABLE
            0x0400
```

The stack will not use the well defined Hilscher default values for I&M. The user has to transfer the values to the stack with the SET\_OEM\_PARAMETERS service.



**Note:** Setting the OEM parameters has to be done before sending this Set Configuration Request.

```
#define PNS_IF_SYSTEM_CHECK_IND_ALL_MODULES_DISABLED
            0x0000
```

The stack will not send a Check Indication to the user if the submodule plugged into a specific slot/subslot matches the one expected by the IO-Controller.

```
#define PNS_IF_SYSTEM_CHECK_IND_ALL_MODULES_ENABLED
            0x0800
```

The stack will send a Check Indication to the user for every submodule expected by the IO-Controller – even if the submodule plugged into a specific slot/subslot matches the one expected by the IO-Controller. This may be sensible if the application needs to know which submodules / modules are used for data exchange.

```
#define PNS_IF_SYSTEM_HANDLE_LINK_DOWN_AS_FATAL_DISABLED
0x0000
```

The stack will not call fatal error callback or send Error Indication if link down is detected.

```
#define PNS_IF_SYSTEM_HANDLE_LINK_DOWN_AS_FATAL_ENABLED
0x1000
```

The stack will call the fatal error callback or send an Error Indication if link down is detected.

```
#define PNS_IF_SYSTEM_CHECK_IND_UNUSED_MODULES_DISABLED
0x0000
```

The stack will not send a Check Indication to the user if the submodule plugged into a specific slot/subslot is not expected (unused) by the IO-Controller.

```
#define PNS_IF_SYSTEM_CHECK_IND_UNUSED_MODULES_ENABLED
0x2000
```

The stack will send a Check Indication to the user for every submodule plugged into a specific slot/subslot which is not expected by the IO-Controller. This may be sensible if the application needs to know which submodules / modules are not used for data exchange.



---

**Important:** If the stack has no flash available this flag will be ignored and the storage of the remanent data has to be handled by the application.

---

```
#define PNS_IF_SYSTEM_DISABLE_STORE_REMANENT_DISABLE
0x0000
```

The stack will not store the remanent data itself. If new Remanent data are available the stack will send an indication to the application.

```
#define PNS_IF_SYSTEM_DISABLE_STORE_REMANENT_ENABLE 0x4000
```

The stack will store and handle the remanent data. If the stack has no flash available this flag will be ignored and the application is responsible for the handling of remanent data. In this case the stack will send a Indication for each change of the remanent data.

```
#define PNS_IF_SYSTEM_ENABLE_LINK_STATE_INDICATION 0x0000
```

The stack will send a Link state change Indication to the application for each detected link change.

```
#define PNS_IF_SYSTEM_DISABLE_LINK_STATE_INDICATION 0x0800
```

The stack will not send any Link state change Indication to the application.

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 36: 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..1024	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..1024	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 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 or if DPM is not used this value shall be set to 0xFFFFFFFF.  See subchapter 7.4 for further information
	ulDPMOffsetOut	UINT32		Offset in DPM OutputArea where provided data of the submodule shall be taken from. This data is sent by IO-Device and received by IO-Controller.  If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.  See subchapter 7.4 for further information
	usOffsetIOPSP rovider	UINT16		Offset where to put IOPS provider state for this submodule relative to beginning of IOPS block in DPM output area too.  See subchapter 7.4 for further information
	usOffsetIOPSC onsumer	UINT16		Offset where to take IOPS provider state of this submodule relative to beginning of IOPS block in DPM input area from.  See subchapter 7.4 for further information
	ulReserved	UINT32[ 2]	0	Reserved for future use. Set to zero.

Table 37: Structure `PNS_IF_SUBMODULE_STRUCT_T` for stack version v3.3

- 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 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 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 follow. The structure of IO\_SIGNALS\_CONFIGURE\_REQ\_DATA\_T is the same as the data part of the Configure Signal Request described in section 7.19.1

## 7.6.2 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 38: PNS\_IF\_SET\_CONFIGURATION\_CNF\_T - Set Configuration Confirmation

### 7.6.3 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.

## 7.7 Warmstart Service

This warmstart service is implemented for compatibility reasons. It was introduced with the PROFINET IO-Device Stack v2.

Using this service the stack will automatically adjust its module configuration to the one requested by the IO-Controller. The user application will not be able to get information about the submodules used inside the stack.

However the Device Access Point (DAP) which resides in slot 0 is always fix and will not be adapted. This includes the PDEV-submodules. It is required that the GSDML-file contains the same ModuleIdNumber and SubmoduleIdNumbers for these modules and submodules as the example GSDML-file delivered together with the protocol stack.

The stack will automatically generate the DPM-offsets where the IO-data are copied to and taken from. The following rules apply:

- The DPM-offsets start at offset 0 and are used in ascending order without any break
- The submodules with the lowest subslot of the module in the slot with the lowest number will be put at offset 0
- For all submodules of the same slot the DPM-offsets are used in ascending order
- When all submodules of the module with the lowest slot number are copied than the next module will be used



---

**Important:** It is highly recommended **not** to use this service to configure the PROFINET IO-Device Stack v3 this way as the restrictions described above will not make it possible for the application to really work with IO-data.

---

## 7.7.1 Warmstart Request

To cause the stack to automatically adjust its module configuration to the one requested by the PROFINET IO-Controller this warmstart request has to be sent to the PROFINET IO-Device stack.

### Packet structure Reference

```
typedef struct PNS_IF_SET_WARMSTART_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_SET_WARMSTART_REQ_DATA_T    tData;
} PNS_IF_SET_WARMSTART_REQ_T;

typedef struct PNS_IF_SET_WARMSTART_REQ_DATA_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 ulSubnetworkMask;
    TLR_UINT32 ulGateway;
} PNS_IF_SET_WARMSTART_REQ_DATA_T;
```

## Packet Description

structure PNS_IF_SET_WARMSTART_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	576	Packet data 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	0x1F6A	PNS_IF_SET_WARMSTART_PARAM_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_WARMSTART_REQ_DATA_T			
	ulSystemFlags	UINT32		Flags for system behavior. See section Set Configuration Request for details.
	ulWdgTime	UINT32	Default value: 1000  Allowed values: 0, 20..65535	Watchdog time (in milliseconds). 0 indicates the 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 ... $(2^{16} - 1)$ *, 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..1024 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.

structure PNS_IF_SET_WARMSTART_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
	ulCompleteOutputSize	UINT32	Default value:128 Allowed values: 0..1024 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.
	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.
	ulIpAddress	UINT32	Valid IP address, default: 0.0.0.0	IP address
	ulSubnetworkMask	UINT32	Valid network mask, default: 0.0.0.0	Network mask
	ulGateway	UINT32	Valid gateway address, default: 0.0.0.0	Gateway address

Table 39: PNS\_IF\_SET\_WARMSTART\_REQ\_T – Set Warmstart Request



## 7.7.2 Warmstart Confirmation

This confirmation will be returned to the application.

### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T      PNS_IF_SET_WARMSTART_CNF_T;
```

### Packet Description

structure PNS_IF_SET_WARMSTART_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	0x1F6B	PNS_IF_SET_WARMSTART_PARAM_CNF -Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 40: PNS\_IF\_SET\_WARMSTART\_CNF\_T – Set Warmstart Confirmation

## 7.8 Get Device Handle Service

Using this service the application can request a handle to the device. This handle is for example required in case the application wants to send a process alarm. Since Stack versions V3.4.12.0 using this request is not recommended anymore. The stack will provide the application with a device handle which is closely related to the connection. E.g. First connection gets device handle 1, second connection gets device handle 2.

As a special case device handle NULL may be used in requests to indicate to the stack to automatically select the correct connection.

### 7.8.1 Get Device Handle Request

The request packet the application has to send to the stack in order to get a device handle.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_GET_DEVICE_HANDLE_REQ_T;
```

#### Packet Description

structure PNS_IF_GET_DEVICE_HANDLE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		Source Queue-Handle
	ulDestId	UINT32		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		Status not in use for requests.
	ulCmd	UINT32	0x1FB0	PNS_IF_GET_DEVICE_HANDLE_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 41: PNS\_IF\_GET\_DEVICE\_HANDLE\_REQ\_T - Get Device Handle Request

## 7.8.2 Get Device Handle Confirmation

The confirmation packet containing the requested device handle looks like:

### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_HANDLE 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_GET_DEVICE_HANDLE_CNF_T;
```

### Packet Description

structure PNS_IF_GET_DEVICE_HANDLE_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	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		Status has to be okay for this service.
	ulCmd	UINT32	0x1FB1	PNS_IF_GET_DEVICE_HANDLE_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	HANDLE		The device handle

Table 42: PNS\_IF\_GET\_DEVICE\_HANDLE\_CNF\_T - Get Device Handle Confirmation

## 7.9 Register Application Service

Using the Register Application Service the user application provides the stack an endpoint to send indications to.

The service is described in “DPM Interface Manual for netX based Products” [4].

If the user application is registered at the stack, it will receive the following indications (if the event triggering any of these indications occurs):

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



---

**Note:**

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

---

### 7.9.1 Register Application Request

This service is described in “DPM Interface Manual for netX based Products” [4].

### 7.9.2 Register Application Confirmation

This service is described in “DPM Interface Manual for netX based Products” [4].

## 7.10 Unregister Application Service

Using this Service the protocol stack will delete information about the registered user application.

This service is described in the “*DPM Interface Manual for netX based Products*” [4].

The user application will no longer receive any indication from the stack.

**Note:**

Without a registered application the stack is not fully functional as some acyclic events have to be handled by user application.

---

### 7.10.1 Unregister Application Request

This service is described in the “*DPM Interface Manual for netX based Products*” [4].

### 7.10.2 Unregister Application Confirmation

This service is described in the “*DPM Interface Manual for netX based Products*” [4].

## 7.11 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:

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 Dualport Memory is used.



### Note:

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.

### 7.11.1 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 ... $2^{32}-1$	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 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 43: 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);
```

## 7.11.2 Register Fatal Error Callback Confirmation

With his 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 44: PNS\_REG\_FATAL\_ERROR\_CALLBACK\_CNF\_T - Register Fatal Error Callback Confirmation



## 7.12 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 be reported fatal errors using the service described in 9.14.

### 7.12.1 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 45: PNS\_UNREG\_FATAL\_ERROR\_CALLBACK\_REQ\_T - Unregister Fatal Error Callback Request

## 7.12.2 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 46:** PNS\_UNREG\_FATAL\_ERROR\_CALLBACK\_CNF\_T - Unregister Fatal Error Callback Confirmation

## 7.13 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 can not 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 Hilscher address range is "Chassis MAC-address +1" and "Chassis MAC-address +2" for the 2 Ethernet ports".

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

### 7.13.1 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 47: PNS\_IF\_SET\_PORT\_MAC\_REQ\_T - Set Port MAC Address Request

## 7.13.2 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 48: PNS\_IF\_SET\_PORT\_MAC\_CNF\_T - Set Port MAC Address Confirmation

## 7.14 Set OEM Parameters Service

For some special needs (like I&M) of the user this request is used. It is designed for multiple purpose 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 user he 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.

### 7.14.1 Set OEM Parameters Request

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



### Important:

This packet should be sent prior to the *Set Configuration Service* packet, however it will be accepted later on as well.



### 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_6      0x06

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_6_Ttag
{
    TLR_UINT16 usIRTCycleCounterOffset;
} PNS_IF_SET_OEM_PARAMETERS_TYPE_6_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_6_T      tType6Param;

} 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
Data	structure PNS_IF_SET_OEM_PARAMETERS_REQ_DATA_T			
	ulParamType	UINT32		This parameter describes what struct 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 49: PNS\_IF\_SET\_OEM\_PARAMETERS\_REQ\_T - Set OEM Parameters Request

### 7.14.1.1 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
	abSerialNumbe r	UINT8[1 6]		The SerialNumber to report inside I&M 0 response. The serial number shall be left aligned and space (hex 0x20) padded.
	usProfileId	UINT16		The ProfileId to report inside I&M.
	usRevisionCou nter	UINT16		RevisionCounter to report inside I&M.
	usProfileSpec ificType	UINT16		The ProfileSpecificType to report in I&M.

Table 50: PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_1\_T - Set OEM Parameters for ulParamType = 1



### 7.14.1.2 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 SerialNumber to report inside I&M 0 response. The serial number shall be left aligned and space (hex 0x20) padded.

Table 51: PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_2\_T - Set OEM Parameters for ulParamType = 2

### 7.14.1.3 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 PNS_IF_SET_OEM_PARAMETERS_TYPE_6_T				
Area	Variable	Type	Value / Range	Description
	usIRTCycleCounterOffset	UINT16	not 0 and less than input DPM area size	Offset for IRT cycle counter value. IRT cycle counter shows the actual IRT cycle to which the input data belongs to.  The offset value shall not be zero and be less than DPM input area size. It also should not overlap with configured input modules.

Table 52: PNS\_IF\_SET\_OEM\_PARAMETERS\_TYPE\_6\_T - Set OEM Parameters for ulParamType = 6

## 7.14.2 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.

structure PNS_IF_SET_OEM_PARAMETERS_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
	ulCmd	UINT32	0x1FE9	PNS_IF_SET_OEM_PARAMETERS_CNF - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch

*Table 53: PNS\_IF\_SET\_OEM\_PARAMETERS\_CNF\_T - Set OEM Parameters Confirmation*

## 7.15 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 a special task startup parameter is set properly. Therefore this service is only available if the user is working with the linkable object module (or source code) of the PROFINET IO-Device stack.



---

**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.

---

## 7.15.1 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 the data will be used.

If the stack is accessed via Dual Port Memory it may be required for the application the fragment the request packet. This is due to the fact that the mailbox size is limited. How the fragmented service shall be handled by the application and the stack is described in [4].

**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

```
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 [4].
	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 54: PNS\_IF\_LOAD\_REMANENT\_DATA\_REQ\_T - Load Remanent Data Request

## 7.15.2 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 55: PNS\_IF\_LOAD\_REMANENT\_DATA\_CNF\_T - Load Remanent Data Confirmation

## 7.16 Configuration Delete Service

Using the Configuration Delete Service the user application can delete the current configuration.

This service will **not** delete configuration file (e.g. a SYCON.net database). This has to be done by the user.

**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.

---

### 7.16.1 Configuration Delete Request

This service is described in “DPM Interface Manual for netX based Products” [4].

### 7.16.2 Configuration Delete Confirmation

This service is described in “DPM Interface Manual for netX based Products” [4].

## 7.17 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 7.2 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.

### 7.17.1 Set IO-Image Request

#### Packet Structure Reference

```
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			
	ulConslImageSize	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.
	pbConslImage	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 subchapter 7.2 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 subchapter 7.2 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 56: PNS\_IF\_SET\_IOIMAGE\_REQ\_T – Set IO-Image Request

## 7.17.2 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
);

#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

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

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 ... $2^{32}-1$	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	0x1FF1	PNS_IF_SET_IOIMAGE_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_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.
	pfnUpdateExtendedStatusBlock	PNS_IF_UPDATE_EXTENDED_STATUS_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 57: PNS\_IF\_SET\_IOIMAGE\_CNF\_T – Set IO-Image Confirmation

## 7.18 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 subchapter 7.2.1 which contains an example.



### Note:

This applies to stack version 3.3 and above only. Previous stack versions do not provide this functionality. This functionality is supported for all methods of cyclic data exchange.

### 7.18.1 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    ulConsImageIOPSMODE;
    TLR_UINT32    ulConsImageIOPSOFFSET;
    TLR_UINT32    ulProvImageIOPSMODE;
    TLR_UINT32    ulProvImageIOPSOFFSET;

    TLR_UINT32    ulConsImageIOCSMODE;
    TLR_UINT32    ulConsImageIOCSOFFSET;
    TLR_UINT32    ulProvImageIOCSMODE;
    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			
	ulConslmageIO PSMode	UINT32	0 ... 2	How the consumer data provider states shall be encoded by the stack. Either disabled, bit list (valid invalid) or byte list (complete IOPS)
	ulConslmageIO PSOffset	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 subchapter 7.2.1 for further information
	ulProvImageIO PSMode	UINT32	0 ... 2	How the provider data provider states shall be decoded by the stack. Either disabled, bit list (valid invalid) or byte list (complete IOPS)
	ulProvImageIO PSOffset	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 subchapter 7.2.1 for further information
	ulConslmageIO CSMode	UINT32	0	Reserved for future usage. Shall be set to zero for compatibility.
	ulConslmageIO CSOffset	UINT32	0	Reserved for future usage. Shall be set to zero for compatibility. See subchapter 7.2.1 for further information
	ulProvImageIO CSMode	UINT32	0	Reserved for future usage. Shall be set to zero for compatibility.
	ulProvImageIO CSOffset	UINT32	0	Reserved for future usage. Shall be set to zero for compatibility. See subchapter 7.2.1 for further information

Table 58: PNS\_IF\_SET\_IOXS\_CONFIG\_REQ\_T – Set IOXS Config Request

## 7.18.2 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: 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		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 59: PNS\_IF\_SET\_IOXS\_CONFIG\_CNF\_T – Set IOXS Config Confirmation

## 7.19 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.



**Note:** This request is available for stack version **3.4.0 and above** only.

### 7.19.1 Configure Signal Request

The request packet has to be send 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.



structure IO_SIGNALS_CONFIGURE_SIGNAL_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
Data	structure IO_SIGNALS_CONFIGURE_SIGNAL_REQ_DATA_T			
	ulFieldbusSpecif1	UINT32	0	Unused, set to zero.
	ulFieldbusSpecif2	UINT32	0	The api of the submodule. Currently only 0 supported.
	ulFieldbusSpecif3	UINT32	0..0x7FFF	The slot of the submodule.
	ulFieldbusSpecif4	UINT32	1..0x7FFF	The subslot of the submodule
	ulFieldbusSpecif5	UINT32	0	Unused. Set to zero
	ulFieldbusSpecif6	UINT32	0	Unused. Set to zero
	ulFieldbusSpecif7	UINT32	0	Unused. Set to zero
	ulFieldbusSpecif8	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. First byte is signal type, 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 60: IO\_SIGNALS\_CONFIGURE\_SIGNAL\_REQ\_T - Configure Signal Request

## 7.19.2 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 61: IO\_SIGNALS\_CONFIGURE\_SIGNAL\_CNF\_T – Configure Signal Confirmation

## 7.19.3 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 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;
```

## 7.20 Usage of Linkable Object Module

If the stack is used as Linkable Object Module, the user has to create its own configuration file (which among others contains the task start-up parameters). Furthermore, depending on the application interface – Shared Memory Interface or not -, the user has to implement support for one of the provided data exchange mechanisms described in section 5. This section tries to give some hints what can be influenced by the user and what he has to take care of.



