



**Protocol API**  
**PROFINET IO Controller**

**V2.7**

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

DOC050901API19EN | Revision 19 | English | 2015-05 | Released | Public

# Table of Contents

<b>1</b>	<b>Introduction.....</b>	<b>5</b>
1.1	About this Document.....	5
1.2	List of Revisions.....	5
1.3	Functional Overview.....	5
1.4	System Requirements.....	6
1.5	Intended Audience.....	6
1.6	Specifications.....	6
1.6.1	Supported Protocols.....	6
1.6.2	Technical Data.....	7
1.7	Terms, Abbreviations and Definitions.....	9
1.8	References to Documents.....	10
1.9	Legal Notes.....	11
1.9.1	Copyright.....	11
1.9.2	Important Notes.....	11
1.9.3	Exclusion of Liability.....	12
1.9.4	Export.....	12
<b>2</b>	<b>Fundamentals.....</b>	<b>13</b>
2.1	General Access Mechanisms on netX Systems.....	13
2.2	Accessing the Protocol Stack by Programming the AP Task's Queue.....	14
2.2.1	Getting the Receiver Task Handle of the Process Queue.....	14
2.2.2	Meaning of Source- and Destination-related Parameters.....	15
2.3	Accessing the Protocol Stack via the Dual Port Memory Interface.....	15
2.3.1	Communication via Mailboxes.....	15
2.3.2	Using Source and Destination Variables correctly.....	16
2.3.3	Obtaining useful Information about the Communication Channel.....	19
2.4	Client/Server Mechanism.....	21
2.4.1	Application as Client.....	21
2.4.2	Application as Server.....	22
<b>3</b>	<b>Dual-Port Memory.....</b>	<b>23</b>
3.1	Cyclic Data (Input/Output Data).....	23
3.1.1	Input Process Data.....	24
3.1.2	Output Process Data.....	24
3.2	Acyclic Data (Mailboxes).....	25
3.2.1	General Structure of Messages or Packets for Non-Cyclic Data Exchange.....	26
3.2.2	Status & Error Codes.....	28
3.2.3	Differences between System and Channel Mailboxes.....	28
3.2.4	Send Mailbox.....	28
3.2.5	Receive Mailbox.....	28
3.2.6	Channel Mailboxes (Details of Send and Receive Mailboxes).....	29
3.3	Status.....	30
3.3.1	Common Status.....	30
3.3.2	Extended Status.....	38
3.4	Control Block.....	40
3.5	Example: DPM Layout of Input Area.....	41
<b>4</b>	<b>Getting started / Configuration.....</b>	<b>42</b>
4.1	Overview about Essential Functionality.....	42
4.2	Configuration of the Profinet IO Controller.....	43
4.2.1	General Configuration Structure.....	43
4.2.2	Using the Packet Interface with Write Access to the Dual-Port Memory.....	49
4.2.3	Using the Configuration Tool SYCON.net.....	50
4.3	Port Settings concerning Cross-over (manual Configuration).....	50
4.4	Task Structure of the PROFINET IO Controller Stack.....	51
4.5	Device Handle and obtainin detailed Information of an IO Device.....	55
4.5.1	Identifying an PROFINET IO Device by a Handle.....	55
4.5.2	Obtaining detailed and Diagnostic Information from connected Slaves.....	55
4.6	IOPS Interface.....	58
4.6.1	What is IOPS?.....	58
4.6.2	How does the interface work?.....	58
4.6.3	IOPS Modes.....	58
4.6.4	Important Notes.....	59

<b>5</b>	<b>The Application Interface .....</b>	<b>60</b>
5.1	General Packets for the Configuration Data .....	60
5.1.1	Packet Header.....	61
5.1.2	Packet Subheader.....	61
5.1.3	Configuration Data .....	61
5.2	Download of large Datasets .....	62
5.2.1	General Packet Header of Request Packet.....	63
5.2.2	Packet Header of Confirmation Packet.....	63
5.3	Configuration Data Packets .....	64
5.3.1	Extended PNM (IO Controller) Dataset .....	67
5.3.2	PNM (IO Controller) Dataset (legacy).....	75
5.3.3	Extended PNM_IOD (IO-Device) Datasets.....	78
5.3.4	PNM_IOD (IO-Device) Datasets (legacy) .....	83
5.3.5	PNM_IOD_IOCRR (Communication Relation) Datasets.....	86
5.3.6	PNM_IOD_AP (Application Process) Datasets .....	90
5.3.7	PNM_IOD_Module (Module) Datasets .....	92
5.3.8	PNM_IOD_Submodule (Submodule) Datasets .....	94
5.3.9	PNM_IOD_SubmDescr_Ext (Extended Submodule Description) Datasets.....	97
5.3.10	PNM_IOD_SubmDescr (Submodule Description) Datasets .....	99
5.3.11	PNM_IOD_IO_Signals (SubmDescr Signal Configuration) Dataset .....	103
5.3.12	PNM_IOD_IntfSubm (Interface Submodule) Datasets .....	105
5.3.13	PNM_IOD_PortSubm (Port Submodule) Datasets .....	109
5.3.14	PNM_IOD_RecData (Record Data) Datasets.....	112
5.3.15	PNM Download Finish.....	115
5.3.16	PNM Set VersionInfo Service.....	117
5.4	Registering and Unregistering an Application.....	119
5.4.1	Register Service .....	119
5.4.2	Deregister Service.....	122
5.5	Acyclic Requests.....	125
5.5.1	Read Service.....	126
5.5.2	Write Service .....	129
5.5.3	Read Implicit Service.....	132
5.5.4	Device Diagnosis Service.....	135
5.5.5	Obtain Device Connection Information.....	138
5.5.6	Common Status Service (Firmware Diagnosis) .....	141
5.5.7	ModuleDiffBlock Service.....	143
5.5.8	DCP Signal.....	146
5.5.9	DCP SET Name .....	150
5.5.10	DCP SET IP .....	154
5.5.11	DCP RESET FACTORYSETTINGS.....	158
5.5.12	DCP IDENT ALL.....	161
5.5.13	DCP GET .....	163
5.5.14	Alarm Acknowledge Service.....	167
5.5.15	Release IO-Device Service .....	171
5.6	Acyclic Indications to Application .....	174
5.6.1	DCP IDENT ENTRY Service .....	174
5.6.2	DCP IDENT ALL Finished Service .....	177
5.6.3	Alarm Service .....	180
5.6.4	Diagnosis Service.....	184
<b>6</b>	<b>Status/Error Codes Overview.....</b>	<b>187</b>
6.1	Error Codes of the ACP – Task .....	187
6.1.1	Diagnostic Codes of the ACP – Task .....	188
6.2	Error Codes of the APCFG –Task .....	189
6.2.1	Diagnostic Codes of the APCFG -Task .....	194
6.3	Error Codes of the APCTL –Task .....	195
6.3.1	Diagnostic Codes of the APCTL-Task.....	198
6.4	Error Codes of the CMCTL–Task .....	199
6.4.1	Diagnostic Codes of the CMCTL–Task .....	203
6.5	Error Codes of the CMDEV–Task.....	204
6.5.1	Diagnostic Codes of the CMDEV–Task.....	207
6.6	Error Codes of the DCP –Task .....	208
6.6.1	Diagnostic Codes of the DCP –Task .....	210
6.7	Error Codes of the EDD–Task .....	210
6.7.1	Diagnostic Codes of the EDD–Task .....	210
6.8	Error Codes of the IO Signal Task .....	211
6.9	Error Codes of the LLDP–Task.....	211
6.10	Error Codes of the MGT–Task.....	216

---

6.10.1	Diagnostic Codes of the MGT –Task.....	218
6.11	Error Codes of the RPC–Task .....	219
6.11.1	Diagnostic Codes of the RPC–Task .....	221
6.12	Other Relevant Error Codes.....	223
<b>7</b>	<b>Definition of the LEDs.....</b>	<b>226</b>
<b>8</b>	<b>Appendix .....</b>	<b>227</b>
8.1	List of Tables.....	227
8.2	List of Figures.....	229
8.3	Contacts .....	230

# 1 Introduction

## 1.1 About this Document

This manual describes the user interface of the PROFINET RT IO Controller implementation (stack version 2.6.9.0 and newer) 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 driver functions or direct access to the dual-port memory.

The general mechanism of data transfer, for example how to send and receive a message is independent from the protocol. These procedures are common to all devices and are described in the 'netX DPM Interface manual'

## 1.2 List of Revisions

Rev	Date	Name	Chapter	Revision
15	2011-12-07	RG/TK/ BM		Firmware/ stack version 2.6.3.x Reference to netX Dual-Port Memory Interface Manual Revision 12 Added description of <code>bStateTypeID</code> and Table 2: Terms, Abbreviations and Definitions Added description of <code>ulTrDevId</code> in <i>Table 29: Configure extended PNM Confirmation</i> Corrected command code of DCP Reset FactorySettings Confirmation packet in <i>Table 108: APIOC_DCP_RESET_FACTORY_CNF – DCP RESET FACTORY Confirmation Packet</i> Added new error messages (Completely new sections 6.5, 6.9 and further additions)
16	2012-07-12	BM		Firmware/ stack version 2.6.7.x Correction: Read Implicit possible during data exchange Section DCP GET added
17	2013-02-15	BM		Firmware/ stack version v2.6.9.x Section PNM Set VersionInfo Service added.
18	2013-09-16	RG/BM	5.3.11.1	Firmware/ stack version v2.6.9.x Improved description meaning of parameters of Configure PNM_IOD_IO_Signals Request Added some error message descriptions
19	2015-05-28	RG		Firmware/stack version V2.7.x.x Change in <i>Table 90: DIAG_INFO_GET_COMMON_STATE_CNF - Get Common Status Block Confirmation</i>

Table 1: List of Revisions

## 1.3 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 RT IO Controller Context Management
- Realization of the PROFINET RT IO Controller cyclic data exchange
- Realization of the PROFINET RT IO Controller acyclic data exchange
- Realization of the DCP protocol

## 1.4 System Requirements

This software package has following system requirements to its environment:

- netX-Chip as CPU hardware platform with PROFINET master license
- operating system rcX

If your hardware has no master license, the PROFINET RT IO Controller stack will not work.

## 1.5 Intended Audience

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

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

## 1.6 Specifications

The data below applies to IO Controller RT firmware and stack version V2.6.9.0.

### 1.6.1 Supported Protocols

- RTC – Real time Cyclic Protocol
- RTA – Real time Acyclic Protocol
- DCP – Discovery and Configuration Protocol
- CL-RPC – Connectionless Remote Procedure Call
- LLDP – Link Layer Discovery Protocol
- SNMP – Simply Network Management Protocol (SNMP v1)

## 1.6.2 Technical Data

Maximum number of total cyclic input data	5712 bytes (including IOxS status bytes)
Maximum number of total cyclic output data	5760 bytes (including IOxS status bytes)
Maximum number of cyclic input data	1440 bytes per device (= IOCR data length including IOxS status bytes)
Maximum number of cyclic output data	1440 bytes per device (= IOCR data length including IOxS status bytes)
Maximum number of configured devices	128
Acyclic communication	Read/Write Record Limited to 1392 bytes per telegram Limited to 4096 bytes per request
Alarm processing	yes but requires handling in application program
Diagnostis data	One 200 Byte buffer per IO device
DCP functions via API	Name Assignment IO-Devices (DCP SET NameOfStation) Set IO-Devices IP (DCP SET IP) Signal IO-Device (DCP SET SIGNAL) Reset IO-Device to factory settings (DCP Reset FactorySettings) Bus scan (DCP IDENTIFY ALL) DCP GET
Context management by CL-RPC	supported
Minimum cycle time	1ms (Different IO-Devices can be configured with different cycle times)
Functions	Fast Startup of PROFINET IO Devices supported
Baud rate	100 MBit/s Full-Duplex mode
Data transport layer	Ethernet II, IEEE 802.3
Size of configuration file	Max. 1 MByte

### Firmware/stack available for netX

netX 50	no
netX 100, netX 500	yes

### PCI

DMA Support for PCI targets	yes
-----------------------------	-----

### Slot number

Slot number support for	CIFX 50-RE
-------------------------	------------

### Configuration

Configuration by tool SYCON.net (Download or exported configuration of two files named *config.nxd* and *nwid.nxd*)

Configuration by packets to transfer bus and device parameters

## Diagnostic

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

Firmware has a buffer for IO-Device's Profinet diagnosis data (one 200 byte buffer per IO-Device, see information on limitation below). This buffer contains the raw diagnosis data directly taken from the bus. Additionally the Profinet Diagnosis alarms are directly indicated to the user application.

## Limitations

The size of the bus configuration file is limited by the size of the RAM Disk (1 Megabyte)

The configuration database interface and its structure and definitions apply for netX products which use the PROFINET IO Controller - firmware only.

Structures and functions described in this document apply only to hardware from 3rd party vendors insofar as original Hilscher firmware is concerned. Therefore, whenever the term "netX firmware" is used throughout this manual it refers to ready-made firmware provided by Hilscher. Although 3rd party vendors are free to implement the same structures and functions in their product, no guarantee for compatibility of drivers etc. can be given.

The following limitations apply

- The usable (minimum) cycle time depends on the number of used IO Devices, the number of used input and output data. The cycle-time, the number of configured IO-Devices and the amount of IO-data depend on each other. For example it is not possible due to performance reasons to have 128 IO-Devices communication with cycle-time 1ms.
- RT over UDP not supported
- Multicast communication not supported
- DHCP is not supported (neither for PROFINET IO Controller nor for the IO-Devices)
- Only one IOCR per IO-Device
- NameOfStation of IO Controller CANNOT be set using the DCP SET NameOfStation service but only at start-up while configuring the IO Controller
- The buffer for IO-Device diagnosis data will be overwritten in case of multiple diagnostic events. Only one (the last) event is stored at the same time. If a single event produces more than 200 bytes of diagnosis data (which is a really improbable situation), only the first 200 bytes will be taken care of.
- WriteMultiple-Record service is not supported

## Known Restrictions

Provider state in cyclic frames is set to "bad" until an application explicitly sets new input data

For compatibility reasons with some IO-Devices the firmware changes the gateway address of an IO-Device from 0.0.0.0 to the IO-Device's IP-address. Both values have the meaning "No gateway" according to Profinet specification.



## 1.7 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
CS	Consumer Status
PS	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
MRP	Media Redundancy Protocol
IOPS	IO Provider Object Status (on a submodule basis)
IOCS	IO Consumer Object Status (on a submodule basis)

Table 2: Terms, Abbreviations and Definitions

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

All IP addresses in this document have host byte order.

## 1.8 References to Documents

This document refers to the following documents:

- [1] Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Interface Manual, netX based products, Revision 12, english, 2012
- [2] PROFIBUS Nutzerorganisation e.V.: Application Layer **protocol** for decentralized periphery and distributed automation; Specification for PROFINET; Version 2.2, October 2007, Order No: 2.722
- [3] PROFIBUS Nutzerorganisation e.V.: Application Layer **services** for decentralized periphery and distributed automation; Specification for PROFINET; Version 2.2, October 2007, Order No: 2.712

*Table 3: References to Documents*

## **1.9 Legal Notes**

### **1.9.1 Copyright**

© Hilscher, 2006-2015, 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.

### **1.9.2 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.

### 1.9.3 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.

### 1.9.4 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:

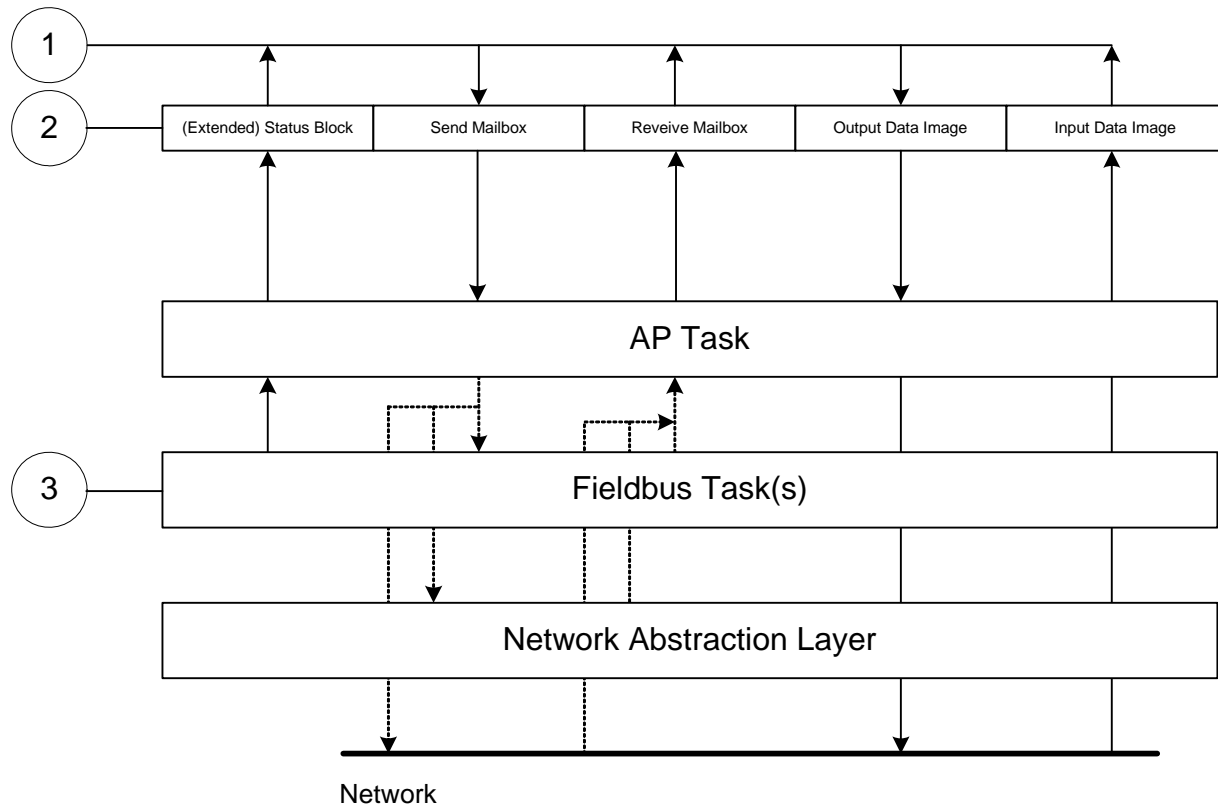


Figure 1: The 3 different Ways to access a Protocol Stack running on a netX System

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 a PROFINET IO Controller 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 different tasks which you have to use as current value for the first parameter (`pszIdn`) is

ASCII Queue name	Description
"QUE_PNIO_ACP"	Name of the ACP-Task process queue
"QUE_PNIO_APCFG"	Name of the APCFG-Task process queue
"QUE_PNIO_APCTL"	Name of the APCTL-Task process queue
"QUE_PNIO_CMCTL"	Name of the CMCTL -Task process queue
"QUE_PNIO_DCP"	Name of the DCP -Task process queue
"QUE_PNIO_EDD"	Name of the EDD -Task process queue
"QUE_PNIO_MGT"	Name of the MGT -Task process queue
"QUE_RPC_TASK"	Name of the RPC -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 other tasks. 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
ulDest	Application mailbox used for confirmation
ulSrc	Queue handle returned by TLR_QUEUE_IDENTIFY() as described above.
ulSrcId	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:

4. Section 7 "Queue-Packets"
5. 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 Controller 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 firmware

Receive Mailbox          Packet transfer from firmware to host system

For more details about acyclic data transfer via mailboxes see section *Acyclic Data (Mailboxes)* on page 25 in this context, is described in detail in section *General Structure of Messages or Packets for Non-Cyclic Data Exchange* on page 26 while the possible codes that may appear are listed in section *Status & Error Codes* on page 28.

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:

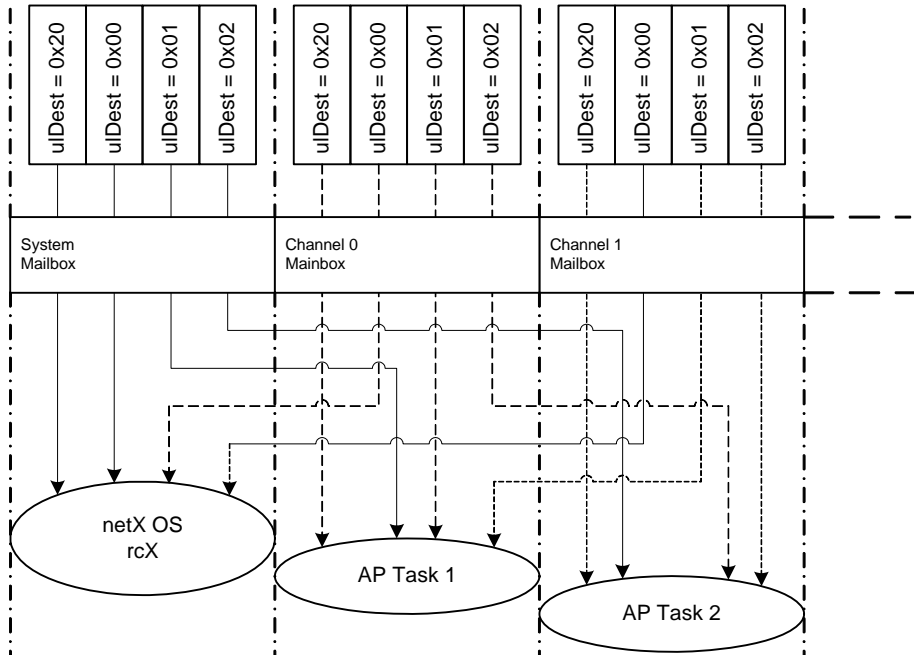


Figure 2: Use of `ulDest` in Channel and System Mailbox

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
0x0000	Packet is passed to the <code>netX</code> operating system <code>rcX</code>
0x0001	Packet is passed to communication channel 0
0x0002	Packet is passed to communication channel 1
0x0003	Packet is passed to communication channel 2
0x0004	Packet is passed to communication channel 3
0x0020	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 0x0020 (= *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:

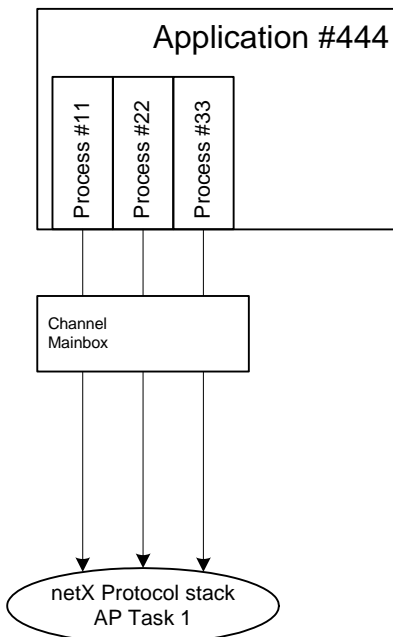


Figure 3: Using ulSrc and ulSrcId

**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 (0x0020)	This value needs always to be set to 0x0020 (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:

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

Check each of the hardware assembly options whether its value has been set to `RCX_HW_ASSEMBLY_CAN = 0x0030`. If true, this denotes that this xCPort is suitable for running the CANopen protocol stack. Otherwise, this port is designed for another communication protocol. In most cases, xC Port[2] will be used for field-bus systems, while xC Port[0] and xC Port[1] are normally used for Ethernet communication.

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

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

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

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

The information is delivered as follows:

Size of Channel in Bytes

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

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

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

## 2.4 Client/Server Mechanism

### 2.4.1 Application as Client

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

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

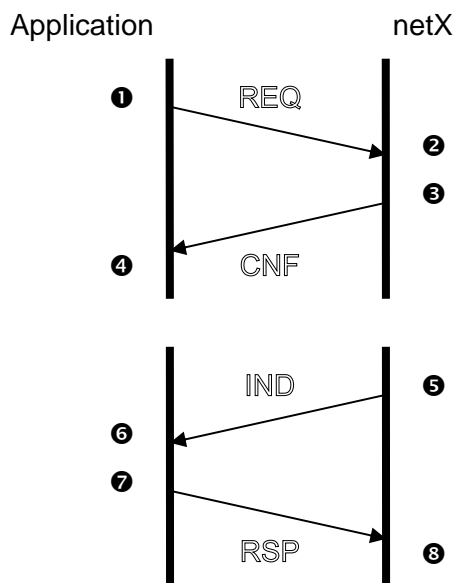


Figure 4: Transition Chart Application as Client

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

REQ	Request	CNF	Confirmation
IND	Indication	RSP	Response

## 2.4.2 Application as Server

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

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

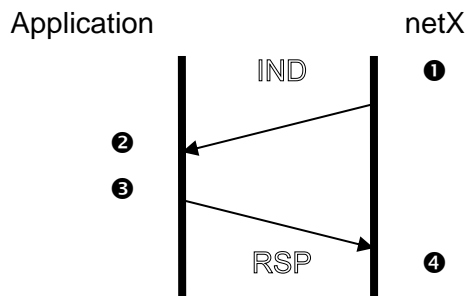


Figure 5: Transition Chart Application as Server

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

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

IND Indication      RSP Response

## 3 Dual-Port Memory

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

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

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

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

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

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



**Note:** Profinet offers a data status on a submodule basis (IOPS, see section IOPS Interface). If the IOPS interface is not used the IO Controller firmware may copy invalid data into the DPM input area without the user having a possibility to detect this invalidity.

### 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*).



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

The contents of these 48 byte is identical to the contents of the second part of the Extended Status Block beginning at address 0x0100, see *Table 18: Extended Status Block for PROFINET IO Controller – Second Part (State Field Definition Block)*.



**Note:** If the IOPS interface is used the IOPS information reduces the amount of usable input data, too.

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

Table 8: 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 *netX DPM Manual*).



**Note:** If the IOPS interface is used the IOPS information reduces the amount of usable output data.

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

Table 9: Output Data Image



## 3.2 Acyclic Data (Mailboxes)

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

Send Mailbox                      Packet transfer from host system to firmware

Receive Mailbox                Packet transfer from firmware to host system

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

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

---

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

---

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

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

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

Table 10: 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 Handler

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

### Destination Identifier

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

### Source Identifier

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

### Length of Data Field

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

### Identifier

The *ulId* field is used to identify a specific packet among others of the same kind. That way the application or driver can match a specific reply or confirmation packet to a previous request packet. The receiving task does not change this field and passes it back to the originator of the packet. Its use is optional in most of the cases. But it is mandatory for sequenced packets. Example: Downloading big amounts of data that does not fit into a single packet. For a sequence of packets the identifier field is incremented by one for every new packet.

### Status / Error Code

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

### Command / Response

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

### Extension

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

### Routing Information

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

### User Data Field

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

## 3.2.2 Status & Error Codes

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

## 3.2.3 Differences between System and Channel Mailboxes

The mailbox system on netX provides a non-cyclic data transfer channel for 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 11: 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

Common Status			
<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 12: 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 13: Communication State of Change



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

Bit	Definition / Description
0	Ready (RCX_COMM_COS_READY) 0 - ... 1 - The <i>Ready</i> flag is set as soon as the protocol stack is started properly. Then the protocol stack is awaiting a configuration. As soon as the protocol stack is configured properly, the <i>Running</i> flag is set, too.
1	Running (RCX_COMM_COS_RUN) 0 - ... 1 - The <i>Running</i> flag is set when the protocol stack has been configured properly. Then the protocol stack is awaiting a network connection. Now both the <i>Ready</i> flag and the <i>Running</i> flag are set.
2	Bus On (RCX_COMM_COS_BUS_ON) 0 - ... 1 - The <i>Bus On</i> flag is set to indicate to the host system whether or not the protocol stack has the permission to open network connections. If set, the protocol stack has the permission to communicate on the network; if cleared, the permission was denied and the protocol stack will not open network connections.
3	Configuration Locked (RCX_COMM_COS_CONFIG_LOCKED) 0 - ... 1 - The <i>Configuration Locked</i> flag is set, if the communication channel firmware has locked the configuration database against being overwritten. Re-initializing the channel is not allowed in this state. To unlock the database, the application has to clear the <i>Lock Configuration</i> flag in the control block (see section 3.2.4 of the netX DPM Interface Manual).
4	Configuration New (RCX_COMM_COS_CONFIG_NEW) 0 - ... 1 - The <i>Configuration New</i> flag is set by the protocol stack to indicate that a new configuration became available, which has not been activated. This flag may be set together with the <i>Restart Required</i> flag.
5	Restart Required (RCX_COMM_COS_RESTART_REQUIRED) 0 - ... 1 - The <i>Restart Required</i> flag is set when the channel firmware requests to be restarted. This flag is used together with the <i>Restart Required Enable</i> flag below. Restarting the channel firmware may become necessary, if a new configuration was downloaded from the host application or if a configuration upload via the network took place.
6	Restart Required Enable (RCX_COMM_COS_RESTART_REQUIRED_ENABLE) 0 - ... 1 - The <i>Restart Required Enable</i> flag is used together with the <i>Restart Required</i> flag above. If set, this flag enables the execution of the Restart Required command in the netX firmware (for details on the <i>Enable</i> mechanism see section 2.3.2 of the netX DPM Interface Manual)).
7 ... 31	Reserved, set to 0

Table 14: Meaning of Communication Change of State Flags

### Communication State (All Implementations)

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

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

### Communication Channel Error (All Implementations)

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

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

### Runtime Failures

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

### Initialization Failures

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

### Configuration Failures

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

## Network Failures

- (General) NETWORK FAULT  
#define RCX\_COMM\_NETWORK\_FAULT 0xC0000140
- CONNECTION CLOSED  
#define RCX\_COMM\_CONNECTION\_CLOSED 0xC0000141
- CONNECTION TIMED OUT  
#define RCX\_COMM\_CONNECTION\_TIMEOUT 0xC0000142
- LONELY NETWORK #define RCX\_COMM\_LONELY\_NETWORK 0xC0000143
- DUPLICATE NODE #define RCX\_COMM\_DUPLICATE\_NODE 0xC0000144
- CABLE DISCONNECT #define RCX\_COMM\_CABLE\_DISCONNECT 0xC0000145

## Version (All Implementations)

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

- STRUCTURE VERSION #define RCX\_STATUS\_BLOCK\_VERSION 0x0001

## Watchdog Timeout (All Implementations)

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

## Host Watchdog (All Implementations)

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

## Error Count (All Implementations)

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

## Error Log Indicator (All Implementations)

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

### 3.3.1.2 Master Implementation

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

#### Master Status Structure Definition

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

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

Table 15: Master Status Structure Definition

## Slave State

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

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

Table 16: Status and Error Codes

## Slave Error Log Indicator

The error log indicator field holds the number of entries in the internal error log. If all entries are read from the log, the field is set to zero.

**Note:** This function is not yet supported.

## Number of Configured Slaves

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

## Number of Active Slaves

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

## Number of Faulted Slaves

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

### 3.3.1.3 Slave Implementation

The slave firmware uses only the common structure as outlined in section 3.2.5.1 of the Hilscher netX Dual-Port-Memory Manual.

### 3.3.2 Extended Status

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

#### netX Extended Status Field Definition Structure

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

Extended Status Block			
Offset	Type	Name	Description
0x0050	UINT8[] (the first 64 bytes correspond to CANOPEN_MASTER_GLOBAL_STATE_T)	abExtendedStatus[172]	Area containing PROFINET-IO-related information. Currently, this area is not used
0x00FC	Structure NETX_EXTENDED_STATE_FIELD_T	tExtStateField	Structure to define Status Fields and their Properties. Status type and properties are specific to protocol implementation

Table 17: Extended Status Block

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

The definition of the first structure remains specific to correspondent protocol and contains additional information about network status (i.e. flags, error counters, events etc.). The exact definition of this structure can be found in Protocol API Manual of chosen protocol implementation.

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

The Extended Status Block for PROFINET IO Controller is structured as follows:

At the location of the `bReserved` field (Address `0x0050`), no data structure is stored.

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

Extended Status Block for PROFINET IO Controller – Second part (State Field Definition Block)			
Offset	Type	Name	Description/Value
0x00FC	unsigned char	bReserved[3]	Reserved. Do not use.
0x00FF	unsigned char	bNumStateStructs	Number of State Structures defined below = 3
↓	NETX_EXTENDED_STATE_STRUCT_T	atStateStruct[0]	Structure to define State field properties
0x0100	unsigned char	bStateArea	=0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	=1. Corresponds to a bit list (one bit per node ) of configured nodes
	unsigned short	usNumOfStateEntries	=128. Corresponds to 128 bits, each representing a slave
	unsigned long	ulStateOffset	Contains an offset pointer to a state field inside input data area 0, which contains the slave configuration area
↓	NETX_EXTENDED_STATE_STRUCT_T	atStateStruct[1]	Structure to define State field properties
0x0108	unsigned char	bStateArea	=0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	=2. Corresponds to a bit list (one bit per node ) of active nodes
	unsigned short	usNumOfStateEntries	=128. Corresponds to 128 bits, each representing a slave
	unsigned long	ulStateOffset	Contains an offset pointer to a state field inside input data area 0, which contains the slave state information area
↓	NETX_EXTENDED_STATE_STRUCT_T	atStateStruct[2]	Structure to define State field properties
0x0110	unsigned char	bStateArea	=0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	=3. Corresponds to a bit list (one bit per node ) of diagnostic nodes
	unsigned short	usNumOfStateEntries	=128. Corresponds to 128 bits, each representing a slave
	unsigned long	ulStateOffset	Contains an offset pointer to a state field inside input data area 0, which contains the slave diagnostic area

Table 18: Extended Status Block for PROFINET IO Controller – Second Part (State Field Definition Block)

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

In case IOPS interface is used the extended status field will contain information about the location of this fields as well leading to up to five entries in `atStateStruct`.

For more information about the extended status of the PROFINET IO Controller stack also see manual '*DTM for Hilscher PROFINET IO Controller - Configuration of Hilscher PROFINET IO Controller*'.

The PROFINET IO Controller protocol stack may write the values shown in the following table into `bStateTypeID`:

Value	Definition /	Description
6	TLR_EXTDEVSTAT_IOPS_BYTEWISE	PROFINET Provider State Byte list
7	TLR_EXTDEVSTAT_IOPS_BITWISE	PROFINET Provider State Bit list
8	TLR_EXTDEVSTAT_IOCS_BYTEWISE	PROFINET Consumer State Byte list
9	TLR_EXTDEVSTAT_IOCS_BITWISE	PROFINET Consumer State Bit list

Table 19: Status Type IDs used by the PROFINET IO Controller Protocol Stack

### 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.

The following gives an example of the use of control and status block. The host application intends 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 20: 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 Example: DPM Layout of Input Area

Assuming a configuration with a single PROFINET-IO Device in it and both input- and output-IOPS mode enables bitwise. The IO Device shall have two Ethernet-Ports, PDEV functionality and 3 IO Modules:

- Slot 1: 8 Byte Input
- Slot 2: 3 Byte Input
- Slot 3: 8 Byte Output

For instance, this could lead to a DPM layout as displayed in *Figure 6: Example: DPM Layout of Input Area*.

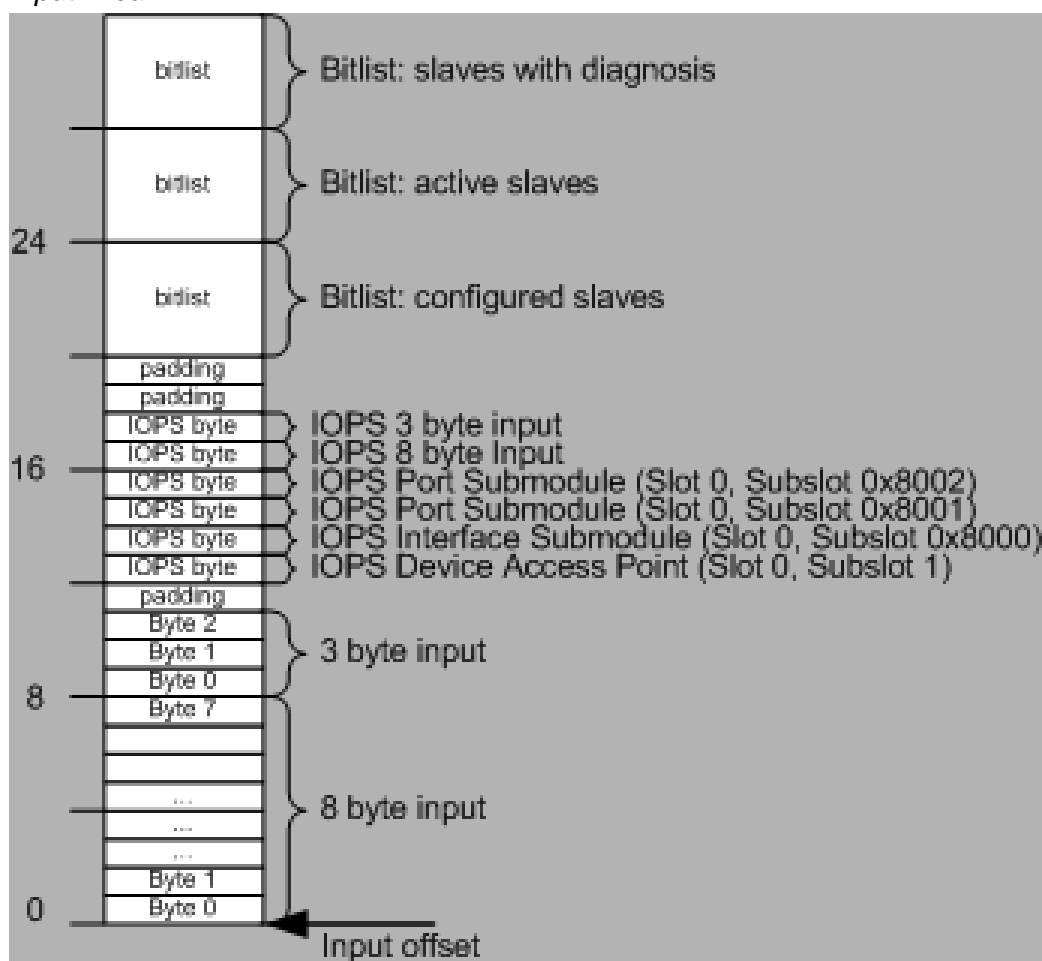


Figure 6: Example: DPM Layout of Input Area

**Note:** This is an example to understand the principle of IOPS in DPM. Creating a real configuration like this may result in a different DPM layout.

## 4 Getting started / Configuration

This section explains some essential information you should know when starting to work with the Profinet IO Controller Protocol Interface.

### 4.1 Overview about Essential Functionality

You can find the most commonly used functionality of the Profinet IO Controller Protocol API within the following sections of this document:

Topic	Section	Section
Acyclic data transfer (Records)	5.5.1	Read Service
	5.5.2	Write Service
Alarm management	5.6.3	Alarm Service
Register Application	5.4.1	Register Service

*Table 21: Overview about essential Functionality (cyclic and acyclic Data Transfer, Alarm Handling and Registering Application to Firmware)*

If no application is registered (section *Register Service* on page 119), the IO Controller will not work properly in case of incoming alarms.

## 4.2 Configuration of the Profinet IO Controller

### 4.2.1 General Configuration Structure

Figure 7 shows the bus and device structure of PROFINET IO. Every IO-Device consists of several APs that are partitioned into modules. These modules are partitioned into submodules. Each module can belong to several APs. The submodules keep one or two submodule descriptions (input, output or both) and they also can have some record data. Module 0 is reserved. Your configuration data download must follow this structure. For this purpose the downloaded packets have a special sequence and they are prefixed with a header for the addressing. The sequence of packets is based on the tree structure of the configuration data and is always from the root to the leaves. This means the configuration data of a packet always need an anchor point in a part of configuration dataset which was sent in a packet before.