### 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.

### 7.20.1 Task Startup Parameters

Every single task (as described in section *Commissioning the PROFINET IO Device Stack V3* on page 62) requires its own start-up parameters to work correctly. If the parameters are not valid the stack will not work properly.

An example of the configuration-file should be delivered together with the Linkable Object Module and this Manual.

#### 7.20.1.1 PNSIF Task

The user can influence the behavior of the stack by changing the startup parameters of the PNSIF task. The task startup parameters are defined as follows:

```
typedef struct PNS_IF_STARTUPPARAMETER_Ttag    /* task startup parameter */
{
    TLR_TASK_PARAMETERHEADER;

    TLR_UINT32    ulDPMChannel;    /* DPM channel */
    TLR_STR*      pszEddName;      /* name of EDD to identify */
    TLR_UINT32    ulPNSIFFlags;    /* flags */
    TLR_STR*      pszSFLEDName;    /* name of SF LED to identify */
    TLR_STR*      pszBFLEDName;    /* name of BF LED to identify */
    TLR_STR*      pszSignalLEDName; /* name of LED to use for special signal event */

    /* the following parameters only apply to stack v3.3.x and above */
    /** amount of Ethernet ports available */
    TLR_UINT32    ulEthPortCnt;
} PNS_IF_STARTUPPARAMETER_T;
```

If the DualPortMemory is used, the parameter `ulDPMChannel` defines which Communication Channel will be used by the PROFINET IO-Device stack. Typically this value should be set to the first valid Communication Channel 0.

The name of the EDD shall be put to `pszEddName`. This name is used by other tasks, too and shall always have the same value. Typically the default value is "ETHERNET".

The behavior of the stack can be influenced by the parameter `ulPNSIFFlags`.

The following values (or a combination of them) can be set here:

```
typedef enum
{
    PNS_IF_DISABLE_DBM                = 0x00000001,
    PNS_IF_DISABLE_LEDS               = 0x00000002,
    PNS_IF_DISABLE_STACK_STORE_REMANENT = 0x00000004,
} PNS_IF_STARTUP_PARAM_FLAGS_E;
```

PNS\_IF\_DISABLE\_DBM will disable SYCON.net database support. Even if a database exists in the flash memory, it will not be evaluated.

PNS\_IF\_DISABLE\_LEDS will disallow the stack to access the LEDs. If this choice is made, the SF and BF LED are not set by the stack. If this parameter is set the three LED-Name startup parameters will be ignored.

PNS\_IF\_DISABLE\_STACK\_STORE\_REMANENT will disallow the stack to store the remanent data in the SYSVOLUME. If this choice is made, the stack will indicate the remanent data to the user application with the Store Remanent Data Service. This value should be set if the hardware does not have a flash SYSVOLUME and/or the user application shall permanently store the remanent parameters.

The name of the SF-LED shall be put to `pszSFLedName`. An LED with this name shall be registered to rcX (via config.c) so that the stack can identify it. If this succeeds, the LED will be handled by the stack as described in xxx.

The name of the BF-LED shall be put to `pszBFLedName`. An LED with this name shall be registered to rcX (via config.c) so that the stack can identify it. If this succeeds, the LED will be handled by the stack as described in xxx.

The name of the Signal-LED shall be put to `pszSignalLedName`. An LED with this name shall be registered to rcX (via config.c) so that the stack can identify it. If this succeeds, the LED will be used by the stack to indicate the receipt of a DCP Signal request. In parallel, this receipt will be indicated to the application with the Start LED Blinking Service.

The amount of Ethernet Ports shall be put in `ulEthPortCnt`. Typically this value will be 2 for all netX based products.

### 7.20.1.2 CMDEV Task

This task's startup parameters are defined as follows:

```
typedef struct PNIO_CMDEV_STARTUPPARAMETER_Ttag /* task startup parameter */
{
    TLR_TASK_PARAMETERHEADER;

    TLR_UINT uiMaxCmdevPm /* max. amount of parallel CMDEV instances */
    TLR_UINT uiMinDeviceInterval; /* GSDML file parameter MinDeviceInterval */
} PNIO_CMDEV_STARTUPPARAMETER_T_V2;
```

The parameter `uiMaxCmdev` specifies how many CMDEV instances are allowed to exist simultaneously. Currently only the value 1 is supported.

The parameter `uiMinDeviceInterval` must be set to the value `MinDeviceInterval` from the GSDML file corresponding to the final product the stack is used for.

### 7.20.1.3 MGT Task

This task's startup parameters do not have influence on the behavior of the stack. They are defined as follows:

```
typedef struct PNIO_MGT_STARTUPPARAMETER_Ttag    /* task startup parameter */
{
    TLR_TASK_PARAMETERHEADER;

    TLR_STR*   pszEddName;           /* name of EDD to identify */
} PNIO_MGT_STARTUPPARAMETER_T;
```

The name of the EDD shall be put to `pszEddName`. This name is used by other tasks, too, and shall always have the same value. Typically the default value is "ETHERNET".

### 7.20.1.4 DCP Task

This task's startup parameters do not have influence on the behavior of the stack. They are defined as follows:

```
typedef struct PNIO_DCP_STARTUPPARAMETER_Ttag    /* task startup parameter */
{
    TLR_TASK_PARAMETERHEADER;

    TLR_UINT   uiMaxDcpPm;           /* max. amount of DCP protocol machines */
} PNIO_DCP_STARTUPPARAMETER_T;
```

The parameter `uiMaxDcpPm` specifies the number of DCP protocol machine instances that are allowed to exist simultaneously. Currently only the value 2 is supported.

### 7.20.1.5 ACP Task

This task's startup parameters do not have influence on the behavior of the stack. They are defined as follows:

```
typedef struct PNIO_ACP_STARTUPPARAMETER_V2_Ttag /* task startup parameter */
{
    TLR_TASK_PARAMETERHEADER;

    TLR_STR*   pszEddName;           /* name of EDD to identify */
    TLR_UINT   uiMaxAlpm;            /* max. amount of alarm protocol machines */
    TLR_STR*   pszRTIrqName;         /* RT Scheduler Interrupt name */
    TLR_STR*   pszTimerIrqName;      /* Timer Interrupt name */
    TLR_STR*   pszIRTMSyncCh0Name;   /* IRT MSYNC Interrupt name Channel 0 */
    TLR_STR*   pszIRTMSyncCh1Name;   /* IRT MSYNC Interrupt name Channel 1 */
    TLR_STR*   pszPhy0Name;          /* name of PHY 0 to identify */
    TLR_STR*   pszPhy1Name;          /* name of PHY 1 to identify */
} PNIO_ACP_STARTUPPARAMETER_V2_T;
```

The name of the EDD shall be put to `pszEddName`. This name is used by other tasks, too, and shall always have the same value. Typically the default value is "ETHERNET".

The parameter `uiMaxAlpm` specified the number of Alarm protocol machine instances that are allowed to exist simultaneously. Currently only the value 2 is supported.

The name of the RT Scheduler Interrupt shall be put to `pszRTIrqName`. The interrupt with the corresponding name (see Additional Hardware Resources) shall be defined in the configuration file.

The name of the Timer Interrupt shall be put to `pszTimerIrqName`. The interrupt with the corresponding name (see Additional Hardware Resources) shall be defined in the configuration file.

The name of the IRT MSync Interrupt for Channel 0 shall be put to `pszIRTMSyncCh0Name`. The interrupt with the corresponding name (see Additional Hardware Resources) shall be defined in the configuration file.

The name of the IRT MSync Interrupt for Channel 1 shall be put to `pszIRTMSyncCh1Name`. The interrupt with the corresponding name (see Additional Hardware Resources) shall be defined in the configuration file.

The name of the PHYs shall be put to `pszPhy0Name` and `pszPhy1Name`. The name of the PHY shall be defined in the configuration file. The ACP task will identify and use this PHY resource via `rcX` functions.

### 7.20.1.6 RPC Task

This task's startup parameters do not have influence on the behavior of the stack. They are defined as follows:

```
typedef struct RPC_TASK_STARTUPPARAMETER_V2_Ttag /* task startup parameter */
{
    TLR_TASK_PARAMETERHEADER;

    TLR_STR*   pszEddName; /* name of EDD to identify */
} RPC_TASK_STARTUPPARAMETER_V2_T;
```

The name of the EDD shall be put to `pszEddName`. This name is used by other tasks, too and shall always have the same value. Typically the default value is "ETHERNET".

### 7.20.1.7 LLDP Task

This task's startup parameters do not have influence on the behavior of the stack. However, it is important to use the correct values here. Otherwise the stack will consume more memory than necessary and may not work as expected. The parameters are defined as follows:

```
typedef struct LLDP_STARTUPPARAMETER_Ttag
{
    TLR_TASK_PARAMETERHEADER;

    TLR_UINT    uiRemoteMibElemMax; /* max. entries in remote systems MIB */
    TLR_UINT    uiPortsNum; /* number of available Ethernet ports */
    TLR_UINT    uiManAddrNum; /* max number of management addresses for either
                               local or remote system information entry */
    TLR_UINT    uiVlanNameNum; /* max number of XDOT1 Vlan Names for either
                               local or remote system information entry */
    TLR_UINT    uiPpvidNum; /* max number of XDOT1 PPVIDs for either local or
                             remote system information entry */
    TLR_UINT    uiPidNum; /* max number of XDOT1 PIDs for either local or
                           remote system information entry */

    TLR_UINT    uiUnknownNum; /* max number of either basic set unknown TLVs or
                               extension set unknown TLVs for either local or
                               remote system information entry */

    TLR_STR*    pszEddName; /* name of EDD to identify*/
    TLR_STR*    pszPhy0Name; /* name of PHY 0 to identify */
    TLR_STR*    pszPhy1Name; /* name of PHY 1 to identify */
} LLDP_STARTUPPARAMETER_T;
```

The parameter `uiRemoteMibElemMax` represents the number of entries the remote system MIB can handle. It shall be set to 2.

The parameter `uiPortsNum` represents the amount of Ethernet ports of the hardware. The default value is 2.

The parameter `uiManAddrNum` represents the maximum amount of management addresses. The default value is 10.

The parameter `uiVlanNameNum` represents the maximum amount of VLAN Name entries. The default value is 3.

The parameter `uiPpvidNum` represents the maximum amount of PPVID entries. The default value is 3.

The parameter `uiPidNum` represents the maximum amount of PID entries. The default value is 3.

The parameter `uiUnknownNum` represents the maximum amount of unknown entries. The default value is 5.

The name of the EDD shall be put to `pszEddName`. This name is used by other tasks, too, and shall always have the same value. Typically the default value is "ETHERNET".

The name of the PHYs shall be put to `pszPhy0Name` and `pszPhy1Name`. The name of the PHY shall be defined in the configuration file. The ACP task will identify and use this PHY resource via `rcX` functions.

### 7.20.1.8 SNMP Task

This task's startup parameters do not have influence on the behavior of the stack. They are defined as follows:

```
typedef struct SNMP_SERVER_STARTUPPARAMETER_Ttag
{
    TLR_TASK_PARAMETERHEADER;
    TLR_UINT32                ulReserved;
} SNMP_SERVER_STARTUPPARAMETER_T;
```

The parameter `ulReserved` is reserved. Its value shall be set to 0.

### 7.20.1.9 MIB Task

This task's startup parameters do not have influence on the behavior of the stack. They are defined as follows:

```
typedef struct MIB_DATABASE_STARTUPPARAMETER_Ttag
{
    TLR_TASK_PARAMETERHEADER;
    TLR_UINT32                ulReserved;
} MIT_DATABASE_STARTUPPARAMETER_T;
```

The parameter `ulReserved` is reserved. Its value shall be set to 0.



### 7.20.1.10 TCP/IP Task

This task's startup parameters are described in the TCP/IP Protocol Interface Manual [5].

For correct operation of the PROFINET IO-Device stack it is required that the parameters are as follows:

```

STATIC CONST TCPIP_TCP_TASK_STARTUPPARAMETER_T g_tTcpIpPrm =
{
    TLR_TASK_TCPUDP,                /* ulTaskIdentifier      */
    TCPIP_STARTUPPARAMETER_VERSION_V4, /* ulParamVersion        */
    TCPIP_SRT_QUE_ELEM_CNT_AP_DEFAULT, /* ulQueElemCntAp        */
    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, /* ulQueFreeElemCnt     */
    TCPIP_SRT_SOCKET_MAX_CNT_DEFAULT,  /* ulSocketMaxCnt        */
    TCPIP_SRT_ARP_CACHE_SIZE_DEFAULT,  /* ulArpCacheSize        */
    PNS_EDD_IDENTIFY_NAME,             /* name of EDD to identify */
    TCPIP_SRT_EDD_QUE_POOL_ELEM_CNT_DEFAULT, /* ulEddQuePoolElemCnt */
    TCPIP_SRT_EDD_OUT_BUF_MAX_CNT_DEFAULT, /* ulEddOutBufMaxCnt    */
    NULL,                             /* EthInterf. Parameters */
    180,                             /* ARP Cache Timeout     */
};

```

The only difference to the default parameters are the elements `ulStartFlags` whose value shall be set to `TCPIP_SRT_FLAG_FAST_START` and the ARP Cache Timeout which shall be 60 seconds for PROFINET.

## 7.20.2 Task priorities

It is recommended to use the following task priorities as can be seen in the example configuration file:



### Note:

Any change in these priorities may lead to **problems which will be difficult to detect**. It is highly recommended **not to create any task with a higher priority than PNSIF-Task**. **Otherwise** the stack is expected **not to work properly**.

Name	Priority
XC0 Comm.-Interrupt	TSK_PRIO_3
XC1 Comm.-Interrupt	TSK_PRIO_4
RT Scheduler Interrupt	TSK_PRIO_10
ACP-Task	TSK_PRIO_11
TLR-Timer Task	TSK_PRIO_12
DCP-Task	TSK_PRIO_13
TCP/IP-Task	TSK_PRIO_15
RPC-Task	TSK_PRIO_16
MGT-Task	TSK_PRIO_17
CMDEV-Task	TSK_PRIO_18
LLDP-Task	TSK_PRIO_19
PNSIF-Task	TSK_PRIO_21
SNMP-Task	TSK_PRIO_22
SNMP MIB-Task	TSK_PRIO_23
rcX MidSys-Task	TSK_PRIO_25

Table 62: Overview about the recommended Task Priorities

## 7.20.3 Additional Hardware Resources

The PROFINET IO-Device stack requires some additional resources to work correctly. They have to be added to the configuration file.

### 7.20.3.1 Hardware Timer

The PROFINET IO-Device stack requires the usage of one of the hardware timers from the GPIO component. This timer is used for cyclic data exchange and will be reprogrammed by the stack whenever necessary.

It shall be configured as follows:

```
{ {"PNS_RT_TIMER", RX_PERIPHERAL_TYPE_TIMER, 0},
  1,                               /* can be freely chosen; here: use GPIO_counter1 */
  1000,                             /* don't care, will be reprogrammed */
  TRUE,                             /* Continuous Mode */
  TRUE,                             /* Interrupt enabled */
  FALSE,                           /* No external Clock */
  RX_HWTIMER_TRIGGER_NONE,         /* No Trigger */
  0,                               /* No I/O reference */
  0                                /* No Prescaler */
},
```

Note that the name of this resource has to be handed over to the ACP Task via its startup parameter `pszTimerIrqName`.

### 7.20.3.2 Interrupts for stack v3.3

The PROFINET IO-Device stack requires additional interrupts. There is a difference between netX50 and netX100/500.