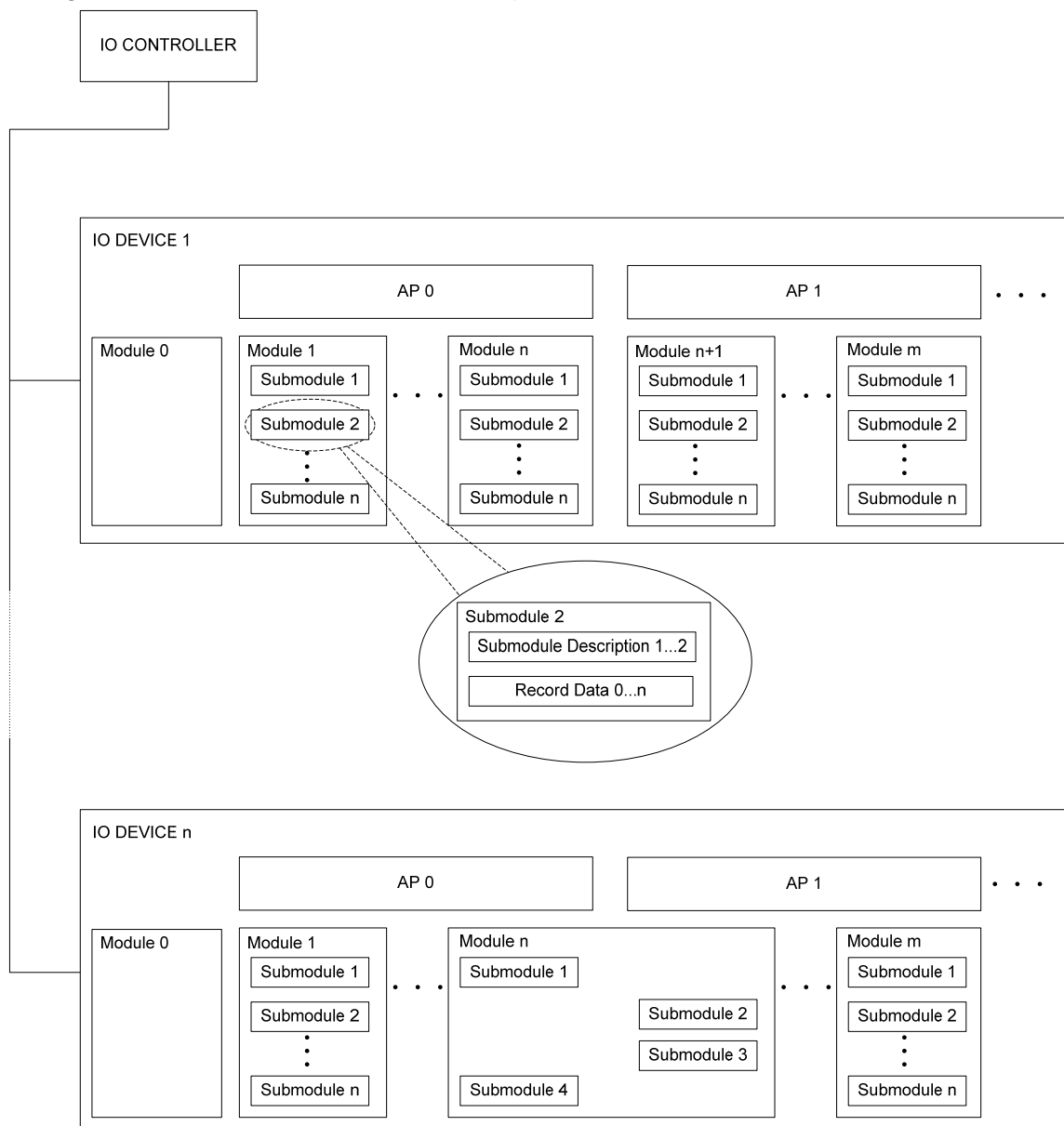


Figure 7: Bus and Device Structure in PROFINET IO

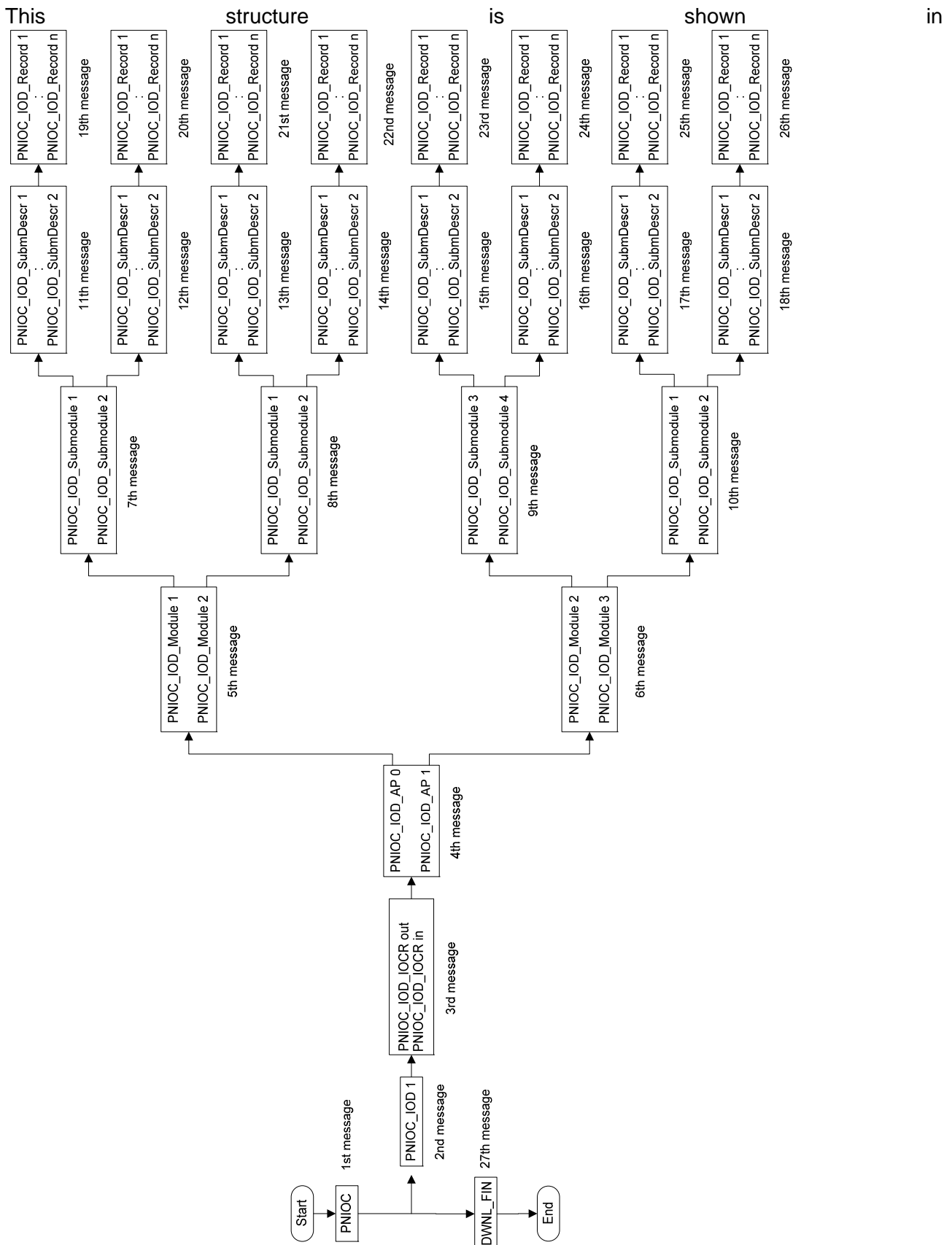
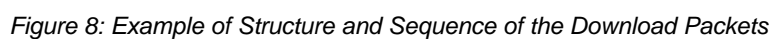


Figure 8. You can see the configuration data from one device with two APs each with an own module and one module shared by both.



This applies only to the case that no optional PNIOC\_IOD\_PORT\_SUBM and PNIOC\_IOD\_INTF\_SUBM packets are sent- Otherwise, the following picture shows the example if all of these messages are sent beginning with the fourth message (as the first messages are identical): All packets marked in blue are optional:

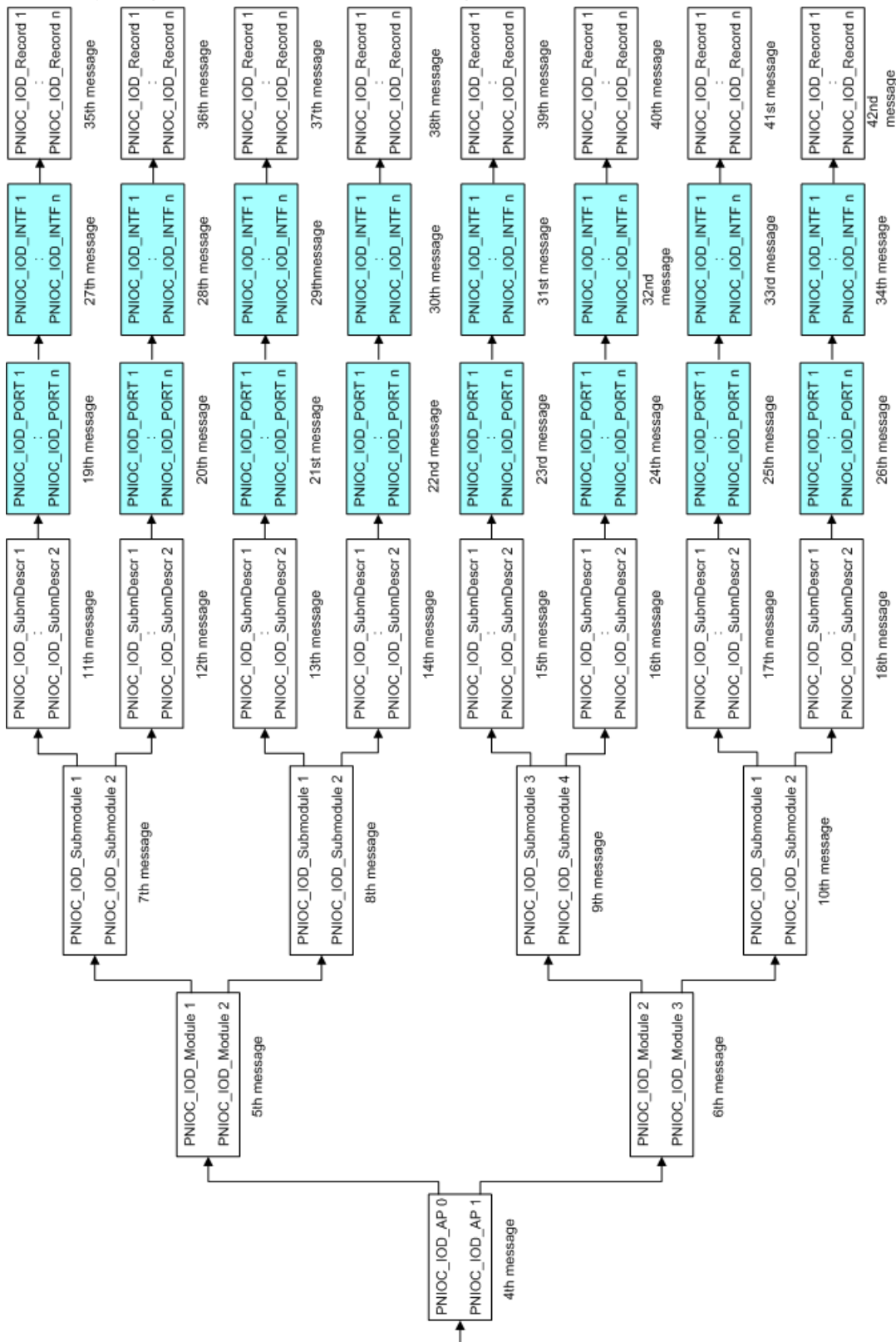


Figure 9: Example of Structure and Sequence of the Download Packets (with optional Packets)

The whole record of configuration data consists of several packets.

Figure 9: Example of Structure and Sequence of the Download Packets (with optional Packets) and the following figure display a more general view on the packets. Each box in the picture corresponds to one packet.

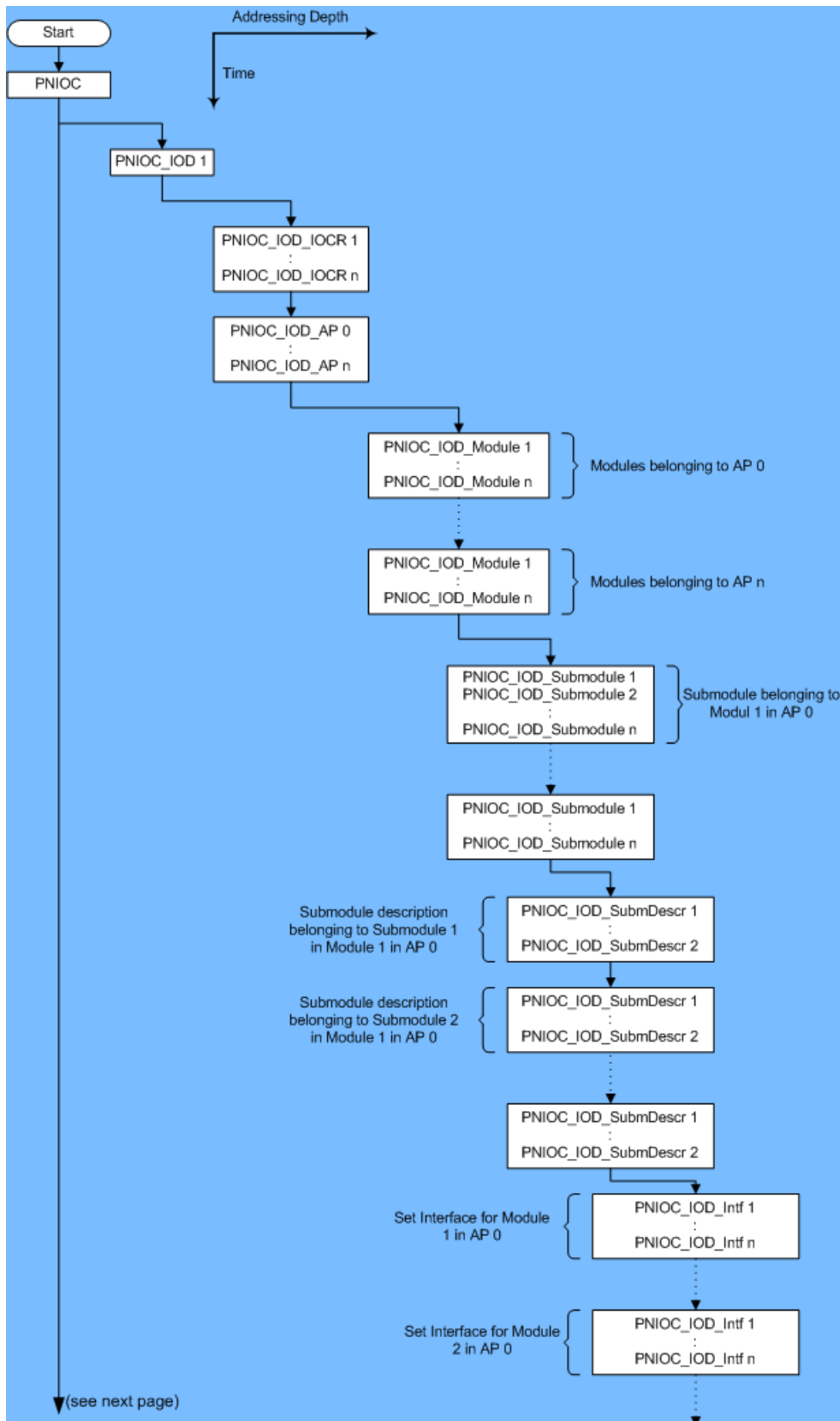


Figure 10: Sequence and Nesting of the Configuration Data (Part 1)

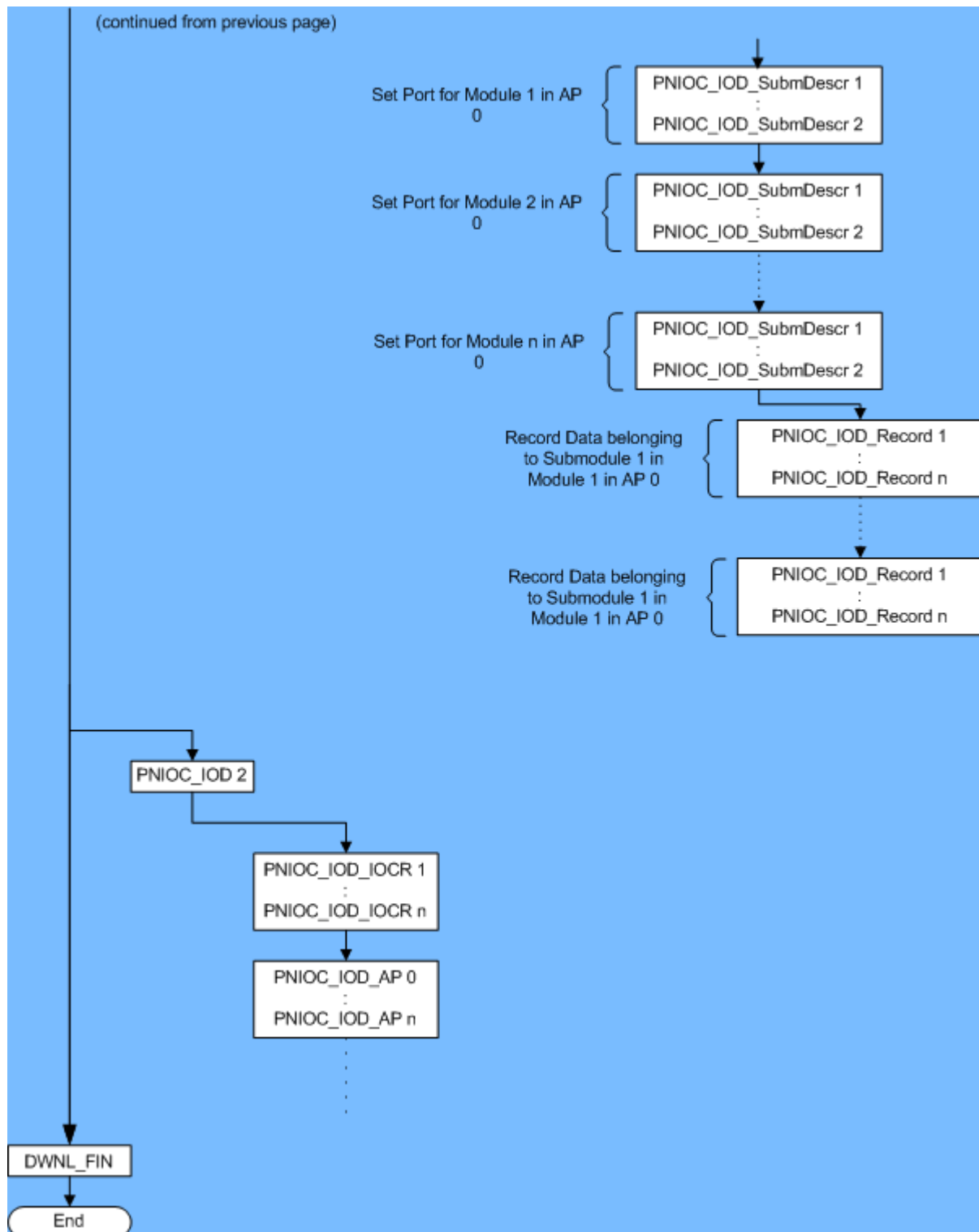


Figure 11: Sequence and Nesting of the Configuration Data (Part 2)



## 4.2.2 Using the Packet Interface with Write Access to the Dual-Port Memory

You can configure the IO Controller firmware using the packet interface described in this document. You need access via DPM to the protocol stack.

All configuration possibilities are described in section Configuration Data Packets on page 64.

First a *Configure PNM Request* (page 75) has to be sent to the protocol stack.

Similarly, in order to configure the devices and to supply them with parameters, the *Configure PNM\_IOD Request* (page 83) has to be sent to the protocol stack.

---

**Note:** During further development of the IO Controller firmware more features were introduced. They need to be configured. Therefore some configuration packets are explained in different versions inside this document. It is recommended to always use the “newest” structure definition matching the firmware version used. The newest version will be presented first in every subchapter.

---

### 4.2.3 Using the Configuration Tool SYCON.net

The easiest way to configure the IO Controller is using Hilscher's configuration tool SYCON.net.

SYCON.net creates a binary configuration file, which can be downloaded into the netX running the IO Controller firmware. The configuration tool is described in an own manual.

The general procedure is as follows:

- First, you need to create a project in SyCon.net. This is described in detail in the SyCon.net documentation (refer to document "DTM for Hilscher PROFINET IO Controller").
- Configure the bus and controller-related parameters as described in the SyCon.net documentation.
- After you completed your project, you can right-click on the icon of the PROFINET IO Controller and select "Connect".
- You will see that the name of the PROFINET IO Controller will get a green background. Now right-click on the icon again and select "Download".
- This will download the configuration files into the firmware. It is stored in a RAM Disk in a channel dependent directory ("PORT\_0" for channel 0, "PORT\_1" for channel 1, etc.).
- After the download is finished, the driver requests the PROFINET IO Controller firmware to perform a Channel-Init. All current connections will be shut down by the firmware and a restart will be performed.
- During this restart, the configuration that has been downloaded previously will be evaluated and used.
- Now the PROFINET IO Controller is first checking its own NameOfStation and IP for uniqueness before it starts to try to connect to the configured IO-Devices.

## 4.3 Port Settings concerning Cross-over (manual Configuration)

If the port configuration (parameter `usMAUTypePort0` for Ethernet port 0, `usMAUTypePort1` for Ethernet port 1) of the PROFINET IO Controller is not set to the option "Automatic" (i.e. `usMAUTypePort0/1 = 16`), the following is valid:

- Auto-Negotiation is not active.
- Port 0 is set to uncrossed operation (MDI mode).
- Port 1 is set to crossed operation (MDIX mode).

Otherwise, if the PROFINET IO Controller is set to the option "Automatic" (i.e. `usMAUTypePort0/1 = 0`), the following is valid:

- Auto-Negotiation is active.
- Auto-Crossover is active.

Also see section *Configure Extended PNM Request* on page 67 concerning this topic.

#### 4.4 Task Structure of the PROFINET IO Controller Stack

The figure below displays the internal structure of the tasks which together represent the PROFINET IO Controller Stack:

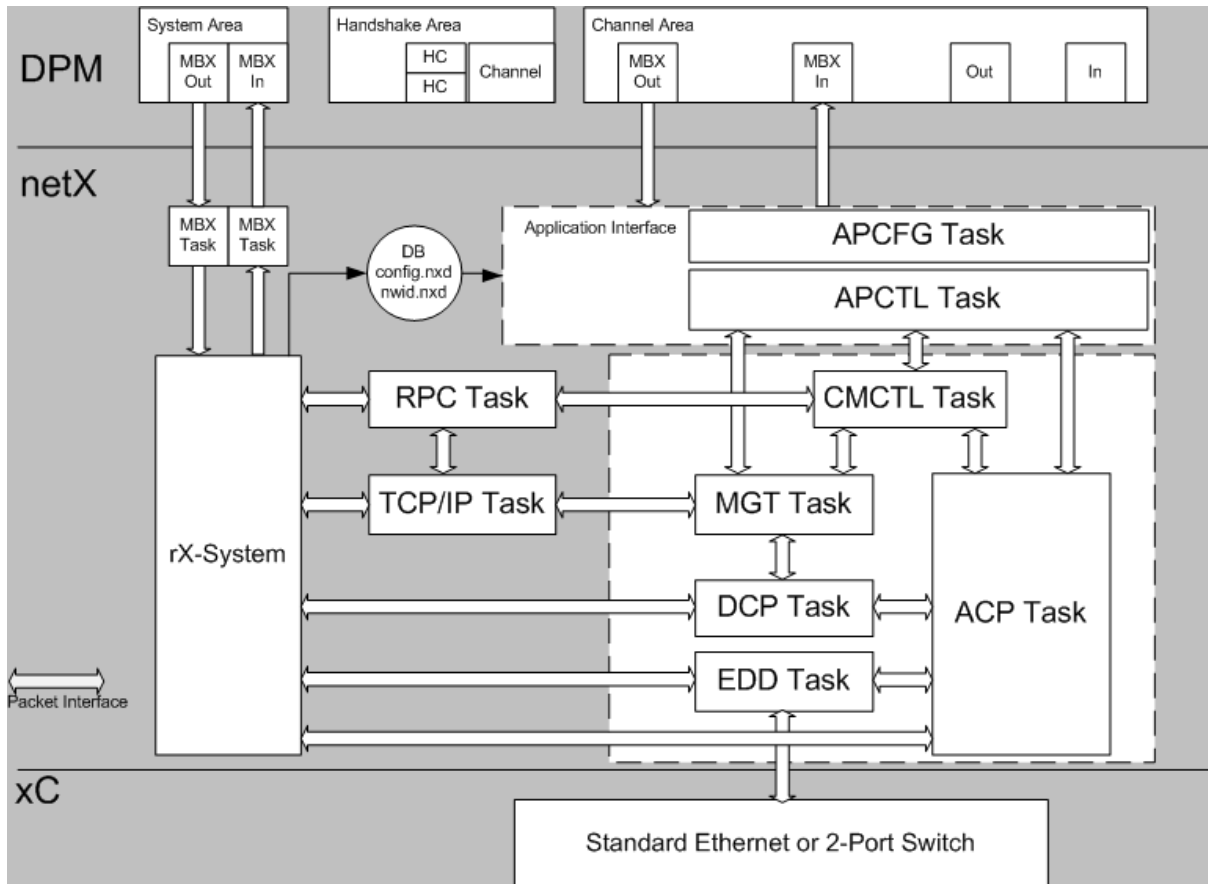


Figure 12: Task Structure of the PROFINET IO Controller Stack

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

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

The user application only accesses the tasks located in the highest layer namely the APCTL task and the APCFG task which together constitute the application interface of the PROFINET IO Controller stack.

The ACP task, CMCTL task, MGT task, DCP task and EDD task represent the core of the PROFINET IO Controller Stack.

The RPC task and the TCP/IP task are required to perform connection-less RPC using the UDP protocol.

Handling of Ethernet frames is required for the ACP, DCP, EDD and TCP/IP tasks and performed using appropriate rcX functionality.

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

### APCTL Task

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

- Handling the communication channels DPM-interface
  - Process data exchange
  - channel mailboxes
  - Watchdog
  - MasterStatusBlock
- Handling applications packets (all packets described in Protocol Interface Manual)
  - Configuration packets
  - DCP Services
  - Acyclic Read/Write Record Service
- Start connection establishment of IO-Devices
  - Provide all necessary information to CMCTL
  - Add a delay between the connection establishments of several IO-Devices to avoid high CPU load peaks and bursts of frames on the network. This is done by sending parameters for IO-Devices to CMCTL with a delay in-between two packets.
- Handling stacks indication packets
  - Connection problems
  - Alarms
- Provide information about state of every IO-Device contained in configuration

### APCFG Task

The APCFG task evaluates the data base and is used for packet configuration. In detail, it is responsible for the following:

- Evaluation of data base files
- Preparation of configuration data
- Handling applications configuration packets if data base is not used
- Providing configuration data to APCTL

After the configuration is successfully loaded and sent to APCTL the APCFG task is inactive.

### ACP Task

The ACP task is highly interconnected with the EDD task. The ACP task provides the following functionality:

- It implements the Profinet IO state machines APMS, APMR, ALPMR, PPM and CPM.
  - State machines PPM and CPM are used by both, ACP and EDD task.
- It receives all acyclic real-time telegrams with Ether Type 0x8892. These Include alarms and also DCP telegrams.
- All DCP telegrams are directed to the DCP task without any change and processed there.
- All alarm telegrams are processed in the corresponding state machine inside this task. The state machines will perform the action required to fulfill the Profinet specification. Indications about the alarms may be sent to CMCTL.

- All cyclic telegrams are received and handled by this task. However the evaluation of these cyclic telegrams is done in EDD task.
- The current link status (link working or broken) is known by this task. The ACP task informs the CMCTL and APCTL task (and therefore the user) about the current link status.

### CMCTL Task

The CMCTL task implements the context management (CMCTL state machine) defined in the PROFINET IO specification. It coordinates and controls all lower level tasks.

It is responsible for the connection establishment including the start-up of consumer and provider protocol state machines, alarm state machines and RPC Client.

For every IO-Device the CMCTL task creates an instance of CMCTL state machine.

During connection establishment that CMCTL state machine instance first advises the MGT-task to search for the IO-Device via DCP Identify Request. If the IO-Device is found the RPC client will be created.

The Profinet content of RPC-packets is created and sent via RPC to the IO-Device. This includes Connect Request, Write Requests (containing parameters) and Parameter End.

IO-Devices ApplicationReady received by RPC Task is forwarded to CMCTL and after its receipt CMCTL informs APCTL about the successful connection establishment.

If at any time an error occurs this is indicated to APCTL as well. Depending on CMCTL state machines current state the existing connection may be shut down and the other state machines are shut down, too. Afterwards the CMCTL instance is deleted.

### MGT Task

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

It configures the TCP/IP task.

It advises DCP task to create and shut down DCP state machine instances and to send different DCP telegrams.

It provides

- The resolution of names (NRPM state machine of Profinet IO specification)
- Resource management according to the RMPM state machine of Profinet IO specification)
- Internal interface to APCTL task to make it possible for the user application to directly use DCP Services

## DCP Task

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

The state machines DCPMCS, DCPMCR, DCPUCS and DCPUCR are implemented.

One instance of the DCP receiver state machines (DCPMCR, DCPUCR) is always present to handle incoming DCP requests (like DCP Identify). Their receipt is directly forwarded to MGT task.

Instances of the DCP sender state machines (DCPMCS, DCPUCS) only exist during connection establishment of an IO-Device. If the connection is successfully established the state machine instance is closed. It is possible that multiple instances of the sender state machines exist at the same time.

## EDD Task

The EDD task is highly interconnected with the ACP task. It is responsible for the following activities:

- Evaluation of all incoming cyclic telegrams (state machine CPM)
- Producing all outgoing cyclic telegrams (state machine PPM)
- Control of sending lists.

The EDD task is started once per millisecond by a timer. It performs all actions required at that point in time and sleeps afterwards.

## RPC Task

The RPC task is required for connection-less RPC

It provides:

- RPC client and RPC server 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.

## TCP/IP Task

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

## 4.5 Device Handle and obtainin detailed Information of an IO Device

### 4.5.1 Identifying an PROFINET IO Device by a Handle

The netX operating system rcX uses handles in order to identify the IO devices (slaves). A handle is not equal to the IP address of the slave.

Retrieve the handle by the *Get Slave Handle* request (`RCX_GET_SLAVE_HANDLE_REQ`, Command code `0x2F08`) which is described in [1], chapter 5.2.2.1.

The handle is necessary for

- *Read Service* page 126,
- *Write Service* page 129,
- *Device Diagnosis Service* page 135,
- *ModuleDiffBlock Service* page 143,
- *Alarm Acknowledge Service* page 167,
- *Release IO-Device Service* page 171,
- *Alarm Service* page 180,
- *Diagnosis Service* page 184.

### 4.5.2 Obtaining detailed and Diagnostic Information from connected Slaves

An application which is based on Hilscher's DPM for netX can obtain diagnostic and status information about all slaves connected to and administered by this master from the master firmware as described in general in [1]. This information only concerns cyclic data transfer. The PROFINET IO Controller firmware supports this feature.

The diagnostic information consists of three lists for the administration of:

- the configured slaves
- the activated slaves
- the slaves with a known fault

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

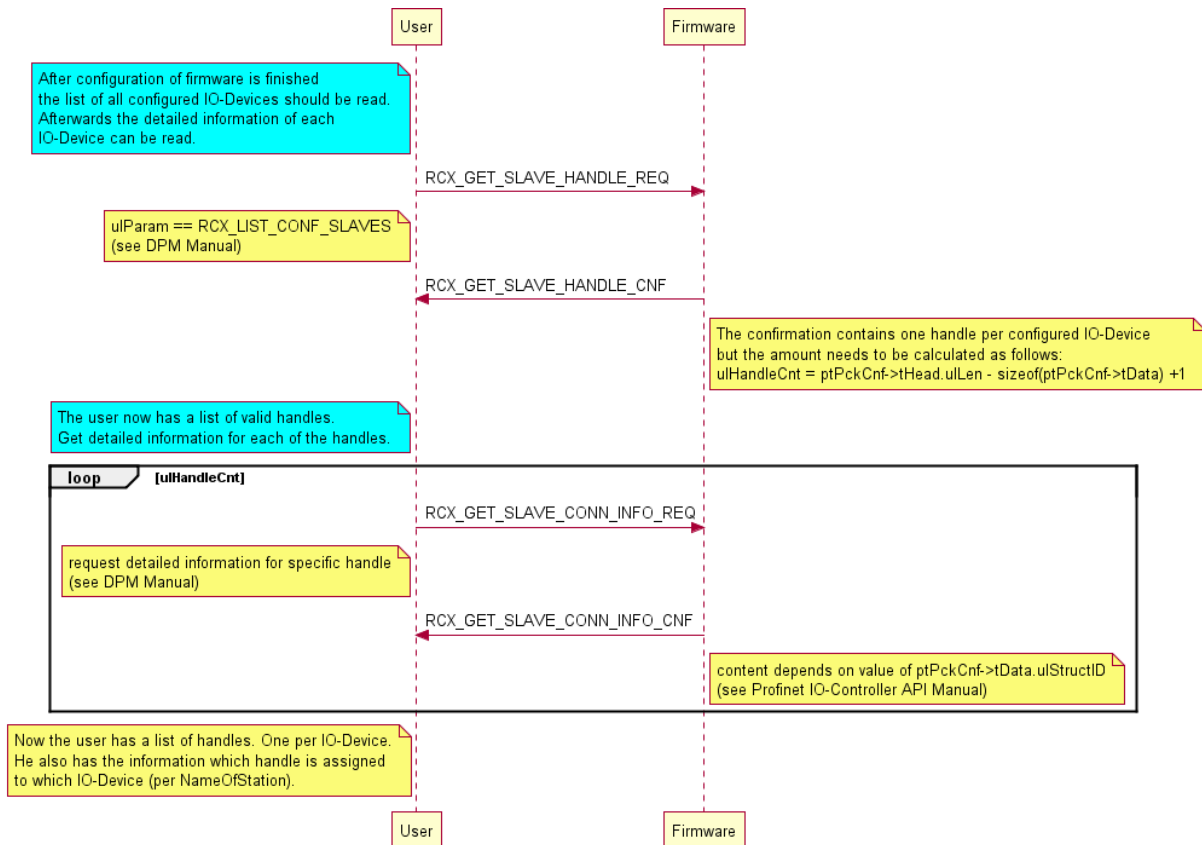


Figure 13: Sequence Diagram to get Slave Handle and Connection Information

1. Retrieve the handle by the *Get Slave Handle* request (`RCX_GET_SLAVE_HANDLE_REQ`, Command code `0x2F08`) which is described in [1], chapter 5.2.2.1. In order to do so, you need to choose which kind of list of the above mentioned slave lists you want to obtain. The confirmation packet you will receive (`RCX_GET_SLAVE_HANDLE_CNF`, Command code `0x2F09`) will then deliver an array of handles to the elements of the selected list.
2. This allows to obtain a diagnosis structure for the specific slave by the *Get Slave Connection Information* request (`RCX_GET_SLAVE_CONN_INFO_REQ`, Command code `0x2F0A`). This packet requires the handle of the specific slave taken from the array of handles to the elements of the selected list obtained in the first step. You will then receive the confirmation packet (`RCX_GET_SLAVE_CONN_INFO_CNF`, Command code `0x2F0B`) delivering – besides others – a structure `tState` containing the following structure with information about the selected IO-Device from the master firmware (see reference [1] for more details).



For an active PROFINET IO-Device this structure looks as follows:

```
typedef struct
{
    /** length of NameOfStation */
    TLR_UINT32          ulLenName;
    /** length of TypeOfStation */
    TLR_UINT32          ulLenType;
    /** IP address */
    TLR_UINT32          ulIPAddress;
    /** Flags to code diagnosis information */
    TLR_UINT32          ulDiagFlags;
    /** device ID */
    TLR_UINT16          usDeviceID;
    /** vendor ID */
    TLR_UINT16          usVendorID;
    /** MAC address */
    TLR_UINT8           abMac[PNIO_LEN_MAC_ADDR];
    /** reserved */
    TLR_UINT16          usReserved;
    /** NameOfStation */
    TLR_UINT8           abName[PNIO_MAX_NAME_OF_STATION];
    /** TypeOfStation */
    TLR_UINT8           abType[PNIO_MAX_TYPE_OF_STATION];
} APIOC_ACTIVE_DEVICE_CONNECT_INFO_T;
```

For the structure ID, the value 0x2245 will be returned in case of an active PROFINET IO-Device.

For an inactive PROFINET IO- Device this structure looks as follows:

```
typedef struct
{
    /** length of device name */
    TLR_UINT32          ulLenName;
    /** Flags to code diagnosis information */
    TLR_UINT32          ulDiagFlags;
    /** device ID */
    TLR_UINT16          usDeviceID;
    /** vendor ID */
    TLR_UINT16          usVendorID;
    /** NameOfStation */
    TLR_UINT8           abName[PNIO_MAX_NAME_OF_STATION];
} APIOC_INACTIVE_DEVICE_CONNECT_INFO_T;
```

For the structure ID, the value 0x2244 will be returned in case of an inactive PROFINET IO-Device.

## 4.6 IOPS Interface

Since firmware version v2.6.x.x the IO Controller now supports access to the Profinet Input/Output Object Provider State (IOPS). It is possible for the user to read the IOPS sent by IO-Devices to the IO Controller as well as setting the IOPS sent by IO Controller to IO-Devices. This feature can be enabled and is disabled by default.

### 4.6.1 What is IOPS?

In contrast to other realtime Ethernet systems Profinet offers the possibility to declare data valid / invalid on a submodule level. A cyclic Profinet frame containing data from a provider may contain any combination of valid and invalid data at the same time. This means that the provider of data can easily inform the consumer about the fact that e.g. only five of six submodules provide valid data. This allows the whole system to continue working if e.g. a single input module of an IO-Device fails to deliver data.

### 4.6.2 How does the interface work?

If configured IOPS received from IO-Devices is put directly into the DPM Input area so that the user gets the information if the input data is valid directly together with the input data. In the opposite direction the user declares its output data valid or invalid at the same time he provides new data to the IO Controller firmware.

The IO Controller firmware will not interpret the IOPS. This means that even if an IO-Device marks its IO-Data as invalid ("bad") the IO-Data from the cyclic frame received is copied into the DPM Input area. It is in the responsibility of the user to interpret the data for validity.

If the IOPS interface is not used the user will not be able to detect whether the data read from DPM input area is valid.

The DPM area is divided in a section containing the IO data and a section containing the IOPS. This applies for the input area as well as for the output area.

The IO Controller firmware supports three different IOPS modes which are described in the next section.

### 4.6.3 IOPS Modes

The IOPS interface supports three modes. The modes for input and output direction can be configured independently. This allows the user to e.g. read the IOPS in input area and let IO Controller firmware set the IOPS sent to IO-Devices.

#### 4.6.3.1 IOPS disabled

The IOPS is not copied to DPM / taken from DPM. In case of input area this leads to the situation that the user cannot detect if the IO data inside the input area is valid. In case of output area this forces the IO Controller to declare the output data sent to IO-Devices as valid ("good").

#### 4.6.3.2 IOPS bitwise

The IOPS is handled as a bit list in DPM. Each submodule description is represented by a single bit. If this bit is set to 1 the data is valid, if the bit is set to 0 the data is invalid.

---

**Note:** Submodules with input and output data at the same time have IOPS in input and output direction.

---

#### **4.6.3.3 IOPS bitwise**

The IOPS is handled as a byte array in DPM. Each submodule description is represented by a byte. If the byte is set to 0x80 the data is valid, otherwise it is invalid. In this mode the whole IOPS-byte is directly copied from / to the cyclic frame giving the user the possibility to access all bits of IOPS. Typically only the first bit of this IOPS byte is interesting as it stated valid or invalid. For details about the other bits see [2].

#### **4.6.4 Important Notes**

Please be aware of the fact that in case of using IOPS interface for output direction the responsibility of declaring IO-data sent to IO-Devices as valid is shifted to the user. If the IOPS of the IO-data is not set to valid an invalid IOPS will be sent by IO Controller firmware to the IO-Device. This results in the fact that the IO-Device will not use the IO-data provided (as it is invalid).

Please be aware of the fact that in case of using the IOPS interface for input direction the IO Controller does not interpret the IOPS. It simply copies it in DPM input area together with the cyclic data received in the frame. The data is even copied if the IOPS declares the data invalid. It is in the responsibility of the user to evaluate the IOPS and ignore the IO-data in case the IOPS is invalid.

## 5 The Application Interface

The configuration data are written to the Send Mailbox of the Dual-Port Memory from netX. A packet consists of three parts: a general packet header, a data header and the data itself. The packet header is used to route the packet via the Dual-Port Memory to the destination (IO Controller) and to identify the packet. The data header identifies the configuration dataset. It is used to route the configuration dataset to the right place in the tree structure. For further information about the data exchange via Dual-Port Memory see also chapter “*Dual-Port Memory Function*” in reference [DPM].

### 5.1 General Packets for the Configuration Data

The packet structure in Table 22 is common for all packets within the configuration data download.

Structure Information				Type
Variable	Type	Value / Range	Description	
<b>tHead - Structure Information</b>				
ulDest	UINT32		Destination Queue Handle	
ulSrc	UINT32		Source Queue Handle	
ulDestId	UINT32		Destination Queue Reference	
ulSrcId	UINT32		Source Queue Reference	
ulLen	UINT32		Packet Data Length (In Bytes)	
ulId	UINT32		Packet Identification As Unique Number	
ulState	UINT32		Status / Error Code	
ulCmd	UINT32		Command / Response	
ulExt	UINT32		Reserved	
ulRout	UINT32		Routing Information	
<b>tSubHead - Structure Information</b>				
ulTrCntrId	UINT32		Tree identification of the IO Controller	
ulTrDevId	UINT32		Tree identification of the IO-Device	
ulTrApId	UINT32		Tree identification of the AP	
ulTrModId	UINT32		Tree identification of the module	
ulTrSubModId	UINT32		Tree identification of the submodule	
ulTrIdCnt	UINT32		Number of equal datasets in tData	
<b>tData - Structure Information</b>				
ulTrId	UINT32		Tree identification number	
...	...	...	Configuration data set	
ulTrId	UINT32		Tree identification number	
...	...	...	Configuration data set	
...	...	...	...	

Table 22: Packet Structure

### 5.1.1 Packet Header

For a general description of the packet header see section “Dual-Port Memory Function” in reference [1].

### 5.1.2 Packet Subheader

To build the tree structure presented in *Figure 5: Transition Chart Application as Server* it is necessary to address each level from the root to the leaf. The following identifiers represent a pointer to each address level. Unused identifiers of higher levels are padded with zero.

#### Tree IO Controller Identifier

The *ulTrCntrId* field is the identification of the IO Controller address level in the tree structure. The standard value is 1.

#### Tree Device Identifier

The *ulTrDevId* field contains the identification of the IO-Device address level. Usually a IO Controller accesses several devices.

#### Tree AP Identifier

The *ulTrApId* field keeps the identification of the AP address level. Each device can provide several APs.

#### Tree Module Identifier

The *ulTrModId* field identifies the module address level.

#### Tree Submodule Identifier

The *ulTrSubModId* field contains the identifier of the submodule address level.

#### Tree Identifier Count

The *ulIdxCnt* field keeps the number of equal datasets (number of module configuration data, submodule configuration data or submodule descriptions etc.) in one packet.

### 5.1.3 Configuration Data

Each dataset in the packet is prefixed with the *ulTrId* field. This field keeps a number within the address level to have a handle to the next higher level. This value starts with number 1 and is incremented by one for each following dataset.

## 5.2 Download of large Datasets

For the reason a packet exceeds the maximum data size the packet must be partitioned. The Flowchart of the entire download including the sequenced transfer of large datasets is shown in Figure 14: Flowchart of the entire Configuration Data download. The management of the different fragments is done in the packet header. Only the first packet includes a subheader.

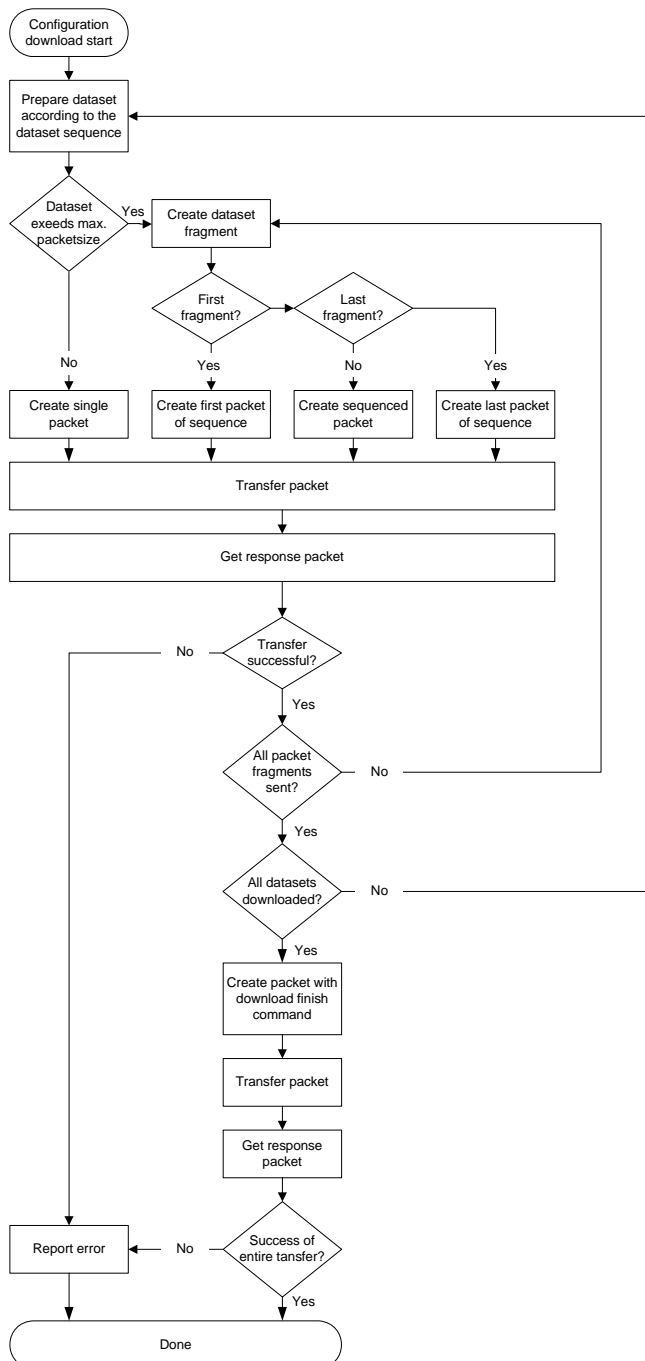


Figure 14: Flowchart of the entire Configuration Data download

If an error occurs during the download, the process is canceled by sending an error response.

## 5.2.1 General Packet Header of Request Packet

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>tHead - Structure Information</b>			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION
ulSrc	UINT32	x	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	y	Source Queue Reference
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number
ulState	UINT32	0x0000	Status
ulCmd	UINT32		Command
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension Single Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0x0000	Routing Information, Not Used
<b>tSubHead - Not present in Sequenced Packet or Last Packet of Sequence</b>			
<b>tData - Structure Information</b>			
...	...	...	...

Table 23: Packet Header of Request Packet

### Parameter ulExt

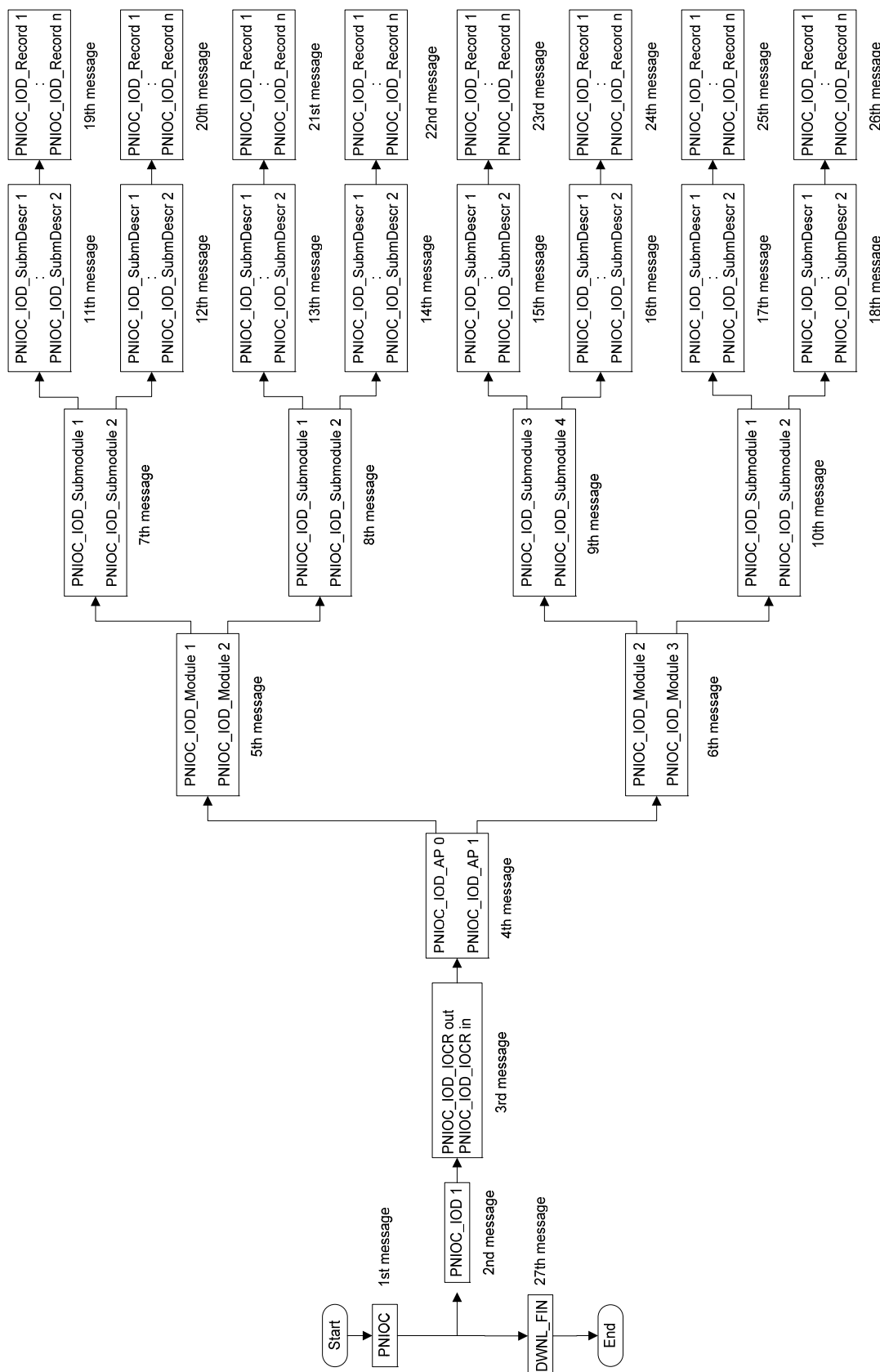
```
#define PNM_APCTL_PACKET_SEQ_NONE    0x0000
Single Packet
#define PNM_APCTL_PACKET_SEQ_FIRST   0x0080
First Packet of Sequence
#define PNM_APCTL_PACKET_SEQ_MIDDLE  0x00C0
Sequenced Packet
#define PNM_APCTL_PACKET_SEQ_LAST    0x0040
Last Packet of Sequence
```

The fragment number `ulId` starts with zero for the first data packet and is incremented by one for each following packet. Single packets use always zero in `ulId`.

## 5.2.2 Packet Header of Confirmation Packet

The returned confirmation packets are explained in the following chapter for each packet.

## 5.3 Configuration Data Packets



Each box in

Figure 8 and represents one separate packet. The figures also show the sequence of the packets and the dependencies of the data.



**Note:** All of the configuration data packets described in this section need to be executed successfully in order to perform a correct configuration. If even only one packet of this sequence of configuration packets cannot be processed correctly, the whole sequence needs to be repeated starting with the Configure PNM Request packet described just below as all configuration data that have been transmitted and accepted previously are erased in this case.

To start the configuration sequence or to delete the current configuration a ChannelInit packet (request) has to be send before the first configuration packet.

**Note:** The application program has to wait at least 100 ms between the channel init packet and the first configuration packet. As long as the channel init is not finished all configuration packets are rejected with an error.

In detail, the following functionality is provided by the configuration packets of the PROFINET IO Controller Protocol Stack:

Overview over the Configuration Packets of the PROFINET IO Controller Protocol Stack			
No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
5.3.1	Configure Extended PNM Request (structure version == 2)	0x0C42	67
	Configure Extended PNM Request (structure version == 1)	0x0C42	68
	Configure Extended PNM Confirmation	0x0C43	71
5.3.2	Configure PNM Request	0x0C30	75
	Configure PNM Confirmation	0x0C31	77
5.3.3	Configure PNM_IOD Ext_Request	0x0C44	78
	Configure PNM_IOD Ext_Confirmation	0x0C45	82
5.3.3	Configure PNM_IOD Request	0x0C32	83
	Configure PNM_IOD Confirmation	0x0C33	85
5.3.5	Configure PNM_IOD_IOCRR Request	0x0C34	86
	Configure PNM_IOD_IOCRR Confirmation	0x0C35	89
5.3.6	Configure PNM_IOD_AP Request	0x0C36	90
	Configure PNM_IOD_AP Confirmation	0x0C37	91
5.3.7	Configure PNM_IOD_Module Request	0x0C38	92
	Configure PNM_IOD_Module Confirmation	0x0C39	93
5.3.8	Configure PNM_IOD_Submodule Request	0x0C3A	94
	Configure PNM_IOD_Submodule Confirmation	0x0C3B	96
5.3.9	Configure PNM_IOD_SubmDescr Request	0x0C3C	99
	Configure PNM_IOD_Submdescr Confirmation	0x0C3D	102
5.3.11	Configure PNM_IOD_IO_Signals Request	0x6100	103
	Configure PNM_IOD_IO_Signals Confirmation	0x6101	104
5.3.12	Configure PNM_IOD_IntfSubm Request (structure version == 2)	0x0C46	105
	Configure PNM_IOD_IntfSubm Request (structure version == 1)	0x0C46	106
	Configure PNM_IOD_IntfSubm Confirmation	0x0C47	108
5.3.13	Configure PNM_IOD_PortSubm Request	0x0C48	109
	Configure PNM_IOD_PortSubm Confirmation	0x0C49	111
5.3.14	Configure PNM_IOD_RecData Request	0x0C3E	112

Overview over the Configuration Packets of the PROFINET IO Controller Protocol Stack			
No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
	Configure PNM_IOD_RecData Confirmation	0x0C3F	114
5.3.15	PNM_DWNL_FIN Request	0x0C40	115
	PNM_DWNL_FIN Confirmation	0x0C41	116

Table 24: Overview over the Configuration Packets of the PROFINET IO Controller Protocol Stack

### 5.3.1 Extended PNM (IO Controller) Dataset

The first downloaded packet of the configuration dataset is PNM. This dataset keeps IO Controller specific values.

**Note:** This service exists in different versions. The newest one shall be used but older versions are still supported for compatibility reasons.

**Note:** If one the IOxS.interface is used it is required to configure submodule descriptions using the extended service Configure PNM\_IOD\_SubmDescr\_Ext Request. The old service Configure PNM\_IOD\_SubmDescr Request does not support configuration of IOxS.

#### 5.3.1.1 Configure Extended PNM Request (structure version == 2)

Structure Information			Type: Request
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION
ulSrc	UINT32	x	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	y	Source Queue Reference
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number
ulSta	UINT32	0x0000	Status
ulCmd	UINT32	0x0C42	Command Configure extended PNM
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0x0000	Routing Information, Not Used
Structure tSubHead			
ulTrCntrId	UINT32	0	Tree identification of the IO Controller
ulTrDevId	UINT32	0	Tree identification of the IO-Device
ulTrApId	UINT32	0	Tree identification of the AP
ulTrModId	UINT32	0	Tree identification of the module
ulTrSubModId	UINT32	0	Tree identification of the submodule
ulTrIdCnt	UINT32	1	1 IO Controller dataset
Structure tData			
ulTrId	UINT32	1	Tree identification number of IO Controller dataset
ulSystemFlags	UINT32	see below	System Flags

Structure Information			Type: Request
ulWdgTime	UINT32	0..65535	Watchdog time (in milliseconds)
usTypeOfStation Len	UINT16		Length of Type of Station string
usAlign1	UINT16	0	Padding with zero
abTypeOfStation [ 240 ]	CHAR		Type of Station
usNameOfStation Len	UINT16		Length of Name of Station string
usLenOrderId	UINT16		Length of OrderId string
abNameOfStation [ 240 ]	CHAR		Name of Station
usVendorID	UINT16		Vendor identifier of the IO Controller
usDeviceID	UINT16		Device identifier of the IO- Controller
ulIPAddr	UINT32	see below	IP address
ulNetmask	UINT32	see below	Network mask
ulGateway	UINT32	see below	IP address of gateway
ulIPFlags	UINT32	see below	IP Flags
ulStructVersion	UINT32	2	Structure version.
usMAUTypePort0	UINT16	see below	MAUType to be used on EthernetPort 0
usMAUTypePort1	UINT16	see below	MAUType to be used on EthernetPort 1
abOrderId[ 20 ]	CHAR		Profinet OrderId
ulIOPSMODEConsumer	UINT32	see below	The IOPS mode for consumed data (Profinet Input data)
ulIOPSOFFSETConsumer	UINT32		The start offset in DPM Input area where the IOPS status information is copied to
ulIOPSMODEProvider	UINT32	see below	The IOPS mode for provided data (Profinet Output data)
ulIOPSOFFSETProvider	UINT32		The start offset in DPM Output area where the IOPS status information is taken from
ulIOCSMODEConsumer	UINT32	0	Reserved for future usage, not supported
ulIOCSOFFSETConsumer	UINT32	0	Reserved for future usage, not supported
ulIOCSMODEProvider	UINT32	0	Reserved for future usage, not supported
ulIOCSOFFSETProvider	UINT32	0	Reserved for future usage, not supported
ulBitlistStartOffset	UINT32		The start offset in DPM Input area where the bit lists for configured, active and faulty IO-Devices start

Table 25: Configure extended PNM Request Packet (Structure Version == 2)

### 5.3.1.2 Configure Extended PNM Request (structure version == 1)

This older request packet is described for legacy reasons only and shall not be used by new implementations.

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION

Structure Information				Type: Request
ulSrc	UINT32	x	Source Queue Handle	
ulDestId	UINT32	0x0000	Destination Queue Reference	
ulSrcId	UINT32	y	Source Queue Reference	
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence	
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number	
ulState	UINT32	0x0000	Status	
ulCmd	UINT32	0x0C42	Command Configure extended PNM	
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence	
ulRout	UINT32	0x0000	Routing Information, Not Used	
Structure tSubHead				
ulTrCntrId	UINT32	0	Tree identification of the IO Controller	
ulTrDevId	UINT32	0	Tree identification of the IO-Device	
ulTrApId	UINT32	0	Tree identification of the AP	
ulTrModId	UINT32	0	Tree identification of the module	
ulTrSubModId	UINT32	0	Tree identification of the submodule	
ulTrIdCnt	UINT32	1	1 IO Controller dataset	
Structure tData				
ulTrId	UINT32	1	Tree identification number of IO Controller dataset	
ulSystemFlags	UINT32	see below	System Flags	
ulWdgTime	UINT32	0..65535	Watchdog time (in milliseconds)	
usTypeOfStation Len	UINT16		Length of Type of Station string	
usAlign1	UINT16	0	Padding with zero	
abTypeOfStation [ 240 ]	CHAR		Type of Station	
usNameOfStation Len	UINT16		Length of Name of Station string	
usAlign2	UINT16	0	Padding with zero	
abNameOfStation [ 240 ]	CHAR		Name of Station	
usVendorID	UINT16		Vendor identifier of the IO Controller	
usDeviceID	UINT16		Device identifier of the IO- Controller	
ulIPAddr	UINT32	see below	IP address	
ulNetmask	UINT32	see below	Network mask	
ulGateway	UINT32	see below	IP address of gateway	
ulIPFlags	UINT32	see below	IP Flags	
ulStructVersion	UINT32	1	Structure version.	
usMAUTypePort0	UINT16	see below	MAU Type to be used on Ethernet Port 0	
usMAUTypePort1	UINT16	see below	MAU Type to be used on Ethernet Port 1	

Table 26: Configure extended PNM Request Packet (Structure Version == 1)

**Parameter** `ulCmd`

```
#define PNM_APCTL_CMD_SET_CONFIG_PNM_EXT_REQ    0x00000C42
```

## Parameter MAU Type

This parameter adjusts the PHY settings of an Ethernet port. Two modes are possible. Automatic settings with Auto-Negotiation and Auto-Crossover enabled is set by using the value 0. Fixed setting of 100 MBit/s FullDuplex without Auto-Negotiation and without Auto-Crossover is set by using the value 16. In that case one Ethernet port will be set to MDI mode and the other one will be set do MDIX mode.

## Parameter OrderId

This parameter represents the Profinet OrderId of the whole device which includes the Profinet IO Controller firmware (and netX hardware). This string is used by the firmware only for information purposes of other stations. E.g. a superposes engineering system might see this value when performing SNMP walk or RPC EndPointMapper Lookup to the IO Controller firmware.

## Parameters ulIOPSMODEConsumer and ulIOPSMODEProvider:

These parameters specify how to handle IOPS in input and output direction. Possible values are:

```
#define PNM_APCFG_CFG_IOXS_MODE_DISABLED 0
```

IOPS is disabled (not copied to DPM / taken from DPM).

```
#define PNM_APCFG_CFG_IOXS_MODE_BITWISE 1
```

IOPS is handled as a bit list in DPM. Each IOPS is represented by a single bit.

```
#define PNM_APCFG_CFG_IOXS_MODE_BYTEWISE 2
```

IOPS is handled as a byte list. Each IOPS is represented as a byte.

## Parameter ulBitlistStartOffset:

This parameter specified the offset in Input area where the bit lists shall start. If the user does not care about the bit lists automatic mode can be used. The firmware will put bit lists behind existing IOPS area. The offset can be read from extended status block during runtime.

```
#define PNM_APCFG_CFG_BITLIST_OFFSET_AUTOMATIC 0xFFFFFFFF
```

## Parameter ulSystemFlags - IO status length

Bit	Description
D4 ... D31	unused, set to zero
D3	I/O data format in Dual-Port Memory
D2	Length of I/O Data Status field
D1	I/O Data Status field
D0	Start mode

Table 27: System Flags

### Start mode

```
#define PNM_APCFG_STARTMODE_MASK 0x0001
Start mode mask
#define PNM_APCFG_STARTMODE_AUTO 0x0000
Auto start
#define PNM_APCFG_STARTMODE_APPLICATION 0x0001
Start by application (host)
```

I/O Data Status field in Dual-Port Memory (this feature is currently not supported)

```
#define PNM_APCFG_IODATASTAT_MASK    0x0002
I/O Data Status field mask
#define PNM_APCFG_IODATASTAT_DISABLED    0x0000
I/O Data Status field is disabled
#define PNM_APCFG_IODATASTAT_ENABLED 0x0002
I/O Data Status field is enabled
```

### Length of I/O Data Status field in Dual-Port Memory (this feature is currently not supported)

```
#define PNM_APCFG_IODATASTATLEN_MASK 0x0004
I/O Data length mask
#define PNM_APCFG_IODATASTATLEN_32BIT    0x0000
Length of I/O Data Status field is 4 byte
#define PNM_APCFG_IODATASTATLEN_8BIT 0x0004
Length of I/O Data Status field is one byte
I/O Data format in Dual-Port Memory
#define PNM_APCFG_IODATA_DPM_FORMAT_MASK    0x0008
I/O Data format mask
#define PNM_APCFG_IODATA_DPM_FORMAT_DEFAULT    0x0000
I/O Data format default value
#define PNM_APCFG_IODATA_DPM_FORMAT_MSB_FIST_BIG_ENDIAN    0x0000
I/O Data format is big endian
#define PNM_APCFG_IODATA_DPM_FORMAT_LSB_FIRST_LITTLE_ENDIAN    0x0008
I/O Data format is little endian
```

### Parameter ulFlags - IP Flag Definition

Bit	Description
D5.. D31	unused, set to zero
D4	Parameter by DHCP server
D3	Parameter by BOOTP server
D2	Gateway by ulGateway parameter
D1	Netmask by ulNetMask parameter
D0	IP address by ulIpAddr parameter

Table 28: IP Flags

### IP address by ulIpAddr parameter (this feature is currently not supported)

```
#define PNM_CFG_IPFLAG_IP_ADDR_MASK    (0x0001)
IP Address Flag Mask
#define PNM_CFG_IPFLAG_IP_ADDR_NONE    (0x0000)
The content of the ulIpAddr parameter will be ignored
#define PNM_CFG_IPFLAG_IP_ADDR_USED    (0x0001)
The content of the ulIpAddr parameter will be evaluated.
```

### Netmask by ulNetMask parameter

```
#define PNM_CFG_IPFLAG_NETMASK_MASK    (0x0002)
Net Mask Flag Mask
#define PNM_CFG_IPFLAG_NETMASK_NONE    (0x0000)
The IO Controller will assume to be an isolated host which is not connected to any
network. The ulGateway parameter will be ignored in this case.
#define PNM_CFG_IPFLAG_NETMASK_USED    (0x0002)
The content of the ulNetMask parameter will be evaluated.
```

### Gateway by ulGateway parameter

```
#define PNM_CFG_IPFLAG_GATEWAY_MASK    (0x0004)
Gateway Flag Mask
#define PNM_CFG_IPFLAG_GATEWAY_NONE    (0x0000)
The IO Controller will assume that there exists no gateway.
#define PNM_CFG_IPFLAG_GATEWAY_USED    (0x0004)
The content of the ulGateway parameter will be evaluated.
```

### Parameter by BOOTP server

```
#define PNM_CFG_IPFLAG_BOOTP_MASK    (0x0008)
BOOTP Flag Mask
```



```
#define PNM_CFG_IPFLAG_BOOTP_DISABLED      (0x0000)
Configuration via BOOTP is disabled.
#define PNM_CFG_IPFLAG_BOOTP_ENABLED (0x0008)
The IO Controller obtains its IP configuration from a BOOTP server.
```

### Parameter by DHCP server

```
#define PNM_CFG_IPFLAG_DHCP_MASK          (0x0010)
DHCP Flag Mask
#define PNM_CFG_IPFLAG_DHCP_DISABLED (0x0000)
Configuration via DHCP server is disabled.
#define PNM_CFG_IPFLAG_DHCP_ENABELD  (0x0010)
The IO Controller obtains its IP configuration from DHCP server.
```

Please note, that there exists a fallback procedure between the different configuration methods, if more than one is enabled in the `ulFlags` parameter. If enabled, the stack will first try to configure using DHCP. If DHCP configuration fails, the stack will fallback to BOOTP, if this is enabled. In case of a BOOTP failure, the values found in the `ulIpAddress`, `ulNetMask` and `ulGateway` parameters will be used, if enabled in `ulFlags`. If none of these configuration mechanisms succeeds, the stack will report an error.

### Parameter `ulIpAddress`

The stack's IP address can be configured using the `ulIpAddress` parameter. Additionally `PNM_CFG_IPFLAG_IP_ADDR_USED` must be set in `ulFlags`.

### Parameter `ulNetMask`

The `ulNetMask` parameter holds the Netmask for the subnet the device is connected to. Additionally `PNM_CFG_IPFLAG_NETMASK_USED` must be set in `ulFlags`.

### Parameter `ulGateway`

The `ulGateway` parameter stores the IP address of the default gateway. Additionally `PNM_CFG_IPFLAG_GATEWAY_USED` must be set in `ulFlags`.

### 5.3.1.3 Configure Extended PNM Confirmation

The IO Controller returns the following confirmation packet.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C43	Confirmation Configure extended PNM
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 29: Configure extended PNM Confirmation

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_EXT_PNM_CNF 0x00000C43
```

### 5.3.2 PNM (IO Controller) Dataset (legacy)

The first downloaded packet of the configuration dataset is PNM. This dataset contains IO Controller specific values.

**Note:** This packet is only described for compatibility reasons. Use the new *Extended PNM (IO Controller)* Dataset packet instead.

#### 5.3.2.1 Configure PNM Request

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION
ulSrc	UINT32	x	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	y	Source Queue Reference
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number
ulState	UINT32	0x0000	Status
ulCmd	UINT32	0x0C30	Command Configure PNM
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0x0000	Routing Information, Not Used
<b>Structure tSubHead</b>			
ulTrCntrId	UINT32	0	Tree identification of the IO Controller
ulTrDevId	UINT32	0	Tree identification of the device
ulTrApId	UINT32	0	Tree identification of the AP
ulTrModId	UINT32	0	Tree identification of the module
ulTrSubModId	UINT32	0	Tree identification of the submodule
ulTrIdCnt	UINT32	1	1 IO Controller dataset
<b>Structure tData</b>			
ulTrId	UINT32	1	Tree identification number of IO Controller dataset
ulSystemFlags	UINT32	see preceding section	System Flags
ulWdgTime	UINT32	0..65535	Watchdog time (in milliseconds)
usTypeOfStation Len	UINT16		Length of Type of Station string
usAlign1	UINT16	0	Padding with zero
abTypeOfStation [240]	CHAR		Type of Station
usNameOfStation Len	UINT16		Length of Name of Station string
usAlign2	UINT16	0	Padding with zero

Structure Information			Type: Request
abNameOfStation [ 240]	CHAR		Name of Station
usVendorID	UINT16		Vendor identifier of the IO Controller
usDeviceID	UINT16		Device identifier of the IO- Controller
ulIPAddr	UINT32	see preceding section	IP address
ulNetmask	UINT32	see preceding section	Network mask
ulGateway	UINT32	see preceding section	IP address of gateway
ulIPFlags	UINT32	see preceding section	IP Flags

Table 30: Configure PNM Request Packet

**Parameter ulCmd**

```
#define PNM_APCTL_CMD_SET_CONFIG_PNM_REQ 0x00000C30
```

For descriptions of the other parameters: see the preceding section!

### 5.3.2.2 Configure PNM Confirmation

The IO Controller returns the following confirmation packet.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C31	Confirmation Configure PNM
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 31: Configure PNM Confirmation Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_PNM_CNF 0x00000C31
```

### 5.3.3 Extended PNM\_IOD (IO-Device) Datasets

#### 5.3.3.1 Configure PNM\_IOD Ext\_Request

Structure Information				Type: Request
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION	
ulSrc	UINT32	x	Source Queue Handle	
ulDestId	UINT32	0x0000	Destination Queue Reference	
ulSrcId	UINT32	y	Source Queue Reference	
ulLen	UINT32	24 + n  n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence	
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number	
ulState	UINT32	0x0000	Status	
ulCmd	UINT32	0x00000C44	Command Configure PNM_IOD Extended	
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence	
ulRout	UINT32	0x0000	Routing Information, Not Used	
Structure tSubHead				
ulTrCntrId	UINT32	1	Tree identification of the IO Controller	
ulTrDevId	UINT32	0	Tree identification of the IO-Device	
ulTrApId	UINT32	0	Tree identification of the AP	
ulTrModId	UINT32	0	Tree identification of the module	
ulTrSubModId	UINT32	0	Tree identification of the submodule	
ulTrIdCnt	UINT32	a	Number of IO-Devices	
Structure tData				
ulTrId	UINT32	1	Tree identification number of IO-Device dataset	
ulFlags	UINT32	see below	Flags	
usNameOfStationLen	UINT16		Length of Name of Station string	
usAlign1	UINT16	0	Padding with zero	
abNameOfStation[240]	CHAR		Name of Station	
usVendorID	UINT16		Vendor identifier of the IO-Device	
usDeviceID	UINT16		Device identifier of the IO-Device	
ulIPFlags	UINT32	see below	IP flags (IP_CFG_DCP, IP_CFG_DNS)	
ulIPAddr	UINT32		IP address	
ulNetworkMask	UINT32		Network mask	
ulGatewayAddr	UINT32		IP address of gateway	
ulArUuidData1	UINT32		UUID data 1 of application relationship	

Structure Information			Type: Request
usArUuidData2	UINT16		UUID data 2 of application relationship
usArUuidData3	UINT16		UUID data 3 of application relationship
abArUuidData4[8]	UINT8		UUID data 4 of application relationship
usARType	UINT16	see below	Type of application relationship
usAlign2	UINT16	0	Padding with zero
ulARProp	UINT32	see below	Properties of application relationship
usAlarmCRType	UINT16	see below	Type of alarm CR
usAlign3	UINT16	0	Padding with zero
ulAlarmCRProp	UINT32	see below	Properties of alarm CR
usAlarmCRVLANID	UINT16		VLAN ID of alarm CR
usRTATimeoutFact	UINT16		Timeout Factor of alarms (multiple of 100ms)
usRTARetries	UINT16		Maximum number of retries of alarms
usAlign4	UINT16	0	Padding with zero
usObjUUIDInstance	UINT16		ObjectUUID_LocalIndex from GSDML
usAlign5	UINT16	0	Padding with zero
ulStructVersion	UINT32	1	Structure version
ulFSParameterMode	UINT32	see below	Parameter mode speedup. If enabled FSParmUuid shall contain a valid and unique UUID.
ulFSParamUuidData1	UINT32		UUID data 1 of parameter values
usFSParamUuidData2	UINT16		UUID data 2 of parameter values
usFSParamUuidData3	UINT16		UUID data 3 of parameter values
abFSParamUuidData4[8]	UINT8		UUID data 4 of parameter values
fIsWriteMultipleSupported	BOOL32	True False	The IO-Device supports WriteMultiple. The IO-Device does not support WriteMultiple. Note: The current IO Controller implementation does not support WriteMultiple.
ulTrId	UINT32	2	Tree identification number of IO-Device dataset (only if a>1)
...	...	...	...
ulTrId	UINT32	a	Tree identification number of IO-Device dataset
...	...	...	...

Table 32: Configure PNM\_IOD extended Request Packet

**Parameter ulCmd**

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_EXT_REQ 0x00000C44
```

**Parameter ulFlags**

```
#define PNM_APCFG_IOD_FLAG_ACTIVE    0x0001
IO-Device is activated in configuration
```

**Parameter ulIPFlags**

The following definition defines the IP Flags. The flags describe the resolution routines. This feature is currently not supported:

```
#define PNM_CFG_IOD_IPFLAG_DEFAULT  (0x0000)
#define PNM_CFG_IOD_IPFLAG_DNS      (0x0001)
IP address of IO-Device will be requested from DNS
#define PNM_CFG_IOD_IPFLAG_DCP      (0x0002)
IP parameters of IO-Device will be configured by means of DCP services
```

**Parameter usARType**

The following definition describes the AR Type according to PROFINET IO specification. The types are not defined as flags. Only one of the AR Type is possible to use:

```
#define PNIO_API_AR_TPE_SINGLE  (0x0001)
Single AR
```

**Parameter ulARProp**

The following definition describes the AR Properties according to PROFINET IO specification. The types are defined as flags. One or more flags can be combined:

- Property State (bit 0 - 2)
- Supervisor Takeover Allowed (bit 3)
- Parameterization Server (bit 4)
- Data Rate (bit 5 - 6)
- Device Access (bit 8)

Bit	Description
D9 ... D31	unused, set to zero
D8	Device Access
D7	unused, set to zero
D6.. D5	Data Rate
D4	Parameterization Server
D3	Supervisor Takeover Allowed
D2.. D0	Property State

Table 33: AR Property Flags

**Property State**

```
#define PNIO_API_AR_PROP_STATE_MASK  (0x0007)
Property State mask
#define PNIO_API_AR_PROP_STATE_BACKUP      (0x0000)
Backup AR
#define PNIO_API_AR_PROP_STATE_PRIMARY     (0x0001)
Primary AR
```



## Supervisor Takeover Allowed

```
#define PNIO_API_AR_PROP_SUPERVISOR_MASK (0x0008)
Property Supervisor Takeover Allowed mask
#define PNIO_API_AR_PROP_SUPERVISOR_NONE (0x0000)
Takeover by Supervisor not allowed
#define PNIO_API_AR_PROP_SUPERVISOR_ALLOWED (0x0008)
Takeover by Supervisor allowed
```

## Parameter usAlarmCRType

The following definition describes the Alarm CR Type according to PROFINET IO specification. The types are not defined as flags. Only one of the Alarm CR Type is possible to use:

```
#define PNIO_API_ALCR_TYPE_ALARM (0x0001)
Alarm CR
```

## Parameter ulAlarmCRProp

The following definition describes the Alarm CR Properties according to PROFINET IO specification. The types are defined as flags. One or more flags can be combined:

- Priority (bit 0)
- Transport (bit 1)

Bit	Description
D2 ... D31	unused, set to zero
D1	Transport
D0	Priority

Table 34: Alarm CR Property Flags

## Priority

```
#define PNIO_API_ALCR_PROP_PRIO_MASK (0x0001)
Priority mask
#define PNIO_API_ALCR_PROP_PRIO_DEFAULT (0x0000)
Use default priority
#define PNIO_API_ALCR_PROP_PRIO_LOW (0x0001)
Use low priority
```

## Transport (this feature is currently not supported)

```
#define PNIO_API_ALCR_PROP_TRANS_MASK (0x0002)
Transport mask
#define PNIO_API_ALCR_PROP_TRANS_DATA (0x0000)
Transport via Data-RTA-PDU
#define PNIO_API_ALCR_PROP_TRANS_UDP (0x0002)
Transport via UDP-RTA-PDU
```

**Note:** If this command fails the whole sequence of configuration commands starting with the Configure PNM Request needs to be repeated in order to perform a correct configuration. This means starting right from the beginning. See note at section 5.3 for more information.

## Parameter ulFSPParamMode

Describes the mode of the IO-Device feature Parameterization speedup. If mode is enabled the parameter FSPParamUuid shall be set. Possible values for ulFSPParamMode are:

```
#define PNM_APCFG_CFG_FSPARAM_MODE_DISABLE (0x00000000)
#define PNM_APCFG_CFG_FSPARAM_MODE_ENABLE (0x00000001)
```

### 5.3.3.2 Configure PNM\_IOD Ext Confirmation

The IO Controller returns the following confirmation packet.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C45	Confirmation Configure PNM_EXT_IOD Extended
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 35: Configure PNM\_IOD extended Confirmation Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_EXT_CNF 0x00000C45
```

### 5.3.4 PNM\_IOD (IO-Device) Datasets (legacy)

**Note:** This packet is only described for compatibility reasons. Use the new *Extended PNM\_IOD (IO-Device) Datasets* packet instead.