#### Interrupts required for both netX50 and netX100/500

The RT-Scheduler needs an interrupt:

```
{ {"PNS_RT_IRQ", RX_PERIPHERAL_TYPE_INTERRUPT, 0}, /* */
  SRT_NETX_VIC_IRQ_STAT_timer1, /* gpio timer 1 irq, must match the hw-timer */
  21, /* Interrupt Priority 21 */
  RX_INTERRUPT_MODE_TASK, /* Allow interrupt to be a thread */
  RX_INTERRUPT_EOI_AUTO, /* EOI self by RX */
  RX_INTERRUPT_TRIGGER_RISING_EDGE, /* Edge triggered */
  RX_INTERRUPT_PRIORITY_STANDARD, /* Normal Priority */
  RX_INTERRUPT_REENTRANCY_DISABLED, /* Interrupt is not reentrant */
  TSK_PRIO_10, /* Task priority 10 */
  TSK_TOK_10, /* Task Token 10 */
  2048 /* */
}
```

**Interrupts required for IRT-Switch (supported for netX50 and netX100/500)**

The IRT-Switch and IRT-Scheduler requires these interrupts:

```
{ {"PNS_IRT_MSNC",RX_PERIPHERAL_TYPE_INTERRUPT,0}, /* Comm. Channel, Instance 0 */
  SRT_NETX_VIC_IRQ_STAT_msnc0,          /* ext. comm. channel 0 IRQ */
  16,                                   /* Interrupt Priority 16 */
  RX_INTERRUPT_MODE_SYSTEM,             /* System mode */
  RX_INTERRUPT_EOI_AUTO,                 /* EOI self by RX */
  RX_INTERRUPT_TRIGGER_RISING_EDGE,     /* Edge triggered */
  RX_INTERRUPT_PRIORITY_STANDARD,        /* Normal Priority */
  RX_INTERRUPT_REENTRANCY_DISABLED,      /* Interrupt is not reentrant */
},
{ {"PNS_IRT_MSNC",RX_PERIPHERAL_TYPE_INTERRUPT,1}, /* Comm. Channel Instance 1 */
  SRT_NETX_VIC_IRQ_STAT_msnc1,          /* external comm. channel 1 IRQ */
  17,                                   /* Interrupt Priority 17 */
  RX_INTERRUPT_MODE_SYSTEM,             /* System mode */
  RX_INTERRUPT_EOI_AUTO,                 /* EOI self by RX */
  RX_INTERRUPT_TRIGGER_RISING_EDGE,     /* Edge triggered */
  RX_INTERRUPT_PRIORITY_STANDARD,        /* Normal Priority */
  RX_INTERRUPT_REENTRANCY_DISABLED,      /* Interrupt is not reentrant */
},
{ {"PNS_COM",RX_PERIPHERAL_TYPE_INTERRUPT,0}, /* comm. Channel, Instance 0 */
  SRT_NETX_VIC_IRQ_STAT_com0,          /* ext. comm. channel 0 IRQ */
  18,                                   /* Interrupt Priority 18 */
  RX_INTERRUPT_MODE_SYSTEM,             /* System-mode */
  RX_INTERRUPT_EOI_AUTO,                 /* EOI self by RX */
  RX_INTERRUPT_TRIGGER_RISING_EDGE,     /* Edge triggered */
  RX_INTERRUPT_PRIORITY_STANDARD,        /* Normal Priority */
  RX_INTERRUPT_REENTRANCY_DISABLED,      /* Interrupt is not reentrant */
  TSK_PRIO_4,
  TSK_TOK_4,
},
{ {"PNS_COM",RX_PERIPHERAL_TYPE_INTERRUPT,1}, /* comm. Channel, Instance 1 */
  SRT_NETX_VIC_IRQ_STAT_com1,          /* ext. comm. channel 1 IRQ */
  19,                                   /* Interrupt Priority 19 */
  RX_INTERRUPT_MODE_SYSTEM,             /* System mode */
  RX_INTERRUPT_EOI_AUTO,                 /* EOI self by RX */
  RX_INTERRUPT_TRIGGER_RISING_EDGE,     /* Edge triggered */
  RX_INTERRUPT_PRIORITY_STANDARD,        /* Normal Priority */
  RX_INTERRUPT_REENTRANCY_DISABLED,      /* Interrupt is not reentrant */
  TSK_PRIO_4,
  TSK_TOK_4,
},
{ {"PNS_COM",RX_PERIPHERAL_TYPE_INTERRUPT,3}, /* comm. Channel, Instance 3 */
  SRT_NETX_VIC_IRQ_STAT_com3,          /* ext. comm. channel 1 IRQ */
  20,                                   /* Interrupt Priority 20 */
  RX_INTERRUPT_MODE_SYSTEM,             /* System mode */
  RX_INTERRUPT_EOI_AUTO,                 /* EOI self by RX */
  RX_INTERRUPT_TRIGGER_RISING_EDGE,     /* Edge triggered */
  RX_INTERRUPT_PRIORITY_STANDARD,        /* Normal Priority */
  RX_INTERRUPT_REENTRANCY_DISABLED,      /* Interrupt is not reentrant */
  TSK_PRIO_4,
  TSK_TOK_4,
}
```

## Interrupts required for 2PortSwitch (not IRT capable!) (supported for netX50 and netX100/500)

The 2port-Switch requires these interrupts:

```
{ {"PNS_COM",RX_PERIPHERAL_TYPE_INTERRUPT,0}, /* comm. Channel, Instance 0 */
  SRT_NETX_VIC_IRQ_STAT_com0, /* ext. comm. channel 0 IRQ */
  16, /* Interrupt Priority 16 */
  RX_INTERRUPT_MODE_TASK, /* Allow interrupt to be a thread */
  RX_INTERRUPT_EOI_AUTO, /* EOI self by RX */
  RX_INTERRUPT_TRIGGER_RISING_EDGE, /* Edge triggered */
  RX_INTERRUPT_PRIORITY_STANDARD, /* Normal Priority */
  RX_INTERRUPT_REENTRANCY_DISABLED, /* Interrupt is not reentrant */
  TSK_PRIO_3,
  TSK_TOK_3,
  2048
},
{{ "PNS_COM",RX_PERIPHERAL_TYPE_INTERRUPT,1}, /* comm. Channel, Instance 1 */
  SRT_NETX_VIC_IRQ_STAT_com1, /* ext. comm. channel 1 IRQ */
  17, /* Interrupt Priority 17 */
  RX_INTERRUPT_MODE_TASK, /* Allow interrupt to be a thread */
  RX_INTERRUPT_EOI_AUTO, /* EOI self by RX */
  RX_INTERRUPT_TRIGGER_RISING_EDGE, /* Edge triggered */
  RX_INTERRUPT_PRIORITY_STANDARD, /* Normal Priority */
  RX_INTERRUPT_REENTRANCY_DISABLED, /* Interrupt is not reentrant */
  TSK_PRIO_4,
  TSK_TOK_4,
  2048
}
```

## 8 Connection Establishment

After the stack is configured with the `Set_Configuration_Req` packet (see section *Set Configuration Service* of this document) it is ready to start communicating with an IO-Controller. If the IO-Controller is configured to connect to an IO-Device who's `NameOfStation` equals the one the IO-Device stack was handed over a `Connect_Request` will be sent from IO-Controller to IO-Device. The stack will indicate this with several packets to the application.

These packets are described in this section 8.

The typically interaction between stack and application is shown in the next figure.

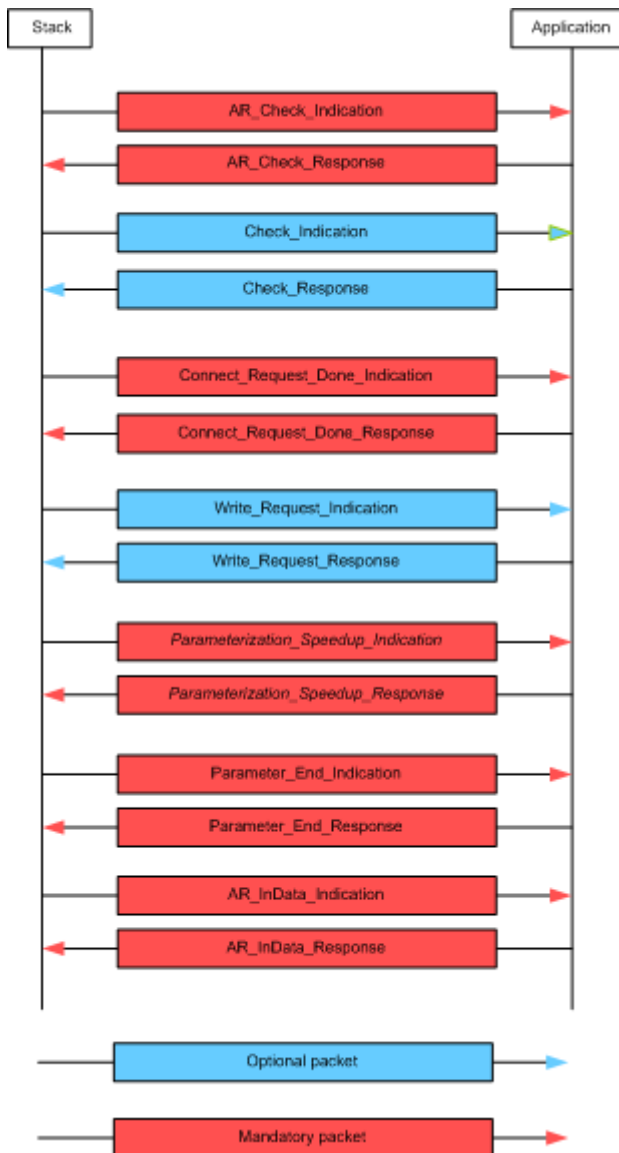


Figure 8: Exchange of Packets between Stack and Application during Connection Establishment

The first packet is the **AR\_Check\_Indication** (see section 8.1.1). The information inside this indication is informative only for the application. It may or may not use/store it. It indicates the beginning of a connection establishment phase.

In the next step the stack may indicate inconsistent module-configuration to the application. This is done in the case that the IO-Controller expects different submodules to be plugged at the IO-Device. In this case this is indicated to the application with a **Check\_Indication** (see section 8.2.1)

for every inconsistent submodule. The application has to decide what to do in this case. It may reconfigure the module configuration at this point. Depending on applications choice a ModuleDiffBlock containing information about the module configuration difference may be reported by the stack to the IO-Controller.

The stack sends the **Connect\_Request\_Done\_Indication** (see section 8.3.1) to the application to inform it that no more Check\_Indication will be sent. This is informative for the application only.

The **Write\_Record\_Indication** (see section 9.2.1) may be indicated to the application. It contains additional parameter data which has to be examined by the application.

After the last Write\_Record was sent from the IO-Controller it will indicate this to the IO-Device with a Parameter\_End Request. The stack checks the configured parameterization speedup UUID and informs the application using the **Parameterization\_Speedup\_Indication**. If the contained UUID is NIL (all values 0) parameterization speedup is not used or was disabled. If the UUID has a valid value the application shall store this UUID together with all parameters indicated with Write\_Record\_Indications and use them after the next power cycle. The stored UUID shall be compared with the one indicated for fast detection of parameter changes.

Afterwards the end of parameterization is indicated to the application with a **Parameter\_End\_Indication** (see section 8.4.1). If the application and all necessary submodules are ready the application returns this packet with a positive status. In this case the stack will automatically send the Application\_Ready to the IO-Controller to indicate that the connection establishment is finished. If the application returns the Parameter\_End\_Indication with a negative status code the stack will not send the Application\_Ready to IO-Controller. This has to be forced by application using service 8.5.

Finally, the **AR\_InData\_Indication** (see section 8.6.1) informs the application about the fact that the first cyclic frame from the IO-Controller was received. From now on the application can read the input IO-data from the Input TripleBuffer.

The AR\_InData\_Indication may be sent at a different time. It is sent by the stack when the first cyclic frame of the IO-Controller is received. This depends on the configured cycle time and is typically shortly after the Connect\_Request\_Done\_Indication is sent by the stack.

In detail, the following connection establishment functionality 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
8.1	AR Check Indication	0x1F14	129
	AR Check Response	0x1F15	131
8.2	Check Indication	0x1F16	132
	Check Response	0x1F17	134
8.3	Connect Request Done Indication	0x1FD4	138
	Connect Request Done Response	0x1FD5	140
8.4	Parameter End Indication	0x1F0E	143
	Parameter End Response	0x1F0F	143
8.5	Application Ready Request	0x1F10	145
	Application Ready Confirmation	0x1F11	147
8.6	AR InData Indication	0x1F28	148
	AR InData Response	0x1F29	150
8.7	Store Remanent Data Indication	0x1FEA	151
	Store Remanent Data Response	0x1FEB	153

Table 63: Overview over the Connection Establishment Packets of the PROFINET IO Device IRT StAR Check Service

After the stack has examined the Application Relation (AR)-block which is always the very first block inside IO-Controllers Connect-Request the application will be informed about the parameters from the AR block. The indication is for information only and the response is not further evaluated by the stack. The device handle may be used to distinguish between different application relations.



## 8.1 AR Check

### 8.1.1 AR Check Indication

This packet represents the indication with the information from the AR block. It is sent from the stack to application.

#### Packet Structure Reference

```
typedef struct PNS_IF_AR_CHECK_IND_DATA_Ttag
{
    TLR_HANDLE      hDeviceHandle;
    TLR_UINT16      usARType;
    TLR_UINT32      ulARProperties;
    TLR_UINT32      ulRemoteIpAddr;
    TLR_UINT16      usRemoteNameOfStationLen;
    TLR_UINT8       abRemoteNameOfStation[PNIO_MAX_NAME_OF_STATION];
} 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 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	256	Packet data 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	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	HANDLE	1-2	The device handle. The handle is described at chapter 0. The handling of this indication is dependent on this handle.
	usARType	UINT16		Connect-Request's AR-Type
	ulARProperties	UINT32		Connect-Request's AR-Properties.
	ulRemoteIpAddress	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.

Table 64: PNS\_IF\_AR\_CHECK\_IND\_T - AR Check Indication

## 8.1.2 AR Check Response

The application has to return this packet as AR check response.

### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_HANDLE 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	HANDLE		Handle to the IO-Device.

Table 65: PNS\_IF\_AR\_CHECK\_RSP\_T - AR Check Response

## 8.2 Check Indication Service

If the IO-Controller wants to use (sub)modules which are configured wrong (or are even not configured at all) in the IO-Device stack will indicate this to application with a Check Indication packet. This indication contains information about the requested parameters. The application has to decide what to do. It has to set the state for the corresponding (sub)module. Its decision will be the base for the stack to create the ModuleDiffBlock which is reported to IO-Controller.



### Note:

It is allowed for the application to correct the module configuration at this time using the Pull- (see sections 10.9 and 10.10) and Plug-Services (see sections 10.7 and 10.8). They shall be used **before** the Check Response is returned to the stack.



### Note:

For this service the stack has implemented a timeout. If the application needs more than 2 seconds to handle the request the stack will automatically generate a negative response the IO-Controller.

### 8.2.1 Check Indication

This packet indicates the parameters of a wrong/missing (sub)module to application.

#### Packet Structure Reference

```
typedef struct PNS_IF_CHECK_IND_IND_DATA_Ttag
{
    TLR_HANDLE 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	HANDLE		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 66: PNS\_IF\_CHECK\_IND\_T - Check Indication

## 8.2.2 Check Response

With this response packet the application influences the `ModuleDiffBlock` 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.



### Notes:

- 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.
- Currently only one single API is supported.

### Packet Structure Reference

```
typedef struct PNS_IF_CHECK_IND_RSP_DATA_Ttag
{
    TLR_HANDLE 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	HANDLE		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 ModuleID the IO-Controller expected
	usModuleState	UINT16	See below	The ModuleState
	ulSubmodId	UINT32		The SubmoduleID the IO-Controller expected.
	usSubmodState	UINT16	See below	The SubmoduleState

Table 67: 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.
PNIO_MODSTATE_CORRECT_MODULE (0xFFFF)	The correct module has been plugged by the application while evaluating this Check Indication.

Table 68: 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.
PNIO_SUBSTATE_CORRECT_SUBMODULE (0xFFFF)	The correct submodule has been plugged by the application while evaluating this Check Indication.

Table 69: Field `usSubmodState`



### 8.2.3 Module Reconfiguration after a Check Indication

The situation may occur at the PROFINET IO Device that the registered modules at the application differ from those requested from the PROFINET IO Controller via the bus. Such a conflict situation will lead to a check indication as described above.

The application has the possibility to change the configuration of the PROFINET IO stack according to its requirements. The exchange of packets between stack and application for reconfiguration is displayed below:

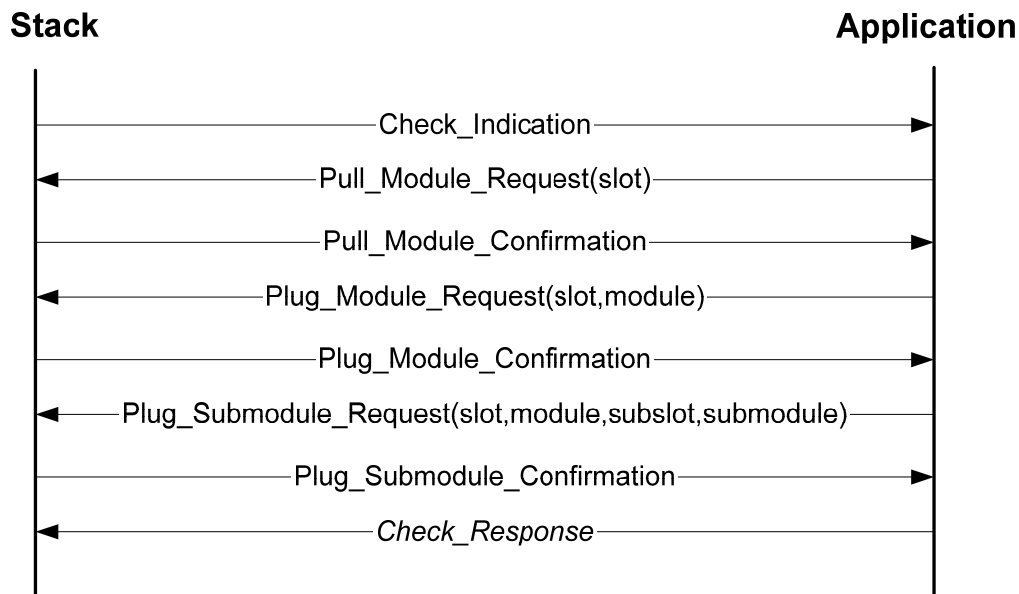


Figure 9: Exchange of Packets between Stack and Application for Reconfiguration



**Important:** If this dynamic reconfiguration process (which may consist of several Check Indications in a row) in total takes too much time (more than a few seconds) this may lead to the situation that the IO-Controller aborts the connection it currently tries to open. This will be indicated to the application with the "AR Abort Indication Service". Inside the IO-Controller a timer watches the time the IO-Device needs to answer the connect request. If this timer expires the connection will be closed before it was completely opened.

## 8.3 Connect Request Done Service

The stack sends the indication Connect Request Done to application to indicate that no more Check Indications will be sent by the stack. Now the application has to examine all information it was given by the stack to be able to have exact knowledge about the submodules and the frame offsets of their IO-data. The application has to put together the information from the Check\_Indication (see section 8.2) to do so.

### 8.3.1 Connect Request Done Indication

The indication packet is informative only.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_HANDLE 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 ... $2^{32}-1$	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	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	HANDLE		The device handle

Table 70: PNS\_IF\_CONNECTREQ\_DONE\_IND\_T - Connect Request Done Indication

## 8.3.2 Connect Request Done Response

The application has to return the packet.

### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_HANDLE 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	HANDLE		The device handle

Table 71: PNS\_IF\_CONNECTREQ\_DONE\_RSP\_T - Connect Request Done Response

## 8.4 Parameter End Service

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

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

If the application is ready, the packet shall be responded 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 become ready, the application shall set the field `fSendApplicationReady` to false and respond to the indication immediately. When the application becomes ready, it shall use the Application Ready Service described in section 8.5 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 8.5).



### Note:

For this service the stack has implemented a timeout. If the application needs more than 2 seconds to handle the request the stack will automatically generate a negative response the IO-Controller.

### 8.4.1 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_HANDLE      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 specifi
                                     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	HANDLE		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 72: PNS\_IF\_PARAM\_END\_IND\_T - Parameter End Indication

## 8.4.2 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_HANDLE      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	HANDLE		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 8.5.  The stack shall automatically send ApplicationReady.

Table 73: PNS\_IF\_PARAM\_END\_RSP\_T - Parameter End Response



## 8.5 Application Ready Service

With this service the application shall indicate to the stack, that its 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 can not 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 by 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 back an error (COM-Flag not set) which can be ignored.

### 8.5.1 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_HANDLE 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	HANDLE		The device handle

Table 74: PNS\_IF\_APPL\_READY\_IND\_T - Application Ready

## 8.5.2 Application Ready Confirmation

The stack will respond with this packet.

### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_HANDLE 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	HANDLE		Handle to the IO-Device.

Table 75: PNS\_IF\_APPL\_READY\_CNF\_T - Application Ready Confirmation

## 8.6 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.

From now on the application may access the Input-TripleBuffer to access the input IO-data if stack v3.2 is used.

### 8.6.1 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_HANDLE 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	HANDLE		The device handle

Table 76: PNS\_IF\_AR\_IN\_DATA\_IND\_T - AR InData Indication

## 8.6.2 AR InData Response

The application has to return this packet.

### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_HANDLE 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	HANDLE		Handle to the IO-Device.

Table 77: PNS\_IF\_AR\_IN\_DATA\_RSP\_T - AR InData Response

## 8.7 Store Remanent Data Service

Using this service the stack indicates the presence of remanent data to the user. The user is responsible for storing this data remanent e.g. to a flash memory. After the next power cycle the user has to hand this data over to the stack using the Load Remanent Data Service.

**Note:**

This service is optional and only used by the stack if special task startup parameters are used. It is only available for customers using the linkable object module of the PROFINET IO-Device stack.

### 8.7.1 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`. Most likely the amount of data is very small, but the user 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.
	ulCmd	UINT32	0x1FEA	PNS_IF_STORE_REMANENT_DATA_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			
	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 78: PNS\_IF\_STORE\_REMANENT\_DATA\_IND\_T - Store Remanent Data Indication



## 8.7.2 Store Remanent Data Response

The application has to return this packet.

### 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	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 79: PNS\_IF\_STORE\_REMANENT\_DATA\_RES\_T - Store Remanent Data Response

## 9 Acyclic Events indicated by the Stack

This section describes the acyclic services the stack indicates to the application. Depending on the service the application has to perform an action or simply return the packet.

Some of the 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.

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

Overview over the Packets for Acyclic Events indicated by the PROFINET IO Device IRT Stack			
No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
9.1	Read Record Indication	0x1F36	156
	Read Record Response	0x1F37	158
9.2	Write Record Indication	0x1F3A	169
	Write Record Response	0x1F3B	171
9.3	AR Abort Indication Indication	0x1F2A	173
	AR Abort Indication Response	0x1F2B	175
9.4	Save Station Name Indication	0x1F1A	176
	Save Station Name Response	0x1F1B	178
9.5	Save Station Type Indication	0x1F1C	179
	Save Station Type Response	0x1F1D	181
9.6	Save IP Address Indication	0x1FB8	182
	Save IP Address Response	0x1FB9	184
9.7	Start LED Blinking Indication	0x1F1E	185
	Start LED Blinking Response	0x1F1F	187
9.8	Stop LED Blinking Indication	0x1F20	188
	Stop LED Blinking Response	0x1F21	189
9.9	Reset Factory Settings Indication	0x1F18	190
	Reset Factory Settings Response	0x1F19	192
9.10	APDU Status Changed Indication	0x1F2E	193
	APDU Status Changed Response	0x1F2F	195
9.11	Alarm Indication Indication	0x1F30	196
	Alarm Indication Response	0x1F31	199
9.12	Release Request Indication	0x1FD6	200
	Release Request Indication Response	0x1FD7	202

Overview over the Packets for Acyclic Events indicated by the PROFINET IO Device IRT Stack			
No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
9.13	Link Status Changed Indication	0x1F70	204
	Link Status Changed Response	0x1F71	206
9.14	Error Indication	0x1FDC	207
	Error Indication Response	0x1FDD	209

*Table 80: Overview over the Packets for Acyclic Events indicated by the PROFINET IO Device IRT Stack*

## 9.1 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. Application has to return the Read Record Response packet to the stack so that the stack can send the Read Response to the IO-Controller.



### Note:

For this service the stack has implemented a timeout. If the application needs more than 2 seconds to handle the request the stack will automatically generate a negative response the IO-Controller.



### Note:

Since stack V3.4.11.0 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.

### 9.1.1 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_HANDLE      hRecordHandle;
    TLR_HANDLE      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	HANDLE		A stack internal identifier which belongs to this indication.
	hDeviceHandle	HANDLE		The device handle. <u>The handle is described at chapter 0.</u>
	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. For DPM targets as well as for stack versions prior to 3.4.11.0 n may be up to 1024. n may be up to 32768 for LOM target on stack version since 3.4.11.0.

Table 81: PNS\_IF\_READ\_RECORD\_IND\_T - Read Record Indication

## 9.1.2 Read Record Response

This packet has to be sent by the application as response to a Read Record Indication.



### Note:

The application will not get any feedback if the stack was able to successfully send the Read Record Response to the IO-Controller. However if something was wrong with the Read Record Response packet 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 9.14).

### Packet Structure Reference