#### 5.3.4.1 Configure PNM\_IOD Request

Structure Information				Type: Request
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION	
ulSrc	UINT32	x	Source Queue Handle	
ulDestId	UINT32	0x0000	Destination Queue Reference	
ulSrcId	UINT32	y	Source Queue Reference	
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence	
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number	
ulState	UINT32	0x0000	Status	
ulCmd	UINT32	0x00000C32	Command Configure PNM_IOD	
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence	
ulRout	UINT32	0x0000	Routing Information, Not Used	
Structure tSubHead				
ulTrCntrId	UINT32	1	Tree identification of the IO Controller	
ulTrDevId	UINT32	0	Tree identification of the IO-Device	
ulTrApId	UINT32	0	Tree identification of the AP	
ulTrModId	UINT32	0	Tree identification of the module	
ulTrSubModId	UINT32	0	Tree identification of the submodule	
ulTrIdCnt	UINT32	a	Number of IO-Devices	
Structure tData				
ulTrId	UINT32	1	Tree identification number of IO-Device dataset	
ulFlags	UINT32	see preceding section	Flags	
usNameOfStationLen	UINT16		Length of Name of Station string	
usAlign1	UINT16	0	Padding with zero	
abNameOfStation[240]	CHAR		Name of Station	
usVendorID	UINT16		Vendor identifier of the IO-Device	
usDeviceID	UINT16		Device identifier of the IO-Device	
ulIPFlags	UINT32	see preceding section	IP flags (IP_CFG_DCP, IP_CFG_DNS)	
ulIPAddr	UINT32		IP address	

Structure Information			Type: Request
ulNetworkMask	UINT32		Network mask
ulGatewayAddr	UINT32		IP address of gateway
ulArUuidData1	UINT32		UUID data 1 of application relationship
usArUuidData2	UINT16		UUID data 2 of application relationship
usArUuidData3	UINT16		UUID data 3 of application relationship
abArUuidData4[8]	UINT8		UUID data 4 of application relationship
usARType	UINT16	see preceding section	Type of application relationship
usAlign2	UINT16	0	Padding with zero
ulARProp	UINT32	see preceding section	Properties of application relationship
usAlarmCRType	UINT16	see preceding section	Type of alarm CR
usAlign3	UINT16	0	Padding with zero
ulAlarmCRProp	UINT32	see preceding section	Properties of alarm CR
usAlarmCRVLANID	UINT16		VLAN ID of alarm CR
usRTATimeoutFact	UINT16		Timeout Factor of alarms (multiple of 100ms)
usRTARetries	UINT16		Maximum number of retries of alarms
usAlign4	UINT16	0	Padding with zero
usObjUUIDInstance	UINT16		ObjectUUID_LocallIndex from GSDML
usAlign5	UINT16	0	Padding with zero
ulTrId	UINT32	2	Tree identification number of IO-Device dataset (only if a>1)
...	...	...	...
ulTrId	UINT32	a	Tree identification number of IO-Device dataset
...	...	...	...

Table 36: Configure PNM\_IOD Request Packet

**Parameter ulCmd**

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_REQ 0x00000C32
```

### 5.3.4.2 Configure PNM\_IOD Confirmation

The IO Controller returns the following confirmation packet.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C33	Confirmation Configure PNM_IOD
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 37: Configure PNM\_IOD Confirmation Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_CNF 0x00000C33
```

## 5.3.5 PNM\_IOD\_IOCRR (Communication Relation) Datasets

### 5.3.5.1 Configure PNM\_IOD\_IOCRR Request

Structure Information				Type: Request
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION	
ulSrc	UINT32	x	Source Queue Handle	
ulDestId	UINT32	0x0000	Destination Queue Reference	
ulSrcId	UINT32	y	Source Queue Reference	
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence	
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number	
ulState	UINT32	0x0000	Status	
ulCmd	UINT32	0x00000C34	Command Configure PNM_IOD_IOCRR	
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence	
ulRout	UINT32	0x0000	Routing Information, Not Used	
Structure tSubHead				
ulTrCntrId	UINT32	1	Tree identification of the IO Controller	
ulTrDevId	UINT32	1 ... a	Tree identification of the IO-Device	
ulTrApId	UINT32	0	Tree identification of the AP	
ulTrModId	UINT32	0	Tree identification of the module	
ulTrSubModId	UINT32	0	Tree identification of the submodule	
ulTrIdCnt	UINT32	b	Number of IOCRRs	
Structure tData				
ulTrId	UINT32	1	Tree identification number of the IOCRR	
usType	UINT16	see below	Type of IOCRR	
usVLANID	UINT16		VLAN ID of IOCRR	
ulProp	UINT32	see below	Properties of IOCRR <b>Note:</b> Since firmware v2.6.x.x RT_Class_2 FrameID range (traffic class “green”) is usable and shall be used whenever possible. If the DAP GSDML file of the IO-Device contains the keyword “Supported_RTClasses” the properties shall be set to use RT_Class_2 FrameIDs.	
abMcastMACAddr[6]	UINT8		Multicast MAC address of IOCRR	
usDataLen	UINT16		Data length in frame of IOCRR	
usSendClockFact	UINT16		Send clock factor	
usReductRatio	UINT16		Reduction ratio	
usPhase	UINT16		Phase	
usSequ	UINT16		Sequence	
ulFrameSendOffs	UINT32		Frame send offset	
usWatchdogFact	UINT16		Watchdog factor	

Structure Information			Type: Request
usDataHoldFact	UINT16		Data hold factor
ulTrId	UINT32	2	Tree identification number of the IOCR (only if b>2)
...	...	...	...
ulTrId	UINT32	b	Tree identification number of the IOCR
...	...	...	...

Table 38: Configure PNM\_IOD\_IOCR request packet

**Parameter ulCmd**

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_IOCR_REQ 0x00000C34
```

**Parameter usType**

The following definition describes the IOCR Type according to PROFINET IO specification. The types are not defined as flags. Only one of the IOCR Types is possible to use:

```
#define PNIO_API_IOCR_TYPE_MASK (0x0007)
Type Mask

#define PNIO_API_IOCR_TYPE_INPUT (0x0001)
Input (consumer)

#define PNIO_API_IOCR_TYPE_OUTPUT (0x0002)
Output (provider)
```

**Parameter ulProp**

**Note:** Since firmware version v2.6.x.x the firmware supports more possible values for this parameter.

The following definition describes the IOCR Properties according to PROFINET IO specification. The types are defined as flags. One or more flags can be combined:

- RT Class FrameID selector (bit 0 - 3). This property does not specify the traffic class (e.g. “green”, “orange”, “red”) but only the FrameID range to use. The traffic class is always “green”.
- Interface Type (bit 4 - 5)
- Send Clock Synchronization (bit 6 - 8)
- Address Resolution (bit 9 - 10)

Bit	Description
D11 ... D31	unused, set to zero
D9 ... D10	Address Resolution
D6 ... D8	Send Clock Synchronization
D4 ... D5	Interface Type
D0 ... D3	RT Class

Table 39: IOCR Property Flags

**RT Class**

```
#define PNIO_API_IOCR_PROP_RTCLASS_MASK    (0x000F)
RT Class mask

#define PNIO_API_IOCR_PROP_RTCLASS_DATA1    (0x0001)
RT Class1 Data-RTC-PDU

#define PNIO_API_IOCR_PROP_RTCLASS_DATA2    (0x0002)
RT Class2 Data-RTC-PDU

#define PNIO_API_IOCR_PROP_RTCLASS_DATA3    (0x0003)
RT Class3 Data-RTC-PDU (this feature is currently not supported)

#define PNIO_API_IOCR_PROP_RTCLASS_UDP1     (0x0004)
RT Class1 UDP-RTC-PDU (this feature is currently not supported)
```

**Interface Type (this feature is currently not supported)**

```
#define PNIO_API_IOCR_PROP_IFTYPE_MASK      (0x0030)
Interface Type mask

#define PNIO_API_IOCR_PROP_IFTYPE_ZERO      (0x0000)
Zero in this version
```

**Send Clock Synchronization (this feature is currently not supported)**

```
#define PNIO_API_IOCR_PROP_SCSYNC_MASK      (0x01C0)
Send Clock Synchronization mask

#define PNIO_API_IOCR_PROP_SCSYNC_ZERO      (0x0000)
Zero in this version
```

**Address Resolution (this feature is currently not supported)**

```
#define PNIO_API_IOCR_PROP_ADDRESS_MASK     (0x0600)
Address Resolution mask

#define PNIO_API_IOCR_PROP_ADDRESS_ZERO     (0x0000)
Zero in this version
```

**Note:** If this command fails the whole sequence of configuration commands starting with the Configure PNM Request needs to be repeated in order to perform a correct configuration. This means starting right from the beginning. See note at section 5.3 for more information.



### 5.3.5.2 Configure PNM\_IOD\_IOCRR Confirmation

The IO Controller returns the following confirmation packet.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C35	Confirmation Configure PNM_IOD_IOCRR
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 40: Configure PNM\_IOD\_IOCRR Confirmation Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_IOCRR_CNF 0x00000C35
```

## 5.3.6 PNM\_IOD\_AP (Application Process) Datasets

### 5.3.6.1 Configure PNM\_IOD\_AP Request

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION
ulSrc	UINT32	x	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	y	Source Queue Reference
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number
ulState	UINT32	0x0000	Status
ulCmd	UINT32	0x00000C36	Command Configure PNM_IOD_AP
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0x0000	Routing Information, Not Used
<b>Structure tSubHead</b>			
ulTrCntrId	UINT32	1	Tree identification of the IO Controller
ulTrDevId	UINT32	1 ... a	Tree identification of the IO-Device
ulTrApId	UINT32	0	Tree identification of the AP
ulTrModId	UINT32	0	Tree identification of the module
ulTrSubModId	UINT32	0	Tree identification of the submodule
ulTrIdCnt	UINT32	c	Number of APs
<b>Structure tData</b>			
ulTrId	UINT32	1	Tree identification number of AP
ulAPI	UINT32		Application Process Identifier
ulTrId	UINT32	2	Tree identification number of AP (only if c>1)
...	...	...	...
ulTrId	UINT32	c	Tree identification number of API
...	...	...	...

Table 41: Configure PNM\_IOD\_AP Request Packet

**Parameter ulCmd**

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_AP_REQ 0x00000C36
```

**Note:** If this command fails the whole sequence of configuration commands starting with the Configure PNM Request needs to be repeated in order to perform a correct configuration. This means starting right from the beginning. See note at section 5.3 for more information.

**5.3.6.2 Configure PNM\_IOD\_AP Confirmation**

The IO Controller returns the following confirmation packet.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C37	Confirmation Configure PNM_IOD_AP
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 42: Configure PNM\_IOD\_AP Confirmation Packet

**Parameter ulCmd**

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_AP_CNF 0x00000C37
```

## 5.3.7 PNM\_IOD\_Module (Module) Datasets

### 5.3.7.1 Configure PNM\_IOD\_Module Request

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION
ulSrc	UINT32	x	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	y	Source Queue Reference
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number
ulState	UINT32	0x0000	Status
ulCmd	UINT32	0x00000C38	Command Configure PNM_IOD_Module
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0x0000	Routing Information, Not Used
<b>Structure tSubHead</b>			
ulTrCntrId	UINT32	1	Tree identification of the IO Controller
ulTrDevId	UINT32	1 ... a	Tree identification of the IO-Device
ulTrApId	UINT32	1 ... c	Tree identification of the AP
ulTrModId	UINT32	0	Tree identification of the module
ulTrSubModId	UINT32	0	Tree identification of the submodule
ulTrIdCnt	UINT32	d	Number modules
<b>Structure tData</b>			
ulTrId	UINT32	1	Tree identification number of module dataset
ulModuleID	UINT32		Module Ident number
usModuleProp	UINT16		Module properties
usSlotNumber	UINT16		Slot number
ulTrId	UINT32	2	Tree identification number of module dataset (only if d>1)
...	...	...	...
ulTrId	UINT32	d	Tree identification number of module dataset
...	...	...	...

Table 43: Configure PNM\_IOD\_Module Request Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_MODULE_REQ 0x00000C38
```

#### Parameter usModuleProp

Not defined yet!

**Note:** If this command fails the whole sequence of configuration commands starting with the Configure PNM Request needs to be repeated in order to perform a correct configuration. This means starting right from the beginning. See note at section 5.3 for more information.

### 5.3.7.2 Configure PNM\_IOD\_Module Confirmation

The IO Controller returns the following confirmation packet.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C39	Confirmation Configure PNM_IOD_Module
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 44: Configure PNM\_IOD\_Module Confirmation Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_MODULE_CNF 0x00000C39
```

## 5.3.8 PNM\_IOD\_Submodule (Submodule) Datasets

### 5.3.8.1 Configure PNM\_IOD\_Submodule Request

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION
ulSrc	UINT32	x	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	y	Source Queue Reference
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number
ulState	UINT32	0x0000	Status
ulCmd	UINT32	0x00000C3A	Command Configure PNM_IOD_Submodule
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0x0000	Routing Information, Not Used
<b>Structure tSubHead</b>			
ulTrCntrId	UINT32	1	Tree identification of the IO Controller
ulTrDevId	UINT32	1 ... a	Tree identification of the IO-Device
ulTrApId	UINT32	1 ... c	Tree identification of the AP
ulTrModId	UINT32	1 ... d	Tree identification of the module
ulTrSubModId	UINT32	0	Tree identification of the submodule
ulTrIdCnt	UINT32	e	Number of submodules
<b>Structure tData</b>			
ulTrId	UINT32	1	Tree identification number of submodule dataset
ulSubmoduleID	UINT32		Submodule Ident number
usSubmoduleProp	UINT16	see below	Submodule properties
usSubslotNumber	UINT16		Subslot number
ulTrId	UINT32	2	Tree identification number of submodule dataset (only if e>1)
...	...	...	...
ulTrId	UINT32	e	Tree identification number of submodule dataset
...	...	...	...

Table 45: Configure PNM\_IOD\_Submodule Request Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_SUBMODULE_REQ 0x00000C3A
```

## Parameter usSubmoduleProp

The following definition describes the Submodule Properties according to PROFINET IO specification. The types are defined as flags. One or more flags can be combined:

- Type (bit 0 - 1)
- Shared Input (bit 2)
- Reduced Input Submodule Data Length (bit 3)
- Reduced Output Submodule Data Length (bit 4)
- Discard IOXS (bit 5)

Bit	Description
D6 ... D31	unused, set to zero
D5	Discard IOXS
D4	Reduced Output Submodule Data Length
D3	Reduced Input Submodule Data Length
D2	Shared Input
D1 ... D0	Type

Table 46: Submodule Property Flags

## Type

```
#define PNIO_API_SUBM_PROP_TYPE_MASK (0x0003)
Type mask

#define PNIO_API_SUBM_PROP_TYPE_NONE (0x0000)
No input and no output

#define PNIO_API_SUBM_PROP_TYPE_INPUT      (0x0001)
Input

#define PNIO_API_SUBM_PROP_TYPE_OUTPUT     (0x0002)
Output

#define PNIO_API_SUBM_PROP_TYPE_BOTH (0x0003)
Input and output
```

**Note:** If this command fails, the whole sequence of configuration commands starting with the Configure PNM Request needs to be repeated in order to perform a correct configuration. This means starting right from the beginning. See note at section 5.3 for more information.

### 5.3.8.2 Configure PNM\_IOD\_Submodule Confirmation

The Profinet-IO Controller returns the following confirmation packet.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C3B	Confirmation Configure PNM_IOD_Submodule
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 47: Configure PNM\_IOD\_Submodule confirmation packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_SUBMODULE_CNF 0x00000C3B
```



### 5.3.9 PNM\_IOD\_SubmDescr\_Ext (Extended Submodule Description) Datasets

This dataset replaces the deprecated PNM\_IOD\_SubmDescr Dataset. Whenever possible this new dataset shall be used. The old dataset is still supported by the firmware for compatibility reasons. Only this extended dataset allows configuration and usage of IOPS in Dual Port Memory.

**Note:** The extended service Configure PNM\_IOD\_SubmDescr\_Ext Request shall be used if application wants to access IOxS.

#### 5.3.9.1 Configure PNM\_IOD\_SubmDescr\_Ext Request

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION
ulSrc	UINT32	x	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	y	Source Queue Reference
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number
ulState	UINT32	0x0000	Status
ulCmd	UINT32	0x00000C4A	Command Configure PNM_IOD_SubmDescr_Ext
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0x0000	Routing Information, Not Used
<b>Structure tSubHead</b>			
ulTrCntrId	UINT32	1	Tree identification of the IO Controller
ulTrDevId	UINT32	1 ... a	Tree identification of the IO-Device
ulTrApId	UINT32	1 ... c	Tree identification of the AP
ulTrModId	UINT32	1 ... d	Tree identification of the module
ulTrSubModId	UINT32	1 ... e	Tree identification of the submodule
ulTrIdCnt	UINT32	f	Number of submodule descriptions
<b>Structure tData</b>			
ulTrId	UINT32	1	Tree identification number of submodule description dataset
usDataDescr	UINT16	see below	Data description
usSubmDataLen	UINT16		Length of submodule data
ulIOCRIdProd	UINT32	see the note below	Link to associated producer IOCR referenced with ulTrId of the IOCRs
ulIOCRIdCons	UINT32	see the note below	Link to associated consumer IOCR referenced with ulTrId of the IOCRs

Structure Information			Type: Request
bIOPSLen	UINT8	1	Length of IOPS
bIOCSLen	UINT8	1	Length of IOCS
usAlign1	UINT16	0	Padding with zero
usFrameOffs	UINT16		Frame offset of data object
usIOCSFrameOffs	UINT16		Frame offset of IOCS
ulIO_Block	UINT32		IO-Block in DPM
ulDPM_Offset	UINT32		Offset
ulSignla_Attrib	UINT32		reserved
ulStructVersion	UINT32	1	Structure Version of this extended dataset
ulDPM_OffsetIop sBit	UINT32		Bit-offset in DPM where the IOPS for this submodule description shall be copied to / taken from. Note: If IOPS is configured in byte-mode this field still contains Bit-offsets (e.g. byte 1 has Bit-offset 8).
ulDPM_OffsetIoc sBit	UINT32	0	Reserved for future usage, not supported
ulTrId	UINT32	2	Tree identification number of submodule description dataset (only if f>1)
...	...	...	...
ulTrId	UINT32	f	Tree identification number of submodule description dataset
...	...	...	...

Table 48: Configure PNM\_IOD\_SubmDescr\_Ext Request

**Parameter ulCmd**

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_SUBMDESCR_EXT_REQ 0x00000C4A
```

**Parameter usDataDescr**

Switch between input and output Data. Only one of the Data Descriptions is possible to use:

```
#define PNIO_API_SUBMDESCR_IO_MASK (0x0003)
Description Mask

#define PNIO_API_SUBMDESCR_IO_INPUT (0x0001)
Input

#define PNIO_API_SUBMDESCR_IO_OUTPUT (0x0002)
Output
```

**Note:** The *ulTrId* of the communication relation datasets (PNM\_IOD\_IOCRR; see chapter 5.3.5) is used as a link in the submodule description datasets above. Because each submodule description must know in/out which communication relation to put/get its data.

**Note:** If this command fails the whole sequence of configuration commands starting with the Configure PNM Request needs to be repeated in order to perform a correct configuration. This means starting right from the beginning. See note at section 5.3 for more information.

### 5.3.9.2 Configure PNM\_IOD\_Submdescr Confirmation

The Profinet IO Controller returns the following confirmation packet.

Structure Information				Type: Confirmation
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM	
ulSrc	UINT32	From Request	Source Queue Handle	
ulDestId	UINT32	0x0000	Destination Queue Reference	
ulSrcId	UINT32	From Request	Source Queue Reference	
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data	
ulId	UINT32	From Request	Packet Identification as Unique Number	
ulState	UINT32	see below	Status / Error Code	
ulCmd	UINT32	0x00000C4B	Confirmation Configure PNM_IOD_SubmDescrExt	
ulExt	UINT32	From Request	Extension	
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use	

Table 49: Configure PNM\_IOD\_Submdescr Confirmation

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_SUBMDESCR_EXT_CNF 0x00000C4B
```

### 5.3.10 PNM\_IOD\_SubmDescr (Submodule Description) Datasets

This deprecated service is only shown for compatibility reasons. For new implementations the new PNM\_IOD\_SubmDescr\_Ext dataset service shall be used.

#### 5.3.10.1 Configure PNM\_IOD\_SubmDescr Request

Structure Information			Type: Request
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION
ulSrc	UINT32	x	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	y	Source Queue Reference
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number
ulState	UINT32	0x0000	Status
ulCmd	UINT32	0x00000C3C	Command Configure PNM_IOD_SubmDescr

Structure Information			Type: Request
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0x0000	Routing Information, Not Used
Structure tSubHead			
ulTrCntrId	UINT32	1	Tree identification of the IO Controller
ulTrDevId	UINT32	1 ... a	Tree identification of the IO-Device
ulTrApId	UINT32	1 ... c	Tree identification of the AP
ulTrModId	UINT32	1 ... d	Tree identification of the module
ulTrSubModId	UINT32	1 ... e	Tree identification of the submodule
ulTrIdCnt	UINT32	f	Number of submodule descriptions
Structure tData			
ulTrId	UINT32	1	Tree identification number of submodule description dataset
usDataDescr	UINT16	see below	Data description
usSubmDataLen	UINT16		Length of submodule data
ulIOCRIdProd	UINT32	see the note below	Link to associated producer IOCR referenced with ulTrId of the IOCRs
ulIOCRIdCons	UINT32	see the note below	Link to associated consumer IOCR referenced with ulTrId of the IOCRs
bIOPSLen	UINT8		Length of IOPS
bIOCSLen	UINT8		Length of IOCS
usAlign1	UINT16	0	Padding with zero
usFrameOffs	UINT16		Frame offset of data object
usIOCSFrameOffs	UINT16		Frame offset of IOCS
ulIO_Block	UINT32		IO-Block in DPM
ulDPM_Offset	UINT32		Offset
ulSignla_Attrib	UINT32		reserved
ulTrId	UINT32	2	Tree identification number of submodule description dataset (only if f>1)
...	...	...	...
ulTrId	UINT32	f	Tree identification number of submodule description dataset
...	...	...	...

Table 50: Configure PNM\_IOD\_SubmDescr request packet

**Parameter ulCmd**

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_SUBMDESCR_REQ 0x00000C3C
```

**Parameter usDataDescr**

Switch between input and output Data. Only one of the Data Descriptions is possible to use:

```
#define PNIO_API_SUBMDESCR_IO_MASK (0x0003)
Description Mask

#define PNIO_API_SUBMDESCR_IO_INPUT (0x0001)
Input

#define PNIO_API_SUBMDESCR_IO_OUTPUT (0x0002)
Output
```

---

**Note:** The *ulTrId* of the communication relation datasets (PNM\_IOD\_IOCRR; see chapter 5.3.5) is used as a link in the submodule description datasets above. Because each submodule description must know in/out which communication relation to put/get its data.

---

---

**Note:** If this command fails the whole sequence of configuration commands starting with the Configure PNM Request needs to be repeated in order to perform a correct configuration. This means starting right from the beginning. See note at section 5.3 for more information.

---

### 5.3.10.2 Configure PNM\_IOD\_Submdescr Confirmation

The Profinet IO Controller returns the following confirmation packet.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C3D	Confirmation Configure PNM_IOD_SubmDescr
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 51: Configure PNM\_IOD\_SubmDescr Confirmation Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_SUBMDESCR_CNF 0x00000C3D
```

### 5.3.11 PNM\_IOD\_IO\_Signals (SubmDescr Signal Configuration) Dataset

Using this optional dataset it is possible to configure the individual signals which together form the I/O-data of a submodule description dataset. Without this information the Profinet IO Controller stack is not able to adjust the byte order of I/O-data in Dual Port Memory.

If this dataset is not configured the I/O-data of the submodule description will be treated as byte array with the length defined in submodule descriptions field `usSubmDataLen`.

For each data direction of a submodule a separate configuration packet for the signals has to be sent.

#### Parameter `atSignals[n]`

The following definition describes the signal table entry according to PROFINET IO specification. For each new signal a new such entry needs to be added. In this context, *n* is the number entries in the array, i.e. the value of `ulTotalSignalCount`.

<code>bSignalType</code>	UINT8	0 ... 39	Type of the first signal
<code>bSignalAmount</code>	UINT8	1..255	Amount of this signal (i.e. the count how often this signal appears in this configuration)

#### 5.3.11.1 Configure PNM\_IOD\_IO\_Signals Request

Structure Information				Type: Request
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION	
ulSrc	UINT32	x	Source Queue Handle	
ulDestId	UINT32	0x0000	Destination Queue Reference	
ulSrcId	UINT32	Y	Source Queue Reference	
ulLen	UINT32	42 + n  n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence	
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number	
ulState	UINT32	0x0000	Status	
ulCmd	UINT32	0x00006100	Command IO_SIGNALS_CONFIGURE_SIGNAL_REQ	
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence	
ulRout	UINT32	0x0000	Routing Information, Not Used	
Structure tSubHead				
ulTrCntrId	UINT32	1	Tree identification of the IO Controller	
ulTrDevId	UINT32	1 ... a	Tree identification of the IO-Device	
ulTrApId	UINT32	1 ... c	Tree identification of the AP	
ulTrModId	UINT32	1 ... d	Tree identification of the module	

Structure Information			Type: Request
ulTrSubModId	UINT32	1 ... e	Tree identification of the submodule
ulTrIdCnt	UINT32	f	Number of submodule descriptions
Structure tData			
ulFieldbusSpecific7	UINT32	0	Reserved
ulFieldbusSpecific8	UINT32	0	Reserved
ulSignalDirection	UINT32	1 ... 2	The direction of the signals described in this dataset. 1: Input (IO_SIGNALS_DIRECTION_CONSUMER) 2:Output (IO_SIGNALS_DIRECTION_PROVIDER)
ulTotalSignalCount	UINT32		Number of entries in atSignals In most cases, but not always, this value is equal to the number of signals.
atSignals[n]	UINT8[]		Array containing signal information (signal type and amount of signals), see above n means the value of ulTotalSignalCount here.

Table 52: Configure PNM\_IOD\_IO\_Signals Request Packet

### 5.3.11.2 Configure PNM\_IOD\_IO\_Signals Confirmation

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00006101	Command IO_SIGNALS_CONFIGURE_SIGNAL_CNF
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 53: Configure PNM\_IOD\_IO\_Signals Confirmation Packet



### 5.3.12 PNM\_IOD\_IntfSubm (Interface Submodule) Datasets

This service is used to additionally configure InterfaceSubmodules which are described in the GSDML file. It shall only be used to configure those special submodules. The usage of this service is optional. If no feature configured with this packet shall be used the packet can be skipped.

**Note:** This service exists in different versions. The newest one shall be used but older versions are still supported for compatibility reasons.

#### 5.3.12.1 Configure PNM\_IOD\_IntfSubm Request (structure version == 2)

Structure Information				Type: Request
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION	
ulSrc	UINT32	x	Source Queue Handle	
ulDestId	UINT32	0x0000	Destination Queue Reference	
ulSrcId	UINT32	Y	Source Queue Reference	
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence	
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number	
ulState	UINT32	0x0000	Status	
ulCmd	UINT32	0x00000C46	Command Configure PNM_IOD_IntfSubm Request	
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence	
ulRout	UINT32	0x0000	Routing Information, Not Used	
Structure tSubHead				
ulTrCntrId	UINT32	1	Tree identification of the IO Controller	
ulTrDevId	UINT32	1 ... a	Tree identification of the IO-Device	
ulTrApId	UINT32	1 ... c	Tree identification of the AP	
ulTrModId	UINT32	1 ... d	Tree identification of the module	
ulTrSubModId	UINT32	1 ... e	Tree identification of the submodule	
ulTrIdCnt	UINT32	1	Interface submodule	
Structure tData				
ulTrId	UINT32	1	Tree Identifier of the Interface submodule	
ulStructVersion	UINT32	2	Structure Version	
fEnableFsu	BOOL32	True False	IO-Device shall be configured to issue Hello Requests. IO-Device shall be configured to not issue Hello Requests.	
ulFSHelloMode	UINT32	see below	Hello mode	
ulFSHelloInterval	UINT32	see below	Hello interval	
ulFSHelloRetry	UINT32	1..15	Hello retry	
ulFSHelloDelay	UINT32	see below	Hello delay	
usMrpRole	UINT16	See below	MRP Role	

Table 54: Configure PNM\_IOD Interface Submodule Request Packet with Structure Version == 2

**5.3.12.2 Configure PNM\_IOD\_IntfSubm Request (structure version == 1)**

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION
ulSrc	UINT32	x	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	Y	Source Queue Reference
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number
ulState	UINT32	0x0000	Status
ulCmd	UINT32	0x00000C46	Command Configure PNM_IOD_IntfSubm Request
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0x0000	Routing Information, Not Used
<b>Structure tSubHead</b>			
ulTrCntrId	UINT32	1	Tree identification of the IO Controller
ulTrDevId	UINT32	1 ... a	Tree identification of the IO-Device
ulTrApId	UINT32	1 ... c	Tree identification of the AP
ulTrModId	UINT32	1 ... d	Tree identification of the module
ulTrSubModId	UINT32	1 ... e	Tree identification of the submodule
ulTrIdCnt	UINT32	1	Interface submodule
<b>Structure tData</b>			
ulTrId	UINT32	1	Tree Identifier of the Interface submodule
ulStructVersion	UINT32	1	Structure Version
fEnableFsu	BOOL32	True False	IO-Device shall be configured to issue Hello Requests. IO-Device shall be configured to not issue Hello Requests.
ulFSHelloMode	UINT32	see below	Hello mode
ulFSHelloInterval	UINT32	see below	Hello interval
ulFSHelloRetry	UINT32	1..15	Hello retry
ulFSHelloDelay	UINT32	see below	Hello delay

Table 55: Configure PNM\_IOD Interface Submodule Request Packet with Structure Version == 1

**Parameter ulCmd**

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_INTF_SUBM_REQ 0x00000C46
```

**Parameter ulFSHelloMode**

This parameter specifies the HelloMode to be used. Possible values are

```
#define PNM_APCFG_CFG_HELLO_MODE_OFF 0
#define PNM_APCFG_CFG_HELLO_MODE_HELLO_ON_LINKUP 1
#define PNM_APCFG_CFG_HELLO_MODE_HELLO_AFTER_DELAY 2
```

**Parameter ulFSHelloInterval**

This parameter specifies the time interval between sending the Hello Requests. Possible values are

```
#define PNM_APCFG_CFG_HELLO_INTERVAL_30MS 30
#define PNM_APCFG_CFG_HELLO_INTERVAL_50MS 50
#define PNM_APCFG_CFG_HELLO_INTERVAL_100MS 100
#define PNM_APCFG_CFG_HELLO_INTERVAL_300MS 300
#define PNM_APCFG_CFG_HELLO_INTERVAL_500MS 500
#define PNM_APCFG_CFG_HELLO_INTERVAL_1000MS 1000
```

**Parameter ulFSHelloRetry**

This parameter specifies the amount of Hello Requests the IO-Device shall send.

**Parameter ulFSHelloDelay**

This parameter specifies the time the IO-Device shall wait after detecting LinkUp before issuing the first Hello Request. This parameter is only used if ulFSHelloMode is set to PNM\_APCFG\_CFG\_HELLO\_MODE\_HELLO\_AFTER\_DELAY. Possible values are

```
#define PNM_APCFG_CFG_HELLO_DELAY_OFF 0
#define PNM_APCFG_CFG_HELLO_DELAY_50MS 50
#define PNM_APCFG_CFG_HELLO_DELAY_100MS 100
#define PNM_APCFG_CFG_HELLO_DELAY_500MS 500
#define PNM_APCFG_CFG_HELLO_DELAY_1000MS 1000
```

**Parameter usMrpRole**

This parameter specifies whether the IO-Device supports the media redundancy protocol MRP or not.

---

**Note:** The Profinet IO Controller firmware does not support any kind of media redundancy. However in case the feature “Fast StartUp” of an IO-Device is used and this IO-Device supports media redundancy it is required to disable media redundancy inside this IO-Device. Therefore the IO Controller firmware needs the information if the IO-Device supports media redundancy.

---

```
#define PNM_APCFG_CFG_MRP_ROLE_NOT_SUPPORTED 0xFFFF
```

The GSDML keyword “MediaRedundancy” is not contained in the DAP description.

```
#define PNM_APCFG_CFG_MRP_ROLE_MRP_DISABLED 0x0000
```

The GSDML keyword “MediaRedundancy” is contained in the DAP description.

### 5.3.12.3 Configure PNM\_IOD\_IntfSubm Confirmation

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C47	Command Configure PNM_IOD_IntfSubm Confirmation
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 56: Configure PNM\_IOD Interface Submodule Confirmation Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_INTF_SUBM_CNF 0x00000C47
```

### 5.3.13 PNM\_IOD\_PortSubm (Port Submodule) Datasets

This service is used to additionally configure PortSubmodules which are described in the GSDML file. It shall only be used to configure those special submodules. The usage of this service is optional. If no feature configured with this packet shall be used the packet can be skipped.

#### 5.3.13.1 Configure PNM\_IOD\_PortSubm Request

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION
ulSrc	UINT32	x	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	Y	Source Queue Reference
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number
ulState	UINT32	0x0000	Status
ulCmd	UINT32	0x00000C48	Command Configure PNM_IOD_PortSubm Request
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0x0000	Routing Information, Not Used
<b>Structure tSubHead</b>			
ulTrCntrId	UINT32	1	Tree identification of the IO Controller
ulTrDevId	UINT32	1 ... a	Tree identification of the IO-Device
ulTrApId	UINT32	1 ... c	Tree identification of the AP
ulTrModId	UINT32	1 ... d	Tree identification of the module
ulTrSubModId	UINT32	1 ... e	Tree identification of the submodule
ulTrIdCnt	UINT32	1	Port submodule
<b>Structure tData</b>			
ulTrId	UINT32	1	Tree Identifier of the Port submodule
ulStructVersion	UINT32	1	Structure Version
ulAdjustFlags	UINT32	see below	Flags what parameter shall be adjusted
usAdjustMauType	UINT16		The MAU Type to adjust. Supported values are described in the IO-Devices GSDML file.

Table 57: Configure PNM\_IOD Port Submodule Request Packet

**Parameter `ulCmd`**

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_PORT_SUBM_REQ 0x00000C48
```

**Parameter `ulAdjustFlags`**

This parameter specifies which PortSubmodule parameters shall be configured. Currently only two values are supported:

<code>#define PNM_APCFG_CFG_PORT_ADJUST_NOTHING</code>	0
<code>#define PNM_APCFG_CFG_PORT_ADJUST_MAUTYPE</code>	1

### 5.3.13.2 Configure PNM\_IOD\_PortSubm Confirmation

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C49	Command Configure PNM_IOD_PortSubm Confirmation
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 58: Configure PNM\_IOD Port Submodule Confirmation Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_PORT_SUBM_CNF 0x00000C49
```

### 5.3.14 PNM\_IOD\_RecData (Record Data) Datasets

#### 5.3.14.1 Configure PNM\_IOD\_RecData Request

Structure Information				Type: Request
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION	
ulSrc	UINT32	x	Source Queue Handle	
ulDestId	UINT32	0x0000	Destination Queue Reference	
ulSrcId	UINT32	y	Source Queue Reference	
ulLen	UINT32	24 + n n	Packet Data Length (in Bytes) If First Packet of Sequence or No Sequenced Packet If Sequenced Packet or Last Packet of Sequence	
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number	
ulState	UINT32	0x0000	Status	
ulCmd	UINT32	0x00000C3E	Command Configure PNM_IOD_RecData	
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence	
ulRout	UINT32	0x0000	Routing Information, Not Used	
Structure tSubHead				
ulTrCntrId	UINT32	1	Tree identification of the IO Controller	
ulTrDevId	UINT32	1 ... a	Tree identification of the IO-Device	
ulTrApId	UINT32	1 ... c	Tree identification of the AP	
ulTrModId	UINT32	1 ... d	Tree identification of the module	
ulTrSubModId	UINT32	1 ... e	Tree identification of the submodule	
ulTrIdCnt	UINT32	g	Number of record data	
Structure tData				
ulTrId	UINT32	1	Tree identification number of record dataset	
usIndex	UINT16		Index with submodule	
usAlign1	UINT16	0	Padding with zero	
ulDataLen	UINT32		Length of RecordData[...] in byte	
abRecordData[...] ]	UINT8		User specific Record Data	
ulTrId	UINT32	2	Tree identification number of record dataset (only if g>1)	
...	...	...	...	
ulTrId	UINT32	g	Tree identification number of record dataset	
...	...	...	...	

Table 59: Configure PNM\_IOD\_RecData request packet



**Parameter `u1Cmd`**

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_RECDATA_REQ 0x00000C3E
```

---

**Note:** If this command fails the whole sequence of configuration commands starting with the *Configure PNM Request* (page 75) needs to be repeated in order to perform a correct configuration. This means starting right from the beginning. See note at section *Configuration Data Packets* on page 64 for more information.

---

### 5.3.14.2 Configure PNM\_IOD\_RecData Confirmation

The Profinet-IO Controller returns the following confirmation packet.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C3F	Confirmation Configure PNM_IOD_RecData
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 60: Configure PNM\_IOD\_RecData Confirmation Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_IOD_RECDATA_CNF 0x00000C3F
```

## 5.3.15 PNM Download Finish

### 5.3.15.1 PNM\_DWNL\_FIN Request

The download is finished with this packet.

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION
ulSrc	UINT32	x	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	y	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes)
ulId	UINT32	0	Packet Identification as Unique Number
ulState	UINT32	0x0000	Status
ulCmd	UINT32	0x00000C40	Command PNM_DWNL_FIN
ulExt	UINT32	0x0000	Extension
ulRout	UINT32	0x0000	Routing Information, Not Used

Table 61: PNM\_DWNL\_FIN Request Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_DWNL_FIN_REQ 0x00000C40
```

### 5.3.15.2 PNM\_DWNL\_FIN Confirmation

The IO Controller returns the following confirmation packet.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C41	Confirmation PNM_DWNL_FIN
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

Table 62: PNM\_DWNL\_FIN Confirmation Packet

#### Parameter ulCmd

```
#define PNM_APCTL_CMD_SET_CONFIG_DWNL_FIN_CNF 0x00000C41
```

### 5.3.16 PNM Set VersionInfo Service

The application can enforce the firmware to use a specific version number for identification. The version information is accessible from the bus side by issuing RPC EndPointMapper Lookup Requests and SNMP SystemDescription. This version information has no influence to the firmware information shown by the firmware in the dual-port memory.

**Note:** This service is supported since firmware version v2.6.9.0.

#### 5.3.16.1 PNM Set VersionInfo Request

This service shall be used by the application to set the version information.

**Note:** This packet is **not** supported by the firmware if database configuration is used.

**Note:** This packet shall be sent before the packet configuration download is finished. It is not possible to send the packet more than once when the firmware is running. Hence, new version information can only be sent after issuing a real power cycle / system reset.

The version to be seen on the bus is conducted in the following way, where empty characters are used as separator:

“usVersionMajor” + “ ” + “usVersionMinor” + “ ” + “usVersionBuild”

The whole string is limited to 9 characters, so if one of the version numbers has 3 digits, then empty character between the version numbers is invisible.

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0020	Destination Queue Handle PNM_APPLICATION
ulSrc	UINT32	x	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	y	Source Queue Reference
ulLen	UINT32	7	Packet Data Length (in Bytes)
ulId	UINT32	0 ... m	Packet Identification as Unique Number Block or Sequence Number
ulState	UINT32	0x0000	Status
ulCmd	UINT32	0x00000C66	Command PNIO_APCTL_SET_VERSIONINFO_REQ
ulExt	UINT32	0x0000	Extension No Sequenced Packet
ulRout	UINT32	0x0000	Routing Information, Not Used
<b>Structure tData (APIOC_SET_VERSIONINFO_REQ_DATA_T)</b>			
usVersionMajor	UINT16	0 – 999	Major software version (see service description)
usVersionMinor	UINT16	0 – 999	Minor software version (see service description)
usVersionBuild	UINT16	0 – 999	Build number software version (see service description)

Structure Information			Type: Request
bVersionPrefix	UINT8	'V' 'R' 'P' 'U' 'T'	Software version Prefix (check [2] for more detailed information) Released software version Revision software version Prototype software version Fieldtest software version Test software version

**Parameter ulCmd**

```
#define PNM_APCTL_CMD_SET_VERSIONINFO_REQ 0x00000C66
```

**5.3.16.2 PNM Set VersionInfo Confirmation**

The IO Controller returns the following confirmation packet:

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32	0x0000	Destination Queue Handle: SYSTEM
ulSrc	UINT32	From Request	Source Queue Handle
ulDestId	UINT32	0x0000	Destination Queue Reference
ulSrcId	UINT32	From Request	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) No Data
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	see below	Status / Error Code
ulCmd	UINT32	0x00000C67	Confirmation PNM_APCTL_CMD_SET_VERSIONINFO_CNF
ulExt	UINT32	From Request	Extension
ulRout	UINT32	z	Routing Information, Don't Care, Don't Use

**Parameter ulCmd**

```
#define PNM_APCTL_CMD_SET_VERSIONINFO_CNF 0x00000C67
```

## 5.4 Registering and Unregistering an Application

This section describes the packets that shall be used to register an application to IO Controller. IO Controller will indicate alarms, new diagnosis-data and DCP-Ident entries to the registered application.

**If no application is registered, the IO Controller will not work properly in case of incoming alarms.**

In detail, the following functionality is provided by the Registration and Unregistration Packets of the PROFINET IO Controller Protocol Stack:

Overview over Registration and Unregistration Packets of the PROFINET IO Controller Stack			
No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
5.4.1	Register Request	0x0C28	119
	Register Confirmation	0x0C29	121
5.4.2	Deregister Request	0x0C2A	122
	Deregister Confirmation	0x0C2B	124

Table 63: Overview over the Registration and Unregistration Packets of the PROFINET IO Controller Protocol Stack

### 5.4.1 Register Service

The Register Service shall be used to register an application to the IO Controller.

---

**Note:** After a successful registration the IO Controller will directly send all outstanding alarm indication to the Application.

---

#### 5.4.1.1 Register Request

To register an application to the IO Controller the following packet has to be sent to IO Controller. The field "ulSrc" will be used by IO Controller as destination for all indications that will be sent to the Application later.

---

**Note:** This packet will no longer be supported by the firmware described in this document after September 1, 2009.  
Use the registering functionality described in the netX Dual-Port-Memory Manual instead: `RCX_REGISTER_APP_REQ`, code 0x2F10.

---

Structure Information APIOC_APPLICATION_REGISTER_REQ_T			Type: Request
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of APCTL-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	Set to zero as this packet only consists of the header.
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status unused for request, Set to zero
ulCmd	UINT32	0xC28	PNIO_APCTL_CMD_APPL_REGISTER_REQ - 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 64: APIOC\_APPLICATION\_REGISTER\_REQ – Request Packet for Registering an Application to IO Controller

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_APPL_REGISTER_REQ 0x00000C28
```



### 5.4.1.2 Register Confirmation

The IO Controller will return this packet as confirmation for registering the application. The status of registering is contained in field "ulSta".

Structure Information APIOC_APPLICATION_REGISTER_CNF_T			Type: Confirmation
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of APCTL-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	Set to zero as this packet only consists of the header.
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	See below	Status / error code
ulCmd	UINT32	0xC29	PNIO_APCTL_CMD_APPL_REGISTER_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulDest	UINT32		Destination queue handle of APCTL-task process queue

Table 65: APIOC\_APPLICATION\_REGISTER\_CNF – Confirmation Packet for Registering an Application to IO Controller

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_APPL_REGISTER_CNF 0x00000C29
```

#### Parameter ulSta

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC0000201L	TLR_E_APPLICATION_ALREADY_REGISTERED There is already another application registered to IO Controller

Table 66: APIOC\_APPLICATION\_REGISTER\_CNF - Status Codes of Registering an Application to IO Controller

## 5.4.2 Deregister Service

To deregister the registered application this service has to be used. The application will only be deregistered if the request is coming from the same source the register request came from. Otherwise the application will not be deregistered. But you can force the unregistration if this is necessary. Then the application will be deregistered even if the deregister request comes from another source.

### 5.4.2.1 Deregister Request

To deregister the currently registered application this request packet has to be sent to Profinet IO Controller.

**Note:** This packet will no longer be supported by the firmware described in this document after September 1, 2009.  
Use the deregistering functionality described in the netX Dual-Port-Memory Manual instead (RCX\_UNREGISTER\_APP\_REQ).

Structure Information APIOC_APPLICATION_UNREGISTER_REQ_T			Type: Request
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
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	APIOC_APPLICATION_DEREGISTER_REQ_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status unused for request, Set to zero
ulCmd	UINT32	0xC2A	PNIO_APCTL_CMD_APPL_DEREGISTER_REQ - 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
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.
Structure tData (APIOC_APPLICATION_DEREGISTER_REQ_DATA_T)			
ulFlags	UINT32	See below.	Flags.

Table 67: APIOC\_APPLICATION\_DEREGISTER\_REQ – Request Packet for Deregistering the registered Application

Parameter `ulCmd`

```
#define PNIO_APCTL_CMD_APPL_DEREGISTER_REQ 0x00000C2A
```

Parameter `ulFlags` – Request Flag Definition

31	30	...	10	9	8	7	6	5	4	3	2	1	0	
unused, set to zero													Force Unregistration	

Table 68: Deregister Application Flags

```
#define PNIO_APCTL_FORCE_APPL_DEREGISTER 0x0001
```

### 5.4.2.2 Deregister Confirmation

The IO Controller will return this confirmation packet to indicate the state of unregistration.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of APCTL-task process queue
ulSrc	UINT32		Source queue handle of AP-task process queue
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
ulLen	UINT32	4	APIOC_APPLICATION_DEREGISTER_CNF_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status unused for request, Set to zero
ulCmd	UINT32	0xC2B	PNIO_APCTL_CMD_APPL_DEREGISTER_CNF - 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
Structure tData (APIOC_APPLICATION_DEREGISTER_CNF_DATA_T)			
ulReserved	UINT32	0	Reserved.

Table 69: APIOC\_APPLICATION\_DEREGISTER\_CNF – Confirmation Packet for deregistering the registered Application

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_APPL_DEREGISTER_CNF 0x00000C2B
```

#### Parameter ulSta

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC0000202	TLR_E_NO_APPLICATION_REGISTERED Application not deregistered, either no application was registered or the deregister request did not come from the same source the register request came from.

Table 70: APIOC\_APPLICATION\_DEREGISTER\_CNF - Status Codes of deregistering the registered Application

## 5.5 Acyclic Requests

In this section interface- and packet-definitions for acyclic requests are described. Acyclic requests are RPC Read-Record or Write-Record services or Diagnosis-Requests for example. Another example are the DCP-services.

**Note:** If the acyclic service shall contain more data than fits into the mailbox the use of fragmentation mechanisms is required. They are described in [1] section “Transferring Fragmented Packets”.

In detail, the following functionality is provided by the acyclic request packets of the PROFINET IO Controller Protocol Stack:

Overview over Acyclic Request Packets of the PROFINET IO Controller Stack			
No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
5.5.1	Read Request	0x0C1A	126
	Read Confirmation	0x0C1B	127
5.5.2	Write Request	0x0C1C	129
	Write Confirmation	0x0C1D	130
5.5.3	Read Implicit Request	0x0C1E	132
	Read Implicit Confirmation	0x0C1F	133
5.5.4	Device Diagnosis Request	0x0C22	135
	Device Diagnosis Confirmation	0x0C23	136
5.5.5	Device is active and exchanges cyclic data with IO Controller	0x2F0B	139
	Device is inactive	0x2F0B	140
5.5.6	Common Status Service Request	0x2F00	141
	Common Status Service Confirmation	0x2F01	142
5.5.7	ModuleDiffBlock Request	0x0C60	143
	ModuleDiffBlock Confirmation	0x0C61	144
5.5.8	DCP Signal Request	0x0C0E	146
	DCP Signal Confirmation	0x0C0F	148
5.5.9	DCP SET Name Request	0x0C0A	150
	DCP SET Name Confirmation	0x0C0B	152
5.5.10	DCP SET IP Request	0x0C0C	154
	DCP SET IP Confirmation	0x0C0D	156
5.5.12	DCP IDENT ALL Request	0x0C10	161
	DCP IDENT ALL Confirmation	0x0C11	162
5.5.14	Alarm Acknowledge Request	0x0C26	167
	Alarm Acknowledge Confirmation	0x0C27	169
5.5.15	Release IO-Device Request	0x0C2E	171
	Release IO-Device Confirmation	0x0C2F	172

Table 71: Overview over the Acyclic Request Packets of the PROFINET IO Controller Protocol Stack

## 5.5.1 Read Service

With the RPC Read Service you can read specific data from an IO-Device. This service is only available if the IO-Device to read from is in cyclic data exchange with the IO Controller firmware.

This service can be used for multiple IO-Devices at the same time. However, only one request per IO-Device per time is possible.

**Note:** If the more data shall be read than fits into the mailbox the use of fragmentation mechanisms is required. It is described in [1], section “Transferring Fragmented Packets”.

The Profinet IO Controller firmware requires that **every** fragmented packet sent by the host application contains the full section tData. If that is not the case the sequenced transfer will not work.

### 5.5.1.1 Read Request

To initialize a Read Request this packet has to be sent to IO Controller. The requestor has to be able to receive a response with size `ulDataLength`. That size must be set by requestor inside the request packet. As the IO Controller is not able to receive more than 4096 Byte, that is the highest possible value for `ulDataLength`.

Structure Information APIOC_READ_PCK_T			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
<code>ulDest</code>	UINT32		Destination queue handle of APCTL-task process queue
<code>ulSrc</code>	UINT32		Source queue handle of AP-task process queue
<code>ulDestId</code>	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
<code>ulSrcId</code>	UINT32	0 ... $2^{32} - 1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
<code>ulLen</code>	UINT32	18	APIOC_READ_REQ_DATA_T - Packet data length in bytes
<code>ulId</code>	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
<code>ulSta</code>	UINT32	0	Status unused for request, Set to zero
<code>ulCmd</code>	UINT32	0xC1A	PNIO_APCTL_CMD_READ_REQ - Command
<code>ulExt</code>	UINT32	0	Extension not in use, set to zero for compatibility reasons
<code>ulRout</code>	UINT32	0	Routing not in use, set to zero for compatibility reasons
<b>Structure tData (APIOC_READ_REQ_DATA_T)</b>			
<code>ulHandle</code>	UINT32	0 ... $2^{32} - 1$	Handle to IO-Device to read from. How to get the handle see section <i>Device Handle</i> and obtainin detailed Information of an IO Device on page 55.
<code>ulApi</code>	UINT32	0 ... $2^{32} - 1$	Profinet API to read from.
<code>ulDataLength</code>	UINT32	0 .. 4096	Maximum response size the requestor is able to handle.
<code>usSlot</code>	UINT16	0 ... $2^{15} - 1$	Profinet Slot to read from.
<code>usSubSlot</code>	UINT16	0 ... 36863	Profinet SubSlot to read from.
<code>usIndex</code>	UINT16	0 ... $2^{16} - 1$	Profinet Index to read from.

Table 72: APIOC\_READ\_REQ – Request Packet for Reading Data from IO Device

#### Parameter `ulCmd`

```
#define PNIO_APCTL_CMD_READ_REQ    0x00000C1A
```

### 5.5.1.2 Read Confirmation

IO-Controller will send this packet as confirmation to the Read Request. The data read from IO-Device will be contained. Note that the IO-Device may return no data if no data exists for the requested parameters.

If the data read from IO-Device is too big to fit in a single packet the sequence mechanism as described in [DPM] section 4.1.8 (Transferring Fragmented Packets) will be used to transport the data in several packets.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	29 + n	APIOC_READ_CNF_DATA_T - Packet data length + sizeof(IO-Devices Read-Response) - 1 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	Status / error code.
ulCmd	UINT32	0xC1B	PNIO_APCTL_CMD_READ_CNF - Command
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0	Routing not in use, set to zero for compatibility reasons
<b>Structure tData (APIOC_READ_CNF_DATA_T)</b>			
ulHandle	UINT32	0 ... $2^{32} - 1$	Handle to IO-Device read from.
ulPnio	UINT32	0 ... $2^{32} - 1$	Profinet IO status of Read Request, RPC status.
ulApi	UINT32	0 ... $2^{32} - 1$	Profinet API read from.
ulDataLength	UINT32	0 ... $2^{32} - 1$	Total length of Read Response data. Needed in case that the packet is transferred via DPM Mailboxes that cut the packet in small segments.
usSlot	UINT16	0 ... $2^{15} - 1$	Profinet Slot read from.
usSubSlot	UINT16	0 ... 36863	Profinet SubSlot read from.
usIndex	UINT16	0 ... $2^{16} - 1$	Profinet Index read from.
usAddVal1	UINT16	0 ... $2^{16} - 1$	Profinet Additional Value 1.
usAddVal2	UINT16	0 ... $2^{16} - 1$	Profinet Additional Value 2.
usReserved	UINT16	0	Reserved, is set to zero.
abReadData[1]	UINT8[]	0 ... $2^8 - 1$	Profinet Read Response. If more than 1 byte was read the remaining bytes of response follow this byte.

Table 73: APIOC\_READ\_CNF – Confirmation Packet for Reading Data from IO Device

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_READ_CNF 0x00000C1B
```

**Parameter** `ulSta`

Definition / (Value)	Description
0x00000000	TLR_S_OK Status ok
0xC00A0012	TLR_E_PNIO_CMCTL_RESOURCE_OUT_OF_MEMORY Insufficient memory for this request.
0xC00A0014	TLR_E_PNIO_CMCTL_STATE_CONFLICT This request cannot be served in current CMCTL state.
0xC00A0018	TLR_E_PNIO_CMCTL_PACKET_SEND_FAILED Error while sending a packet to another task.
0xC00A0040	TLR_E_PNIO_CMCTL_INVALID_PM_INDEX The CMCTL protocol-machine restored from index is invalid.
0xC00A0041	TLR_E_PNIO_CMCTL_INVALID_PM The index of CMCTL protocol-machine is invalid.
0xC00C0030	TLR_E_PNIO_APCTL_RPC_REQUEST_LIMIT_REACHED Too many outstanding RPC-requests for this IO-Device.
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error while sending internal message to another task.
0xC00C0032	TLR_E_PNIO_APCTL_INVALID_CMCTL_HANDLE The handle <code>ulHandle</code> used for IO-Device is wrong.
0xC00C0051	TLR_E_PNIO_APCTL_BUS_STATE_OFF The current bus state if OFF and no frames can be sent.
0xC02E0100	TLR_E_RPC_STATUS Generic RPC-error code. See Profinet-status <code>ulPnio</code> code for details.
0xC02E0200	TLR_E_CLRPC_PACKET_SEND_FAILED Error while sending internal message to another task.
0xC02E0201	TLR_E_CLRPC_TIMER_OUT_OF_MEMORY Creating a TLR-Timer-packet in RPC task failed due to insufficient memory.
0xC02E0605	TLR_E_CLRPC_CLIENT_HANDLE_INVALID The handle to RPC-Client instance is invalid.
0xC02E0606	TLR_E_CLRPC_CLIENT_REQUEST_LIMIT_EXCEEDED The maximum amount of outstanding RPC-Requests for this RPC-Clients instance is reached.
0xC02E0607	TLR_E_CLRPC_CLIENT_OPCODE_SEQUENCE RPC-Client instances can only connect to an IO-Device if there are no outstanding RPC-Requests. Currently at least one RPC-Request is outstanding.
0xC02B0024	TLR_E_MID_SYS_PACKET_OUT_OF_SEQUENCE The message ID of request is incorrect; it is out of sequence.
0xC00C0067	TLR_E_PNIO_APCTL_TOO_MUCH_DATA_REQUESTED The maximum amount of data supported by this service is exceeded.

*Table 74: APIOC\_READ\_CNF - Packet Sstatus/Error*



## 5.5.2 Write Service

With the RPC Write Service you can write specific data to an IO-Device.

This service is only available if the IO-Device to write data to is in cyclic data exchange with the IO-Controller firmware.

This service can be used for multiple IO-Devices at the same time. However, only one request per IO-Device per time is possible.

**Note:** If more data shall be written than fits into the mailbox, the use of fragmentation mechanisms is required. It is described in [1] section “Transferring Fragmented Packets”.

The Profinet IO-Controller firmware requires that **every** fragmented packet sent by the host application contains the full section tData. If that is not the case the sequenced transfer will not work. This means that the users application shall ensure that all fields of tData aside from abWriteData contain the same data for all fragments of a request. The field tData,abWriteData shall contain the data fragment of the record to write

### 5.5.2.1 Write Request

To initialize a Write Request this packet has to be sent to IO-Controller. If the data to write to IO-Device is too big to fit in a single packet the sequence mechanism as described in [DPM] section 4.1.8 (Transferring Fragmented Packets) shall be used to transport the data in several packets.

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	21 + n	APIOC_WRITE_REQ_DATA_T - Packet data length + sizeof(Write-Request) -1 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 unused for request, Set to zero
ulCmd	UINT32	0xC1C	PNIO_APCTL_CMD_WRITE_REQ - Command
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0	Routing not in use, set to zero for compatibility reasons
<b>Structure tData (APIOC_WRITE_REQ_DATA_T)</b>			
ulHandle	UINT32	0 ... $2^{32}-1$	Handle to IO-Device to write to. How to get the handle see section <i>Device Handle</i> and obtainin detailed Information of an IO Device on page 55.
ulApi	UINT32	0 ... $2^{32}-1$	Profinet API to write to.
ulDataLength	UINT32	0 ... $2^{32}-1$	Total length of Write data. Needed in case that the packet is transferred via DPM Mailboxes that cut the packet in small segments.

Structure Information			Type: Request
usSlot	UINT16	0 ... $2^{15} - 1$	Profinet Slot to write to.
usSubSlot	UINT16	0 ... 36863	Profinet SubSlot to write to.
usIndex	UINT16	0 ... $2^{16} - 1$	Profinet Index to write to.
usReserved	UINT16	0	Reserved, set to zero.
abWriteData [1]	UINT8[]	0 ... $2^8 - 1$	First byte of data to be written. Remaining bytes must follow this one.

Table 75: *APIOC\_WRITE\_REQ – Request Packet for Writing Data to IO Device***Parameter ulCmd**

```
#define PNIO_APCTL_CMD_WRITE_REQ 0x00000C1C
```

**5.5.2.2 Write Confirmation**

The Profinet IO-Controller will send this packet as confirmation to the Write Request.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
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	APIOC_WRITE_CNF_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	See below	Status / error code.
ulCmd	UINT32	0xC1D	PNIO_APCTL_CMD_WRITE_CNF - Command
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
ulRout	UINT32	0	Routing not in use, set to zero for compatibility reasons
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.
<b>Structure tData (APIOC_WRITE_CNF_DATA_T)</b>			
ulHandle	UINT32	0 ... $2^{32} - 1$	Handle to IO-Device written to.
ulPnio	UINT32	0 ... $2^{32} - 1$	Profinet IO status of Write Request, RPC status.
ulApi	UINT32	0 ... $2^{32} - 1$	Profinet API written to.
usSlot	UINT16	0 ... $2^{15} - 1$	Profinet Slot written to.
usSubSlot	UINT16	0 ... 36863	Profinet SubSlot written to.
usIndex	UINT16	0 ... $2^{16} - 1$	Profinet Index written to.
usAddVal1	UINT16	0 ... $2^{16} - 1$	Profinet Additional Value 1.
usAddVal2	UINT16	0 ... $2^{16} - 1$	Profinet Additional Value 2.

Table 76: *APIOC\_WRITE\_CNF – Confirmation Packet for Writing Data to IO Device*

**Parameter ulCmd**

```
#define PNIO_APCTL_CMD_WRITE_CNF      0x00000C1D
```

**Parameter ulSta**

Definition / (Value)	Description
0x00000000	TLR_S_OK Status ok
0xC00A0012	TLR_E_PNIO_CMCTL_RESOURCE_OUT_OF_MEMORY Insufficient memory for this request.
0xC00A0014	TLR_E_PNIO_CMCTL_STATE_CONFLICT This request cannot be served in current CMCTL state.
0xC00A0018	TLR_E_PNIO_CMCTL_PACKET_SEND_FAILED Error while sending a packet to another task.
0xC00A0040	TLR_E_PNIO_CMCTL_INVALID_PM_INDEX The CMCTL protocol-machine restored from index is invalid.
0xC00A0041	TLR_E_PNIO_CMCTL_INVALID_PM The index of CMCTL protocol-machine is invalid.
0xC00C0030	TLR_E_PNIO_APCTL_RPC_REQUEST_LIMIT_REACHED Too many outstanding RPC-requests for this IO-Device.
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error while sending internal message to another task.
0xC00C0032	TLR_E_PNIO_APCTL_INVALID_CMCTL_HANDLE The handle ulHandle used for IO-Device is wrong.
0xC00C0051	TLR_E_PNIO_APCTL_BUS_STATE_OFF The current bus state is OFF and no frames can be sent.
0xC02E0100	TLR_E_RPC_STATUS Generic RPC-error code. See Profinet-status ulPnio code for details.
0xC02E0200	TLR_E_CLRPC_PACKET_SEND_FAILED Error while sending internal message to another task.
0xC02E0201	TLR_E_CLRPC_TIMER_OUT_OF_MEMORY Creating a TLR-Timer-packet in RPC task failed due to insufficient memory.
0xC02E0605	TLR_E_CLRPC_CLIENT_HANDLE_INVALID The handle to RPC-Client instance is invalid.
0xC02E0606	TLR_E_CLRPC_CLIENT_REQUEST_LIMIT_EXCEEDED The maximum amount of outstanding RPC-Requests for this RPC-Clients instance is reached.
0xC02E0607	TLR_E_CLRPC_CLIENT_OPCODE_SEQUENCE RPC-Client instances can only connect to an IO-Device if there are no outstanding RPC-Requests. Currently at least one RPC-Request is outstanding.
0xC02B0024	TLR_E_MID_SYS_PACKET_OUT_OF_SEQUENCE The message ID of request is incorrect; it is out of sequence.
0xC00C0067	TLR_E_PNIO_APCTL_TOO_MUCH_DATA_REQUESTED The maximum amount of data supported by this service is exceeded.

Table 77: APIOC\_WRITE\_CNF - Packet Status/Error

### 5.5.3 Read Implicit Service

With the RPC Read Service you can read specific data from an IO-Device.

This service cannot be used in parallel to read from multiple IO-Devices. Only access to one IO-Device per time is supported.

**Note:** If the more data shall be read than fits into the mailbox the use of fragmentation mechanisms is required. It is described in [1] section “Transferring Fragmented Packets”.

The Profinet IO-Controller firmware requires that **every** fragmented packet sent by the host application contains the full section tData. If that is not the case the sequenced transfer will not work.

#### 5.5.3.1 Read Implicit Request

To initialize a Read Request this packet has to be sent to IO-Controller. The requestor has to be able to receive a response with size ulDataLength. That size must be set by requestor inside the request packet. As the IO-Controller is not able to receive more than 4096 Byte, that is the highest possible value for ulDataLength.

Structure Information			Type: Request
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of APCTL-task process queue
ulSrc	UINT32		Source queue handle of AP-task process queue
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
ulLen	UINT32	26	APIOC_READ_IMPL_REQ_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status unused for request, Set to zero
ulCmd	UINT32	0xC1E	PNIO_APCTL_CMD_READ_IMPL_REQ - 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
Structure tData (APIOC_READ_REQ_IMPL_DATA_T)			
ulIPAddress	UINT32		The IP-Address of the IO-Device to read from.
usDeviceId	UINT16		The Device ID of the IO-Device to read from.
usVendorId	UINT16		The Vendor ID of the IO-Device to read from.
usInstanceId	UINT16		The Instance ID of the IO-Device to read from (GSDML parameter).
usReserved	UINT16	0	Reserved, set to zero.
ulApi	UINT32	0 ... 2 <sup>32</sup> -1	Profinet API to read from.
ulDataLength	UINT32	0 .. 4096	Maximum response size the requestor is able to handle.
usSlot	UINT16	0 ... 2 <sup>15</sup> -1	Profinet Slot to read from.
usSubSlot	UINT16	0 ... 36863	Profinet SubSlot to read from.
usIndex	UINT16	0 ... 2 <sup>16</sup> -1	Profinet Index to read from.

Table 78: APIOC\_READ\_IMPL\_REQ – Request Packet for implicitly Reading Data from IO Device

**Parameter ulCmd**

```
#define PNIO_APCTL_CMD_READ_IMPL_REQ 0x00000C1E
```

**5.5.3.2 Read Implicit Confirmation**

IO-Controller will send this packet as confirmation to the Read Request. The data read from IO-Device will be contained. Note that the IO-Device may return no data if no data exists for the requested parameters.

If the data read from IO-Device is too big to fit in a single packet the sequence mechanism as described in [DPM] section 4.1.8 (Transferring Fragmented Packets) will be used to transport the data in several packets.

Structure Information				Type: Confirmation
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32		Destination queue handle of APCTL-task process queue	
ulSrc	UINT32		Source queue handle of AP-task process queue	
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons	
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.	
ulLen	UINT32	25 + n	APIOC_READ_IMPL_CNF_DATA_T - Packet data length + sizeof(IO-Devices Read-Response) -1 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	Status / error code.	
ulCmd	UINT32	0xC1F	PNIO_APCTL_CMD_READ_IMPL_CNF - Command	
ulExt	UINT32	0x0000 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence	
ulRout	UINT32	0	Routing not in use, set to zero for compatibility reasons	
Structure tData (APIOC_READ_IMPL_CNF_DATA_T)				
ulPnio	UINT32	0 ... 2 <sup>32</sup> -1	Profinet IO status of Read Request, RPC status.	
ulApi	UINT32	0 ... 2 <sup>32</sup> -1	Profinet API read from.	
ulDataLength	UINT32	0 ... 2 <sup>32</sup> -1	Total length of Read Response data. Needed in case that the packet is transferred via DPM Mailboxes that cut the packet in small segments.	
usSlot	UINT16	0 ... 2 <sup>15</sup> -1	Profinet Slot read from.	
usSubSlot	UINT16	0 ... 36863	Profinet SubSlot read from.	
usIndex	UINT16	0 ... 2 <sup>16</sup> -1	Profinet Index read from.	
usAddVal1	UINT16	0 ... 2 <sup>16</sup> -1	Profinet Additional Value 1.	
usAddVal2	UINT16	0 ... 2 <sup>16</sup> -1	Profinet Additional Value 2.	
usReserved	UINT16	0	Reserved, is set to zero.	
abReadData[1]	UINT8[]	0 ... 2 <sup>8</sup> -1	Profinet Read Response. If more than 1 byte was read the remaining bytes of response follow this byte.	

Table 79: APIOC\_READ\_IMPL\_CNF – Confirmation Packet for implicitly Reading Data from IO Device

**Parameter ulCmd**

```
#define PNIO_APCTL_CMD_READ_IMPL_CNF 0x00000C1F
```

**Parameter** `ulSta`

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC00A0012	TLR_E_PNIO_CMCTL_RESOURCE_OUT_OF_MEMORY Insufficient memory for this request.
0xC00A0014	TLR_E_PNIO_CMCTL_STATE_CONFLICT This request cannot be served in current CMCTL state.
0xC00A0018	TLR_E_PNIO_CMCTL_PACKET_SEND_FAILED Error while sending a packet to another task.
0xC00A0040	TLR_E_PNIO_CMCTL_INVALID_PM_INDEX The CMCTL protocol-machine restored from index is invalid.
0xC00A0041	TLR_E_PNIO_CMCTL_INVALID_PM The index of CMCTL protocol-machine is invalid.
0xC00C0030	TLR_E_PNIO_APCTL_RPC_REQUEST_LIMIT_REACHED Too many outstanding RPC-requests for this IO-Device.
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error while sending internal message to another task.
0xC00C0032	TLR_E_PNIO_APCTL_INVALID_CMCTL_HANDLE The handle <code>ulHandle</code> used for IO-Device is wrong.
0xC00C0051	TLR_E_PNIO_APCTL_BUS_STATE_OFF The current bus state is OFF and no frames can be sent.
0xC02E0100	TLR_E_RPC_STATUS Generic RPC-error code. See Profinet-status <code>ulPnio</code> code for details.
0xC02E0200	TLR_E_CLRPC_PACKET_SEND_FAILED Error while sending internal message to another task.
0xC02E0201	TLR_E_CLRPC_TIMER_OUT_OF_MEMORY Creating a TLR-Timer-packet in RPC task failed due to insufficient memory.
0xC02E0605	TLR_E_CLRPC_CLIENT_HANDLE_INVALID The handle to RPC-Client instance is invalid.
0xC02E0606	TLR_E_CLRPC_CLIENT_REQUEST_LIMIT_EXCEEDED The maximum amount of outstanding RPC-Requests for this RPC-Clients instance is reached.
0xC02E0607	TLR_E_CLRPC_CLIENT_OPCODE_SEQUENCE RPC-Client instances can only connect to an IO-Device if there are no outstanding RPC-Requests. Currently at least one RPC-Request is outstanding.
0xC02B0024	TLR_E_MID_SYS_PACKET_OUT_OF_SEQUENCE The message ID of request is incorrect; it is out of sequence.
0xC000001A	TLR_E_REQUEST_RUNNING There is already an implicit Read Request running.
0xC0140048	TLR_E_PNIO_APCFG_INVALID_IP_ADDRESS The IP address is not reachable with the local netmask set.
0xC014004A	TLR_E_PNIO_APCFG_INVALID_GATEWAY The IP address is not reachable with the local Gateway address.
0xC00C0067	TLR_E_PNIO_APCTL_TOO_MUCH_DATA_REQUESTED The maximum amount of data supported by this service is exceeded.

Table 80: `APIOC_READ_IMPL_CNF` - Packet Status/Error

## 5.5.4 Device Diagnosis Service

With the Device Diagnosis Service you can read existing diagnosis-data of an IO-Device (which is reported to application by IO-Controller with service described in section *Diagnosis Service* on page 184). If diagnosis data for the requested device exists, it will be put in confirmation-packet. With the flags in request-packet you can specify if the diagnosis data shall be deleted in IO-Controller.

### 5.5.4.1 Device Diagnosis Request

To initialize a Device Diagnosis Request this packet has to be sent to the IO-Controller.

Structure Information			Type: Request
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of APCTL-task process queue
ulSrc	UINT32		Source queue handle of AP-task process queue
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
ulLen	UINT32	8	APIOC_DEVICE_DIAG_REQ_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status unused for request, Set to zero
ulCmd	UINT32	0xC22	PNIO_APCTL_CMD_GET_DEVICE_DIAGNOSIS_REQ - 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
Structure tData (APIOC_DEVICE_DIAG_REQ_DATA_T)			
ulHandle	UINT32	0 ... 2 <sup>32</sup> -1	Handle to IO-Device to get diagnosis data for. How to get the handle see section <i>Device Handle</i> and obtainin detailed Information of an IO Device on page 55.
ulFlags	UINT32		Flags. See below.

Table 81: APIOC\_DEVICE\_DIAG\_REQ – Request Packet for getting diagnosis Data for an IO Device

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_GET_DEVICE_DIAGNOSIS_REQ 0x00000C22
```

#### Parameter ulFlags – diagnosis data Request Flag Definition

31	30	...	10	9	8	7	6	5	4	3	2	1	0	
unused, set to zero													0	Delete diagnosis buffer

Table 82: Diagnosis Data Flags

#### IO-Device status parameters

```
#define PNIO_IOD_DIAG_IOC_DELETE_BUFFER (0x0001)
```

Delete the content of diagnosis buffer in IO-Controller after handling the diagnosis request.

### 5.5.4.2 Device Diagnosis Confirmation

IO-Controller will send this packet as confirmation to the Device Diagnosis Request. If diagnosis data for the requested device is existing, it will be put in confirmation-packet.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of APCTL-task process queue
ulSrc	UINT32		Source queue handle of AP-task process queue
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
ulLen	UINT32	13 + n	APIOC_DEVICE_DIAG_CNF_DATA_T - Packet data length + sizeof(diagnosis-data) -1 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	Status / error code.
ulCmd	UINT32	0xC23	PNIO_APCTL_CMD_GET_DEVICE_DIAGNOSIS_CNF - 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
Structure tData (APIOC_GET_DEVICE_DIAG_CNF_DATA_T)			
ulHandle	UINT32	0 ... 2 <sup>32</sup> -1	Handle to IO-Device the diagnosis data belongs to.
ulFlags	UINT32		Flags. See below.
ulDiagLength	UINT32	0 ... 1560	Length of diagnosis data in this packet.
abDiagData[1 ]	UINT8[]	0 ... 2 <sup>16</sup> -1	First byte of diagnosis data. Remaining byte follow this one. This buffer contains the raw diagnosis data directly taken from the bus. No swapping is done by the firmware.

Table 83: APIOC\_DEVICE\_DIAG\_CNF – Confirmation Packet with Diagnosis Data of an IO Device

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_GET_DEVICE_DIAGNOSIS_CNF 0x00000C23
```

#### Parameter ulsta

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC00C0032	TLR_E_PNIO_APCTL_INVALID_CMCTL_HANDLE The IO-Device handle ulHandle is invalid.

Table 84: TCPIP\_TCP\_UDP\_CMD\_GET\_SOCKET\_OPTION\_CNF - Packet Status/Error

#### Parameter ulFlags – Diagnosis Data Confirmation Flag Definition

Bit	Description
D12.. D31	unused, set to zero
D11	ModuleDiffBlock present
D10	Packet to small
D9	Diagnosis buffer overwritten
D8	Diagnosis buffer overflow



Bit	Description
D7	Diagnosis-Disappeared
D6	Diagnosis data present for IO-Device
D5	IO-Device deactivated
D4	IO-Device parameter fault
D3	IO-Device invalid response
D2	IO-Device configuration fault
D1	IO-Device not ready
D0	IO-Device does not exist

Table 85: Diagnosis Data Flags

### IO-Device status parameters

```
#define PNIO_IOD_DIAG_IOD_NON_EXIST (0x0001)
The IO-Device does not exist / respond to DCP Ident Requests.

#define PNIO_IOD_DIAG_IOD_NOT_READY (0x0002)
The IO-Device is not ready.

#define PNIO_IOD_DIAG_IOD_CFG_FAULT (0x0004)
A configuration fault exists for this IO-Device (e.g. its NameOfStation or IP is used
more than once in the network).

#define PNIO_IOD_DIAG_IOD_INVALID_RESPONSE (0x0008)
The IO-Device send an invalid response (e.g. DCP Set IP was not successful).

#define PNIO_IOD_DIAG_IOD_PRM_FAULT (0x0010)
A parameter fault exists for this IO-Device (e.g. Connect_Request or Write_Record on
Startup is rejected with error code).

#define PNIO_IOD_DIAG_IOD_DEACTIVATED (0x0020)
This IO-Device is deactivated by user.
```

### Status of diagnosis data

```
#define PNIO_IOD_DIAG_IOD_DIAG_DATA_PRESENT (0x0040)
There is diagnosis data existing for the IO-Device.

#define PNIO_IOD_DIAG_IOD_DIAG_DISAPPEARED (0x0080)
The IO-Device send a "Diagnosis disappeared Alarm".

#define PNIO_IOD_DIAG_IOC_BUFFER_OVERFLOW (0x0100)
IO-Controllers buffer for diagnosis data was to small for the diagnosis data send by IO-
Device.

#define PNIO_IOD_DIAG_IOC_BUFFER_OVERWRITTEN (0x0200)
IO-Controllers buffer for diagnosis data was overwritten by new diagnosis data of IO-
Device before old data was read.

#define PNIO_IOD_DIAG_IOC_PACKET_TOO_SMALL (0x0400)
The packet requesting diagnosis data is to small to carry the diagnosis data for this IO-
Device.