```
typedef struct PNS_IF_READ_RECORD_RSP_DATA_Ttag
{
    TLR_HANDLE      hRecordHandle;
    TLR_HANDLE      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
	ulRout	UINT32	x	Routing, do not touch
Data	structure PNS_IF_READ_RECORD_RSP_DATA_T			
	hRecordHandle	HANDLE		A stack internal identifier which belongs to this indication.
	hDeviceHandle	HANDLE		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 Table 83: Coding of the field <u>ulPnio</u> .
	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 82: PNS\_IF\_READ\_RECORD\_RSP\_T - Read Record Response

Table 83 explains the coding of the fields ErrorCode, ErrorDecode, ErrorCode1 and ErrorCode2 within variable ulPnio:

Bits	Description
31-24	ErrorCode
23-16	ErrorDecode
15-8	ErrorCode1
7-0	ErrorCode2

Table 83: Coding of the field `ulPnio`**Example:**

The well defined error code “Read Record error – Application feature not supported” (ErrorCode 0xDE, ErrorDecode 0x80, ErrorCode1 0xA9, ErrorCode2 0x00) shall be coded as ulPnio 0xDE80A900.

**Coding of ErrorCode:**

Value	Description
0x01 - 0x80	Reserved
0x81	PNIO
0x82 - 0xCE	Reserved
0xCF	RTA error
0xD0 - 0xD9	Reserved
0xDA	AlarmAck
0xDB	IODConnectRes
0xDC	IODReleaseRes
0xDD	IODControlRes
0xDE	IODReadRes
0xDF	IODWriteRes
0xE0 - 0xEF	Reserved
0xF0 - 0xFF	Reserved

Table 84: Coding of ErrorCode

**Coding of ErrorDecode:**

Value	Description	Use
0x01 - 0x7F	Reserved	
0x80	PNIORW	Used in context with user error codes of the services Read, Write, Read Input Data, Read Output Data, Read Device Diagnosis, Read AR Data, Read Logbook, Read Expected Identification, Read Real Identification
0x81	PNIO	Used in context with other services or internal e.g. RPC errors
0x82 - 0xFF	Reserved	

Table 85: Coding of ErrorDecode



Coding of ErrorCode1 and ErrorCode 2 for ErrorDecode = PNIOBW:

ErrorCode1 consists of ErrorClass and ErrorDecode. ErrorCode2 is user specific in this case.

ErrorClass (decimal) bit 7 - 4	Description	ErrorDecode (decimal) bit 3- 0
0 - 9	not specified	not specified
10	Application	0 = read error 1 = write error 2 = module failure 3 = not specified 4 = not specified 5 = not specified 6 = not specified 7 = busy 8 = version conflict 9 = feature not supported 10 = user specific 1 11 = user specific 2 12 = user specific 3 13 = user specific 4 14 = user specific 5 15 = user specific 6
11	Access	0 = invalid index 1 = write length error 2 = invalid slot / subslot 3 = type conflict 4 = invalid area / API 5 = state conflict 6 = access denied 7 = invalid range 8 = invalid parameter 9 = invalid type 10 = backup 11 = user specific 7 12 = user specific 8 13 = user specific 9 14 = user specific 10 15 = user specific 11

ErrorClass (decimal) bit 7 - 4	Description	ErrorDecode (decimal) bit 3- 0
12	Resource	0 = read constrain conflict 1 = write constrain conflict 2 = resource busy 3 = resource unavailable 4 = not specified 5 = not specified 6 = not specified 7 = not specified 8 = user specific 12 9 = user specific 13 10 = user specific 14 11 = user specific 15 12 = user specific 16 13 = user specific 17 14 = user specific 18 15 = user specific 19
13 - 15	user specific	not specified

Table 86: Coding of ErrorCode1 and ErrorCode 2 for ErrorDecode = PNIORW

Coding of ErrorCode1 and ErrorCode 2 for ErrorDecode = PNIO:

Value ErrCode1 (decimal)	Description	Value ErrCode2 (decimal)	Description / Use
0	Reserved	0 - 255	Reserved
1	Connect Parameter Error. Faulty ARBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		13	Error in Parameter NameOfStation
		14 - 255	Reserved
2	Connect Parameter Error. Faulty IOCRBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		28	Error in Parameter IOCSFrameOffset production
		29 - 255	Reserved
3	Connect Parameter Error. Faulty ExpectedSubmoduleBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		7	Error in Parameter SubmoduleLength production
		8 - 255	Reserved
4	Connect Parameter Error. Faulty AlarmCRBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		15	Error in Parameter AlarmCRTagHeaderLow
		16 - 255	Reserved
5	Connect Parameter Error. Faulty PrmServerBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		8	Error in Parameter ServerStationName
		9 - 255	Reserved
6	Connect Parameter Error. Faulty MCRBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		8	Error in Parameter ProviderStationName
		9 - 255	Reserved
7	Connect Parameter Error. Faulty ARRPCBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...

Value ErrCode1 (decimal)	Description	Value ErrCode2 (decimal)	Description / Use
		4	Error in Parameter InitiatorRPCServerPort
		5 - 255	Reserved
8	Read Write Record Parameter Error. Faulty Record	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		12	Error in Parameter TargetARUID
		13 - 255	Reserved
9 - 19	Reserved	0 - 255	Reserved
20	IODControl Parameter Error. Faulty ControlBlockConnect	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		7	Error in Parameter ControlBlockProperties
		8 - 255	Reserved
21	IODControl Parameter Error. Faulty ControlBlockPlug	0	Error in Parameter BlockType
		...	...
		7	Error in Parameter ControlBlockProperties
		8 - 255	Reserved
22	IOXControl Parameter Error. Faulty ControlBlock after a connection establishment	0	Error in Parameter BlockType
		...	...
		7	Error in Parameter ControlBlockProperties
		8 - 255	Reserved
23	IOXControl Parameter Error. Faulty ControlBlock a plug alarm	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		7	Error in Parameter ControlBlockProperties
		8 - 255	Reserved
24 - 39	Reserved	0 - 255	Reserved
40	Release Parameter Error. Faulty ReleaseBlock	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		7	Error in Parameter ControlBlockProperties
		8 - 255	Reserved
41 - 49	Reserved	0 - 255	Reserved
50	Response Parameter Error. Faulty ARBlockRes	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...

Value ErrCode1 (decimal)	Description	Value ErrCode2 (decimal)	Description / Use
		8	Error in Parameter ResponderUDPRTPort
		9 - 255	Reserved
51	Response Parameter Error. Faulty IOCRBlockRes	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		6	Error in Parameter FrameID
		7 - 255	Reserved
52	Response Parameter Error. Faulty AlarmCRBlockRes	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		6	Error in Parameter MaxAlarmDataLength
		7 - 255	Reserved
53	Response Parameter Error. Faulty ModuleDiffBlock	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		13	Error in Parameter SubmoduleState
		14 - 255	Reserved
54	Response Parameter Error. Faulty ARRPCBlockRes	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		4	Error in Parameter ResponderRPCServerPort
		5 - 255	Reserved
55 - 59	Reserved	0 - 255	Reserved
60	AlarmAck Error Codes	0	Alarm Type Not Supported
		1	Wrong Submodule State
		2 - 255	Reserved
61	CMDEV	0 - 255	Device CM error
62	CMCTL	0	State conflict
		1	Timeout
		2	No data send
		3 - 255	Reserved
63	NRPM	0	No DCP active
		1	DNS Unknown_RealStationName
		2	DCP No_RealStationName
		3	DCP Multiple_RealStationName
		4	DCP No_StationName
		5	No_IP_Addr
		6	DCP_Set_Error

Value ErrCode1 (decimal)	Description	Value ErrCode2 (decimal)	Description / Use
		7 - 255	Reserved
64	RMPM	0	ArgsLength invalid
		1	Unknown Blocks
		2	IOCR Missing
		3	Wrong AlarmCRBlock count
		4	Out of AR Resources
		5	AR UUID Unknown
		6	State conflict
		7 - 255	Reserved
65	ALPMI	0	Invalid state
		1	Wrong ACK-PDU
		2 - 255	Reserved
66	ALPMR	0	Invalid state
		1	Wrong Notification PDU
		2 - 255	Reserved
67	LMPM	0 - 255	Service handler error
68	MMAC	0 - 255	Mapping MAC Protocol Machine error
69	RPC	0 - 255	
70	APMR	0	Invalid state
		1	LMPM signaled an error
		2 - 255	Reserved
71	APMS	0	Invalid state
		1	LMPM signaled an error
		2	Timeout
		3 - 255	Reserved
72	CPM	0	Invalid state
		1 - 255	Reserved
73	PPM	0	Invalid state
		1 - 255	Reserved
74	Used by DCPUCS	0	Invalid state
		1	LMPM signaled an error
		2	Timeout
		3 - 255	Reserved
75	Used by DCPUCR	0	Invalid state
		1	LMPM signalled an error
		2 - 255	Reserved
76	Used by DCPMCS	0	Invalid state
		1	LMPM signalled an error
		2 - 255	Reserved
77	Used by DCPMCR	0	Invalid state

Value ErrCode1 (decimal)	Description	Value ErrCode2 (decimal)	Description / Use
		1	LMPM signalled an error
		2 - 255	Reserved
78	FSPM	0 - 255	FAL Service Protocol Machine error
79 - 252	Reserved	0 - 255	Reserved
253	Used by RTA for protocol error (RTA_ERR_CLS_PROTOCOL)	0	Reserved
		1	error within the coordination of sequence numbers (RTA_ERR_CODE_SEQ) error
		2	instance closed (RTA_ERR_ABORT)
		3	AR out of memory (RTA_ERR_ABORT)
		4	AR add provider or consumer failed (RTA_ERR_ABORT)
		5	AR consumer DHT / WDT expired (RTA_ERR_ABORT)
		6	AR cmi timeout (RTA_ERR_ABORT)
		7	AR alarm-open failed (RTA_ERR_ABORT)
		8	AR alarm-send.cnf(-) (RTA_ERR_ABORT)
		9	AR alarm-ack- send.cnf(-) (RTA_ERR_ABORT)
		10	AR alarm data too long (RTA_ERR_ABORT)
		11	AR alarm.ind(err) (RTA_ERR_ABORT)
		12	AR rpc-client call.cnf(-) (RTA_ERR_ABORT)
		13	AR abort.req (RTA_ERR_ABORT)
		14	AR re-run aborts existing (RTA_ERR_ABORT)
		15	AR release.ind received (RTA_ERR_ABORT)
		16	AR device deactivated (RTA_ERR_ABORT)
		17	AR removed (RTA_ERR_ABORT)
		18	AR protocol violation (RTA_ERR_ABORT)
		19	AR name resolution error (RTA_ERR_ABORT)
		20	AR RPC-Bind error (RTA_ERR_ABORT)
		21	AR RPC-Connect error (RTA_ERR_ABORT)

Value ErrCode1 (decimal)	Description	Value ErrCode2 (decimal)	Description / Use
		22	AR RPC-Read error (RTA_ERR_ABORT)
		23	AR RPC-Write error (RTA_ERR_ABORT)
		24	AR RPC-Control error (RTA_ERR_ABORT)
		25	AR forbidden pull or plug after check.rsp and before in- data.ind (RTA_ERR_ABORT)
		26	AR AP removed (RTA_ERR_ABORT)
		27	AR link down (RTA_ERR_ABORT)
		28	AR could not register multicast- mac address (RTA_ERR_ABORT)
		29	not synchronized (cannot start companion-ar) (RTA_ERR_ABORT)
		30	wrong topology (cannot start companion-ar) (RTA_ERR_ABORT)
		31	dcp, station-name changed (RTA_ERR_ABORT)
		32	dcp, reset to factory-settings (RTA_ERR_ABORT)
		33	cannot start companion-AR because a 0x8ipp submodule in the first AR... (RTA_ERR_ABORT)
		34	no irdata record yet (RTA_ERR_ABORT)
		35	PDEV (RTA_ERROR_ABORT)
		36	PDEV, no port offers required speed / duplicity (RTA_ERR_ABORT)
		37	IP-suite [of the IOC] changed by means of DCP set(IPPParameter) or local engineering (RTA_ERR_ABORT)
		38 - 200	Reserved.
		201 - 255	Manufacturer specific
254	Reserved	0 - 255	Reserved
255	User specific	0 - 254	User specific
		255	Recommended for "User abort" without further detail

Table 87: Coding of ErrorCode1 and ErrorCode 2 for ErrorDecode = PNIO



## 9.2 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 is recommended to not directly configure the submodule with the parameters but wait for Parameter End Indication.



### Note:

For this service the stack has implemented a timeout. If the application needs more than 2 seconds to handle the request the stack will automatically generate a negative response the IO-Controller.



### Note:

Since stack V3.4.11.0 the LOM target supports up to 32 kB of write record data payload.

### 9.2.1 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_HANDLE      hRecordHandle;
    TLR_HANDLE      hDeviceHandle;
    TLR_UINT32      ulSequenceNum;
    TLR_UINT32      ulApi;
    TLR_UINT32      ulSlot;
    TLR_UINT32      ulSubslot;
    TLR_UINT32      ulIndex;
    TLR_UINT32      ullLenToWrite;
    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
	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	HANDLE		A stack internal identifier which belongs to this indication.
	hDeviceHandle	HANDLE		The device handle. <u>The handle is described at chapter 0.</u>
	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. For DPM targets n may be up to 1024. For LOM target and stack versions from 3.4.11.0 on n may be up to 32768
	abRecordData[1024]	UINT8[]		The write record data. The actual length of the array depends on ulLenToWrite field.