#define PNIO_IOD_DIAG_MODULE_DIFF_REPORTED (0x0800)
The IO-Device reported a ModuleDiffBlock during connection establishment.
```

## 5.5.5 Obtain Device Connection Information

This section describes the fieldbus specific components of the general mechanism to obtain Slave Connection Information as described in [DPM]. The registered application can request additional information about a specific handle to be able to identify the IO-Device belonging to the handle.

Depending on the connection state of the requested IO-Device one of the following structures will be returned. The application can decide which structure is used by examining the field `ulStructId` of the confirmation.

### 5.5.5.1 Get Slave Connection Information Request

Using the handles described in reference #1, the application can request network status information for each of the configured network slaves.

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
<code>ulDest</code>	UINT32	0x00000020	Destination Queue Handle CHANNEL
<code>ulSrc</code>	UINT32	X	Source Queue Handle
<code>ulDestId</code>	UINT32	0x00000000	Destination Queue Reference
<code>ulSrcId</code>	UINT32	Y	Source Queue Reference
<code>ulLen</code>	UINT32	4	Packet Data Length (in Bytes)
<code>ulId</code>	UINT32	Any	Packet Identification as Unique Number
<code>ulSta</code>	UINT32	0x00000000	Status
<code>ulCmd</code>	UINT32	0x00002F0A	Command Get Slave Connection Information Request
<code>ulExt</code>	UINT32	0x00000000	Reserved
<code>ulRout</code>	UINT32	0x00000000	Routing Information
<b>Structure tData</b>			
<code>ulHandle</code>	UINT32	0 ... 0xFFFFFFFF	Slave Handle How to get the handle see section <i>Device Handle and obtainin detailed Information of an IO Device</i> on page 55.

Table 86: Get Slave Connection Information Request

### 5.5.5.2 Device is active and exchanges cyclic data with IO Controller

In case of an active device the Device Connection Information Confirmation will have the following structure.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	516	Packet length
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	See below	Status / error code.
ulCmd	UINT32	0x2F0B	RCX_GET_SLAVE_CONN_INFO_CNF - 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
<b>Structure tData</b>			
ulHandle	UINT32	0 ... $2^{32}-1$	Handle to IO-Device the information belongs to
ulStructId	UINT32	0x2245	APIOC_ACTIVE_DEVICE_CONNECT_INFO_STRUCT_ID – identifies the following structure
ulLenName	UINT32	1 ... 240	Length of NameOfStation of IO-Device.
ulLenType	UINT32	1 ... 240	Length of TypeOfStation of IO-Device.
ulIPAddress	UINT32		The IP-Address of IO-Device.
ulDiagFlags	UINT32		Flags of Device Diagnosis Information (see description of field ulFlags in section 5.5.4.2 for details).
usDeviceID	UINT16		Device ID of IO-Device.
usVendorID	UINT16		Vendor ID of IO-Device.
abMac[6]	UINT8[]		MAC address of IO-Device.
usReserved	UINT16	0	Reserved, don't care.

Table 87: Device Connection Information Confirmation for active IO Devices

### 5.5.5.3 Device is inactive

In case of inactive device the Device Connection Information Confirmation will have the following structure. The parameters are coming from IO Controller's database. It is trying to connect to an IO Device with this parameters.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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 ... 232 -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
ulLen	UINT32	260	Packet length
ulId	UINT32	0 ... 232 -1	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	See below	Status / error code.
ulCmd	UINT32	0x2F0B	RCX_GET_SLAVE_CONN_INFO_CNF - 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
<b>Structure tData</b>			
ulHandle	UINT32	0 ... 232 -1	Handle to IO-Device the information belongs to
ulStructId	UINT32	0x2244	APIOC_INACTIVE_DEVICE_CONNECT_INFO_STRUCT_ID – identifies the following structure
ulLenName	UINT32	1 ... 240	Length of NameOfStation of IO-Device.
ulDiagFlags	UINT32		Flags of Device Diagnosis Information (see description of field ulFlags in section Device Diagnosis Confirmation on page 136 for details).
usDeviceID	UINT16		Device ID of IO-Device.
usVendorID	UINT16		Vendor ID of IO-Device.
abName[ 240 ]	UINT8[]		IO-Device NameOfStation.

Table 88: Device Connection Information Confirmation for inactive IO Devices

## 5.5.6 Common Status Service (Firmware Diagnosis)

The Common Status Service can be used to read out the Common Status Block via packets. For details about the Common Status Block see section *Common Status* on page 30. This Common Status Block is equal for all Hilscher netX firmware stacks (protocol independent).

### 5.5.6.1 Common Status Service Request

To request the content of Common Status Block from the IO-Controller this packet has to be sent.

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	DIAG_INFO_GET_COMMON_STATE_REQ_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status not in use for request.
ulCmd	UINT32	0x2F00	DIAG_INFO_GET_COMMON_STATE_REQ - 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 89: DIAG\_INFO\_GET\_COMMON\_STATE\_REQ - Get Common Status Block Request

#### Parameter ulCmd

```
#define DIAG_INFO_GET_COMMON_STATE_REQ 0x00002F00
```

### 5.5.6.2 Common Status Service Confirmation

The Common Status Block will be returned inside the packet described below. A detailed description of the meaning of the parameters returned can be found in section *Common Status* on page 30.

Structure Information				Type: Confirmation
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32		Destination queue handle of APCTL-task process queue	
ulSrc	UINT32		Source queue handle of AP-task process queue	
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons	
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.	
ulLen	UINT32	64	DIAG_INFO_GET_COMMON_STATE_CNF_DATA_T - Packet data length in bytes	
ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet	
ulSta	UINT32	0	See below.	
ulCmd	UINT32	0x2F01	DIAG_INFO_GET_COMMON_STATE_CNF - 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	
Structure tData (DIAG_INFO_GET_COMMON_STATE_CNF_DATA_T)				
ulCommunicationCOS	UINT32	0...127	Communication Change of State.	
ulCommunicationState	UINT32	0...4	Communication State.	
ulCommunicationError	UINT32		Communication Error.	
usVersion	UINT16		Version.	
usWatchdogTime	UINT16		Watchdog Timeout.	
ausReserved[2]	UINT16[]		Reserved	
ulHostWatchdog	UINT32		Host Watchdog.	
ulErrorCount	UINT32		Error Count.	
ulErrorLogInd	UINT32		Not supported yet.	
ulReserved[2]	UINT32[]		Reserved.	

Table 90: DIAG\_INFO\_GET\_COMMON\_STATE\_CNF - Get Common Status Block Confirmation

#### Parameter ulCmd

```
#define DIAG_INFO_GET_COMMON_STATE_CNF 0x00002F01
```

## 5.5.7 ModuleDiffBlock Service

The ModuleDiffBlock Service can be used to read out ModuleDiffBlock for an IO-Device. If the configuration the IO-Controller has of the IO-Device does not correspond to the physical configuration of the IO-Device the IO-Device will send a ModuleDiffBlock to IO-Controller.

**Note:** The IO-Controller firmware will only store and return the ModuleDiffBlock as indicated by IO-Device during connection establishment. Any change during runtime is **not** contained in the block returned within this service.

**Note:** If the application needs the current ModuleDiffBlock it is recommended to perform an acyclic read request for the ModuleDiffBlock (index 0xE002) instead of using this service.

### 5.5.7.1 ModuleDiffBlock Request

The ModuleDiffBlock Request service shall be used to get a ModuleDiffBlock for a specific IO Device.

There is (theoretically) no limit for the size of ModuleDiffBlock. Therefore the Application has to tell the firmware the maximum size of ModuleDiffBlock it can handle in parameter `ulMaxRspLength`.

Structure Information <code>APIOC_GET_MOD_DIFF_BLOCK_PCK_T</code>			Type: Request
Variable	Type	Value / Range	Description
Structure <code>TLR_PACKET_HEADER_T</code>			
<code>ulDest</code>	UINT32		Destination queue handle of APCTL-task process queue
<code>ulSrc</code>	UINT32		Source queue handle of AP-task process queue
<code>ulDestId</code>	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
<code>ulSrcId</code>	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
<code>ulLen</code>	UINT32	8	<code>APIOC_DCP_SET_SIGNAL_REQ_DATA_T</code> - Packet data length in bytes
<code>ulId</code>	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
<code>ulSta</code>	UINT32	0	Status not in use for request.
<code>ulCmd</code>	UINT32	0xC60	<code>PNIO_APCTL_CMD_GET_MOD_DIFF_BLOCK_REQ</code> - Command
<code>ulExt</code>	UINT32	0	Extension not in use, set to zero for compatibility reasons
<code>ulRout</code>	UINT32	0	Routing not in use, set to zero for compatibility reasons
Structure <code>tData (APIOC_GET_MOD_DIFF_BLOCK_REQ_DATA_T)</code>			
<code>ulHandle</code>	UINT32		Handle to the IO-Device. How to get the handle see section <i>Device Handle</i> and obtainin detailed Information of an IO Device on page 55.
<code>ulMaxRspLength</code>	UINT32		Maximum supported size of response in bytes.

Table 91: `APIOC_GET_MOD_DIFF_BLOCK_REQ` – Get ModuleDiffBlock Request

#### Parameter `ulCmd`

```
#define PNIO_APCTL_CMD_GET_MOD_DIFF_BLOCK_REQ 0x00000C60
```

### 5.5.7.2 ModuleDiffBlock Confirmation

The application will return this confirmation to the ModuleDiffBlock request. If the ModuleDiffBlock is bigger than requests parameter `ulMaxRspLength` then `ulMaxRspLength` bytes of the ModuleDiffBlock will be reported by IO-Controller and the flag `ulFlag` in confirmation packet will be set correctly.

**Note:** The parameter `ulLen` may have the value 8 although the packet data contains more than 8 byte. `ulLen == 8` indicates that no ModuleDiffBlock exists for the specified IO-Device.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
<code>ulDest</code>	UINT32		Destination queue handle of APCTL-task process queue
<code>ulSrc</code>	UINT32		Source queue handle of AP-task process queue
<code>ulDestId</code>	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
<code>ulSrcId</code>	UINT32	0 ... $2^{32} - 1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
<code>ulLen</code>	UINT32	8 + x	APIOC_DCP_SET_SIGNAL_REQ_DATA_T - Packet data length in bytes
<code>ulId</code>	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
<code>ulSta</code>	UINT32	0	See below.
<code>ulCmd</code>	UINT32	0xC61	PNIO_APCTL_CMD_GET_MOD_DIFF_BLOCK_CNF - Command
<code>ulExt</code>	UINT32	0	Extension not in use, set to zero for compatibility reasons
<code>ulRout</code>	UINT32	0	Routing not in use, set to zero for compatibility reasons
<b>Structure tData (APIOC_GET_MOD_DIFF_BLOCK_CNF_DATA_T)</b>			
<code>ulHandle</code>	UINT32		Handle to the IO-Device.
<code>ulFlag</code>	UINT32		Flags. See below.
<code>usNumAPI</code>	UINT16		Number of APIs inside the ModuleDiffBlock.
<code>abModuleDiffBlock[1]</code>	UINT8[]		First byte of ModuleDiffBlock.

Table 92: APIOC\_GET\_MOD\_DIFF\_BLOCK\_CNF – Get ModuleDiffBlock Confirmation

#### Parameter `ulCmd`

```
#define PNIO_APCTL_CMD_GET_MOD_DIFF_BLOCK_CNF 0x00000C61
```



**Parameter ulFlag – Get ModuleDiffBlock Confirmation Flag**

31	30	...	0
Unused, set to zero.			ModuleDiffBlock is bigger than reported

Table 93: Flags used in Get ModuleDiffBlock confirmation packet

```
#define PNIO_APCTL_MOD_DIFF_BLOCK_TO_BIG (0x0001)
The ModuleDiffBlock is bigger than requests ulMaxRspLength.
```

**Parameter ulSta**

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC00C0002	TLR_E_PNIO_APCTL_RSC_OUTOFMEMORY Not enough free memory for request
0xC0000007	TLR_E_INVALID_PACKET_LENGTH Requests packet length is invalid.
0xC000000E	TLR_E_UNKNOWN_HANDLE The IO-Device handle is invalid.
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error sending request to another task.
0xC00C0032	TLR_E_PNIO_APCTL_INVALID_CMCTL_HANDLE The IO-Device handle is invalid.

Table 94: GET\_MOD\_DIFF\_BLOCK\_CNF - Status/Error Codes

## 5.5.8 DCP Signal

The DCP Signal Service is used to make a PROFINET device (IO-Device or IO-Controller) blink with a LED to be able to identify a single device.

The Signal Service in Hilscher PROFINET IO-Controller is implemented as follows: The request packet makes the IO-Controller send a DCP request packet to the specified IO-Device. That packet is resend by IO-Controller to IO-Device until IO-Controller receives another request packet with flag "PNIO\_APCTL\_DCP\_STOP\_SIGNAL" set in "ulFlags" from Application.

**Note:** This service cannot be used in parallel to signal multiple IO-Devices. Only access to one IO-Device per time is supported.

### 5.5.8.1 DCP Signal Request

The DCP Signal request requests the IO-Controller to send DCP SIGNAL packets to the specified IO-Device.

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
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	10	APIOC_DCP_SET_SIGNAL_REQ_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status not in use for request.
ulCmd	UINT32	0xC0E	PNIO_APCTL_CMD_DCP_SET_SIGNAL_REQ - 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
ulSrc	UINT32		Source queue handle of AP-task process queue
<b>Structure tData (APIOC_DCP_SET_SIGNAL_REQ_DATA_T)</b>			
ulFlags	UINT32		Flags for request. See below.
tMac[6]	UINT8[]		MAC-Address of device to signal.

Table 95: APIOC\_DCP\_SET\_SIGNAL\_REQ – DCP SIGNAL Request Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_DCP_SET_SIGNAL_REQ    0x00000C0E
```

**Parameter `ulFlags` – DCP Signal Request Flags**

Bit	Description
D5 ... D31	unused, set to zero
D4	Do not use this flag
D3	Do not use this flag
D2	Do not use this flag
D1	DCP START Signal
D0	DCP STOP Signal

*Table 96: Flags to use in DCP SIGNAL Request Packet***DCP Signal request Flag values:**

```
#define PNIO_APCTL_DCP_STOP_SIGNAL (0x0001)
STOP sending SIGNAL-requests to device.

#define PNIO_APCTL_DCP_START_SIGNAL (0x0002)
START sending SIGNAL-requests to device.
```

### 5.5.8.2 DCP Signal Confirmation

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	APIOC_DCP_SET_CNF_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32		See below.
ulCmd	UINT32	0xC0F	PNIO_APCTL_CMD_DCP_SET_SIGNAL_CNF - 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
<b>Structure tData (APIOC_DCP_SET_CNF_DATA_T)</b>			
ulDcpStatus	UINT32		DCP status.

Table 97: APIOC\_DCP\_SET\_CNF – DCP SIGNAL Confirmation Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_DCP_SET_SIGNAL_CNF 0x00000C0F
```

#### Parameter ulSta

Definition / (Value)	Description
0x00000000	TLR_S_OK Status ok
0xC00C0002	TLR_E_PNIO_APCTL_RSC_OUTOFMEMORY Not enough free memory for request
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error sending request to another task.
0xC00C0051	TLR_E_PNIO_APCTL_BUS_STATE_OFF The current bus state is OFF and no frames can be sent.
0xC0120302	TLR_E_PNIO_DCPUCS_RESOURCE_LIMIT_EXCEEDED Maximum amount of DCPUCS protocol machine instances reached
0xC0120303	TLR_E_PNIO_DCPUCS_RESOURCE_OUT_OF_MEMORY Not enough free memory in DCPUCS for request
0xC0120304	TLR_E_PNIO_DCPUCS_RESOURCE_STATE_INVALID DCPUCS state is invalid for this request
0xC0120305	TLR_E_PNIO_DCPUCS_RESOURCE_HANDLE_INVALID Handle to DCPUCS protocol machine instance is invalid
0xC0120306	TLR_E_PNIO_DCPUCS_TIMER_CREATE_FAILED Error while creating timer packet in DCPUCS
0xC0120307	TLR_E_PNIO_DCPUCS_TIMER_OUT_OF_MEMORY Not enough memory in DCPUCS to create a TLR-timer

Definition / (Value)	Description
0xC012030A	TLR_E_PNIO_DCPUCS_FRAME_OUT_OF_MEMORY DCPUCS did not get a frame from EDD
0xC012030B	TLR_E_PNIO_DCPUCS_FRAME_SEND_FAILED EDD reported error sending the DCP frame
0xC0130013	TLR_E_PNIO_NRPM_IDENTIFY_FLAG_INVALID Invalid identify flag requested from NRPM
0xC0130014	TLR_E_PNIO_NRPM_RESOURCE_LIMIT_EXCEEDED
0xC0130018	TLR_E_PNIO_NRPM_DCP_TYPE_INVALID The requested DCP type is invalid

*Table 98: DCP\_SIGNAL\_CNF - Status/Error Codes*

## 5.5.9 DCP SET Name

The DCP SET Name service can be used to set the name of an IO-Device.

**Note:** This service **cannot** be used in parallel to set the Name of multiple IO-Devices. Only access to one IO-Device per time is supported.

### 5.5.9.1 DCP SET Name Request

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	254	APIOC_DCP_SET_NAME_REQ_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status not in use for request.
ulCmd	UINT32	0xC0A	PNIO_APCTL_CMD_DCP_SET_NAME_REQ - 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
<b>Structure tData (APIOC_DCP_SET_NAME_REQ_DATA_T)</b>			
ulLenName	UINT32	1 .. 240	Length of the Name to set for device
ulFlags	UINT32		Flags for request. See below.
tMac[6]	UINT8[]		MAC-Address of device whose name shall be set.
tName[240]	UINT8[]		Name to set for specified device.

Table 99: APIOC\_DCP\_SET\_NAME\_REQ – DCP SET Name Request Packet

**Parameter ulCmd**

```
#define PNIO_APCTL_CMD_DCP_SET_NAME_REQ 0x00000C0A
```

**Parameter ulFlags – DCP SET Name Request Flags**

Bit	Description
D5 ... D31	Unused, set to zero
D4	Use STOP DCP Transaction Block
D3	Use START DCP Transaction Block
D2	Use no DCP Transaction Block
D1 ... D0	Unused, set to zero

*Table 100: Flags to use in DCP SET Name Request Packet***DCP Signal request Flag values:**

```
#define PNIO_PRM_NO_TR_BLOCK_TYPE (0x0004)  
Do not add a transaction block.  
  
#define PNIO_PRM_START_TR_BLOCK_TYPE (0x0008)  
Add a START transaction block.  
  
#define PNIO_PRM_STOP_TR_BLOCK_TYPE (0x0010)  
Add a STOP transaction block.
```

### 5.5.9.2 DCP SET Name Confirmation

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of APCTL-task process queue
ulSrc	UINT32		Source queue handle of AP-task process queue
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
ulLen	UINT32	4	APIOC_DCP_SET_CNF_DATA_T - 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	0xC0B	PNIO_APCTL_CMD_DCP_SET_NAME_CNF - 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
Structure tData (APIOC_DCP_SET_CNF_DATA_T)			
ulDcpStatus	UINT32		DCP status.

Table 101: APIOC\_DCP\_SET\_CNF – DCP SET NAME Confirmation Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_DCP_SET_NAME_CNF 0x00000C0B
```

#### Parameter ulsta

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC00C0002	TLR_E_PNIO_APCTL_RSC_OUTOFMEMORY Not enough free memory for request
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error sending request to another task.
0xC00C0033	TLR_E_PNIO_APCTL_INVALID_NAME_OF_STATION_LENGTH Requested NameOfStation has invalid length
0xC00C0034	TLR_E_PNIO_APCTL_DCP_REQUEST_LIMIT_REACHED Another DCP request is running
0xC00C0051	TLR_E_PNIO_APCTL_BUS_STATE_OFF The current bus state if OFF and no frames can be sent.
0xC0120302	TLR_E_PNIO_DCPUCS_RESOURCE_LIMIT_EXCEEDED Maximum amount of DCPUCS protocol machine instances reached
0xC0120303	TLR_E_PNIO_DCPUCS_RESOURCE_OUT_OF_MEMORY Not enough free memory in DCPUCS for request
0xC0120304	TLR_E_PNIO_DCPUCS_RESOURCE_STATE_INVALID DCPUCS state is invalid for this request



Value	Definition / Description
0xC0120305	TLR_E_PNIO_DCPUCS_RESOURCE_HANDLE_INVALID Handle to DCPUCS protocol machine instance is invalid
0xC0120306	TLR_E_PNIO_DCPUCS_TIMER_CREATE_FAILED Error while creating timer packet in DCPUCS
0xC0120307	TLR_E_PNIO_DCPUCS_TIMER_OUT_OF_MEMORY Not enough memory in DCPUCS to create a TLR-timer
0xC012030A	TLR_E_PNIO_DCPUCS_FRAME_OUT_OF_MEMORY DCPUCS did not get a frame from EDD
0xC012030B	TLR_E_PNIO_DCPUCS_FRAME_SEND_FAILED EDD reported error sending the DCP frame
0xC0130013	TLR_E_PNIO_NRPM_IDENTIFY_FLAG_INVALID Invalid identify flag requested from NRPM
0xC0130014	TLR_E_PNIO_NRPM_RESOURCE_LIMIT_EXCEEDED
0xC0130015	TLR_E_PNIO_NRPM_RESOURCE_OUT_OF_MEMORY Not enough free memory in NRPM for this request
0xC0130016	TLR_E_PNIO_NRPM_PACKET_SEND_FAILED NRPM could not send the packet to another task
0xC0130017	TLR_E_PNIO_NRPM_PACKET_OUT_OF_MEMORY Not enough free memory in NRPM to create a packet.
0xC0130018	TLR_E_PNIO_NRPM_DCP_TYPE_INVALID The requested DCP type is invalid

Table 102: DCP\_SET\_NAME\_CNF - Status/Error Codes

## 5.5.10 DCP SET IP

The DCP SET IP service can be used to set the IP of an IO-Device.

**Note:** This service cannot be used in parallel to set IPs of multiple IO-Devices. Only access to one IO-Device per time is supported.

### 5.5.10.1 DCP SET IP Request

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	22	APIOC_DCP_SET_IP_REQ_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status not in use for request.
ulCmd	UINT32	0xC0C	PNIO_APCTL_CMD_DCP_SET_IP_REQ - 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
<b>Structure tData (APIOC_DCP_SET_IP_REQ_DATA_T)</b>			
ulAddr	UINT32		New IP-Address.
ulMask	UINT32		New Subnet-mask.
ulGate	UINT32		New Gateway.
ulFlags	UINT32		Flags for request. See below.
tMac[6]	UINT8[]		MAC-Address of device whose name shall be set.

Table 103: APIOC\_DCP\_SET\_IP\_REQ – DCP SET IP Request Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_DCP_SET_IP_REQ 0x00000C0C
```

**Parameter `ulFlags` – DCP SET IP Request Flags**

Bit	Description
D6 ... D31	unused, set to zero
D5	The IO-Device shall store the IP parameters permanently. If this bit is not set the IP parameters are marked as “use temporarily”.
D4	Use STOP DCP Transaction Block
D3	Use START DCP Transaction Block
D2	Use no DCP Transaction Block
D1 ... D0	Unused, set to zero

*Table 104: Flags to use in DCP SET IP Request Packet***DCP Signal request Flag values:**

```
#define PNIO_PRM_NO_TR_BLOCK_TYPE      (0x0004)
Do not add a transaction block.

#define PNIO_PRM_START_TR_BLOCK_TYPE (0x0008)
Add a START transaction block.

#define PNIO_PRM_STOP_TR_BLOCK_TYPE  (0x0010)
Add a STOP transaction block.

#define PNIO_APCTL_DCP_SET_IP_PERMANENT      (0x0020)
Store the IP parameters permanently.
```

### 5.5.10.2 DCP SET IP Confirmation

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	APIOC_DCP_SET_CNF_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32		See below.
ulCmd	UINT32	0xC0D	PNIO_APCTL_CMD_DCP_SET_IP_CNF - 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
<b>Structure tData (APIOC_DCP_SET_CNF_DATA_T)</b>			
ulPnio	UINT32		DCP status.

Table 105: APIOC\_DCP\_SET\_CNF – DCP SET IP Confirmation Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_DCP_SET_IP_CNF    0x00000C0D
```

#### Parameter ulsta

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC00C0002	TLR_E_PNIO_APCTL_RSC_OUTOFMEMORY Not enough free memory for request
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error sending request to another task.
0xC00C0034	TLR_E_PNIO_APCTL_DCP_REQUEST_LIMIT_REACHED Another DCP request is running
0xC00C0038	TLR_E_PNIO_APCTL_DCP_REQUEST_NO_ANSWER The requested IO-Device did not answer to the DCP-request.
0xC00C0051	TLR_E_PNIO_APCTL_BUS_STATE_OFF The current bus state if OFF and no frames can be sent.
0xC0120302	TLR_E_PNIO_DCPUCS_RESOURCE_LIMIT_EXCEEDED Maximum amount of DCPUCS protocol machine instances reached
0xC0120303	TLR_E_PNIO_DCPUCS_RESOURCE_OUT_OF_MEMORY Not enough free memory in DCPUCS for request
0xC0120304	TLR_E_PNIO_DCPUCS_RESOURCE_STATE_INVALID DCPUCS state is invalid for this request
0xC0120305	TLR_E_PNIO_DCPUCS_RESOURCE_HANDLE_INVALID Handle to DCPUCS protocol machine instance is invalid

Value	Definition / Description
0xC0120306	TLR_E_PNIO_DCPUCS_TIMER_CREATE_FAILED Error while creating timer packet in DCPUCS
0xC0120307	TLR_E_PNIO_DCPUCS_TIMER_OUT_OF_MEMORY Not enough memory in DCPUCS to create a TLR-timer
0xC012030A	TLR_E_PNIO_DCPUCS_FRAME_OUT_OF_MEMORY DCPUCS did not get a frame from EDD
0xC012030B	TLR_E_PNIO_DCPUCS_FRAME_SEND_FAILED EDD reported error sending the DCP frame
0xC0130013	TLR_E_PNIO_NRPM_IDENTIFY_FLAG_INVALID Invalid identify flag requested from NRPM
0xC0130014	TLR_E_PNIO_NRPM_RESOURCE_LIMIT_EXCEEDED
0xC0130015	TLR_E_PNIO_NRPM_RESOURCE_OUT_OF_MEMORY Not enough free memory in NRPM for this request
0xC0130016	TLR_E_PNIO_NRPM_PACKET_SEND_FAILED NRPM could not send the packet to another task
0xC0130017	TLR_E_PNIO_NRPM_PACKET_OUT_OF_MEMORY Not enough free memory in NRPM to create a packet.
0xC0130018	TLR_E_PNIO_NRPM_DCP_TYPE_INVALID The requested DCP type is invalid

Table 106: DCP\_SET\_IP\_CNF - Status/Error Codes

## 5.5.11 DCP RESET FACTORYSETTINGS

The DCP Reset Factory Settings service can be used to reset an IO-Device to factory settings.

**Note:** This service **cannot** be used in parallel to reset multiple IO-Devices. Only access to one IO-Device per time is supported.

### 5.5.11.1 DCP Reset FactorySettings Request

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	APIOC_DCP_RESET_FACTORY_REQ_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status not in use for request.
ulCmd	UINT32	0xC86	PNIO_APCTL_CMD_DCP_RESET_FACTORY_REQ - 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
<b>Structure tData (APIOC_DCP_RESET_FACTORY_REQ_DATA_T)</b>			
tMac[6]	UINT8[]		MAC-Address of device to be reset.

Table 107: APIOC\_DCP\_RESET\_FACTORY\_REQ – DCP RESET FACTORY Request Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_DCP_RESET_FACTORY_REQ    0x00000C86
```

### 5.5.11.2 DCP Reset FactorySettings Confirmation

Structure Information			Type: Request
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of APCTL-task process queue
ulSrc	UINT32		Source queue handle of AP-task process queue
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
ulLen	UINT32	4	APIOC_DCP_SET_CNF_DATA_T - 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	0xC87	PNIO_APCTL_CMD_DCP_RESET_FACTORY_CNF - 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
Structure tData (APIOC_DCP_SET_CNF_DATA_T)			
ulPnio	UINT32		DCP status.

Table 108: APIOC\_DCP\_RESET\_FACTORY\_CNF – DCP RESET FACTORY Confirmation Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_DCP_RESET_FACTORY_CNF 0x00000C87
```

#### Parameter ulSta

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC00C0002	TLR_E_PNIO_APCTL_RSC_OUTOFMEMORY Not enough free memory for request
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error sending request to another task.
0xC00C0034	TLR_E_PNIO_APCTL_DCP_REQUEST_LIMIT_REACHED Another DCP request is running
0xC00C0038	TLR_E_PNIO_APCTL_DCP_REQUEST_NO_ANSWER The requested IO-Device did not answer to the DCP-request.
0xC00C0051	TLR_E_PNIO_APCTL_BUS_STATE_OFF The current bus state if OFF and no frames can be sent.
0xC0120302	TLR_E_PNIO_DCPUCS_RESOURCE_LIMIT_EXCEEDED Maximum amount of DCPUCS protocol machine instances reached
0xC0120303	TLR_E_PNIO_DCPUCS_RESOURCE_OUT_OF_MEMORY Not enough free memory in DCPUCS for request
0xC0120304	TLR_E_PNIO_DCPUCS_RESOURCE_STATE_INVALID DCPUCS state is invalid for this request

Value	Definition / Description
0xC0120305	TLR_E_PNIO_DCPUCS_RESOURCE_HANDLE_INVALID Handle to DCPUCS protocol machine instance is invalid
0xC0120306	TLR_E_PNIO_DCPUCS_TIMER_CREATE_FAILED Error while creating timer packet in DCPUCS
0xC0120307	TLR_E_PNIO_DCPUCS_TIMER_OUT_OF_MEMORY Not enough memory in DCPUCS to create a TLR-timer
0xC012030A	TLR_E_PNIO_DCPUCS_FRAME_OUT_OF_MEMORY DCPUCS did not get a frame from EDD
0xC012030B	TLR_E_PNIO_DCPUCS_FRAME_SEND_FAILED EDD reported error sending the DCP frame
0xC0130013	TLR_E_PNIO_NRPM_IDENTIFY_FLAG_INVALID Invalid identify flag requested from NRPM
0xC0130014	TLR_E_PNIO_NRPM_RESOURCE_LIMIT_EXCEEDED
0xC0130015	TLR_E_PNIO_NRPM_RESOURCE_OUT_OF_MEMORY Not enough free memory in NRPM for this request
0xC0130016	TLR_E_PNIO_NRPM_PACKET_SEND_FAILED NRPM could not send the packet to another task
0xC0130017	TLR_E_PNIO_NRPM_PACKET_OUT_OF_MEMORY Not enough free memory in NRPM to create a packet.
0xC0130018	TLR_E_PNIO_NRPM_DCP_TYPE_INVALID The requested DCP type is invalid

Table 109: DCP\_RESET\_FACTORY\_CNF - Status/Error Codes



## 5.5.12 DCP IDENT ALL

The DCP IDENT ALL service can be used to identify all devices (IO-Controller and IO-Device) that are connected to the network. If the field `ulSta` contains the value `TLR_S_OK` the IO-Controller was able to handle the DCP IDENT ALL Request.

Every single device connected to the network of IO-Controller will answer with some of its parameters which are then put in an Indication packet of type `APIOC_DCP_IDENT_ENTRY_PCK_T`. That packet is described in section *Acyclic Indications to Application* on page 174 and is sent to the application that is registered to APCTL-Task.

After the timeout specified in `APIOC_DCP_IDENT_ALL_PCK_T` is reached, a packet of type `APIOC_DCP_IDENT_FINISHED_PCK_T` will be sent to Application as indication for the finished DCP IDENT ALL request. That packet is described in section *DCP IDENT ALL Finished Service* on page 177.

### 5.5.12.1 DCP IDENT ALL Request

By sending this packet to IO-Controller the application requests it to send a DCP IDENT ALL packet to the network. The Application has to be ready to receive packets of type `APIOC_DCP_IDENT_ENTRY_PCK_T` from now on until the timeout specified is reached.

Structure Information				Type: Request
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32		Destination queue handle of APCTL-task process queue	
ulSrc	UINT32		Source queue handle of AP-task process queue	
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons	
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.	
ulLen	UINT32	4	APIOC_DCP_IDENT_ALL_REQ_DATA_T - Packet data length in bytes	
ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet	
ulSta	UINT32	0	Status not in use for request.	
ulCmd	UINT32	0xC10	PNIO_APCTL_CMD_DCP_IDENT_ALL_REQ - 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	
Structure tData (APIOC_DCP_IDENT_ALL_REQ_DATA_T)				
ulTimeout	UINT32	0..60	Timeout [is seconds] how long IO-Controller shall wait for incoming DCP IDENT ALL responses.	

Table 110: `APIOC_DCP_IDENT_ALL_REQ` – DCP IDENT ALL Request Packet

#### Parameter `ulCmd`

```
#define PNIO_APCTL_CMD_DCP_IDENT_ALL_REQ 0x00000C10
```

### 5.5.12.2 DCP IDENT ALL Confirmation

This packet is returned by IO-Controller as confirmation for the DCP IDENT ALL Request.

Structure Information			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	APIOC_DCP_IDENT_ALL_CNF_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32		See below.
ulCmd	UINT32	0xC11	PNIO_APCTL_CMD_DCP_IDENT_ALL_CNF - 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
<b>Structure tData (APIOC_DCP_IDENT_ALL_CNF_DATA_T)</b>			
ulReserved	UINT32	0	reserved

Table 111: APIOC\_DCP\_IDENT\_ALL\_CNF – DCP IDENT ALL Confirmation Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_DCP_IDENT_ALL_CNF 0x00000C11
```

#### Parameter ulSta

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC00C0002	TLR_E_PNIO_APCTL_RSC_OUTOFMEMORY Not enough free memory for request
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error sending request to another task.
0xC00C0034	TLR_E_PNIO_APCTL_DCP_REQUEST_LIMIT_REACHED Another DCP request is running
0xC00C0051	TLR_E_PNIO_APCTL_BUS_STATE_OFF The current bus state if OFF and no frames can be sent.

Table 112: DCP\_IDENT\_ALL\_CNF - Status/Error Codes

### 5.5.13 DCP GET

The DCP GET service can be used to read parameters from a device (IO-Controller or IO-Device) that is connected to the network via DCP.

It is possible to read either a single parameter only or multiple parameters at the same time from a device. However, it is not possible to use this service in parallel to multiple devices.

It is not possible to use this service in parallel to other DCP services like

- DCP Signal,
- DCP SET Name,
- DCP SET IP,
- DCP RESET FACTORYSETTINGS,
- DCP IDENT ALL

or Generic bus scan defined in [1].

#### 5.5.13.1 DCP GET Request

By sending this packet to IO-Controller the application requests it to send a DCP GET packet to the network. The Application has to provide the MAC address of the device the request shall be sent to.

Structure Information				Type: Request
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32		Destination queue handle of APCTL-task process queue	
ulSrc	UINT32		Source queue handle of AP-task process queue	
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons	
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.	
ulLen	UINT32	10	APIOC_DCP_GET_REQ_DATA_T - Packet data length in bytes	
ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet	
ulSta	UINT32	0	Status not in use for request.	
ulCmd	UINT32	0xC98	PNIO_APCTL_CMD_DCP_GET_REQ - 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	
Structure tData (APIOC_DCP_GET_REQ_DATA_T)				
ulGetOptFlags	UINT32	1..127	Field containa flags indicating which parameters shall be read using DCP GET (see Table 114: DCP GET Options Flags below)	
tMac[6]	UINT8[]		MAC-Address of device to be read from	

Table 113: APIOC\_DCP\_GET\_REQ\_T – DCP GET Request Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_DCP_GET_REQ 0x00000C98
```

**Parameter `ulGetOptFlags` – DCP GET options flag**

Bit	Description
D7 ... D31	unused, set to zero
D6	Read Device Instance <b>Note:</b> this option is not support by devices according to Profinet specification v2.2
D5	Read Device Role
D4	Read Device Identification (VendorID, DeviceID)
D3	Read NameOfStation
D2	Read TypeOfStation
D1	Read IP Parameters (IP address, network mask, gateway address)
D0	Read MAC Address

*Table 114: DCP GET Options Flags*

### 5.5.13.2 DCP GET Confirmation

This packet is returned by the IO-Controller as confirmation for the DCP GET Request.

Structure Information			Type: Indication
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of APCTL-task process queue
ulSrc	UINT32		Source queue handle of AP-task process queue
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
ulLen	UINT32	516	APIOC_DCP_GET_CNF_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status not in use for request.
ulCmd	UINT32	0xC99	PNIO_APCTL_CMD_DCP_GET_CNF - 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
Structure tData (APIOC_DCP_GET_CNF_DATA_T)			
ulValidFlags	UINT32	0...127	Field indicating which fields inside this packet contain valid information (see Table 114: DCP GET Options Flags)
ulIpAddr	UINT32		IP address of device.
ulNetMask	UINT32		Network mask of device.
ulGateway	UINT32		Gateway address of device.
usVendorId	UINT16		Vendor ID of device.
usDeviceId	UINT16		Device ID of device.
usInstance	UINT16		Instance of device
usRole	UINT16		Role of device
usLenNameOfStation	UINT16	1..240	Length of NameOfStation.
usLenTypeOfStation	UINT16	1..240	Length of TypeOfStation.
abNameOfStation[240]	UINT8[]		NameOfStation of device.
abTypeOfStation[240]	UINT8[]		TypeOfStation of device.
abMacAddr[6]	UINT8[]		MAC address of device.
bErrorCount	UINT8		Amount of options that could not be read successfully.
bBlockError	UINT8		Error Code returned by the last faulty option (if multiple errors occur only the last one is reported here).

Table 115: APIOC\_DCP\_GET\_CNF\_T – DCP GET Confirmation Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_DCP_GET_CNF 0x00000C99
```

**Parameter ulSta**

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC0000007	TLR_E_INVALID_PACKET_LEN The request packet length is invalid.
0xC0000009	TLR_E_INVALID_PARAMETER One of the parameters of request packet is invalid.
0xC000001A	TLR_E_REQUEST_RUNNING There is another DCP based request running.
0xC00C0002	TLR_E_PNIO_APCTL_RSC_OUTOFMEMORY Not enough free memory for request
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error sending request to another task.
0xC00C0034	TLR_E_PNIO_APCTL_DCP_REQUEST_LIMIT_REACHED Another DCP request is running
0xC00C0038	TLR_E_PNIO_APCTL_DCP_REQUEST_NO_ANSWER The device did not answer to DCP GET Request.
0xC00C0051	TLR_E_PNIO_APCTL_BUS_STATE_OFF The current bus state is OFF and no frames can be sent.

Table 116: DCP\_GET\_CNF - Status/Error Codes

## 5.5.14 Alarm Acknowledge Service

The Alarm Acknowledge Service shall be used by the application to request the IO-Controller to send an Alarm Data Acknowledge to an IO-Device that reported an alarm. This service should be used to acknowledge an alarm from an IO-Device after receiving the alarm indication which described in section *Alarm Service* on page 180.

### 5.5.14.1 Alarm Acknowledge Request

This packet shall be send by application to IO-Controller to request it sending an Alarm Data Acknowledge packet to an IO-Device. The alarm specifier shall be taken from the alarm indication packet (see section 5.6.3.1 “*Alarm Indication*”). It is necessary to identify the alarm inside IO-Controller as it may be possible to have more than one alarm per IO-Device.

**Note:** There exist two different request packets. The newer one is presented first and shall be used whenever possible. The older one is only shown for backwards compatibility reasons and shall not be used.

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	10	APIOC_ALARM_ACK_REQ_DATA_V2_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status not in use for request.
ulCmd	UINT32	0xC26	PNIO_APCTL_CMD_APPL_ALARM_ACK_REQ - 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
<b>Structure tData (APIOC_ALARM_ACK_REQ_DATA_V2_T)</b>			
ulHandle	UINT32		Handle to IO-Device to send the Alarm Data Acknowledge to. How to get the handle see section <i>Device Handle</i> and obtainin detailed Information of an IO Device on page 55.
usAlarmSpeci fier	UINT16		Alarm specifier, use value from Alarm Indication Service (see section 5.6.3.1)
usReserved	UINT16	0	Reserved, set to zero.
usAlarmPrior ity	UINT16		Priority of alarm (value shall match the priority of the alarm indication this acknowledge belongs to)

Table 117: APIOC\_ALARM\_ACK\_REQ - Alarm Acknowledge Request Packet

The following table describes the packet which was used up to stack version 2.5.x.x

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	APIOC_ALARM_ACK_REQ_DATA_V1_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status not in use for request.
ulCmd	UINT32	0xC26	PNIO_APCTL_CMD_APPL_ALARM_ACK_REQ - 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
<b>Structure tData (APIOC_ALARM_ACK_REQ_DATA_V1_T)</b>			
ulHandle	UINT32		Handle to IO-Device to send the Alarm Data Acknowledge to. How to get the handle see section <i>Device Handle and obtainin detailed Information of an IO Device</i> on page 55.
usAlarmSpeci fier	UINT16		Alarm specifier, use value from Alarm Indication Service (see section <i>Alarm Indication</i> on page 180)
usReserved	UINT16	0	Reserved, set to zero.

Table 118: APIOC\_ALARM\_ACK\_REQ\_V1 - Alarm Acknowledge Request Packet (legacy)

### Parameter ulCmd

```
#define PNIO_APCTL_CMD_APPL_ALARM_ACK_REQ    0x00000C26
```



### 5.5.14.2 Alarm Acknowledge Confirmation

The IO-Controller will return the Alarm Acknowledge Request packet as Alarm Acknowledge Confirmation packet with the following structure. The status of sending the Alarm Acknowledge will be reported.

Structure Information				Type: Confirmation
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32		Destination queue handle of APCTL-task process queue	
ulSrc	UINT32		Source queue handle of AP-task process queue	
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons	
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.	
ulLen	UINT32	12	APIOC_ALARM_ACK_CNF_DATA_T - 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	Status / Error code	
ulCmd	UINT32	0xC27	PNIO_APCTL_CMD_APPL_ALARM_ACK_CNF - 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	
Structure tData (APIOC_ALARM_ACK_CNF_DATA_T)				
ulHandle	UINT32		Handle to IO-Device to send the Alarm Data Acknowledge to	
usAlarmSpecifier	UINT16		Alarm_Specifier the Alarm ACK was sent for.	
usReserved	UINT16	0	Reserved, set to zero.	
ulPnio	UINT32		PNIO status	

Table 119: APIOC\_ALARM\_ACK\_CNF - Alarm Acknowledge Confirmation Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_APPL_ALARM_ACK_CNF    0x00000C27
```

**Parameter ulSta**

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC00A0002	TLR_E_PNIO_STATUS Generic error code. See packets data-status ulPnio for details.
0xC00C0002	TLR_E_PNIO_APCTL_RSC_OUTOFMEMORY Not enough free memory for request
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error sending request to another task.
0xC00C0032	TLR_E_PNIO_APCTL_INVALID_CMCTL_HANDLE The IO-Device handle is invalid
0xC0110030	TLR_E_PNIO_ALPMR_PRIORITY_INVALID Invalid alarm priority in request packet of ALPMR_AlarmAck_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.
0xC0110037	TLR_E_PNIO_ALPMR_RESOURCE_INDEX_INVALID The index of ALPMR's protocol machine 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.
0xC0110057	TLR_E_PNIO_APMS_TIMER_OUT_OF_MEMORY Insufficient memory for APMS_Send_req_Data() to allocate a timer-indication packet.

Table 120: Alarm Acknowledge Confirmation - Status Codes

## 5.5.15 Release IO-Device Service

This service can be used to actively release an IO-Device. The IO-Controller will send a RPC Release Request to the IO-Device and afterwards all state machines related to the IO-Device are shut down.