Table 88: PNS\_IF\_WRITE\_RECORD\_IND\_T - Write Record Indication

## 9.2.2 Write Record Response

This packet has to be sent by the application to the stack in order to respond to a Write Record Indication.



### 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 9.14).

### Packet Structure Reference

```
typedef struct PNS_IF_WRITE_RECORD_RSP_DATA_Ttag
{
    TLR_HANDLE      hRecordHandle;
    TLR_HANDLE      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		Status: 0: indicates successful execution other: indicates an error (ulPnio also needs to be returned to the stack in this case in order to inform the stack about the error that has occurred) .
	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	HANDLE		A stack internal identifier which belongs to this indication.
	hDeviceHandle	HANDLE		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 <i>Table 83: Coding of the field ulPnio</i> .
	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.

Table 89: PNS\_IF\_WRITE\_RECORD\_RSP\_T - Write Record Response

## 9.3 AR Abort Indication service

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

- stack did not receive cyclic frames from IO-Controller
- IO-Controller closed the connection (RPC Release, RPC Abort, abort alarm)
- Application disallowed communication (Set Bus state OFF)
- Application reconfigured the stack using Channel Init or Configuration Reload

The following applies only if no Dual-Port Memory is used:



### Note:

If Callback function interface is used to exchange IO-Data with stack v3.3 it is recommended to call the UpdateConsumerData-function when this indication is received to allow the stack to invalidate the consumer IO-data.

From now on the stack will be waiting for establishment of a new connection (which must be initiated by the IO-Controller) if it is allowed to access the bus. The new connection will be indicated to application with the AR Check Service (see section 0).

### 9.3.1 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_HANDLE      hDeviceHandle;
    /* PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2 */
    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
	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	HANDLE		The device handle.  The handle is described at chapter 0. The handling of this indication is dependent on this handle.
	ulPnio	UINT32		PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See Table 83: Coding of the field <u>ulPnio</u> .

Table 90: PNS\_IF\_AR\_ABORT\_IND\_IND\_T - AR Abort Indication Indication

## 9.3.2 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_HANDLE 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	HANDLE		Handle to the IO-Device.

Table 91: PNS\_IF\_AR\_ABORT\_IND\_RSP\_T - AR Abort Indication Response

## 9.4 Save Station Name Service

With this service the stack indicates to the application that a DCP Set NameOfStation request was received. The new NameOfStation is automatically set by the stack. The application has to take care that this NameOfStation is stored permanently if requests element `bRemanent` is set and that the stack is configured with this new NameOfStation after the next PowerUp cycle.

**Note:**

If the stack is configured using a SYCON.net database then the stack will automatically change the NameOfStation stored in the database to the new NameOfStation if `bRemanent` is set.

### 9.4.1 Save Station Name Indication

This packet is sent by the stack to indicate the change of NameOfStation to application.

#### Packet Structure Reference

```
typedef struct PNS_IF_SAVE_STATION_NAME_IND_DATA_Ttag
{
    TLR_UINT16 usNameLen;
    TLR_UINT8  bRemanent;
    TLR_UINT8  abNameOfStation[PNIO_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 92: PNS\_IF\_SAVE\_STATION\_NAME\_IND\_T - Save Station Name Indication

## 9.4.2 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 93: PNS\_IF\_SAVE\_STATION\_NAME\_RSP\_T - Save Station Name Response

## 9.5 Save Station Type Service

With this service the stack indicates to the application that a DCP Set TypeOfStation request was received. The new TypeOfStation is automatically set by the stack. The application has to take care that this TypeOfStation is stored permanently if element `bRemanent` is set and that the stack is configured with this new TypeOfStation after the next PowerUp cycle.

### 9.5.1 Save Station Type Indication

This packet is sent by the stack to indicate the change of TypeOfStation to application.

#### Packet Structure Reference

```
typedef struct PNS_IF_SAVE_STATION_TYPE_IND_DATA_Ttag
{
    TLR_UINT16 usTypeLen;
    TLR_UINT8  bRemanent;
    TLR_UINT8  abTypeOfStation[PNIO_MAX_TYPE_OF_STATION];
} PNS_IF_SAVE_STATION_TYPE_IND_DATA_T;

typedef struct PNS_IF_SAVE_STATION_TYPE_IND_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_SAVE_STATION_TYPE_IND_DATA_T tData;
} PNS_IF_SAVE_STATION_TYPE_IND_T;
```

**Packet Description**

Structure PNS_IF_SAVE_STATION_TYPE_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	0x1F1C	PNS_IF_SAVE_STATION_TYPE_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_TYPE_IND_DATA_T			
	usTypeLen	UINT16	1..240	Length of the new TypeOfStation.
	bRemanent	UINT8	0 1	Do not save the new TypeOfStation remanent. Save the TypeOfStation remanent.
	abTypeOfStation[240]	UINT8		The new TypeOfStation as ASCII byte-array.

Table 94: PNS\_IF\_SAVE\_STATION\_TYPE\_IND\_T - Save Station Type Indication

## 9.5.2 Save Station Type Response

The application acknowledges the indication with this packet.

### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T PNS_IF_SAVE_STATION_TYPE_RSP_T;
```

### Packet Description

Structure PNS_IF_SAVE_STATION_TYPE_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	0x1F1D	PNS_IF_SAVE_STATION_TYPE_RES-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 95: PNS\_IF\_SAVE\_STATION\_TYPE\_RSP\_T - Save Station Type Response

## 9.6 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. If the flag `bRemanent` is set the application has to store the IP permanently and to configure the stack with this new IP after the next PowerUp cycle.



### 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:

If the stack is configured using a SYCON.net database then the stack will automatically change the IP parameters stored in the database to the new IP parameters if `bRemanent` is set.

### 9.6.1 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 96: PNS\_IF\_SAVE\_IP\_ADDRESS\_IND\_T - Save IP Address Indication

## 9.6.2 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_ACK-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 97: PNS\_IF\_SAVE\_IP\_ADDRESS\_RSP\_T - Save IP Address Response



## 9.7 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 frequency specified.

The following note only applies when the linkable Object Module is used.

**Note:**

If the stack is configured to use LEDs (see section Task Startup Parameters) and if a valid Blinking LED-Name (see section Task Startup Parameters) is given the stack will automatically blink with the LED and this indication is only informal.

### 9.7.1 Start LED Blinking Indication

With this indication packet the stack informed 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 98: PNS\_IF\_START\_LED\_BLINKING\_IND\_T - Start LED Blinking Indication

## 9.7.2 Start LED Blinking Response

The application shall return this response packet. If it is not able to start blinking with the LED 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 99: PNS\_IF\_START\_LED\_BLINKING\_RSP\_T - Start LED Blinking Response

## 9.8 Stop LED Blinking Service

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

The following note only applies when the linkable Object Module is used.



### Note:

If the stack is configured to use LEDs (see section Task Startup Parameters) and if a valid Blinking LED-Name (see section Task Startup Parameters) is given the stack will automatically stop blinking with the LED and this indication is only informal.

### 9.8.1 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 100: PNS\_IF\_STOP\_LED\_BLINKING\_IND\_T - Stop LED Blinking Indication

## 9.8.2 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 101: PNS\_IF\_STOP\_LED\_BLINKING\_RSP\_T - Stop LED Blinking Response

## 9.9 Reset Factory Settings Service

With this service the stack indicates to the application, that a Reset to Factory Settings request has been received. The application shall delete all permanently stored information like NameOfStation and IP and set them to the default values:

- NameOfStation to "" (empty)
- IP to 0.0.0.0
- I&M1 ... I&M3 record data shall be reset
- I&M4 shall be reset but handling may be application specific if a profile using I&M4 is used

Any remanent data indicated by the stack to the application using the Store Remanent Data Service shall also be deleted by the application and shall no longer be used.

### 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 application is informed about this with the Abort Indication service.



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.

### 9.9.1 Reset Factory Settings Indication

The indication packet sent by the stack.

#### Packet Structure Reference

```
typedef TLR_EMPTY_PACKET_T 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	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	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

Table 102: PNS\_IF\_RESET\_FACTORY\_SETTINGS\_IND\_T - Reset Factory Settings Indication

## 9.9.2 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 PNS_IF_RESET_FACTORY_SETTINGS_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	0x1F19	PNS_IF_RESET_FACTORY_SETTINGS_RES-Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 103. PNS\_IF\_RESET\_FACTORY\_SETTINGS\_RSP\_T - Reset Factory Settings Response



## 9.10 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.

### 9.10.1 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_HANDLE 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	HANDLE		The device handle
	ulStatus	UINT32		Status Byte, see <i>Table 105: Status Byte of PNS_IF_APDU_STATUS_CHANGED_IND_T</i> packet below

Table 104. PNS\_IF\_APDU\_STATUS\_CHANGED\_IND\_T - APDU Status Changed Indication

For the meaning of the status byte, see the following table for the APDU status of IO-Controller.

<b>ulstatus – Status Byte of PNS_IF_APDU_STATUS_CHANGED_IND_T packet</b>		
<b>Bit</b>	<b>Name</b>	<b>Meaning</b>
<b>D7</b>	Reserved 4	
<b>D6</b>	Reserved 3	
<b>D5</b>	Provider operation	1: The provider operates. 0: The provider has a problem.
<b>D4</b>	State of Provider station	1: Provider station is in Run state. 0: The provider station is in Stop state.
<b>D3</b>	Reserved 2	
<b>D2</b>	Data are valid / invalid	1: Valid 0: Invalid
<b>D1</b>	Reserved 1	
<b>D0</b>	Primary/ Backup	1: Primary 0: Backup

*Table 105: Status Byte of PNS\_IF\_APDU\_STATUS\_CHANGED\_IND\_T packet*

## 9.10.2 APDU Status Changed Response

The application shall return this packet.

### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_HANDLE 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	HANDLE		Handle to the IO-Device.

Table 106: PNS\_IF\_APDU\_STATUS\_CHANGED\_RSP\_T - APDU Status Changed Response

## 9.11 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.

### 9.11.1 Alarm Indication Indication

This packet informed 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_HANDLE    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	HANDLE		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.

Structure PNS_IF_ALARM_IND_T				
Type: Indication				
Area	Variable	Type	Value / Range	Description
	usAlarmDataLen	UINT16		The length of alarm data.
	abAlarmData[1024]	UINT8		Alarm data.

Table 107. PNS\_IF\_ALARM\_IND\_T - Alarm Indication Indication

## 9.11.2 Alarm Indication Response

The application shall return this packet.

### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_HANDLE 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	HANDLE		The device handle

Table 108. PNS\_IF\_ALARM\_RSP\_T - Alarm Indication Response

## 9.12 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.

### 9.12.1 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_HANDLE      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	HANDLE		The device handle
	usSessionKey	UINT16		The session key.

Table 109: PNS\_IF\_RELEASE\_REQ\_IND\_T - Release Request Indication

## 9.12.2 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_HANDLE 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	HANDLE		The device handle

Table 110. PNS\_IF\_RELEASE\_REQ\_RSP\_T - Release Request Indication Response

## 9.13 Link Status Changed Service

With this service the stack informed 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 point in time when a new Link Status Changed Indication is received.



---

**Note:**

After Power ON the application may receive up to 4 indications in a short time.

---

### 9.13.1 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 ... $2^{32}-1$	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 indication.
	ulCmd	UINT32	0x1F70	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 port.

Table 111. 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 112. Structure PNS\_IF\_LINK\_STATUS\_DATA\_T

## 9.13.2 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	0x1F71	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 113. PNS\_IF\_LINK\_STATUS\_CHANGED\_RSP\_T - Link Status Changed Response

## 9.14 Error Indication Service

With this service the stack informs the user application about an error. 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 that command has nothing to do with the error reported in `ulErrorCode`.

---

### 9.14.1 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 PNS_IF_USER_ERROR_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 indication.
	ulCmd	UINT32	0x1FDC	PNS_IF_USER_ERROR_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_USER_ERROR_IND_DATA_T			
	ulErrorCode	UINT32		The error code.
	ulCommand	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 114. PNS\_IF\_USER\_ERROR\_IND\_T - Error Indication Service



## 9.14.2 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 115: PNS\_IF\_USER\_ERROR\_RSP\_T - Error Indication Response

## 9.15 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 same I&M0-4 data when reading the I&M0-4 data from the new location.



### Note:

This service is available since stack version V3.4.12.0. Furthermore the service will only be used if the stack was configured to not to handle I&M requests by its own.



### Note:

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



### Note:

I&M Data may be read using a Device Access Application Relation. The Host Application should be designed to deal properly with additional Application Relations

### 9.15.1 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 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.

structure PNS_IF_READ_IM_IND_T				
Type: Indication				
Area	Variable	Type	Value / Range	Description
	ulLen	UINT32	12	Packet data length in bytes. n depends on the parameter type contained in the packet.
	ulId	UINT32	0 ... 2 <sup>32</sup> -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
	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: I&M0-4 I&M0 FilterData
	abReserved[3]	UINT8		Reserved for future usage / Padding

Table 116: PNS\_IF\_READ\_IM\_IND\_T – Read I&amp;M Indication

## 9.15.2 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
tData	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 117: PNS\_IF\_READ\_IM\_RES\_T – Read I&amp;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
	abManufacture rSpecific	UINT8[10]	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
	usManufacture rId	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))
	abSerialNumbe r	UINT8[16]		The Serial number of the (sub)module. (padded with spaces)
	usHardwareRev ision	UINT16		Hardware revision of the (sub)module
	tSoftwareRevi sion.bPrefix	UINT8		Character describing the software of the (sub)module. Allowed values: 'V', 'R', 'P', 'U' and 'T'
	tSoftwareRevi sion.bX	UINT8		Function enhancement. (Major version number) of the (sub)module.
	tSoftwareRevi sion.bY	UINT8		Bug fix (Minor version number) of the (sub)module
	tSoftwareRevi sion.bZ	UINT8		Internal Change (Build version number) of the (sub)module
	usRevisionCou nter	UINT16		Starting from 0, shall increment on each parameter change.
	usProfileId	UINT16		The profile of the (sub)module.
	usProfileSpec ificType	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 118: 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
	abManufacture rSpecific[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 119: PNS\_IF\_IM1\_DATA\_T – Structure of I&amp;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
	abManufacture rSpecific[10]	UINT8	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
	abInstallatio nDate[16]	UINT8		Installation Date of the (sub)module. (padded with spaces)
	abReserved[28 ]	UINT8		Reserved. Set to zero. Not evaluated by stack.

Table 120: PNS\_IF\_IM2\_DATA\_T – Structure of I&amp;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
	abManufacture rSpecific[10]	UINT8	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
	abDescriptor[ 54]	UINT8		Description text of the (sub)module. (padded with spaces)

Table 121: PNS\_IF\_IM3\_DATA\_T – Structure of I&amp;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
	abManufacture rSpecific[10]	UINT8	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
	abSignature[5 4]	UINT8		Signature generated by engineering system. (padded with spaces, default value: <b>ZERO</b> )

Table 122: PNS\_IF\_IM4\_DATA\_T – Structure of I&amp;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 needs 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 123: PNS\_IF\_IM0\_FILTER\_DATA\_T – Structure of I&amp;M0 Filter Information

## 9.16 Write I&M Service

The stack uses this service to make 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:**

This service is available since stack version V3.4.12.0. Furthermore the service will only be used if the stack was configured to not to handle I&M requests by its own.



**Note:**

Writing of I&M is only defined for I&M1-4 records. Therefore write support is optional. If write is not supported, the user application shall respond the write indication with error code TLR\_E\_FAIL



**Note:**

I&M Data may be written using a Device Access Application Relation. The Host Application should be designed to deal properly with additional Application Relations

### 9.16.1 Write I&M Indication

Send 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.
	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 9.15.2.

Table 124: 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:**

Its 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.

## 9.16.2 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 125: PNS\_IF\_WRITE\_IM\_RES\_T – Write I&M Response

## 9.17 Parameterization Speedup Support

This service is used to indicate to the user application if Parameterization Speedup is enabled. The Parameterization speedup UUID received by the stack in ARFSUDataAdjust (0xE050) record will be delivered to the application. If Parameterization speedup is not (or no longer) activated the stack indicates this with a NIL-UUID (all values are 0).



### Note:

This service is available since stack version V3.4.13.0.



### 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:

Applications not using FSU or the feature Parameterization speedup may ignore the content of this indication but still have to return the response to the stack.

### 9.17.1 Parameterization Speedup Support Indication

Send by the stack whenever a write indication to ARFSUDataAdjust has been received. To receive a non-NIL UUID (indicating Parameterization speedup enabled) FSU must also be configured and the GSDML must contain the keyword "ParameterizationSpeedupSupported".

#### Packet Structure Reference

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

typedef __PACKED_PRE struct PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_DATA_Ttag
{
    FSUUUID_T tFSUuuid;
} __PACKED_POST 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 PNS_IF_PARAMET_SPEEDUP_SUPPORTED_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	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		Status not used for request.
	ulCmd	UINT32	0x1FF8	PNS_IF_PARAMET_SPEEDUP_SUPPORTED_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_PARAMET_SPEEDUP_SUPPORTED_IND_DATA_T			
	tFSUUid	FSUUUID_T	0-0xFFFFFFFF	The UUID send to the application. TLR_UINT32 ulData1 TLR_UINT16 usData2 TLR_UINT16 usData3 TLR_UINT8 abData4[8]

Table 126: PNS\_IF\_PARAMET\_SPEEDUP\_SUPPORTED\_IND\_T – Parameterization Speedup Supported Indication

## 9.17.2 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 success or failure of the write.

### 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_RES - Command
	ulExt	UINT32	0	Extension, untouched
	ulRout	UINT32	x	Routing, do not touch

Table 127: PNS\_IF\_PARAMET\_SPEEDUP\_SUPPORTED\_RES\_T- Parameterization Speedup Supported Response

## 10 Acyclic Events requested by the Application

All acyclic events and requests the application can send to the stack are described in this section.

The three alarm services (see sections 10.2, 10.4 and 10.5) are only available while cyclic data exchange is established.

All other services may be used by the application at any time after the stack has been configured.

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

Overview over the Packets of the PROFINET IO Device IRT Stack for Acyclic Events requested by the Application			
No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
10.1	Get Diagnosis Request	0x1FB2	224
	Get Diagnosis Confirmation	0x1FB3	225
10.2	Get XMAC (EDD) Diagnosis Request	0x1FE4	229
	Get XMAC (EDD) Diagnosis Confirmation	0x1FE5	230
10.3	Process Alarm Request	0x1F52	233
	Process Alarm Confirmation	0x1F53	235
10.4	Diagnosis Alarm Request	0x1F4C	237
	Diagnosis Alarm Confirmation	0x1F4D	239
10.5	Return of Submodule Alarm Request	0x1F50	241
	Return of Submodule Alarm Confirmation	0x1F51	243
10.6	AR Abort Request Service	0x1FD8	244
	AR Abort Request Confirmation	0x1FD9	246
10.7	Plug Module Request	0x1F04	247
	Plug Module Confirmation	0x1F05	249
10.8	Extended Plug Submodule Request	0x1F08	254
	Extended Plug Submodule Confirmation	0x1F09	257
10.9	Pull Module Request	0x1F06	260
	Pull Module Confirmation	0x1F07	262
10.10	Pull Submodule Request	0x1F0A	264
	Pull Submodule Confirmation	0x1F0B	266
10.11	Get Station Name Request	0x1F8E	272
	Get Station Name Confirmation	0x1F8F	273
10.13	Get Station Type Request	0x1F8C	274
	Get Station Type Confirmation	0x1F8D	275
10.14	Get IP Address Request	0x1FBC	276

Overview over the Packets of the PROFINET IO Device IRT Stack for Acyclic Events requested by the Application			
No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
	Get IP Address Confirmation	0x1FBD	277
10.15	Add Channel Diagnosis Request	0x1F46	278
	Add Channel Diagnosis Confirmation	0x1F47	283
10.16	Add Extended Channel Diagnosis Request	0x1F54	285
	Add Extended Channel Diagnosis Confirmation	0x1F55	287
10.17	Add Generic Channel Diagnosis Request	0x1F58	289
	Add Generic Channel Diagnosis Confirmation	0x1F59	291
10.18	Remove Diagnosis Request	0x1FE6	293
	Remove Diagnosis Confirmation	0x1FE7	294

Table 128: Overview over the Packets of the PROFINET IO Device IRT Stack for Acyclic Events requested by the Application

## 10.1 Get Diagnosis Service

With this service the user application can request a collection of diagnostic data concerning the PROFINET IO RT/IRT Device V3 protocol stack.



### Note:

In this service the confirmation packet is larger than the request packet. If the application is running on the netX processor and DPM is not used the application has to provide a buffer which is large enough.

### 10.1.1 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 129: PNS\_IF\_GET\_DIAGNOSIS\_REQ\_T - Get Diagnosis Request



## 10.1.2 Get Diagnosis Confirmation

With this packet the stack informs the application about the collected diagnosis data.

### 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;
} 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 PNS_IF_GET_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	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 PROFINET IO RT/IRT Device V3 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.

Table 130: 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	FiberOptic Maintenance Required Record exists for Port 1 <b>Note:</b> Only valid in case of fiber optic hardware.
D15	FiberOptic Maintenance Demanded Record exists for Port 1 <b>Note:</b> Only valid in case of fiber optic hardware.
D14	FiberOptic Maintenance Required Record exists for Port 0 <b>Note:</b> Only valid in case of fiber optic hardware.
D13	FiberOptic 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 131: Meaning of single Bits in ulPnsState

### ■ ulLastRslt

This parameter denotes the error code of the last encountered error of the RCX/API Task, see the TLR error codes documented in section 11 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 132: Values and their corresponding Meanings of ulLinkState

### ■ ulConfigState

This parameter denotes the configuration state. It may have the following values which are listed here along with their meanings:

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 133: 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*.

## 10.2 Get XMAC (EDD) Diagnosis Service

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



### Note:

In this service the confirmation packet is bigger 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.

### 10.2.1 Get XMAC (EDD) Diagnosis Request

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

#### Packet Structure Reference

```
/*request packet */
typedef TLR_EMPTY_PACKET_T                                PNS_IF_GET_XMAC_DIAGNOSIS_REQ_T;
```

#### Packet Description

structure PNS_IF_GET_XMAC_DIAGNOSIS_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		Source Queue-Handle
	ulDestId	UINT32	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	PNS_IF_GET_XMAC_DIAGNOSIS_REQ
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 134: PNS\_IF\_GET\_XMAC\_DIAGNOSIS\_REQ\_T - Get XMAC (EDD) Diagnosis Request

## 10.2.2 Get XMAC (EDD) Diagnosis Confirmation

This packet confirms the Get XMAC (EDD) diagnosis Request and delivers the requested data within structure EDD\_XMAC\_COUNTERS\_T, see Table 136 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_X MAC_C OUNTE RS_T		Array containing statistics stored in XMAC counters

Table 135: 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 136: Structure `EDD_XMAC_COUNTERS_T`



## 10.3 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.

### 10.3.1 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_HANDLE    hDeviceHandle;
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_HANDLE    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			
	hDeviceHandle	HANDLE		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.
	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 137. PNS\_IF\_SEND\_PROCESS\_ALARM\_REQ\_T - Process Alarm Request

## 10.3.2 Process 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.

### Packet Structure Reference

```
typedef struct PNS_IF_SEND_PROCESS_ALARM_CNF_DATA_Ttag
{
    TLR_HANDLE      hDeviceHandle;
    TLR_HANDLE      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	HANDLE		The device handle
	hAlarmHandle	HANDLE		The user specific alarm handle.
	ulPnio	UINT32		PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See Table 83: Coding of the field ulPnio.

Table 138: PNS\_IF\_SEND\_PROCESS\_ALARM\_CNF\_T - Process Alarm Confirmation

## 10.4 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 10.15, 10.16, 10.17). 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 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.

### 10.4.1 Diagnosis Alarm Request

This request packet must be used by application to force the stack to send a diagnosis alarm.

#### Packet Structure Reference

```
typedef struct PNS_IF_SEND_DIAG_ALARM_REQ_DATA_Ttag
{
    TLR_HANDLE    hDeviceHandle;
    TLR_HANDLE    hAlarmHandle;
    TLR_HANDLE    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.
	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			
	hDeviceHandle	HANDLE		The device handle
	hAlarmHandle	HANDLE		A user specific alarm handle. The application is free to choose any value.
	hDiagHandle	HANDLE		The handle to the diagnosis record the diagnosis alarm shall be sent for.

Table 139: PNS\_IF\_SEND\_DIAG\_ALARM\_REQ\_T - Diagnosis Alarm Request

## 10.4.2 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_HANDLE    hDeviceHandle;
    TLR_HANDLE    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	HANDLE		The device handle
	hAlarmHandle	HANDLE		The user specific alarm handle.
	ulPnio	UINT32		PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See Table 83: Coding of the field ulPnio.

Table 140: PNS\_IF\_DIAG\_ALARM\_CNF\_T - Diagnosis Alarm Confirmation



## 10.5 Return of Submodule Alarm Service

The application has to use this service whenever the IOPS of a submodule changes from BAD to GOOD. This service makes the stack send a Return of Submodule alarm to the IO-Controller.

Return of Submodule 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 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.

### 10.5.1 Return of Submodule Alarm Request

This packet has to be sent to the stack to send a Return of Submodule alarm.

#### Packet Structure Reference

```
typedef struct PNS_IF_RETURN_OF_SUB_ALARM_REQ_DATA_Ttag
{
    TLR_HANDLE    hDeviceHandle;
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_HANDLE    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.
	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			
	hDeviceHandle	HANDLE		The device handle
	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 141: PNS\_IF\_RETURN\_OF\_SUB\_ALARM\_REQ\_T - Return of Submodule Alarm Request

## 10.5.2 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_HANDLE hDeviceHandle;
    TLR_HANDLE 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 <sup>32</sup> -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 <sup>32</sup> -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	HANDLE		The device handle
	hAlarmHandle	HANDLE		The user specific alarm handle.
	ulPnio	UINT32		PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See Table 83: Coding of the field <u>ulPnio</u> .

Table 142: PNS\_IF\_RETURN\_OF\_SUB\_ALARM\_CNF\_T - Return of Submodule Alarm Confirmation

## 10.6 AR Abort Request Service

With this service the application forces the stack to abort the established connection to an IO-Controller.

### 10.6.1 AR Abort Request Request

This packet has to be sent by application to force the stack to abort an established connection.

#### Packet Structure Reference

```
typedef struct PNS_IF_HANDLE_DATA_Ttag
{
    TLR_HANDLE 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	HANDLE		The device handle

Table 143. PNS\_IF\_ABORT\_CONNECTION\_REQ\_T - AR Abort Request Request

## 10.6.2 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_HANDLE 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	HANDLE		Handle to the IO-Device.

Table 144: PNS\_IF\_ABORT\_CONNECTION\_CNF\_T - AR Abort Request Confirmation

## 10.7 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.

### 10.7.1 Plug Module Request

Receiving this packet forces the stack to automatically send a Plug-alarm to the IO-Controller if a connection is established.

#### Packet Structure Reference

```
typedef struct PNS_IF_PLUG_MODULE_REQ_DATA_Ttag
{
    TLR_HANDLE 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;
```

**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			
	hDeviceHandle	HANDLE		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	Module state. See below.

Table 145: PNS\_IF\_PLUG\_MODULE\_REQ\_T - Plug Module Request

Possible values for usModState:

```
#define PNS_IF_PLUG_CORRECT_MODULE 0
```

This value shall be used if the module is not a substitute module.

```
#define PNS_IF_PLUG_SUBSTITUTE_MODULE 1
```

This value shall be used if the module is a substitute module.



## 10.7.2 Plug Module Confirmation

The stack will return this packet to application.

### Packet Structure Reference

```
typedef struct PNS_IF_PLUG_MODULE_REQ_DATA_Ttag
{
    TLR_HANDLE 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	HANDLE		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 146: PNS\_IF\_PLUG\_MODULE\_CNF\_T - Plug Module Confirmation

## 10.8 Plug Submodule Service

With this service the application can plug additional submodules after the `Set_Configuration Req` packet (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.

Plugging a submodule may force the IO-Controller to send additional parameters via Write Record Service to the application. If so this is finished by the IO-Controller by sending another Parameter End to the IO-Device. This will be indicated to the application, too.



### Note:

Since stack version 3.4 an extended plug submodule request is available allowing the user to plug module and submodule using one single request packet.

Receiving this packet forces the stack to automatically send a Plug-alarm to the IO-Controller if a connection is established and the submodule plugged is contained in the expected configuration of the IO-Controller.

### Packet Structure Reference

```
/* Request packet */
typedef struct PNS_IF_PLUG_SUBMODULE_REQ_DATA_Ttag
{
    TLR_HANDLE hDeviceHandle;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulSubslot;
    TLR_UINT32 ulSubmodId;
    /* ProvData is data sent by IO-Device and received by IO-Controller */
    TLR_UINT32 ulProvDataLen;
    /* ConsData is data sent by IO-Controller and received by IO-Device */
    TLR_UINT32 ulConsDataLen;
    /* offset in DPM where consumer data is copied to */
    TLR_UINT32 ulDPMOffsetCons;
    /* offset in DPM where provider data is taken from */
    TLR_UINT32 ulDPMOffsetProv;
    /* offset where to put IOPS provider state for this submodule relative */
    /* to beginning of IOPS block in dpm output area to */
    TLR_UINT16 usOffsetIOPSProvider;
    /* offset where to take IOPS provider state of this submodule relative */
    /* to beginning of IOPS block in dpm input area from */
    TLR_UINT16 usOffsetIOPSConsumer;
    /* reserved */
    TLR_UINT32 ulReserved1;
    /* reserved for future use - maybe needed for DPM Area later */
    TLR_UINT32 ulReserved;
    TLR_UINT16 usSubmodState; /* submodule state: 0 = correct submodule, 1 = substitute
submodule */
} PNS_IF_PLUG_SUBMODULE_REQ_DATA_T;

typedef struct PNS_IF_PLUG_SUBMODULE_REQ_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T tHead;
    /** packet data */
    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			
	hDeviceHandle	HANDLE		The device handle
	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.
	ulDPMOffsetConsumer	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.
	ulDPMOffsetProvider	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
	usOffsetIOPSCConsumer	UINT16		Offset where to take IO provider state of this submodule relative to beginning of IOPS block in dpm input area from
	ulReserved1	UINT32	0	Reserved.

Structure PNS_IF_PLUG_SUBMODULE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
	ulReserved	UINT32	0	Reserved for future use. Set to zero.
	usSubmodState	UINT16	0..1	The submodule state. See below.

Table 147: PNS\_IF\_PLUG\_SUBMODULE\_REQ\_T - Plug Submodule Request for stack v3.3

Possible values for usSubmodState:

```
#define      PNS_IF_PLUG_CORRECT_SUBMODULE
                0
```

This value shall be used if the submodule is not a substitute submodule.

```
#define      PNS_IF_PLUG_SUBSTITUTE_SUBMODULE
                1
```

This value shall be used if the submodule is a substitute submodule.

## Packet Structure Reference

```
/* Confirmation packet */
typedef struct PNS_IF_PLUG_SUBMODULE_REQ_DATA_Ttag
{
    TLR_HANDLE hDeviceHandle;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulSubslot;
    TLR_UINT32 ulSubmodId;
    /* ProvData is data sent by IO-Device and received by IO-Controller */
    TLR_UINT32 ulProvDataLen;
    /* ConsData is data sent by IO-Controller and received by IO-Device */
    TLR_UINT32 ulConsDataLen;
    /* offset in DPM where consumer data is copied to */
    TLR_UINT32 ulDPMOffsetCons;
    /* offset in DPM where provider data is taken from */
    TLR_UINT32 ulDPMOffsetProv;
    /* offset of IOxS status of submodule relative to beginning
       of IOxS block in dpm */
    TLR_UINT32 ulDPMOffsetIoxs;
    /* Reserved */
    TLR_UINT32 ulReserved1;
    /* reserved for future use - maybe needed for DPM Area later */
    TLR_UINT32 ulReserved;
    /* submodule state: 0 = correct submodule, 1 = substitute submodule */
    TLR_UINT16 usSubmodState;
} PNS_IF_PLUG_SUBMODULE_REQ_DATA_T;

typedef struct PNS_IF_PLUG_SUBMODULE_REQ_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    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	HANDLE		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.
	ulDPMOffsetIn	UINT32		Offset in DPM where Input data for the submodule shall be copied to.
	ulDPMOffsetOut	UINT32		Offset in DPM where Output data for the submodule shall be taken from.
	ulDPMOffsetIoCsIn	UINT32		Offset in DPM where IOCS is copied to.
	ulDPMOffsetIoCsOut	UINT32		Offset in DPM where IOCS is taken from.
	ulReserved	UINT32	0	Reserved for future use. Set to zero.
	usSubmodState	UINT16	0..1	The submodule state.

Table 148: PNS\_IF\_PLUG\_SUBMODULE\_CNF\_T - Plug Submodule Confirmation of stack v3.3

## 10.8.1 Extended Plug Submodule Request

Receiving this packet the stack adds the described submodule and module to its internal configuration to use it for data exchange. The module will be added if necessary only. 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.



### Note:

This request is available for stack version 3.4.0 and above only.

### Packet Structure Reference

```
/* Extended plug submodule data structure (since stack versions V3.4) */
typedef struct PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_DATA_Ttag
{
    TLR_HANDLE hDeviceHandle;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulSubslot;
    TLR_UINT32 ulSubmodId;
    /* provider data is data sent by IO-Device and received by IO-Controller */
    TLR_UINT32 ulProvDataLen;
    /* consumer data is data sent by IO-Controller and received by IO-Device */
    TLR_UINT32 ulConsDataLen;
    /* offset in DPM where Output data (consumed by IO-Device from IO-Controller) is copied
to */
    TLR_UINT32 ulDPMOffsetCons;
    /* offset in DPM where Input data (provided by IO-Device to IO-Controller) is taken
from */
    TLR_UINT32 ulDPMOffsetProv;
    /* offset where to put IOPS provider state for this submodule relative to beginning of
IOPS block in dpm output area to */
    TLR_UINT16 usOffsetIOPSPROvider;
    /* offset where to take IOPS provider state of this submodule relative to beginning of
IOPS block in dpm input area from */
    TLR_UINT16 usOffsetIOPSCONSUMER;
    /* reserved for future usage */
    TLR_UINT32 ulReserved1;
    /* reserved for future use - maybe needed for DPM Area later */
    TLR_UINT32 ulReserved;
    /* submodule state: 0 = correct submodule, 1 = substitute submodule */
    TLR_UINT16 usSubmodState;
    /* Module identifier (new since V3.4 - Submodules can now be plugged without prior
plugging the modules)*/
    TLR_UINT32 ulModuleId;
    /* Module state (new since V3.4 - Submodules can now be plugged without prior plugging
the modules)*/
    TLR_UINT16 usModuleState;
} PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_DATA_T;

typedef struct PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T tHead;
    /** packet data */
    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: 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)
	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			
	hDeviceHandle	HANDLE		The device handle. Unused since V3.4. 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.
	usOffsetIOPSPProvider	UINT16		Offset where to put IO provider state for this submodule relative to beginning of IOPS block in dpm output area to
	usOffsetIOPSCConsumer	UINT16		Offset where to take IO provider state of this submodule relative to beginning of IOPS block in dpm input area from
	ulReserved1	UINT32	0	Reserved.
	ulReserved	UINT32	0	Reserved for future use. Set to zero.

Structure PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
	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 10.7.1

Table 149: PNS\_IF\_PLUG\_SUBMODULE\_EXTENDED\_REQ\_T – Extended Plug Submodule Request

Possible values for usSubmodState:

```
#define      PNS_IF_PLUG_CORRECT_SUBMODULE                                0
```

This value shall be used if the submodule is not a substitute submodule.

```
#define      PNS_IF_PLUG_SUBSTITUTE_SUBMODULE                            1
```

This value shall be used if the submodule is a substitute submodule.



## 10.8.2 Extended Plug Submodule Confirmation

Confirmation of extended plug submodule request.