**The IO-Controller will NOT try to reconnect to that IO-Device any more!**

### 5.5.15.1 Release IO-Device Request

This packet shall be send by the application to IO-Controller to request it releasing an IO-Device. The IO-Controller will then send an RPC Release Request packet and afterwards shut down all associated protocol machines.

Structure Information APIOC_RELEASE_REQ_T			Type: Request
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of APCTL-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	APIOC_RELEASE_REQ_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status not in use for request.
ulCmd	UINT32	0xC2E	PNIO_APCTL_CMD_RELEASE_REQ - 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
Structure tData (APIOC_RELEASE_REQ_DATA_T)			
ulHandle	UINT32		Handle to IO-Device to release. How to get the handle see section <i>Device Handle and obtainin detailed Information of an IO Device</i> on page 55.

Table 121: Release IO-Device Request Packet Definition

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_RELEASE_REQ 0x00000C2E
```

### 5.5.15.2 Release IO-Device Confirmation

The IO-Controller will return the Release Request packet as Release Confirmation packet with the following structure. The status of sending the Release will be reported.

Structure Information (APIOC_RELEASE_CNF_T)			Type: Confirmation
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	APIOC_RELEASE_CNF_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	See below	Status / Error code
ulCmd	UINT32	0xC2F	PNIO_APCTL_CMD_RELEASE_CNF - 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
<b>Structure tData (APIOC_RELEASE_CNF_DATA_T)</b>			
ulHandle	UINT32		Handle to IO-Device that was released

Table 122: Release IO-Device Confirmation Packet Definition

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_RELEASE_CNF 0x00000C2F
```

**Parameter ulSta**

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC00A0012	TLR_E_PNIO_CMCTL_RESOURCE_OUT_OF_MEMORY Insufficient memory for this request.
0xC00A0014	TLR_E_PNIO_CMCTL_STATE_CONFLICT This request cannot be served in current CMCTL state.
0xC00A0018	TLR_E_PNIO_CMCTL_PACKET_SEND_FAILED Error while sending a packet to another task.
0xC00A0040	TLR_E_PNIO_CMCTL_INVALID_PM_INDEX The CMCTL protocol-machine restored from index is invalid.
0xC00A0041	TLR_E_PNIO_CMCTL_INVALID_PM The index of CMCTL protocol-machine is invalid.
0xC00C0002	TLR_E_PNIO_APCTL_RSC_OUTOFMEMORY Not enough free memory for request
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error sending request to another task.
0xC00C0032	TLR_E_PNIO_APCTL_INVALID_CMCTL_HANDLE The IO-Device handle is invalid
0xC02E0200	TLR_E_CLRPC_PACKET_SEND_FAILED Error while sending internal message to another task.
0xC02E0201	TLR_E_CLRPC_TIMER_OUT_OF_MEMORY Creating a TLR-Timer-packet in RPC task failed due to insufficient memory.
0xC02E0605	TLR_E_CLRPC_CLIENT_HANDLE_INVALID The handle to RPC-Client instance is invalid.
0xC02E0606	TLR_E_CLRPC_CLIENT_REQUEST_LIMIT_EXCEEDED The maximum amount of outstanding RPC-Requests for this RPC-Clients instance is reached.

Table 123: Release IO-Device Confirmation - Status Codes

## 5.6 Acyclic Indications to Application

This section describes Indications that are sent by IO-Controller to the registered application. DCP IDENT Entries and alarms are examples that are reported this way. If no application is registered the IO-Controller is not able to indicate alarms and therefore they may be handled incorrectly.

In detail, the following functionality is provided by the configuration packets of the PROFINET IO Controller Protocol Stack:

Overview over Acyclic Indication Packets of the PROFINET IO Controller Stack			
No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
5.6.1	DCP IDENT ENTRY Indication	0x0C24	174
	DCP IDENT ENTRY Response	0x0C25	176
5.6.2	DCP IDENT ALL Finished Indication	0x0C2C	177
	DCP IDENT ALL Finished Response	0x0C2D	179
5.6.3	Alarm Indication	0x0C80	180
	Alarm Response	0x0C81	182
5.6.4	Diagnosis Indication	0x0C82	184
	Diagnosis Response	0x0C83	186

Table 124: Overview over the Acyclic Indication Packets of the PROFINET IO Controller Protocol Stack

### 5.6.1 DCP IDENT ENTRY Service

The DCP IDENT Entry service is used by IO-Controller to report a received response to a DCP IDENT ALL request (see section 5.5.12) to the application registered to APCTL-Task. The application has to return the packet.

#### 5.6.1.1 DCP IDENT ENTRY Indication

This indication packet is send from IO-Controller to application if IO-Controller receives a DCP IDENT ALL response. Pay attention that the value for `TypeOfStation` is not shown in packet structure although it exists. It lies directly behind the last byte of `NameOfStation`. That is done for efficient memory usage inside IO-Controller. You can find the starting position by using the fields `usLenName` and `usLenType`.

Structure Information			Type: Indication
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
<code>ulDest</code>	UINT32		Destination queue handle of APCTL-task process queue
<code>ulSrc</code>	UINT32		Source queue handle of AP-task process queue
<code>ulDestId</code>	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
<code>ulSrcId</code>	UINT32	0 ... $2^{32} - 1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.

Structure Information			Type: Indication
ulLen	UINT32	33 + n	APIOC_DCP_IDENT_ENTRY_IND_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status not in use for request.
ulCmd	UINT32	0xC24	PNIO_APCTL_CMD_DCP_IDENT_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
Structure tData (APIOC_DCP_IDENT_ENTRY_IND_DATA_T)			
ulIp	UINT32		IP address of device.
ulNetMask	UINT32		Network mask of device.
ulGateway	UINT32		Gateway address of device.
usLenName	UINT16	1..240	Length of NameOfStation.
usLenType	UINT16	1..240	Length of TypeOfStation.
usVendorId	UINT16		Vendor ID of device.
usDeviceId	UINT16		Device ID of device.
usDeviceRole	UINT16		Role of device.
abMac[6]	UINT8[]		MAC address of device.
abNameOfStation[1]	UINT8[]		First byte of NameOfStation. Remaining bytes follow. Directly behind the last byte of abNameOfStation the field abTypeOfStation begins.

Table 125: APIOC\_DCP\_IDENT\_ENTRY\_REQ – DCP IDENT ENTRY Indication Packet

**Parameter ulCmd**

```
#define PNIO_APCTL_CMD_DCP_IDENT_IND    0x00000C24
```

### 5.6.1.2 DCP IDENT ENTRY Response

The IDENT ENTRY packet has to be returned by application to IO-Controller.

Structure Information			Type: Response
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	APIOC_DCP_IDENT_RSP_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32		See below
ulCmd	UINT32	0xC25	PNIO_APCTL_CMD_DCP_IDENT_RSP - 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
<b>Structure tData (APIOC_DCP_IDENT_ENTRY_RSP_DATA_T)</b>			
ulReserved	UINT32	0	Reserved value, set to zero.

Table 126: APIOC\_DCP\_IDENT\_ENTRY\_RSP – DCP IDENT ENTRY Response Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_DCP_IDENT_RSP 0x00000C25
```

#### Parameter ulSta

Value	Definition / Description
0x00000000	TLR_S_OK Status ok

Table 127: DCP\_IDENT\_ENTRY\_RSP - Status/Error Codes



## 5.6.2 DCP IDENT ALL Finished Service

The DCP IDENT ALL Finished Service is used by IO-Controller to indicate the end of a DCP IDENT ALL Request to Application.

### 5.6.2.1 DCP IDENT ALL Finished Indication

This packet is sent by IO-Controller to application after the timeout specified in DCP IDENT ALL Requests field `ulTimeout` is reached. No more incoming IDENT ALL Responses of devices will be reported to application afterwards.

Structure Information (APIOC_DCP_IDENT_FINISHED_PCK_T)			Type: Request
Variable	Type	Value / Range	Description
Structure TLR_PACKET_HEADER_T			
<code>ulDest</code>	UINT32		Destination queue handle of APCTL-task process queue
<code>ulSrc</code>	UINT32		Source queue handle of AP-task process queue
<code>ulDestId</code>	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
<code>ulSrcId</code>	UINT32	0 ... $2^{32} - 1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
<code>ulLen</code>	UINT32	4	APIOC_DCP_IDENT_ALL_CNF_DATA_T - Packet data length in bytes
<code>ulId</code>	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
<code>ulSta</code>	UINT32		See below.
<code>ulCmd</code>	UINT32	0xC2C	PNIO_APCTL_CMD_DCP_IDENT_FINISHED_IND - Command
<code>ulExt</code>	UINT32	0	Extension not in use, set to zero for compatibility reasons
<code>ulRout</code>	UINT32	0	Routing not in use, set to zero for compatibility reasons
Structure tData (APIOC_DCP_IDENT_FINISHED_IND_DATA_T)			
<code>ulDevCnt</code>	UINT32	0 ... $2^{32} - 1$	Amount of IDENT ALL responses the IO-Controller received. Can be greater than the amount of IDENT ENTRY packets that were sent to application if requests <code>ulMaxRsp</code> was too small.

Table 128: APIOC\_DCP\_IDENT\_FINISHED\_IND – DCP IDENT ALL Finished Indication Packet

#### Parameter `ulCmd`

```
#define PNIO_APCTL_CMD_DCP_IDENT_FINISHED_IND 0x00000C2C
```

**Parameter ulSta**

Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC00C0002	TLR_E_PNIO_APCTL_RSC_OUTOFMEMORY Not enough free memory for request
0xC00C0031	TLR_E_PNIO_APCTL_PACKET_SEND_FAILED Error sending request to another task.
0xC00C0034	TLR_E_PNIO_APCTL_DCP_REQUEST_LIMIT_REACHED Another DCP request is running
0xC0120102	TLR_E_PNIO_DCPMCS_RESOURCE_LIMIT_EXCEEDED Maximum amount of parallel DCPMCS protocol machine instances reached
0xC0120103	TLR_E_PNIO_DCPMCS_RESOURCE_OUT_OF_MEMORY Not enough memory to create new DCPMCS protocol machine instance
0xC0120104	TLR_E_PNIO_DCPMCS_RESOURCE_STATE_INVALID State of DCPMCS protocol machine instance is invalid
0xC0120105	TLR_E_PNIO_DCPMCS_RESOURCE_HANDLE_INVALID Handle to DCPMCS protocol machine instance is invalid
0xC0120106	TLR_E_PNIO_DCPMCS_TIMER_CREATE_FAILED DCPMCS could not create TLR-timer instance
0xC0120107	TLR_E_PNIO_DCPMCS_TIMER_OUT_OF_MEMORY DCPMCS could not create timer packet
0xC012010A	TLR_E_PNIO_DCPMCS_FRAME_OUT_OF_MEMORY DCPMCS did not get frame from EDD
0xC012010B	TLR_E_PNIO_DCPMCS_FRAME_SEND_FAILED Error in EDD while sending DCP frame
0xC0130013	TLR_E_PNIO_NRPM_IDENTIFY_FLAG_INVALID Invalid identify flag requested from NRPM
0xC0130014	TLR_E_PNIO_NRPM_RESOURCE_LIMIT_EXCEEDED
0xC0130015	TLR_E_PNIO_NRPM_RESOURCE_OUT_OF_MEMORY Not enough free memory in NRPM for this request
0xC0130016	TLR_E_PNIO_NRPM_PACKET_SEND_FAILED NRPM could not send the packet to another task
0xC0130017	TLR_E_PNIO_NRPM_PACKET_OUT_OF_MEMORY Not enough free memory in NRPM to create a packet.
0xC0130018	TLR_E_PNIO_NRPM_DCP_TYPE_INVALID The requested DCP type is invalid

*Table 129: DCP\_IDENT\_FINISHED\_IND - Status/Error Codes*

### 5.6.2.2 DCP IDENT ALL Finished Response

This packet has to be returned to IO-Controller by application.

Structure Information			Type: Request
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	APIOC_DCP_IDENT_ALL_CNF_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32		See below.
ulCmd	UINT32	0xC2D	PNIO_APCTL_CMD_DCP_IDENT_FINISHED_RSP - 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
<b>Structure tData (APIOC_DCP_IDENT_FINISHED_RSP_DATA_T)</b>			
ulReserved	UINT32	0	reserved

Table 130: APIOC\_DCP\_IDENT\_FINISHED\_RSP – DCP IDENT ALL Finished Response Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_DCP_IDENT_FINISHED_RSP 0x00000C2D
```

#### Parameter ulSta

Value	Definition / Description
0x00000000	TLR_S_OK Status ok

Table 131: DCP\_IDENT\_FINISHED\_IND - Status/Error Codes

### 5.6.3 Alarm Service

The Alarm Service is used by IO-Controller to report alarms of connected IO-Devices which were received by IO-Controller to the registered application. All alarms are reported to the application. The application has to confirm the alarm indication at first. Afterwards it has to explicitly request the IO-Controller to handle the alarms (see section 5.5.14).

#### Example:

An IO-Device reports a PLUG-Alarm. The IO-Controller indicates that alarm to the application. Now the application has to confirm the indication to the IO-Controller (by returning the indication packet). Afterwards the Application has to use the Alarm Acknowledge Service (see section 5.5.14) to make IO-Controller perform the corresponding action like sending the configuration to the IO-Device and sending the Alarm-Data-Acknowledge.

Note that alarms that have to be reported to application are stored internally in IO-Controller if no application is registered. After a new application registers, it will get all stored alarm indications at once.

#### 5.6.3.1 Alarm Indication

The Alarm Indication packet is sent by IO-Controller to the registered application to indicate that a connected IO-Device reported an alarm.

Some alarms do not have additional alarm data. For those alarms the packet definition is different in the way that the fields *usUserStructID* and *abAlarmData* are not present. This is indicated to the Application by packet length 26. If the alarm contains additional data the fields *usUserStructID* and *abAlarmData* are present and the packet length is equal or greater than 29.

Structure Information (APIOC_ALARM_IND_T)			Type: Indication
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	26 29 + n	Packet data length in bytes no additional data (packet ends after <i>usAlarmSpecifier</i> ) additional data, packet contains fields <i>usUserStructID</i> and <i>abAlarmData</i>
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	0xC80	PNIO_APCTL_CMD_APPL_ALARM_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
<b>Structure tData (APIOC_ALARM_IND_DATA_T)</b>			
ulHandle	UINT32		Handle of IO-Device that reported the alarm. How to get the handle see section <i>Device Handle and obtainin detailed Information of an IO Device</i> on page 55.
ulApi	UINT32		API of alarm
usSlot	UINT16		Slot of alarm

Structure Information (APIOC_ALARM_IND_T)			Type: Indication
usSubSlot	UINT16		Subslot of alarm
ulModuleID	UINT32		ModuleID of alarm
ulSubmoduleID	UINT32		SubmoduleID of alarm
usAlarmPriority	UINT16		Priority of alarm
usAlarmType	UINT16		Type of alarm
usAlarmSpecifier	UINT16		See below
usUserStructIdent	UINT16		UserStruct Identifier of alarm <b>OPTIONAL, only present if packet-length != 26</b>
abAlarmData[1]	UINT8[]		First byte of additional alarm data, other bytes follow directly behind <b>OPTIONAL, only present if packet-length != 26</b>

Table 132: APIOC\_ALARM\_IND - Alarm Indication Packet

**Parameter ulCmd**

```
#define PNIO_APCTL_CMD_APPL_ALARM_IND 0x00000C80
```

**Parameter usAlarmSpecifier – Alarm specifier (see [PNP] for details)**

Bit	Meaning	Value
0 ... 10	Alarm Sequence Number	0 ... 2047
11	Channel Diagnosis	0 ... 1
12	Manufacturer specific Diagnosis	0 ... 1
13	Submodule Diagnosis State	0 ... 1
14	Reserved	0
15	AR Diagnosis State	0 ... 1

Table 133: Alarm Specifier

### 5.6.3.2 Alarm Response

The alarm indication has to be returned by application to IO-Controller as Alarm response. The packet is defined in table Table 134: APIOC\_ALARM\_RSP - Alarm Response Packet.

**No further action is done internally in the IO-Controller until that alarm response arrives!** If you want to acknowledge the alarm you have to use the Alarm Acknowledge Request Service described in section 5.5.14.

**Note:** There exist two different response packets. The newer one is presented first and shall be used whenever possible. The older one is only shown for backwards compatibility reasons and shall not be used.

Structure Information (APIOC_ALARM_RSP_V2_T)				Type: Response
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32		Destination queue handle of APCTL-task process queue	
ulSrc	UINT32		Source queue handle of AP-task process queue	
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons	
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.	
ulLen	UINT32	10	APIOC_ALARM_RSP_DATA_V2_T - Packet data length in bytes	
ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet	
ulSta	UINT32	0	Status not to be used for this response.	
ulCmd	UINT32	0xC81	PNIO_APCTL_CMD_APPL_ALARM_RSP - 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	
Structure tData (APIOC_ALARM_RSP_DATA_V2_T)				
ulHandle	UINT32		Handle of IO-Device	
usAlarmSpecifier	UINT16		Alarm specifier of alarm that was indicated.	
usReserved	UINT16	0	Reserved, set to zero	
usAlarmPriority	UINT16		Priority of alarm	

Table 134: APIOC\_ALARM\_RSP - Alarm Response Packet

The following table describes the packet which was used up to stack version 2.5.x.x

Structure Information (APIOC_ALARM_RSP_V1_T)				Type: Response
Variable	Type	Value / Range	Description	
Structure TLR_PACKET_HEADER_T				
ulDest	UINT32		Destination queue handle of APCTL-task process queue	
ulSrc	UINT32		Source queue handle of AP-task process queue	
ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons	
ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.	
ulLen	UINT32	8	APIOC_ALARM_RSP_DATA_V1_T - Packet data length in bytes	
ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the source process of the packet	
ulSta	UINT32	0	Status not to be used for this response.	

Structure Information (APIOC_ALARM_RSP_V1_T)			Type: Response
ulCmd	UINT32	0xC81	PNIO_APCTL_CMD_APPL_ALARM_RSP - 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
Structure tData (APIOC_ALARM_RSP_DATA_V1_T)			
ulHandle	UINT32		Handle of IO-Device
usAlarmSpecifier	UINT16		Alarm specifier of alarm that was indicated.
usReserved	UINT16	0	Reserved, set to zero

Table 135: APIOC\_ALARM\_RSP\_V1 - Alarm Response Packet (legacy)

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_APPL_ALARM_RSP    0x00000C81
```

## 5.6.4 Diagnosis Service

The Diagnosis Service is used by IO-Controller to indicate to application that new IO-Device diagnosis data is present. Both diagnosis and diagnosis-disappears alarms are indicated with this service. The Indication packet contains the whole diagnosis-data.

The application may read the diagnosis data later by using the service described in section 5.5.4.

### 5.6.4.1 Diagnosis Indication

The following packet is sent by IO-Controller to registered application to indicate that new diagnosis data exists for the specified IO-Device or that an existing diagnosis is no longer present.

The diagnosis data contained in the array `abDiagData` is directly taken from the Diagnosis Alarm frame and not swapped. The firmware has no knowledge about the content and it is in the responsibility of the application to swap the content if needed according to the diagnosis type indicated in `usUserStructIdent`.

Structure Information (APIOC_DIAG_DATA_IND_T)			Type: Indication
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
<code>ulDest</code>	UINT32		Destination queue handle of APCTL-task process queue
<code>ulSrc</code>	UINT32		Source queue handle of AP-task process queue
<code>ulDestId</code>	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
<code>ulSrcId</code>	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
<code>ulLen</code>	UINT32	28 + n	APIOC_DIAG_DATA_IND_DATA_T - Packet data length in bytes
<code>ulId</code>	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the source process of the packet
<code>ulSta</code>	UINT32	0	Status not in use for indication.
<code>ulCmd</code>	UINT32	0xC82	PNIO_APCTL_CMD_APPL_DIAG_DATA_IND - Command
<code>ulExt</code>	UINT32	0	Extension not in use, set to zero for compatibility reasons
<code>ulRout</code>	UINT32	0	Routing not in use, set to zero for compatibility reasons
<b>Structure tData (APIOC_DIAG_DATA_IND_DATA_T)</b>			
<code>ulHandle</code>	UINT32	0 ... $2^{32}-1$	Handle of IO-Device for which new diagnosis data are available. How to get the handle see section <i>Device Handle</i> and obtainin detailed Information of an IO Device on page 55.
<code>ulApi</code>	UINT32		API of the submodule with diagnosis
<code>usSlot</code>	UINT16		Slot of the submodule with diagnosis
<code>usSubSlot</code>	UINT16		Subslot of the submodule with diagnosis
<code>ulModuleID</code>	UINT32		ModuleID of the module with diagnosis
<code>ulSubmoduleID</code>	UINT32		SubmoduleID of the module with diagnosis
<code>usAlarmPriority</code>	UINT16		Priority of diagnosis alarm
<code>usAlarmType</code>	UINT16		Type of alarm
<code>usAlarmSpecifier</code>	UINT16		See below
<code>usUserStructIdent</code>	UINT16		UserStruct Identifier
<code>abDiagData[1]</code>	UINT8[]		First byte of additional diagnosis data, other bytes follow directly behind <b>OPTIONAL, only present if packet-length &gt; 28</b> The content of this array is in network byte order. If this is e.g. an Extended Channel Diagnosis the fields have to be swapped by the application to get the correct values.

Table 136: APIOC\_DIAG\_DATA\_IND\_T - New Diagnosis Data Indication Packet



**Parameter** `ulCmd`

```
#define PNIO_APCTL_CMD_APPL_DIAG_DATA_IND 0x00000C82
```

### 5.6.4.2 Diagnosis Response

The application shall return the packet indicating new device diagnosis as diagnosis response packet with the following layout.

Structure Information (APIOC_DIAG_DATA_RSP_T)			Type: Response
Variable	Type	Value / Range	Description
<b>Structure TLR_PACKET_HEADER_T</b>			
ulDest	UINT32		Destination queue handle of APCTL-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	APIOC_DIAG_DATA_RSP_DATA_T - Packet data length in bytes
ulId	UINT32	0 ... $2^{32} - 1$	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	Status not to be used for this response.
ulCmd	UINT32	0xC83	PNIO_APCTL_CMD_APPL_DIAG_DATA_RSP - 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
<b>Structure tData (APIOC_DIAG_DATA_RSP_DATA_T)</b>			
ulHandle	UINT32		Handle of IO-Device

Table 137: APIOC\_DIAG\_DATA\_RSP\_T - New Diagnosis Data Response Packet

#### Parameter ulCmd