### Packet Structure Reference

```

/* Extended plug submodule data structure (since stack versions V3.4) */
typedef struct PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_DATA_Ttag
{
    TLR_HANDLE hDeviceHandle;
    TLR_UINT32 ulApi;
    TLR_UINT32 ulSlot;
    TLR_UINT32 ulSubslot;
    TLR_UINT32 ulSubmodId;
    /* provider data is data sent by IO-Device and received by IO-Controller */
    TLR_UINT32 ulProvDataLen;
    /* consumer data is data sent by IO-Controller and received by IO-Device */
    TLR_UINT32 ulConsDataLen;
    /* offset in DPM where Output data (consumed by IO-Device from IO-Controller) is copied
to */
    TLR_UINT32 ulDPMOffsetCons;
    /* offset in DPM where Input data (provided by IO-Device to IO-Controller) is taken
from */
    TLR_UINT32 ulDPMOffsetProv;
    /* offset where to put IOPS provider state for this submodule relative to beginning of
IOPS block in dpm output area to */
    TLR_UINT16 usOffsetIOPSPROVIDER;
    /* offset where to take IOPS provider state of this submodule relative to beginning of
IOPS block in dpm input area from */
    TLR_UINT16 usOffsetIOPSConsumer;
    /* reserved for future usage */
    TLR_UINT32 ulReserved1;
    /* reserved for future use - maybe needed for DPM Area later */
    TLR_UINT32 ulReserved;
    /* submodule state: 0 = correct submodule, 1 = substitute submodule */
    TLR_UINT16 usSubmodState;
    /* Module identifier (new since V3.4 - Submodules can now be plugged without prior
plugging the modules)*/
    TLR_UINT32 ulModuleId;
    /* Module state (new since V3.4 - Submodules can now be plugged without prior plugging
the modules)*/
    TLR_UINT16 usModuleState;
} PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_DATA_T;

typedef struct PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_Ttag
{
    /** packet header */
    TLR_PACKET_HEADER_T          tHead;
    /** packet data */
    PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_DATA_T tData;
} PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_T;

typedef PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_T  PNS_IF_PLUG_SUBMODULE_EXTENDED_CNF_T;

```

**Packet Description**

Structure PNS_IF_PLUG_SUBMODULE_EXTENDED_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. 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_REQ_DATA_T			
	hDeviceHandle	HANDLE		The device handle. Unused since V3.4.
	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.
	usOffsetIOPSPProvider	UINT16		Offset relative to beginning of IOPS block in dpm output area. where the IO provider state for this submodule will be put to.
	usOffsetIOPSCConsumer	UINT16		Offset relative to beginning of IOPS block in dpm input area where the IO provider state of this submodule will be taken from.
	ulReserved1	UINT32	0	Reserved. Ignore.
	ulReserved	UINT32	0	Reserved for future use. Ignore
	usSubmodState	UINT16	0..1	The submodule state
	ulModuleId	UINT32	1..0xFFFFFFFF	The module identifier.

Structure PNS_IF_PLUG_SUBMODULE_EXTENDED_CNF_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
	usModuleState	UINT16	0..1	The module state

*Table 150: PNS\_IF\_PLUG\_SUBMODULE\_EXTENDED\_CNF\_T – Extended Plug Submodule Confirmation*

## 10.9 Pull Module Service

With this service the application can pull modules from the current configuration.

Pulling a module automatically pulls all submodules which belong to the module, too.

If a module is pulled while cyclic data exchange is running and a submodule of the module belongs to the ApplicationRelation an alarm is generated and sent to IO-Controller.

It is recommended to wait for receipt of the Pull Module confirmation before issuing a new Pull Module Request.

It is required to wait for receipt of the Pull Module confirmation before issuing a Plug Module Request or Plug Submodule Request.

### 10.9.1 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_HANDLE hDeviceHandle;
    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			
	hDeviceHandle	HANDLE		The device handle
	ulApi	UINT32		The API of the module.
	ulSlot	UINT32		The Slot to pull the module from.

Table 151: PNS\_IF\_PULL\_MODULE\_REQ\_T - Pull Module Request

## 10.9.2 Pull Module Confirmation

The stack will return this packet to application.

### Packet Structure Reference

```
/* Confirmation packet */
typedef struct PNS_IF_PULL_MODULE_REQ_DATA_Ttag
{
    TLR_HANDLE hDeviceHandle;
    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			
	hDeviceHandle	HANDLE		The device handle
	ulApi	UINT32		The API of the module.
	ulSlot	UINT32		The Slot to pull the module from.

Table 152: PNS\_IF\_PULL\_MODULE\_CNF\_T – Pull Module Confirmation

## 10.10 Pull Submodule Service

With this service the application can pull submodules.

**Note:**

Pulling a submodule forces the application to set the IOPS and IOCS of the submodule to BAD.

### 10.10.1 Pull Submodule 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

```
/* Request packet */
typedef struct PNS_IF_PULL_SUBMODULE_REQ_DATA_Ttag
{
    TLR_HANDLE 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	HANDLE		The device handle
	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 153: PNS\_IF\_PULL\_SUBMODULE\_REQ-T - Pull Submodule Request

## 10.10.2 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_HANDLE 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;

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			
	hDeviceHandle	HANDLE		The device handle
	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 154: PNS\_IF\_PULL\_SUBMODULE\_CNF-T - Pull Submodule Confirmation

## 10.11 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 V3.4.36.1

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 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 need to be set to “bad”. Afterwards the ApplicationReady shall be sent by application using 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.

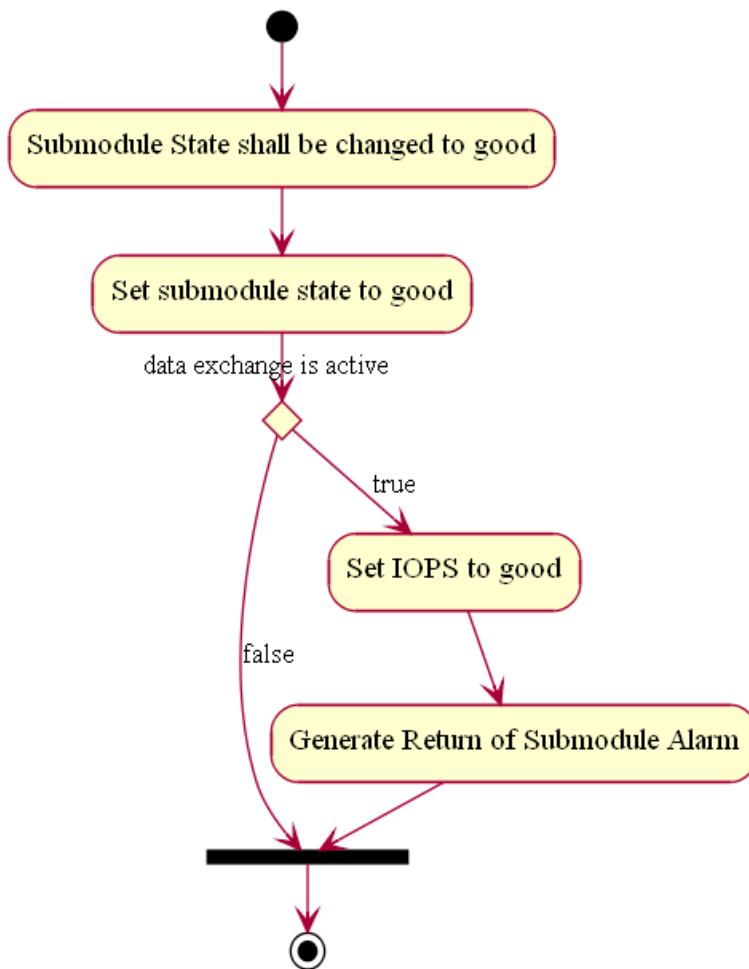


Figure 10: SetSubModuleState from bad to good

### 10.11.1 Set Submodule State Request

To set the Submodule State the following packet shall be used.

Only two states are supported: “ApplicationReady pending” 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”.



#### 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 10.5.1) 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 155: 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 156: Possible Values of usSubmState

## 10.11.2 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 157: PNS\_IF\_SET\_SUBM\_STATE\_CNF-T - Set Submodule State Confirmation

## 10.12 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.

For the station name, only small characters are allowed.

### 10.12.1 Get Station Name Request

The request packet the application has to send 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 158: PNS\_IF\_GET\_STATION\_NAME\_REQ-T - Get Station Name Request



## 10.12.2 Get Station Name Confirmation

The 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 159: PNS\_IF\_GET\_STATION\_NAME\_CNF-T - Get Station Name Confirmation

## 10.13 Get Station Type Service

With this service the application can request the current TypeOfStation 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.

### 10.13.1 Get Station Type Request

The request packet the application has to send to the stack.

#### Packet Structure Reference

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

#### Packet Description

Structure PNS_IF_GET_STATION_TYPE_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	0x1F8C	PNS_IF_GET_STATION_TYPE_REQ-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 160: PNS\_IF\_GET\_STATION\_TYPE\_REQ-T - Get Station Type Request

## 10.13.2 Get Station Type Confirmation

The confirmation packet containing the TypeOfStation will be returned.

### Packet Structure Reference

```
/* Confirmation packet */
typedef struct PNS_IF_GET_STATION_TYPE_CNF_DATA_Ttag
{
    TLR_UINT16 usTypeLen;
    TLR_UINT8  abTypeOfStation[PONIO_MAX_TYPE_OF_STATION];
} PNS_IF_GET_STATION_TYPE_CNF_DATA_T;

typedef struct PNS_IF_GET_STATION_TYPE_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    PNS_IF_GET_STATION_TYPE_CNF_DATA_T  tData;
} PNS_IF_GET_STATION_TYPE_CNF_T;
```

### Packet Description

Structure PNS_IF_GET_STATION_TYPE_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	0x1F8D	PNS_IF_GET_STATION_TYPE_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_TYPE_CNF_DATA_T			
	usTypeLen	UINT16	1..240	Length of the new TypeOfStation.
	abTypeOfStation[240]	UINT8		The current TypeOfStation as ASCII byte-array.

Table 161: PNS\_IF\_GET\_STATION\_TYPE\_CNF-T - Get Station Type Confirmation

## 10.14 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.

### 10.14.1 Get IP Address Request

The request packet the application has to send 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 162: PNS\_IF\_GET\_IP\_ADDR\_REQ-T - Get IP Address Request

## 10.14.2 Get IP Address Confirmation

The 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_C-F -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 163: PNS\_IF\_GET\_IP\_ADDR\_CNF-T - Get IP Address Confirmation

## 10.15 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 10.4).

---

### 10.15.1 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_HANDLE  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_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 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
	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	HANDLE	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 166: Coding of the field usChannelProp
	usChannelErrType	UINT16		See Table 165: Coding of usChannelErrType

Table 164: 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”
0x001A	Manufacturer specific, recommended for “external fault”
0x00-B – 0x001E	Manufacturer specific
0x001F	Manufacturer specific, recommended for “temporary fault”
0x00-0 – 0x00FF	Reserved for common profiles (assigned by PROFIBUS International)
0x01-0 – 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
0x80-A – 0x8FFF	Reserved



Value (Hexadecimal)	Description
0x90-0 - 0x9FFF	Reserved for profiles
0xA0-0 - 0xFFFF	Reserved

Table 165: Coding of usChannelErrType

### Coding of the field usChannelProp

Bit No	Description
D0 - D7	Data type of this channel (see Table 167: Coding of the field Type in field usChannelProp)
D8	Accumulative. It should be set if the diagnosis is accumulated from several channels
D9 - D10	Maintenance (see Table 168: Coding of the field Maintenance in field usChannelProp)
D11 - D13	Specifier. It will be handled by the Stack and shall not be set by application.
D14 - D15	Direction (see Table 169: Coding of the field <b>Direction</b> in field usChannelProp)

Table 166: 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 167: 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 168: 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 169: Coding of the field **Direction** in field `usChannelProp`*

## 10.15.2 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_HANDLE   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	HANDLE		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 166: Coding of the field usChannelProp
	usChannelErrType	UINT16		See Table 165: Coding of usChannelErrType

Table 170: PNS\_IF\_ADD\_CHANNEL\_DIAG\_CNF-T - Add Channel Diagnosis Confirmation

## 10.16 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 10.4).

### 10.16.1 Add Extended Channel Diagnosis Request

Using this packet the user application can add extended diagnosis data to a submodule.

**Packet Structure Reference**

```
/* Request packet */
typedef struct PNS_IF_ADD_EXTENDED_DIAG_Ttag
{
    TLR_UINT32    ulApi;
    TLR_UINT32    ulSlot;
    TLR_UINT32    ulSubslot;
    TLR_HANDLE    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_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 in use for requests. Set to zero.
	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	HANDLE	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>166: Coding of the</b> field usChannelProp
	usChannelErrType	UINT16		See Table 165: <b>Coding of usChannelErrType</b>
	usReserved	UINT16		Reserved
	ulExtChannelAddValue	UINT32	0	Currently not supported, set to zero.
	usExtChannelErrType	UINT16	1 ... 0x7FFF	See Table <b>172: Coding of usExtChannelErrType</b> for Channel Error Type-1 - 0x7FFF

Table 171: 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	
0x00-1 - 0x7FFF	Manufacturer specific	Alarm / Diagnosis
0x8000	Accumulative info	Alarm / Diagnosis
0x80-1 - 0x8FFF	Reserved	
0x90-0 - 0x9FFF	Profile specific error codes	Alarm / Diagnosis
0xA0-0 - 0xFFFF	Reserved	

Table 172: Coding of `usExtChannelErrType` for Channel Error Type-1 - 0x7FFF

## 10.16.2 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_HANDLE    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.
	hDiagHandle	HANDLE		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 166: Coding of the field usChannelProp
	usChannelErrType	UINT16		See Table 165: Coding of usChannelErrType
	usReserved	UINT16		Reserved
	ulExtChannelAddValue	UINT32	0	Currently not supported, set to zero.
	usExtChannelErrType	UINT16		See Table 172: Coding of usExtChannelErrType for Channel Error Type-1 - 0x7FFF

Table 173: PNS\_IF\_ADD\_EXTENDED\_DIAG\_CNF-T - Add Extended Channel Diagnosis Confirmation



## 10.17 Add Generic Channel 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 10.4).

### 10.17.1 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_HANDLE    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_SIZE - Packet data length in bytes + usDiagDataLen
	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	HANDLE	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 166: Coding of the field usChannelProp
	usUserStructId	UINT16	0 - 0x7FFF	See Table 175: Coding of the field usUserStructId
	usReserved	UINT16		Reserved
	usDiagDataLen	UINT16	1..1024	Length of Diagnosis Data in bytes
	abDiagData[1024]	UINT8		Diagnosis Data

Table 174: 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 175: Coding of the field usUserStructId

## 10.17.2 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_HANDLE    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	HANDLE		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 166: Coding of the field usChannelProp
	usUserStructId	UINT16		See Table 175: Coding of the field usUserStructId

Table 176: PNS\_IF\_ADD\_GENERIC\_DIAG\_CNF-T - Add Generic Channel Diagnosis Confirmation

## 10.18 Remove Diagnosis Service

With this service the user application can remove previously added diagnosis data from a submodule. This will be reported to IO-Controller with a diagnosis disappears alarm automatically if necessary.

### 10.18.1 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_HANDLE 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_PACKET_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_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	0x1FE6	<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	HANDLE		The unique diagnosis handle given to application by the stack while adding the diagnosis record.

Table 177: PNS\_IF\_PACKET\_REMOVE\_DIAG\_REQ-T - Remove Diagnosis Request

## 10.18.2 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_PACKET_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 178: PNS\_IF\_PACKET\_REMOVE\_DIAG\_CNF\_T - Remove Diagnosis Confirmation

# 11 Status/Error Codes Overview

## 11.1 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.

Hexadecimal Value	Definition Description
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.
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 "SE"UP" 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 "f <i>"MODULES"</i> linked table.
0xC0300025	TLR_E_PNS_IF_SUBMODULES_GET_REC Error getting record "f <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.



Hexadecimal Value	Definition Description
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 "n "SIGN"LS" table does not correspond to that "n "SUBMODU"ES" table.
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.

Hexadecimal Value	Definition Description
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.
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.

Hexadecimal Value	Definition Description
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.
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.

Hexadecimal Value	Definition Description
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.
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 can not 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.

Hexadecimal Value	Definition Description
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/ .
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.

Hexadecimal Value	Definition Description
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 netmask).
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.
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.

Hexadecimal Value	Definition Description
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.
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.

Hexadecimal Value	Definition Description
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.
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.



Hexadecimal Value	Definition Description
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(-).
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.

Hexadecimal Value	Definition Description
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.
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 179: Status/Error Codes Overview

## 11.2 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.
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.

Hexadecimal Value	Definition Description
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.
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.

Hexadecimal Value	Definition Description
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_Loadlocr() 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.
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.

Hexadecimal Value	Definition Description
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 180: Status/Error Codes for CMCTL Task

## 11.2.1 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 181: CMCTL -Task Diagnosis-Codes

## 11.3 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



Hexadecimal Value	Definition Description
0xC00B0022	TLR_E_PNIO_CMDEV_FUNCTION_RETURN_FAILURE
0xC00B0023	TLR_E_PNIO_CMDEV_WAIT_FOR_PACKET_FAILED
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.

Hexadecimal Value	Definition Description
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.
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

Hexadecimal Value	Definition Description
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
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.

Hexadecimal Value	Definition Description
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
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 182: Status/Error Codes for CM-Dev Task

### 11.3.1 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 183: CM-Dev-Task Diagnosis-Codes

## 11.4 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 184: Status/Error Codes for EDD Task

### 11.4.1 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 185: EDD-Task Diagnosis-Codes

## 11.5 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.

Hexadecimal Value	Definition Description
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.
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().



Hexadecimal Value	Definition Description
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 to 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.
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 to 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 186: Status/Error Codes for ACP Task

## 11.5.1 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 187: ACP-Task Diagnosis-Codes

## 11.6 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.

Hexadecimal Value	Definition Description
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().
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.

Hexadecimal Value	Definition Description
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.
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.

Hexadecimal Value	Definition
	Description
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 188: Status/Error Codes for DCP Task

## 11.6.1 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 189: DCP-Task Diagnosis-Codes

## 11.7 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.

Hexadecimal Value	Definition Description
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.
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().



Hexadecimal Value	Definition Description
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).
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 190: Status/Error Codes for MGT Task

## 11.7.1 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_RMPPM_RESOURCE_INIT_FAILED Failed to initialize RMPPM.
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 191: MGT-Task Diagnosis-Codes

# 12 Appendix

## 12.1 List of Figures

Figure 1: The 3 different Ways to access a Protocol Stack running on a netX System.....	15
Figure 2: Use of ulDest in Channel and System Mailbox.....	18
Figure 3: Using ulSrc and ulSrcId.....	19
Figure 4: Task Structure of the PROFINET IO Device Stack V3.3.....	57
Figure 5: Memory Area Configuration.....	60
Figure 6: Memory area configuration.....	61
Figure 7: Commissioning the PROFINET IO Device Stack V3.....	63
Figure 8: Exchange of Packets between Stack and Application during Connection Establishment.....	126
Figure 9: Exchange of Packets between Stack and Application for Reconfiguration.....	137
Figure 10: SetSubModuleState from bad to good.....	268

## 12.2 List of Tables

Table 1: List of Revisions.....	7
Table 2: Terms, Abbreviations and Definitions.....	12
Table 3: References.....	12
Table 4: Names of Queues in PROFINET Firmware.....	16
Table 5: Meaning of Source- and Destination-related Parameters.....	16
Table 6: Meaning of Destination-Parameter ulDest.Parameters.....	18
Table 7: Example for correct Use of Source- and Destination-related parameters.....	20
Table 8: Hardware Assembly Options for different xC Ports.....	21
Table 9: Communication Channel Addresses in Dual-Port-Memory.....	22
Table 10: Communication Channel-related Information.....	22
Table 11: Communication Channel Addresses in Dual-Port-Memory.....	23
Table 12: Communication Channel-related Information.....	23
Table 13: Supported process data image synchronization modes.....	24
Table 14: Input Data Image.....	25
Table 15: Output Data Image.....	25
Table 16: General Structure of Packets for non-cyclic Data Exchange.....	27
Table 17: Channel Mailboxes.....	31
Table 18: Common Status Structure Definition.....	33
Table 19: Communication State of Change.....	34
Table 20: Meaning of Communication Change of State Flags.....	35
Table 21: Communication Control Block.....	39
Table 22: Configuration Values for Synchronization.....	40
Table 23: Overview about essential Functionality.....	41
Table 24: Parameters of UpdateConsumerImage Callback.....	47
Table 25: Return Codes of UpdateConsumerImage Callback.....	47
Table 26: Parameters of UpdateProviderImage Callback.....	48
Table 27: Return Codes of UpdateProviderImage Callback.....	48
Table 28: Parameters of the Callback.....	49
Table 29: Return Codes of the Callback.....	49
Table 30: Events indicated by the Callback.....	50
Table 31: Parameters of the Callback.....	50
Table 32: Structure PNIO_PDU_ISOCHRONOUSMODEDATA_T.....	55
Table 33: Overview over the Configuration Packets of the PROFINET IO Device Stack.....	56
Table 34: PNS_IF_SET_CONFIGURATION_REQ_T - Set Configuration Request.....	66
Table 35: Structure tDeviceParameters.....	68
Table 36: Structure PNS_IF_API_STRUCT_T.....	72
Table 37: Structure PNS_IF_SUBMODULE_STRUCT_T for stack version v3.3.....	73
Table 38: PNS_IF_SET_CONFIGURATION_CNF_T - Set Configuration Confirmation.....	75
Table 39: PNS_IF_SET_WARMSTART_REQ_T – Set Warmstart Request.....	80
Table 40: PNS_IF_SET_WARMSTART_CNF_T – Set Warmstart Confirmation.....	81
Table 41: PNS_IF_GET_DEVICE_HANDLE_REQ_T - Get Device Handle Request.....	82
Table 42: PNS_IF_GET_DEVICE_HANDLE_CNF_T - Get Device Handle Confirmation.....	83
Table 43: PNS_REG_FATAL_ERROR_CALLBACK_REQ_T - Register Fatal Error Callback Request.....	87
Table 44: PNS_REG_FATAL_ERROR_CALLBACK_CNF_T - Register Fatal Error Callback Confirmation.....	88
Table 45: PNS_UNREG_FATAL_ERROR_CALLBACK_REQ_T - Unregister Fatal Error Callback Request.....	89
Table 46: PNS_UNREG_FATAL_ERROR_CALLBACK_CNF_T - Unregister Fatal Error Callback Confirmation.....	90
Table 47: PNS_IF_SET_PORT_MAC_REQ_T - Set Port MAC Address Request.....	92

Table 48: PNS_IF_SET_PORT_MAC_CNF_T - Set Port MAC Address Confirmation .....	93
Table 49: PNS_IF_SET_OEM_PARAMETERS_REQ_T - Set OEM Parameters Request .....	96
Table 50: PNS_IF_SET_OEM_PARAMETERS_TYPE_1_T - Set OEM Parameters for ulParamType = 1 .....	96
Table 51: PNS_IF_SET_OEM_PARAMETERS_TYPE_2_T - Set OEM Parameters for ulParamType = 2 .....	97
Table 52: PNS_IF_SET_OEM_PARAMETERS_TYPE_6_T - Set OEM Parameters for ulParamType = 6 .....	97
Table 53: PNS_IF_SET_OEM_PARAMETERS_CNF_T - Set OEM Parameters Confirmation .....	98
Table 54: PNS_IF_LOAD_REMANENT_DATA_REQ_T - Load Remanent Data Request .....	101
Table 55: PNS_IF_LOAD_REMANENT_DATA_CNF_T - Load Remanent Data Confirmation .....	102
Table 56: PNS_IF_SET_IOIMAGE_REQ_T - Set IO-Image Request .....	105
Table 57: PNS_IF_SET_IOIMAGE_CNF_T - Set IO-Image Confirmation .....	107
Table 58: PNS_IF_SET_IOXS_CONFIG_REQ_T - Set IOXS Config Request.....	109
Table 59: PNS_IF_SET_IOXS_CONFIG_CNF_T - Set IOXS Config Confirmation.....	110
Table 60: IO_SIGNALS_CONFIGURE_SIGNAL_REQ_T - Configure Signal Request.....	113
Table 61: IO_SIGNALS_CONFIGURE_SIGNAL_CNF_T - Configure Signal Confirmation.....	114
Table 62: Overview about the recommended Task Priorities.....	122
Table 63: Overview over the Connection Establishment Packets of the PROFINET IO Device IRT StAR Check Service .....	128
Table 64: PNS_IF_AR_CHECK_IND_T - AR Check Indication.....	130
Table 65: PNS_IF_AR_CHECK_RSP_T - AR Check Response .....	131
Table 66: PNS_IF_CHECK_IND_T - Check Indication .....	133
Table 67: PNS_IF_CHECK_RSP_T - Check Response .....	135
Table 68: Field usModuleState.....	136
Table 69: Field usSubmodState.....	136
Table 70: PNS_IF_CONNECTREQ_DONE_IND_T - Connect Request Done Indication.....	139
Table 71: PNS_IF_CONNECTREQ_DONE_RSP_T - Connect Request Done Response.....	140
Table 72: PNS_IF_PARAM_END_IND_T - Parameter End Indication .....	142
Table 73: PNS_IF_PARAM_END_RSP_T - Parameter End Response.....	144
Table 74: PNS_IF_APPL_READY_IND_T - Application Ready.....	146
Table 75: PNS_IF_APPL_READY_CNF_T - Application Ready Confirmation.....	147
Table 76: PNS_IF_AR_IN_DATA_IND_T - AR InData Indication .....	149
Table 77: PNS_IF_AR_IN_DATA_RSP_T - AR InData Response .....	150
Table 78: PNS_IF_STORE_REMANENT_DATA_IND_T - Store Remanent Data Indication.....	152
Table 79: PNS_IF_STORE_REMANENT_DATA_RES_T - Store Remanent Data Response.....	153
Table 80: Overview over the Packets for Acyclic Events indicated by the PROFINET IO Device IRT Stack .....	155
Table 81: PNS_IF_READ_RECORD_IND_T - Read Record Indication .....	157
Table 82: PNS_IF_READ_RECORD_RSP_T - Read Record Response .....	159
Table 83: Coding of the field ulPnio .....	160
Table 84: Coding of ErrorCode .....	160
Table 85: Coding of ErrorCode.....	160
Table 86: Coding of ErrorCode1 and ErrorCode 2 for ErrorCode = PNIOIRW.....	162
Table 87: Coding of ErrorCode1 and ErrorCode 2 for ErrorCode = PNIO.....	168
Table 88: PNS_IF_WRITE_RECORD_IND_T - Write Record Indication .....	170
Table 89: PNS_IF_WRITE_RECORD_RSP_T - Write Record Response .....	172
Table 90: PNS_IF_AR_ABORT_IND_IND_T - AR Abort Indication Indication .....	174
Table 91: PNS_IF_AR_ABORT_IND_RSP_T - AR Abort Indication Response.....	175
Table 92: PNS_IF_SAVE_STATION_NAME_IND_T - Save Station Name Indication.....	177
Table 93: PNS_IF_SAVE_STATION_NAME_RSP_T - Save Station Name Response.....	178
Table 94: PNS_IF_SAVE_STATION_TYPE_IND_T - Save Station Type Indication .....	180
Table 95: PNS_IF_SAVE_STATION_TYPE_RSP_T - Save Station Type Response .....	181
Table 96: PNS_IF_SAVE_IP_ADDRESS_IND_T - Save IP Address Indication .....	183
Table 97: PNS_IF_SAVE_IP_ADDRESS_RSP_T - Save IP Address Response .....	184
Table 98: PNS_IF_START_LED_BLINKING_IND_T - Start LED Blinking Indication .....	186
Table 99: PNS_IF_START_LED_BLINKING_RSP_T - Start LED Blinking Response .....	187
Table 100: PNS_IF_STOP_LED_BLINKING_IND_T - Stop LED Blinking Indication .....	188
Table 101: PNS_IF_STOP_LED_BLINKING_RSP_T - Stop LED Blinking Response.....	189
Table 102: PNS_IF_RESET_FACTORY_SETTINGS_IND_T - Reset Factory Settings Indication.....	191
Table 103: PNS_IF_RESET_FACTORY_SETTINGS_RSP_T - Reset Factory Settings Response.....	192
Table 104: PNS_IF_APDU_STATUS_CHANGED_IND_T - APDU Status Changed Indication.....	193
Table 105: Status Byte of PNS_IF_APDU_STATUS_CHANGED_IND_T packet.....	194
Table 106: PNS_IF_APDU_STATUS_CHANGED_RSP_T - APDU Status Changed Response.....	195
Table 107: PNS_IF_ALARM_IND_T - Alarm Indication Indication.....	198
Table 108: PNS_IF_ALARM_RSP_T - Alarm Indication Response.....	199
Table 109: PNS_IF_RELEASE_REQ_IND_T - Release Request Indication.....	201
Table 110: PNS_IF_RELEASE_REQ_RSP_T - Release Request Indication Response.....	203
Table 111: PNS_IF_LINK_STATUS_CHANGED_IND_T - Link Status Changed Indication.....	205

Table 112: Structure PNS_IF_LINK_STATUS_DATA_T .....	205
Table 113: PNS_IF_LINK_STATUS_CHANGED_RSP_T - Link Status Changed Response .....	206
Table 114: PNS_IF_USER_ERROR_IND_T - Error Indication Service.....	208
Table 115: PNS_IF_USER_ERROR_RSP_T - Error Indication Response .....	209
Table 116: PNS_IF_READ_IM_IND_T – Read I&M Indication .....	211
Table 117: PNS_IF_READ_IM_RES_T – Read I&M Response .....	212
Table 118: PNS_IF_IM0_DATA_T – Structure of I&M0 Information .....	213
Table 119: PNS_IF_IM1_DATA_T – Structure of I&M1 Information .....	214
Table 120: PNS_IF_IM2_DATA_T – Structure of I&M2 Information .....	214
Table 121: PNS_IF_IM3_DATA_T – Structure of I&M3 Information .....	214
Table 122: PNS_IF_IM4_DATA_T – Structure of I&M4 Information .....	215
Table 123: PNS_IF_IM0_FILTER_DATA_T – Structure of I&M0 Filter Information .....	215
Table 124: PNS_IF_WRITE_IM_IND_T – Write I&M Indication .....	217
Table 125: PNS_IF_WRITE_IM_RES_T – Write I&M Response .....	218
Table 126: PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_T – Parameterization Speedup Supported Indication.....	220
Table 127: PNS_IF_PARAMET_SPEEDUP_SUPPORTED_RES_T- Parameterization Speedup Supported Response.....	221
Table 128: Overview over the Packets of the PROFINET IO Device IRT Stack for Acyclic Events requested by the Application .....	223
Table 129: PNS_IF_GET_DIAGNOSIS_REQ_T - Get Diagnosis Request.....	224
Table 130: PNS_IF_GET_DIAGNOSIS_CNF_T - Get Diagnosis Confirmation.....	226
Table 131: Meaning of single Bits in ulPnsState .....	227
Table 132: Values and their corresponding Meanings of ulLinkState.....	227
Table 133: Values and their corresponding Meanings of ulConfigState .....	228
Table 134: PNS_IF_GET_XMAC_DIAGNOSIS_REQ_T - Get XMAC (EDD) Diagnosis Request.....	229
Table 135: PNS_IF_GET_XMAC_DIAGNOSIS_CNF_T - Get XMAC (EDD) Diagnosis Confirmation.....	231
Table 136: Structure EDD_XMAC_COUNTERS_T .....	232
Table 137: PNS_IF_SEND_PROCESS_ALARM_REQ_T - Process Alarm Request.....	234
Table 138: PNS_IF_SEND_PROCESS_ALARM_CNF_T - Process Alarm Confirmation .....	236
Table 139: PNS_IF_SEND_DIAG_ALARM_REQ_T - Diagnosis Alarm Request.....	238
Table 140: PNS_IF_DIAG_ALARM_CNF_T - Diagnosis Alarm Confirmation .....	240
Table 141: PNS_IF_RETURN_OF_SUB_ALARM_REQ_T - Return of Submodule Alarm Request .....	242
Table 142: PNS_IF_RETURN_OF_SUB_ALARM_CNF_T - Return of Submodule Alarm Confirmation .....	243
Table 143: PNS_IF_ABORT_CONNECTION_REQ_T - AR Abort Request Request.....	245
Table 144: PNS_IF_ABORT_CONNECTION_CNF_T - AR Abort Request Confirmation.....	246
Table 145: PNS_IF_PLUG_MODULE_REQ_T - Plug Module Request .....	248
Table 146: PNS_IF_PLUG_MODULE_CNF_T - Plug Module Confirmation .....	249
Table 147: PNS_IF_PLUG_SUBMODULE_REQ_T - Plug Submodule Request for stack v3.3 .....	252
Table 148: PNS_IF_PLUG_SUBMODULE_CNF_T - Plug Submodule Confirmation of stack v3.3.....	253
Table 149: PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_T – Extended Plug Submodule Request .....	256
Table 150: PNS_IF_PLUG_SUBMODULE_EXTENDED_CNF_T – Extended Plug Submodule Confirmation .....	259
Table 151: PNS_IF_PULL_MODULE_REQ_T - Pull Module Request.....	261
Table 152: PNS_IF_PULL_MODULE_CNF_T – Pull Module Confirmation.....	263
Table 153: PNS_IF_PULL_SUBMODULE_REQ-T - Pull Submodule Request.....	265
Table 154: PNS_IF_PULL_SUBMODULE_CNF-T - Pull Submodule Confirmation.....	266
Table 155: PNS_IF_SET_SUBM_STATE_REQ-T – Set Submodule State Request .....	270
Table 156: Possible Values of usSubmState .....	270
Table 157: PNS_IF_SET_SUBM_STATE_CNF-T - Set Submodule State Confirmation .....	271
Table 158: PNS_IF_GET_STATION_NAME_REQ-T - Get Station Name Request .....	272
Table 159: PNS_IF_GET_STATION_NAME_CNF-T - Get Station Name Confirmation .....	273
Table 160: PNS_IF_GET_STATION_TYPE_REQ-T - Get Station Type Request.....	274
Table 161: PNS_IF_GET_STATION_TYPE_CNF-T - Get Station Type Confirmation.....	275
Table 162: PNS_IF_GET_IP_ADDR_REQ-T - Get IP Address Request.....	276
Table 163: PNS_IF_GET_IP_ADDR_CNF-T - Get IP Address Confirmation .....	277
Table 164: PNS_IF_ADD_CHANNEL_DIAG_REQ-T - Add Channel Diagnosis Request .....	279
Table 165: Coding of usChannelErrType .....	281
Table 166: Coding of the field usChannelProp.....	281
Table 167: Coding of the field Type in field usChannelProp.....	281
Table 168: Coding of the field Maintenance in field usChannelProp .....	281
Table 169: Coding of the field <b>Direction</b> in field usChannelProp .....	282
Table 170: PNS_IF_ADD_CHANNEL_DIAG_CNF-T - Add Channel Diagnosis Confirmation .....	284
Table 171: PNS_IF_ADD_EXTENDED_DIAG_REQ-T - Add Extended Channel Diagnosis Request.....	286
Table 172: Coding of usExtChannelErrType for Channel Error Type–1 - 0x7FFF .....	287
Table 173: PNS_IF_ADD_EXTENDED_DIAG_CNF-T - Add Extended Channel Diagnosis Confirmation .....	288
Table 174: PNS_IF_ADD_GENERIC_DIAG_REQ_T - Add Generic Channel Diagnosis Request.....	290
Table 175: Coding of the field usUserStructId.....	290

Table 176: PNS_IF_ADD_GENERIC_DIAG_CNF-T - Add Generic Channel Diagnosis Confirmation .....	292
Table 177: PNS_IF_PACKET_REMOVE_DIAG_REQ-T - Remove Diagnosis Request .....	293
Table 178: PNS_IF_PACKET_REMOVE_DIAG_CNF_T - Remove Diagnosis Confirmation .....	294
Table 179: Status/Error Codes Overview .....	306
Table 180: Status/Error Codes for CMCTL Task.....	310
Table 181: CMCTL -Task Diagnosis-Codes.....	311
Table 182: Status/Error Codes for CM-Dev Task.....	316
Table 183: CM-Dev-Task Diagnosis-Codes.....	317
Table 184: Status/Error Codes for EDD Task .....	318
Table 185: EDD-Task Diagnosis-Codes .....	318
Table 186: Status/Error Codes for ACP Task.....	322
Table 187: ACP-Task Diagnosis-Codes.....	322
Table 188: Status/Error Codes for DCP Task .....	326
Table 189: DCP-Task Diagnosis-Codes .....	326
Table 190: Status/Error Codes for MGT Task.....	329
Table 191: MGT-Task Diagnosis-Codes .....	330

## 12.3 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.  
New Delhi - 110 065  
Phone: +91 11 26915430  
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)