```
#define PNIO_APCTL_CMD_APPL_DIAG_DATA_RSP    0x00000C83
```

## 6 Status/Error Codes Overview

The following tables contain all error codes that can be contained in any confirmation/response packet send by Hilscher Profinet IO-Controller to User-Application.

### 6.1 Error Codes of the ACP – Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
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.
0xC0110018	TLR_E_PNIO_ACP_EMPTY_POOL_DETECTED The packet pool of ACP is empty.

Table 138: Error Messages of the ACP-Task

### 6.1.1 Diagnostic Codes of the ACP – Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
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 Failed 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 139: Diagnostic Messages of the ACP-Task

## 6.2 Error Codes of the APCFG –Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC0140002	TLR_E_PNIO_APCFG_DBM_UNKNOWN_VERSION Unknown DBM version.
0xC0140003	TLR_E_PNIO_APCFG_DBM_NO_DATASET No dataset existing.
0xC0140004	TLR_E_PNIO_APCFG_DBM_WRONG_SIZE_OF_DATASET Wrong size of DBM dataset.
0xC0140005	TLR_E_PNIO_APCFG_DBM_WRONG_LEN_TYPEOFSTATION Type of station is too large.
0xC0140006	TLR_E_PNIO_APCFG_DBM_WRONG_LEN_NAMEOFSTATION Name of station is too large.
0xC0140007	TLR_E_PNIO_APCFG_DBM_UNKNOWN_IOCR_KEY Unkwown IOCR relationship in submodule description.
0xC0140008	TLR_E_PNIO_APCFG_DBM_IOCR_ALREADY_IN_USE IOCR is in use by another IO-Device.
0xC0140009	TLR_E_PNIO_APCFG_DBM_WRONG_IOCR_IN_SUBMDESCR Wrong IO-Type of IOCR in submodule description.
0xC014000A	TLR_E_PNIO_APCFG_DBM_WRONG_DATALEN_IN_SUBMDESCR Wrong data length in submodule description.
0xC014000B	TLR_E_PNIO_APCFG_DBM_WRONG_DATADESCR_IN_SUBMDESCR Wrong IO-type in data description of the submodule description.
0xC014000C	TLR_E_PNIO_APCFG_DBM_UNEXP_SUBMDESCR Unexpected submodule description.
0xC014000D	TLR_E_PNIO_APCFG_DBM_MISSING_SUBMDESCR Missing submodule description.
0xC014000E	TLR_E_PNIO_APCFG_DBM_ASSERTION_FAILED Assertion failed.
0xC014000F	TLR_E_PNIO_APCFG_PKT_UNEXP_TREE_IDENTIFICATON Wrong sequence/numbering in the tree identification numbers.
0xC0140010	TLR_E_PNIO_APCFG_PKT_OVERWRITING_CONSISTING_DATA Addressed dataset is already existing and would be overwritten.
0xC0140011	TLR_E_PNIO_APCFG_PKT_MISSING_IOCR Input or output IOCR of module is missing.
0xC0140012	TLR_E_PNIO_APCFG_PKT_WRONG_IO_TYPE_IN_IOCR Wrong input or output type in IOCR.
0xC0140013	TLR_E_PNIO_APCFG_PKT_WRONG_SEQUENCE_OF_FRAGMENTED_PACKETS Wrong sequence of fragmented packets.
0xC0140014	TLR_E_PNIO_APCFG_PKT_WRONG_FRAGMENT_IDENTIFIER Wrong fragment identifier in fragmented packets.
0xC0140015	TLR_E_PNIO_APCFG_PKT_CONFIGURATION_IS_ALREADY_DONE_VIA_DBM Configuration is already done via DBM configuration packets are not accepted.

Hexadecimal Value	Definition Description
0xC0140016	TLR_E_PNIO_APCFG_DBM_INCOMPLETE_CONFIGURATION Incomplete configuration - not all devices are fully developed until submodule descriptions.
0xC0140017	TLR_E_PNIO_APCFG_PKT_DOWNLOAD_ALREADY_FINISHED Packet download is already finished by download finish request.
0xC0140018	TLR_E_PNIO_APCFG_DBM_UNEXP_NUMBER_OF_RECORDS_IN_PNIOC_TABLES More than one entry for one key was found in the tables of the fieldbus specific data.
0xC0140019	TLR_E_PNIO_APCFG_DBM_UNEXP_SIGNAL_ATTRIBUT Unexpected parameter Signal_Attrib in SIGNALS table.
0xC014001A	TLR_E_PNIO_APCFG_DBM_IMPROPER_DPM_OFFSETS_AND_DATA_LENGTHS_I N_SIGNALS Improper DPM offset and data length in SIGNALS.
0xC0140020	TLR_E_PNIO_APCFG_DBM_WRONG_TEST_DATA Wrong test data.
0xC0140021	TLR_E_PNIO_APCFG_DBM_INVALID_IO_DEVICE_AMOUNT Too many IO-Devices are configured.
0xC0140030	TLR_E_PNIO_APCFG_RESOURCE_OUT_OF_MEMORY Not enough memory available for current request.
0xC0140031	TLR_E_PNIO_APCFG_PACKET_SEND_FAILED Error while sending internal message to another task.
0xC0140040	TLR_E_PNIO_APCFG_INVALID_NAME_OF_STATION_LENGTH The length of parameter NameOfStation is invalid.
0xC0140041	TLR_E_PNIO_APCFG_INVALID_NAME_OF_STATION_CHARACTER The NameOfStation contains an invalid character.
0xC0140042	TLR_E_PNIO_APCFG_INVALID_TYPE_OF_STATION_LENGTH The length of parameter TypeOfStation is invalid.
0xC0140043	TLR_E_PNIO_APCFG_INVALID_TYPE_OF_STATION_CHARACTER The TypeOfStation contains an invalid character.
0xC0140044	TLR_E_PNIO_APCFG_INVALID_SYSTEMFLAGS The parameter SystemFlags is invalid.
0xC0140045	TLR_E_PNIO_APCFG_INVALID_WATCHDOG_TIME The parameter WatchdogTime is invalid.
0xC0140046	TLR_E_PNIO_APCFG_INVALID_VENDORID The Parameter VendorID is invalid.
0xC0140047	TLR_E_PNIO_APCFG_INVALID_DEVICEID The parameter DeviceID is invalid.
0xC0140048	TLR_E_PNIO_APCFG_INVALID_IP_ADDRESS The IP-Address to use is invalid.
0xC0140049	TLR_E_PNIO_APCFG_INVALID_NETMASK The NetworkMask to use is invalid.
0xC014004A	TLR_E_PNIO_APCFG_INVALID_GATEWAY The Gateway-Address to use is invalid or unreachable.
0xC014004B	TLR_E_PNIO_APCFG_INVALID_IP_FLAGS The parameter IPFlags is invalid.

Hexadecimal Value	Definition Description
0xC014004C	TLR_E_PNIO_APCFG_INVALID_NAME_OF_STATION_STARTLABEL The NameOfStation shall not start with - . or port-xyz.
0xC014004D	TLR_E_PNIO_APCFG_NAME_OF_STATION_LABEL_TOO_SHORT One label of NameOfStation is too short.
0xC014004E	TLR_E_PNIO_APCFG_NAME_OF_STATION_LABEL_TOO_LONG One label of NameOfStation is too long.
0xC014004F	TLR_E_PNIO_APCFG_NAME_OF_STATION_TOO_MANY_LABELS The NameOfStation contains too many labels.
0xC0140050	TLR_E_PNIO_APCFG_INVALID_RTARETRIES The parameter RTARetries is invalid.
0xC0140051	TLR_E_PNIO_APCFG_INVALID_RTATIMEOUT The parameter RTATimeout is invalid.
0xC0140052	TLR_E_PNIO_APCFG_INVALID_AR_PROPERTIES The parameter ARProperties is invalid.
0xC0140053	TLR_E_PNIO_APCFG_INVALID_AR_TYPE The parameter ARTYPE is invalid.
0xC0140054	TLR_E_PNIO_APCFG_INVALID_AR_UUID The parameter ARUUID is invalid.
0xC0140055	TLR_E_PNIO_APCFG_INVALID_ALARMCR_PROPERTIES The parameter AlarmCRProperties is invalid.
0xC0140056	TLR_E_PNIO_APCFG_INVALID_ALARMCR_TYPE The Parameter AlarmCRType is invalid.
0xC0140060	TLR_E_PNIO_APCFG_INVALID_MCAST_MAC The parameter MulticastMACAddress is invalid.
0xC0140061	TLR_E_PNIO_APCFG_INVALID_FRAMESEND_OFFSET The parameter FrameSendOffset is invalid.
0xC0140062	TLR_E_PNIO_APCFG_INVALID_IOCR_PROPERTIES The parameter IOCProperties is invalid.
0xC0140063	TLR_E_PNIO_APCFG_INVALID_IOC_DATA_LENGTH The parameter IOCDataLength is invalid.
0xC0140064	TLR_E_PNIO_APCFG_INVALID_IOC_TYPE The parameter IOCType is invalid.
0xC0140065	TLR_E_PNIO_APCFG_INVALID_DATAHOLD_FACTOR The parameter DataHoldFactor is invalid.
0xC0140066	TLR_E_PNIO_APCFG_INVALID_WATCHDOG_FACTOR The Parameter WatchdogFactor is invalid.
0xC0140067	TLR_E_PNIO_APCFG_INVALID_SENDCLOCK_FACTOR The parameter SendClockFactor is invalid.
0xC0140068	TLR_E_PNIO_APCFG_INVALID_REDUCTION_RATIO The parameter ReductionRatio is invalid.
0xC0140069	TLR_E_PNIO_APCFG_INVALID_PHASE The Parameter Phase is invalid.
0xC0140070	TLR_E_PNIO_APCFG_INVALID_IOC_LENGTH The parameter IOCLength is invalid.

Hexadecimal Value	Definition Description
0xC0140071	TLR_E_PNIO_APCFG_INVALID_IOPS_LENGTH The parameter IOPSLength is invalid.
0xC0140072	TLR_E_PNIO_APCFG_INVALID_DPM_OFFSET The parameter DPMOffset is invalid.
0xC0140073	TLR_E_PNIO_APCFG_INVALID_FRAME_OFFSET The parameter FrameOffset is invalid.
0xC0140074	TLR_E_PNIO_APCFG_INVALID_IOCS_FRAME_OFFSET The parameter IOCSFrameOffset is invalid.
0xC0140075	TLR_E_PNIO_APCFG_INVALID_SUBMODULE_DATA_LENGTH The parameter SubmoduleDataLength is invalid.
0xC0140076	TLR_E_PNIO_APCFG_INVALID_DATA_DESCRIPTION The Parameter DataDescription is invalid.
0xC0140077	TLR_E_PNIO_APCFG_OVERLAPPING_FRAMEOFFSET_DATA The frame offset to use for IO-Data is already in use by another submodule.
0xC0140078	TLR_E_PNIO_APCFG_OVERLAPPING_FRAMEOFFSET_IOCS The frame offset to use for the IOCS is already in use by another submodule.
0xC0140079	TLR_E_PNIO_APCFG_OVERLAPPING_DPMOFFSET The DPM offset to use for IO-Data is already in use by another submodule.
0xC014007A	TLR_E_PNIO_APCFG_FRAMEOFFSET_OUTSIDE_DEFINED_IOCR The frame offset is bigger than the IOCR-length.
0xC014007B	TLR_E_PNIO_APCFG_IOCS_FRAMEOFFSET_OUTSIDE_DEFINED_IOCR The IOCS frame offset is bigger than the IOCR-length.
0xC014007C	TLR_E_PNIO_APCFG_INVALID_IOCR_PROD_ID The IOCRIdProd is invalid.
0xC014007D	TLR_E_PNIO_APCFG_INVALID_IOCR_CONS_ID The IOCRIcons is invalid.
0xC014007E	TLR_E_PNIO_APCFG_INVALID_RECORD_LENGTH The record data length is invalid.
0xC014007F	TLR_E_PNIO_APCFG_DUPLICATE_ARUUID The ARUUID of this IO-Device is already in use by another IO-Device.
0xC0140080	TLR_E_PNIO_APCFG_MULTIPLE_CR_NOT_SUPPORTED The configuration packet contains more than 1 IOCR for the same direction which is not supported.
0xC0140081	TLR_E_PNIO_APCFG_FAULTY_VERSION_TABLE The content of the version table in database is invalid.
0xC0140082	TLR_E_PNIO_APCFG_UNSUPPORTED_DATABASE_VERSION The version of database is unsupported.
0xC0140083	TLR_E_PNIO_APCFG_INVALID_MAUTYPE The MAUType is invalid.
0xC0140084	TLR_E_PNIO_APCFG_INVALID_SIGNAL_TYPE The signal type is invalid.
0xC0140085	TLR_E_PNIO_APCFG_INVALID_NO_INTF_SUBM The requested submodule is no Interface Submodule.
0xC0140086	TLR_E_PNIO_APCFG_INVALID_NO_PORT_SUBM The requested submodule is no Port Submodule.



Hexadecimal Value	Definition Description
0xC0140087	TLR_E_PNIO_APCFG_INVALID_STRUCTURE_VERSION The value of structure version is invalid.
0xC0140088	TLR_E_PNIO_APCFG_INVALID_HELLO_MODE The Hello Mode is invalid.
0xC0140089	TLR_E_PNIO_APCFG_INVALID_HELLO_RETRY The value of Hello Retry is invalid.
0xC014008A	TLR_E_PNIO_APCFG_INVALID_HELLO_INTERVAL The value of Hello Interval is invalid.
0xC014008B	TLR_E_PNIO_APCFG_INVALID_HELLO_DELAY The value of Hello Delay is invalid.
0xC014008C	TLR_E_PNIO_APCFG_INVALID_MRP_ROLE The value of MRP Role is invalid.
0xC014008D	TLR_E_PNIO_APCFG_INVALID_ORDERID_LENGTH The OrderId length is invalid.
0xC014008E	TLR_E_PNIO_APCFG_INVALID_IOPS_OFFSET The requested IOPS offset does not address a byte although byte mode is requested or is outside valid DPM area.
0xC014008F	TLR_E_PNIO_APCFG_OVERLAPPING_IOPS_OFFSET The requested IOPS offset is already in use by another element (data or IOxS).
0xC0140090	TLR_E_PNIO_APCFG_INVALID_IOPS_MODE The requested IOPS mode is invalid or not supported.
0xC0140091	TLR_E_PNIO_APCFG_INVALID_IOCS_MODE The requested IOCS mode is invalid or not supported.
0xC0140092	TLR_E_PNIO_APCFG_INVALID_IOPS_STARTOFFSET The requested IOPS area offset is invalid.
0xC0140093	TLR_E_PNIO_APCFG_INVALID_IOCS_STARTOFFSET The requested IOCS area offset is invalid.
0xC0140094	TLR_E_PNIO_APCFG_NO_VALID_CONFIG_DATABASE No valid config database was found.
0xC0140095	TLR_E_PNIO_APCFG_NO_VALID_NWID_DATABASE No valid nwid database was found.
0xC0140096	TLR_E_PNIO_APCFG_INVALID_ACTIVE_BITLIST_OFFSET The requested bit list offset for active IO-Devices is invalid.
0xC0140097	TLR_E_PNIO_APCFG_INVALID_CONFIG_BITLIST_OFFSET The requested bit list offset for configured IO-Devices is invalid.
0xC0140098	TLR_E_PNIO_APCFG_INVALID_FAULTY_BITLIST_OFFSET The requested bit list offset for faulty IO-Devices is invalid.
0xC0140099	TLR_E_PNIO_APCFG_INVALID_BITLIST_MODE Either all or no bitlist offsets can be set to automatic mode.
0xC014009A	TLR_E_PNIO_APCFG_INVALID_SUBM_DESCR At least one IOxS interface is active which forces usage of extended submodule description packet.
0xC014009B	TLR_E_PNIO_APCFG_VERSION_INFO_ALREADY_SET The version info has already been set. It can only be set exactly once after power cycle.

Hexadecimal Value	Definition Description
0xC014009C	TLR_E_PNIO_APCFG_INVALID_SW_PREFIX The value of parameter bVersionPrefix is invalid.
0xC014009D	TLR_E_PNIO_APCFG_INVALID_VERSION_STRING_LENGTH The version string conducted of the different version parameters is to long.

*Table 140: Error Messages of the APCFG-Task*

### 6.2.1 Diagnostic Codes of the APCFG -Task

Currently there are no diagnostic messages defined for the APCFG -task of the Profinet IO-Controller firmware for netX-based products.

## 6.3 Error Codes of the APCTL –Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC00C0003	TLR_E_PNIO_APCTL_HIF_IDENT Error identifying HIF.
0xC00C0004	TLR_E_PNIO_APCTL_AREA_IDENT Unable to identify requested DPM Channel.
0xC00C0005	TLR_E_PNIO_APCTL_GETAREA_IDENT Unable to identify DPM section "STD INPUT".
0xC00C0006	TLR_E_PNIO_APCTL_SETAREA_IDENT Unable to identify DPM section "STD OUTPUT".
0xC00C0007	TLR_E_PNIO_APCTL_SETAREA_INFO_GET Unable to get configuration for DPM section "STD OUTPUT".
0xC00C0008	TLR_E_PNIO_APCTL_TOHOST_MBX_NAME_GET Unable to identify DPM section "TOHOST MBX".
0xC00C0009	TLR_E_PNIO_APCTL_TOHOST_MBX_INFO_GET Unable to get configuration for DPM section "TOHOST MBX".
0xC00C000A	TLR_E_PNIO_APCTL_MIDSYS_QUE_IDENT Unable to identify queue to MidSys.
0xC00C000B	TLR_E_PNIO_APCTL_ENABLE_BUSON_CBK Enabling callback function for ApplicationCOS.BUS_ON bit did not succeed.
0xC00C000C	TLR_E_PNIO_APCTL_ENABLE_APPREADY_CBK Enabling callback function for ApplicationCOS.APP_READY bit did not succeed.
0xC00C000D	TLR_E_PNIO_APCTL_COMMONAREA_IDENT Unable to identify DPM section "COMMON STATUS".
0xC00C000E	TLR_E_PNIO_APCTL_STATUSAREA_IDENT Unable to identify DPM section "EXTENDED STATUS".
0xC00C000F	TLR_E_PNIO_APCTL_CONTROLAREA_IDENT Unable to identify DPM section "CONTROL".
0xC00C0010	TLR_E_PNIO_APCTL_GETAREA_INT_ENBL Unable to enable DPM section "STD INPUT".
0xC00C0011	TLR_E_PNIO_APCTL_SETAREA_INT_ENBL Unable to enable DPM section "STD OUTPUT".
0xC00C0012	TLR_E_PNIO_APCTL_ENABLE_INIT_CBK Enabling callback function for ApplicationCOS.INITIALIZATION bit did not succeed.
0xC00C0013	TLR_E_PNIO_APCTL_VOL_IDENT Unable to identify SysVolume.
0xC00C0014	TLR_E_PNIO_APCTL_VOL_MOUNT Unable to mount SysVolume
0xC00C0015	TLR_E_PNIO_APCTL_FAT_FRMT Unable to format SysVolume.
0xC00C0016	TLR_E_PNIO_APCTL_QUE_CREATE The Queue for APCTL task could not be created.

Hexadecimal Value	Definition Description
0xC00C0017	TLR_E_PNIO_APCTL_CMCTL_QUE_IDENT The Queue for CMCTL task could not be identified.
0xC00C0018	TLR_E_PNIO_APCTL_MGT_QUE_IDENT The Queue for MGT task could not be identified.
0xC00C0019	TLR_E_PNIO_APCTL_ACP_QUE_IDENT The Queue for ACP task could not be identified.
0xC00C001A	TLR_E_PNIO_APCTL_GETAREA_INFO_GET Unable to get configuration for DPM section "STD INPUT".
0xC00C001B	TLR_E_PNIO_APCTL_CLR_PCK_GET Unable to get a free packet from packet pool to unregister DPM channel from rcX.
0xC00C001C	TLR_E_PNIO_APCTL_SET_PCK_GET Unable to get a free packet from packet pool to register DPM channel from rcX.
0xC00C001D	TLR_E_PNIO_APCTL_IDENT_QUE_MIDSYS The Queue of rcX MidSys task could not be identified.
0xC00C001E	TLR_E_PNIO_APCTL_CREATE_TIMER_SET The timer for firmware-controlled handling of IO-data in DPM could not be created.
0xC00C0020	TLR_E_PNIO_APCTL_WDG_INIT The Watchdog timer could not be initialized.
0xC00C0021	TLR_E_INFO_FIELD_ONE_CREATE_FAILED The field for extended APCTL task status information could not be created.
0xC00C0022	TLR_E_INFO_FIELD_TWO_CREATE_FAILED An undefined field could not be created (unused).
0xC00C0023	TLR_E_PNIO_APCTL_POOL_CREATE The packet pool for APCTL task could not be created.
0xC00C0024	TLR_E_PNIO_APCTL_IOCR_LIST The number of the input IOCRLs is not equal to the number of the output IOCRLs.
0xC00C0025	TLR_E_PNIO_APCTL_DPM The requested function is not supported because DPM is not initialized.
0xC00C0026	TLR_E_PNIO_APCTL_ENABLE_LOCKCONFIG_CBK Enabling callback function for ApplicationCOS.LOCK_CONFIGURATION bit did not succeed.
0xC00C0027	TLR_E_PNIO_APCTL_LED_CREATE The LEDs could not be initialized.
0xC00C0028	TLR_E_PNIO_APCTL_ENABLE_DMA_CBK Enabling callback function to enable DMA mode did not succeed.
0xC00C0030	TLR_E_PNIO_APCTL_RPC_REQUEST_LIMIT_REACHED Too many outstanding RPC-requests for this IO-Device.
0xC00C0032	TLR_E_PNIO_APCTL_INVALID_CMCTL_HANDLE The handle used for IO-Device is wrong.
0xC00C0033	TLR_E_PNIO_APCTL_INVALID_NAME_OF_STATION_LENGTH The name to be set has incorrect length.
0xC00C0034	TLR_E_PNIO_APCTL_DCP_REQUEST_LIMIT_REACHED Too many outstanding DCP-requests for this IO-Device.
0xC00C0035	TLR_E_PNIO_APCTL_OTHER_CONFIG_PACKET_IN_PROCESS An other configuration packet is in process wait for its confirmation packet.

Hexadecimal Value	Definition Description
0xC00C0036	TLR_E_PNIO_APCTL_APCFG_QUE_IDENT Identification of APCFG queue in remote resources failed.
0xC00C0037	TLR_E_PNIO_APCTL_UNKNOWN_ALARM_SPECIFIER The Alarmspecifier is unknown to IO-Controller.
0xC00C0038	TLR_E_PNIO_APCTL_DCP_REQUEST_NO_ANSWER The requested IO-Device did not answer to the DCP-requests.
0xC00C0040	TLR_E_PNIO_APCTL_APPLICATION_ALREADY_REGISTERED There is already an Application registered to APCTL-Task.
0xC00C0041	TLR_E_PNIO_APCTL_NO_APPLICATION_REGISTERED There is no Application registered to APCTL-Task.
0xC00C0042	TLR_E_PNIO_APCTL_UNREGISTER_APPLICATION_IMPOSSIBLE It is impossible to unregister the Application in APCTL-Task. Either there is no Application registered or the Unregister Request was not send by the registered Application.
0xC00C0050	TLR_E_PNIO_APCTL_CHANNEL_INIT_REQUESTED The request is rejected because a Channel Init was requested.
0xC00C0052	TLR_E_PNIO_APCTL_CYCLIC_WATCHDOG_ERROR The connection to an IO-Device was closed because too many cyclic frames were missing.
0xC00C0053	TLR_E_PNIO_APCTL_CONNECTION_CLOSED_BY_IODEVICE The connection was shutdown by an IO-Device.
0xC00C0054	TLR_E_PNIO_APCTL_WATCHDOG_TIME_TOO_SMALL The watchdog time specified is too small.
0xC00C0055	TLR_E_PNIO_APCTL_WATCHDOG_TIME_TOO_BIG The watchdog time specified is too big.
0xC00C0056	TLR_E_PNIO_APCTL_IO_DEVICE_NOT_IN_DATAEXCHANGE The service is unavailable because the IO-Device is not in cyclic data exchange.
0xC00C0060	TLR_E_PNIO_APCTL_ACYCLIC_REQ_FAILED_REMOTE The acyclic service failed. The IO-Device answered with an error code which is contained in confirmation packet.
0xC00C0061	TLR_E_PNIO_APCTL_ACYCLIC_REQ_FAILED_RPC The acyclic service failed. The RPC-layer detected an error which is contained in confirmation packet.
0xC00C0062	TLR_E_PNIO_APCTL_ACYCLIC_REQ_FAILED_INTERNAL The acyclic service failed. An internal error occurred.
0xC00C0063	TLR_E_PNIO_APCTL_TOO_MUCH_IO_DATA_CONFIGURED The maximum supported IO-data size is exceeded.
0xC00C0064	TLR_E_PNIO_APCTL_INVALID_IP_ADDRESS The IP address is invalid.
0xC00C0065	TLR_E_PNIO_APCTL_INVALID_NETMASK The Netmask is invalid.
0xC00C0066	TLR_E_PNIO_APCTL_INVALID_GATEWAY The gateway address is invalid.
0xC00C0067	TLR_E_PNIO_APCTL_TOO_MUCH_DATA_REQUESTED The maximum supported data size for this service is exceeded.

Hexadecimal Value	Definition Description
0xC00C0068	TLR_E_PNIO_APCTL_CHANNEL_INIT_RUNNING The request cannot be handled because a ChannelInit is currently running.
0xC00C0069	TLR_E_PNIO_APCTL_SEND_CMCTL_CHANNEL_INIT_FAILED Internal sending of Channel Init to CMCTL failed.
0xC00C006A	TLR_E_PNIO_APCTL_SEND_APCFG_CHANNEL_INIT_FAILED Internal sending of Channel Init to APCFG failed.
0xC00C006B	TLR_E_PNIO_APCTL_EMPTY_CONFIGURATION A configuration without IO-Devices has been detected.
0xC0000201	TLR_E_PNIO_APCTL_APPLICATION_ALREADY_REGISTERED There is already an Application registered to APCTL-Task.
0xC0000202	TLR_E_NO_APPLICATION_REGISTERED It is impossible to unregister the Application in APCTL-Task. Either there is no Application registered or the Deregister Request was not send by the registered Application.

Table 141: Error Messages of the APCTL -Task

### 6.3.1 Diagnostic Codes of the APCTL-Task

Currently there are no diagnostic messages defined for the APCTL -task of the Profinet IO-Controller firmware for netX-based products.

## 6.4 Error Codes of the CMCTL–Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
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.
0xC00A0013	TLR_E_PNIO_CMCTL_CLOSED This CMCTL protocol machine was closed.
0xC00A0014	TLR_E_PNIO_CMCTL_STATE_CONFLICT This request cannot 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.
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.
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.

Hexadecimal Value	Definition Description
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 cannot 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 cannot 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.
0xC00A0074	TLR_E_PNIO_CMCTL_AR_PROPERTY The configuration block for AR in CMCTL_RMConnect_req_LoadAr() has an invalid network properties value.



Hexadecimal Value	Definition Description
0xC00A0075	TLR_E_PNIO_CMCTL_AR_REF_UNEXPECTED The AR-Reference for CMCTL protocol-machine is invalid.
(0xC00A007)	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 IOCR in CMCTL_RMConnect_req_LoadIocr() has an invalid length.
0xC00A0091	TLR_E_PNIO_CMCTL_IOCRR_TYPE_UNSUPPORTED The type of IOCR is unsupported.
0xC00A0092	TLR_E_PNIO_CMCTL_IOCRR_TYPE_UNKNOWN The type of IOCR 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 IOCR in CMCTL_RMConnect_req_LoadIocr() has an unsupported interface-type.
0xC00A0096	TLR_E_PNIO_CMCTL_IOCRR_SCSYNC_UNSUPPORTED The expected configuration block for IOCR in CMCTL_RMConnect_req_LoadIocr() has an unsupported value for SendClock.
0xC00A0097	TLR_E_PNIO_CMCTL_IOCRR_ADDRESS_UNSUPPORTED The expected configuration block for IOCR in CMCTL_RMConnect_req_LoadIocr() 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_LoadIocrr() 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_LoadIocrr() does not contain any IO-Data.
0xC00A009B	TLR_E_PNIO_CMCTL_IOCRR_OBJECT_IOS The expected configuration block for IOCRR in CMCTL_RMConnect_req_LoadIocrr() does not contain any IO-Status.
0xC00A009C	TLR_E_PNIO_CMCTL_IOCRR_API The expected configuration block for IOCRR in CMCTL_RMConnect_req_LoadIocrr() 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.
0xC00A00AA	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.
0xC00A00AB	TLR_E_PNIO_CMCTL_ACYCLIC_REQ_FAILED_RPC The acyclic service failed. The RPC-layer detected an error which is contained in confirmation packet.
0xC00A00AC	TLR_E_PNIO_CMCTL_ACYCLIC_REQ_FAILED_INTERNAL The acyclic service failed. An internal error occurred.

Table 142: Error Messages of the CMCTL-Task

### 6.4.1 Diagnostic Codes of the CMCTL-Task

Hexadecimal Value	Definition Description
0x0000	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 143: Diagnostic Messages of the CMCTL-Task

## 6.5 Error Codes of the CMDEV-Task

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.
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.

Hexadecimal Value	Definition Description
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 can not be established.
0xC00B0053	TLR_E_PNIO_CMDEV_TOO_MUCH_ALARM_DATA The packet contains too much alarm data.
0xC00B0060	TLR_E_PNIO_CMDEV_RPC_CONNECT_FAILED The Connect-Response of IO-Device contained an error code; a connection could not be established.
0xC00B0061	TLR_E_PNIO_CMDEV_RPC_WRITE_PARAM_FAILED The Write_Param-Response of IO-Device contained an error code; a connection could not be established.
0xC00B0062	TLR_E_PNIO_CMDEV_RPC_WRITE_FAILED The Write-Response of IO-Device contained an error code.
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.

Hexadecimal Value	Definition Description
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.
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.

Table 144: Error Messages of the CMDEV-Task

### 6.5.1 Diagnostic Codes of the CMDEV-Task

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_FAILED Failed to get handle to ACP message-queue in CMDEV.
0xC00BF005	TLR_DIAG_E_CMDEV_TASK_IDENT_MGT_QUE_FAILED Failed to get handle to MGT message-queue in CMDEV.
0xC00BF006	TLR_DIAG_E_CMDEV_TASK_IDENT_RPC_QUE_FAILED Failed to get handle to RPC message-queue in CMDEV.
0xC00BF007	TLR_DIAG_E_CMDEV_TASK_IDENT_TCP_QUE_FAILED Failed to get handle to TCP/IP message-queue in CMDEV.
0xC00BF008	TLR_DIAG_E_CMDEV_TASK_IDENT_DCP_QUE_FAILED Failed to get handle to DCP message-queue in CMDEV.
0xC00BF009	TLR_DIAG_E_CMDEV_TASK_IDENT_PNSIF_QUE_FAILED Failed to get handle to PNSIF message-queue in CMDEV.

Table 145: Diagnostic Messages of the CMDEV-Task

## 6.6 Error Codes of the DCP –Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC012000A	TLR_E_PNIO_DCPMCR_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().
0xC0120012	TLR_E_PNIO_DCPMCR_RESOURCE_LIMIT_EXCEEDED The index of DCPMCR's protocol machine is invalid.
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.
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.
0xC0120100	TLR_E_PNIO_DCPMCS_INIT_PARAM_INVALID Invalid parameter (uiMaxMcs) in DCPMCS_ResourceInit().
0xC0120102	TLR_E_PNIO_DCPMCS_RESOURCE_LIMIT_EXCEEDED There are too many outstanding DCPMCS requests. New requests will not be accepted.
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.
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().
0xC0120202	TLR_E_PNIO_DCPUCR_RESOURCE_LIMIT_EXCEEDED The index of DCPUCR's protocol machine is invalid.
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.



Hexadecimal Value	Definition Description
0xC0120206)	TLR_E_PNIO_DCPUCR_TIMER_CREATE_FAILED DCPUCR_Activate_req() was unable to create a TLR-timer.
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().
0xC0120302	TLR_E_PNIO_DCPUCS_RESOURCE_LIMIT_EXCEEDED There are too many outstanding DCPUCS requests. New requests will not be accepted.
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.
0xC012030A	TLR_E_PNIO_DCPUCS_FRAME_OUT_OF_MEMORY DCPUCS was not able to get an Edd_FrameBuffer for sending a packet.
0xC012030B	TLR_E_PNIO_DCPUCS_FRAME_SEND_FAILED An error occurred while DCPUCS was trying to send an Edd_Frame.
0xC012030C	TLR_E_PNIO_DCPUCS_FRAME_TIMEOUT DCPUCS did not get a response to an Edd_Frame send .
0xC0120320	TLR_E_PNIO_DCPUCS_DCP_OPTION_UNSUPPORTED The DCP option to set is not supported by IO-Device.
0xC0120321	TLR_E_PNIO_DCPUCS_DCP_SUBOPTION_UNSUPPORTED The DCP suboption to set is not supported by IO-Device.
0xC0120022	TLR_E_PNIO_DCPUCS_DCP_SUBOPTION_NOT_SET The DCP suboption to set was not set inside IO-Device.
0xC0120023	TLR_E_PNIO_DCPUCS_DCP_RESOURCE_ERROR An internal resource error occurred in IO-Device while performing a DCP request.
0xC0120024	TLR_E_PNIO_DCPUCS_DCP_SET_IMPOSSIBLE_LOCAL_REASON The DCP (sub)option could not be set inside IO-Device for IO-Device internal reasons.
0xC0120025	TLR_E_PNIO_DCPUCS_DCP_SET_IMPOSSIBLE_WHILE_OPERATION The DCP (sub)option could not be set inside IO-Device because IO-Device is in operation.

Table 146: Error Messages of the DCP-Task

## 6.6.1 Diagnostic Codes of the DCP –Task

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 147: Diagnostic Messages of the DCP-Task

## 6.7 Error Codes of the EDD–Task

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 148: Error Messages of the EDD-Task

### 6.7.1 Diagnostic Codes of the EDD–Task

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 149: Diagnostic Messages of the EDD-Task

## 6.8 Error Codes of the IO Signal Task

Hexadecimal Value	Definition Description
0xC0910001	TLR_E_IO_SIGNAL_COMMAND_INVALID Invalid command received.
0xC0910002	TLR_E_IO_SIGNAL_INVALID_SIGNAL_DIRECTION The value of signal direction is invalid.
0xC0910003	TLR_E_IO_SIGNAL_INVALID_SIGNAL_AMOUNT The value of signal amount is invalid.
0xC0910004	TLR_E_IO_SIGNAL_INVALID_SIGNAL_TYPE The value of signal type is invalid.
0xC0910005	TLR_E_IO_SIGNAL_UNSUPPORTED_SIGNAL_TYPE The value of signal type is unsupported.

Table 150: Error Messages of the IO Signal –Task

## 6.9 Error Codes of the LLDP–Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC03E0001	TLR_E_LLDP_COMMAND_INVALID Invalid command.
0x403E0002	TLR_I_LLDP_UNKNOWN_TLV Unknwon TLV found.
0xC03E0003	TLR_E_LLDP_PDU_MAX_SIZE_EXCEEDED Maximum Ethernet frame size exceeded.
0xC03E0004	TLR_E_LLDP_TLV_DISCARDED Invalid TLV content.
0xC03E0005	TLR_E_LLDP_FRAME_DISCARDED One of TLVs has a wrong size or invalid mandatory TLV sequence.
0xC03E0006	TLR_E_LLDP_WRONG_PARAMETERS Parameters sent to the task are wrong.
0xC03E0007	TLR_E_NO_MIBS The Task was unable to recreate MIBs during a reset due to insufficient memory.
0xC03E0008	TLR_E_CONFIG_WRONG_PORT_TOO_HIGH Configured Port higher then configured at startup parameter.
0xC03E0009	TLR_E_CONFIG_WRONG_STATUS_RT_2 Configured a too high RT2 Status
0xC03E000A	TLR_E_CONFIG_WRONG_STATUS_RT_3 Configured a too high RT3 Status
0xC03E000B	TLR_E_CONFIG_WRONG_MODE_RT3 Configured a too high RT3 Mode
0xC03E000C	TLR_E_CONFIG_SEND_ENABLE_RANGE Configured a wrong value for TX send enable

Hexadecimal Value	Definition Description
0xC03E000D	TLR_E_CONFIG_WRONG_ADMIN_STATUS Configured a wrong value for admin state
0xC03E000E	TLR_E_CONFIG_WRONG_PARAM_NOTIFIC_ENABLE Configured a wrong value for notification enable
0xC03E000F	TLR_E_CONFIG_WRONG_TX_ENABLE_VALUE Configured a toohigh value for TX enable
0xC03E0010	TLR_E_CONFIG_PORT_ID_SIZE_MAX_FAULT Configured a toolong port id size
0xC03E0011	TLR_E_CONFIG_PORT_ID_SIZE_MIN_FAULT Configured a too short port id size
0xC03E0012	TLR_E_CONFIG_WRONG_PORT_ID_TYPE_MAX Configured a wrong to large port id type
0xC03E0013	TLR_E_CONFIG_WRONG_PORT_ID_TYPE_MIN Configured a wrong to short port id type
0xC03E0014	TLR_E_CONFIG_TO_LONG_PORT_DESCR Configured a too long port description
0xC03E0015	TLR_E_CONFIG_MAX_CHASSIS_ID Configured a too long chassis id type
0xC03E0016	TLR_E_CONFIG_MIN_CHASSIS_ID Configured a too short chassis id type
0xC03E0017	TLR_E_CONFIG_MAX_TX_INTERVAL Configured a too long TX interval
0xC03E0018	TLR_E_CONFIG_MIN_TX_INTERVAL Configured a too short TX interval
0xC03E0019	TLR_E_CONFIG_MAX_PPVID Configured a too large PPVID
0xC03E001A	TLR_E_CONFIG_MIN_PPVID Configured a too short PPVID
0xC03E001B	TLR_E_CONFIG_WRONG_LLDP_SEND_STATE Configured wrong LLDP send state
0xC03E001C	TLR_E_CONFIG_TOO_LARGE_DESCRIPTION Configured a too large system description
0xC03E001D	TLR_E_CONFIG_WRONG_AUTO_NEG_STATE Configured a wrong Autonegation state
0xC03E001E	TLR_E_CONFIG_WRONG_AUTO_NEG_SUPPORTED Configured a wrong Autonegation supported state
0xC03E001F	TLR_E_CONFIG_WRONG_AUTO_NEG_ADVERTISED Configured a wrong Autonegation advertised state
0xC03E0020	TLR_E_CONFIG_MAX_MAU_TYPE Configured a too high MAU type
0xC03E0021	TLR_E_CONFIG_MIN_MAU_TYPE Configured a too low MAU type
0xC03E0022	TLR_E_CONFIG_PPVID_FAILED PPVID was previousley not defined

Hexadecimal Value	Definition Description
0xC03E0023	TLR_E_IF_TYPE_MAX IF TYPE too large
0xC03E0024	TLR_E_IF_TYPE_MIN IF TYPE too small
0xC03E0025	TLR_E_OID_MAX OID too long
0xC03E0026	TLR_E_MANAGEMENT_ADDRESS_MAX Management address size too high
0xC03E0027	TLR_E_MANAGEMENT_ADDRESS_MIN Management address size too low
0xC03E0028	TLR_E_MANAGEMENT_ADDRESS_ID_INVALID Management address id not found. Address shall not exist.
0xC03E0029	TLR_E_TX_HOLD_MAX Value for TX hold too high
0xC03E003A	TLR_E_TX_HOLD_MIN Value for TX hold too low
0xC03E003B	TLR_E_REINIT_DELAY_MAX Value for the reinit delay is too high
0xC03E003C	TLR_E_REINIT_DELAY_MIN
0xC03E003D	TLR_E_TX_DELAY_MAX Value for the tx delay is too high
0xC03E003E	TLR_E_TX_DELAY_MIN Value for the tx delay is too low
0xC03E003F	TLR_E_NOTIFICATION_INTERVALL_MAX Value for the notification intervall too high
0xC03E0040	TLR_E_NOTIFICATION_INTERVALL_MIN Value for the notification intervall too low
0xC03E0041	TLR_E_PVID_MAX Value for the pvid too high
0xC03E0042	TLR_E_PVID_MIN Value for the pvid too low
0xC03E0043	TLR_E_PPVID_MAX Value for the ppvid too high
0xC03E0044	TLR_E_PPVID_ENABLED Value for ppvid enabled out of range
0xC03E0045	TLR_E_PPVID_SUPPORTED Value for ppvid supported out of range
0xC03E0046	TLR_E_WRONG_PARAMETER_COMBINATION The combination of the different configured values doesn't fit
0xC03E0047	TLR_E_NOT_ENOUGH_STORAGE_TLV Not enough storage for so much of these TLV's
0xC03E0048	TLR_E_VLAN_NAME_MAX Vlan name length too long

Hexadecimal Value	Definition Description
0xC03E0049	TLR_E_VLAN_NAME_MIN Vlan name length too short
0xC03E004A	TLR_E_VLAN_ID_MAX Vlan id length too long
0xC03E004B	TLR_E_VLAN_ID_MIN Vlan id length too short
0xC03E004C	TLR_E_PID_MAX Pid length too long
0xC03E004D	TLR_E_MDI_ENABLE_RANGE MDI Enable out of range
0xC03E004E	TLR_E_MDI_SUPPORTED_RANGE MDI Supported out of range
0xC03E004F	TLR_E_PAIR_CONTROLLABEL_RANGE Pair controllable out of range
0xC03E0050	TLR_E_PORT_CLASS_RANGE Port class out of range
0xC03E0051	TLR_E_CLASS_MAX Value for class too high
0xC03E0052	TLR_E_CLASS_MIN Value for class too low
0xC03E0053	TLR_E_PAIRS_MAX Value for pairs too high
0xC03E0054	TLR_E_PAIRS_MIN Value for pairs too low
0xC03E0055	TLR_E_LA_SUPPORTED_RANGE Value for LA Supported out of range
0xC03E0056	TLR_E_LA_ENABLED_RANGE Value for LA Enabled out of range
0xC03E0057	TLR_E_AGG_PORT_ID_MAX Value for Agg Port ID too high
0xC03E0058	TLR_E_MFS_MAX Value for Mfs too high
0xC03E0059	TLR_E_MFS_MIN Value for Mfs too low
0xC03E005A	TLR_E_NOS_MAX Name of station too long
0xC03E005B	TLR_E_MRRT_STATE_RANGE Value of MRRT state out of range
0xC03E005C	TLR_E_LENGTH_PERIOD_MAX Value of Length of period too high
0xC03E005D	TLR_E_RED_PERIOD_BEGIN_MAX Value of Length of red period begin is too high
0xC03E005E	TLR_E_ORANGE_PERIOD_BEGIN_MAX Value of Length of orange period begin is too high

Hexadecimal Value	Definition Description
0xC03E005F	TLR_E_GREEN_PERIOD_BEGIN_MAX Value of Length of green period begin is too high
0xC03E0060	TLR_E_LENGTH_PERIOD_MIN Value of length of period is too low

*Table 151: Error Messages of the LLDP-Task*

## 6.10 Error Codes of the MGT-Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC0130001	TLR_E_PNIO_MGT_PACKET_SEND_FAILED Invalid parameter (uiMaxNrpm) in NRPM_ResourceInit().
0xC0130010	TLR_E_PNIO_NRPM_PARAM_INVALID_INIT The handle to NRPM protocol machine is invalid.
0xC0130011	TLR_E_PNIO_NRPM_HANDLE_INVALID The state of NRPM protocol machine is invalid.
0xC0130012	TLR_E_PNIO_NRPM_STATE_INVALID The identify-flag in NRPM_Init_req() is invalid.
0xC0130013	TLR_E_PNIO_NRPM_IDENTIFY_FLAG_INVALID The requested number of NRPM protocol machines exceeds the highest possible number in NRPM_Init_req().
0xC0130014	TLR_E_PNIO_NRPM_RESOURCE_LIMIT_EXCEEDED Insufficient memory in NRPM_Init_req().
0xC0130015	TLR_E_PNIO_NRPM_RESOURCE_OUT_OF_MEMORY Error while sending a packet to another task in NRPM.
0xC0130016	TLR_E_PNIO_NRPM_PACKET_SEND_FAILED Insufficient memory to allocate a packet in NRPM.
0xC0130017	TLR_E_PNIO_NRPM_PACKET_OUT_OF_MEMORY Received request with invalid type of DCP request in NRPM.
0xC0130018	TLR_E_PNIO_NRPM_DCP_TYPE_INVALID The requested NameOfStation is invalid. Either it has an invalid length or it contains invalid characters.
0xC0130019	TLR_E_PNIO_NRPM_NAME_OF_STATION_INVALID The requested DCP Set operation failed.
0xC013001A	TLR_E_PNIO_NRPM_DCP_SET_ERROR The IP-address the controller shall set for the IO-Device is already in use by another network device.
0xC013001B	TLR_E_PNIO_NRPM_DEVICE_IP_ADDRESS_ALREADY_IN_USE The handle to RMPM is invalid.
0xC013001C	TLR_E_PNIO_NRPM_LATE_ERROR_INCONSISTENT_IP_PARAMETERS The IP parameters of HelloReq and IdentifyRsp are inconsistent.
0xC013001D	TLR_E_PNIO_NRPM_LATE_ERROR_IP_LOOKUP_MAC_CONFLICT While checking the IP an invalid MAC address was found.
0xC013001E	TLR_E_PNIO_NRPM_LATE_ERROR_IP_LOOKUP_STATION_NOT_FOUND Late error detected while checking the IP. No station was found.
0xC013001F	TLR_E_PNIO_NRPM_LATE_ERROR_IP_LOOKUP_MULTIPLE_STATIONS_FOUND Late error detected while checking the IP. Multiple stations were found.
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.



Hexadecimal Value	Definition Description
0xC0130101	TLR_E_PNIO_RMPM_HANDLE_INVALID The state of RMPM is invalid for current request.
0xC0130102	TLR_E_PNIO_RMPM_STATE_INVALID The state of RMPM is closed
0xC0130103	TLR_E_PNIO_RMPM_STATE_CLOSING The number of RMPM state-machines is to high.
0xC0130104	TLR_E_PNIO_RMPM_RESOURCE_LIMIT_EXCEEDED
0xC0130105	TLR_E_PNIO_RMPM_RESOURCE_OUT_OF_MEMORY Insufficient memory to fulfil the current request in RMPM.
0xC0130106	TLR_E_PNIO_RMPM_PACKET_SEND_FAILED Error while sending a packet to another task in RMPM.
0xC0130107	TLR_E_PNIO_RMPM_PACKET_OUT_OF_MEMORY Insufficient memory to allocate a packet in RMPM.
0xC0130108	TLR_E_PNIO_RMPM_ROLE_UNSUPPORTED The parameter "role" is unsupported in RMPM_Init_req_ParameterRole() .
0xC0130109	TLR_E_PNIO_RMPM_ROLE_UNKNOWN The parameter "role" is unknown in RMPM_Init_req_ParameterRole() .
0xC013010A	TLR_E_PNIO_RMPM_ROLE_IN_USE The parameter "role" is already in use in RMPM_Init_req_ParameterRole() .
0xC013010B	TLR_E_PNIO_RMPM_CONFIG_SEQUENCE Incorrect sequence of configuration in RMPM_ConfigSet_req().
0xC013010C	TLR_E_PNIO_RMPM_CONFIG_INVALID_VENDOR_ID Incorrect configuration of Vendor-ID in RMPM_ConfigSet_req().
0xC013010D	TLR_E_PNIO_RMPM_CONFIG_INVALID_NAME Incorrect name of station in RMPM_ConfigSet_req().
0xC013010E	TLR_E_PNIO_RMPM_CONFIG_INVALID_TYPE Incorrect name of type in RMPM_ConfigSet_req().
0xC0130110	TLR_E_PNIO_RMPM_DUPLICATE_NAME_OF_STATION The NameOfStation of IO-Controller is in use by another network device.
0xC0130111	TLR_E_PNIO_RMPM_DUPLICATE_IP The IP-address the IO-Controller shall use is in use by another network device.
0xC0130112	TLR_E_PNIO_RMPM_RPC_PACKET_INVALID The packet length of an RPC-packet received is invalid (most likely too short
0xC0130113	TLR_E_PNIO_RMPM_DCP_PACKET_INVALID The packet length of a 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

Hexadecimal Value	Definition Description
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 NRMC Identity flag invalid
0xC0130204	TLR_E_PNIO_NRMC_RESOURCE_LIMIT_EXCEEDED The number of NRMC state-machines is to high.
0xC0130205	TLR_E_PNIO_NRMC_RESOURCE_OUT_OF_MEMORY Insufficient memory to fullfill 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 NRMC DCP Type invalid

Table 152: Error Messages of the MGT-Task

### 6.10.1 Diagnostic Codes of the MGT –Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC013F010	TLR_DIAG_E_MGT_TASK_RMPM_RESOURCE_INIT_FAILED Failed to initialize RMPM.
0xC013F011	TLR_DIAG_E_MGT_TASK_NRPM_RESOURCE_INIT_FAILED Failed to initialize NRPM.
0xC013F012	TLR_DIAG_E_MGT_TASK_CREATE_QUE_FAILED Failed to create message-queue for MGT task.
0xC013F013	TLR_DIAG_E_MGT_TASK_IDENT_TCPUDP_QUE_FAILED Failed to get handle to TCP/IP task in MGT task.
0xC013F014	TLR_DIAG_E_MGT_TASK_IDENT_DCP_QUE_FAILED Failed to get handle to DCP task in MGT task.
0xC013F015	TLR_DIAG_E_MGT_TASK_IDENT_EDD_FAILED Failed to identify Drv_EDD im MGT task.
0xC011F016	TLR_DIAG_E_ACP_TASK_CREATE_QUE_FAILED Failed to create message-queue for ACP.

Table 153: Diagnostic Messages of the MGT-Task

## 6.11 Error Codes of the RPC-Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC02E0100	TLR_E_RPC_STATUS Generic RPC-error code. See Profinet-status code for details.
0xC02E0101	TLR_E_RPC_CONNECT_OUT_OF_MEMORY There was not enough memory allocated to receive the whole IO-Device's Connect-Response PDU. Most likely it contains a very large ModuleDiff-Block.
0xC02E0102	TLR_E_RPC_FATAL_ERROR_CLB_ALREADY_REGISTERED The fatal error callback function is already registered.
0xC02E0200	TLR_E_CLRPC_PACKET_SEND_FAILED Error while sending internal message to another task.
0xC02E0201	TLR_E_CLRPC_TIMER_OUT_OF_MEMORY Creating a TLR-Timer-packet in RPC task failed due to insufficient memory.
0xC02E0202	TLR_E_CLRPC_REF_COUNTER_INVALID The reference counter value is invalid.
0xC02E0203	TLR_E_CLRPC_INVALID_PORT_HANDLE The port handle is invalid.
0xC02E0204	TLR_E_CLRPC_TIMER_ALREADY_ACTIVE ((TLR_RESULT) The soft timer is already active (expected inactive).
0xC02E0300	TLR_E_CLRPC_MAPPER_INIT_FAILED The parameter "uiMaxReg" (maximum amount of RPC-mapper registrations) is invalid in CLRPC_EPMap_Initialize().
0xC02E0301	TLR_E_CLRPC_MAPPER_RESOURCE_LIMIT_EXCEEDED The requested Endpoint-Mapper index is invalid.
0xC02E0303	TLR_E_CLRPC_MAPPER_STATUS_INVALID The state of Endpoint-Mapper is invalid for this request.
0xC02E0304	TLR_E_CLRPC_MAPPER_STATUS_CLOSING The Endpoint-Mapper is waiting for close-confirmation and therefore its status is invalid for this request.
0xC02E0305	TLR_E_CLRPC_MAPPER_STATUS_UNKNOWN The status of Endpoint-Mapper is unknown.
0xC02E0306	TLR_E_CLRPC_MAPPER_STATUS_CONFLICT The status of Endpoint-Mapper is not "Ready" and therefore request CLRPC_EPMap_Deregister_req() is invalid.
0xC02E0307	TLR_E_CLRPC_MAPPER_PARAMETER_FAILED Invalid parameter in CLRPC_EPMap_Register_req_Compare().
0xC02E0308	TLR_E_CLRPC_MAPPER_SERVER_REGISTERED CLRPC_EPMap_Deregister_req() is not allowed because at least one RPC-Server is registered to this Endpoint-Mapper.
0xC02E0400	TLR_E_CLRPC_SERVER_INIT_FAILED An error occurred in CLRPC_Server_Initialize().
0xC02E0401	TLR_E_CLRPC_SERVER_RESOURCE_LIMIT_EXCEEDED The maximum number of registered RPC-Servers is exceeded or the maximum number of outstanding requests is exceeded.

Hexadecimal Value	Definition Description
0xC02E0402	TLR_E_CLRPC_SERVER_TIMER_CREATE_FAILED Creating TLR-Timer for RPC-Server failed.
0xC02E0403	TLR_E_CLRPC_SERVER_NO_SERVER_REGISTERED There is no RPC-Server registered that could be deregistered (CLRPC_ServerDeregister_req()).
0xC02E0405	TLR_E_CLRPC_SERVER_MAPPER_HANDLE_INVALID The handle to Endpoint-Mapper in CLRPC_ServerRegister_req() is invalid.
0xC02E0406	TLR_E_CLRPC_SERVER_MAPPER_STATUS_INVALID The status of Endpoint-Mapper in CLRPC_ServerRegister_req() is invalid.
0xC02E0407	TLR_E_CLRPC_SERVER_HANDLE_INVALID The handle to RPC-Server instance is invalid.
0xC02E0408	TLR_E_CLRPC_SERVER_OBJECT_REGISTERED There is at least one object registered to RPC-Server instance. CLRPC_ServerDeregister_req() cannot proceed.
0xC02E0409	TLR_E_CLRPC_SERVER_PARAM_RECV_I Invalid parameter "ulMaxRecv" in request-packet in CLRPC_ServerRegister_req().
0xC02E040A	TLR_E_CLRPC_SERVER_PARAM_SEND_INVALID Invalid parameter "ulMaxSend" in request-packet in CLRPC_ServerRegister_req().
0xC02E040B	TLR_E_CLRPC_SERVER_ELEMENT_INVALID Invalid RPC-Server element "ptElem". Internal RPC-Error.
0xC02E040C	TLR_E_CLRPC_SERVER_REQUEST_CANCELED This RPC-Request was canceled.
0xC02E040D	TLR_E_CLRPC_SERVER_STATE_INVALID The state of RPC server is invalid for this request.
0xC02E040E	TLR_E_CLRPC_SERVER_ACTIVITY_ALREADY_INITIALIZED The activity has already been initialized.
0xC02E040F	TLR_E_CLRPC_SERVER_RECEIVED_INVALID_RSP_PACKET The RPC server received an invalid (unexpected) response packet.
0xC02E0501	TLR_E_CLRPC_OBJECT_SERVER_HANDLE_INVALID The handle to RPC-Server instance in CLRPC_ObjectRegister_req() is invalid.
0xC02E0502	TLR_E_CLRPC_OBJECT_SERVER_STATUS_INVALID The status of RPC-Server instance in CLRPC_ObjectRegister_req() is invalid.
0xC02E0503	TLR_E_CLRPC_OBJECT_HANDLE_INVALID The handle to RPC-Object instance in CLRPC_ObjectDeregister_req() is invalid.
0xC02E0600	TLR_E_CLRPC_CLIENT_INIT_FAILED One of the parameters "uiMaxReg" or "uiMaxReq" in CLRPC_Client_Initialize() is invalid.
0xC02E0601	TLR_E_CLRPC_CLIENT_RESOURCE_LIMIT_EXCEEDED The maximum number of parallel RPC-Client instances is reached in CLRPC_ClientRegister_req()
0xC02E0602	TLR_E_CLRPC_CLIENT_TIMER_CREATE_FAILED Creating the TLR-Timer for RPC-Client instance in CLRPC_ClientRegister_req() failed.
0xC02E0604	TLR_E_CLRPC_CLIENT_MAPPER_STATUS_INVALID The state of Endpoint-Mapper is invalid for this request.
0xC02E0605	TLR_E_CLRPC_CLIENT_HANDLE_INVALID The handle to RPC-Client instance is invalid.

Hexadecimal Value	Definition Description
0xC02E0606	TLR_E_CLRPC_CLIENT_REQUEST_LIMIT_EXCEEDED The maximum amount of outstanding RPC-Requests for this RPC-Clients instance is reached.
0xC02E0607	TLR_E_CLRPC_CLIENT_OPCODE_SEQUENCE RPC-Client instances can only connect to an IO-Device if there are no outstanding RPC-Requests. Currently at least one RPC-Request is outstanding.
0xC02E0608	TLR_E_CLRPC_CLIENT_DEREGISTERED The RPC-Client instance you tried to use is going to deregister right now. Aborting your Request !
0xC02E0609	TLR_E_CLRPC_CLIENT_ELEMENT_INVALID Invalid RPC-Client instance element "ptElem". Internal RPC-Error.
0xC02E060A	TLR_E_CLRPC_CLIENT_LONG_TIMEOUT_HIT The LONG timeout TLR-timer for an outstanding RPC-Request hit. Used internally in RPC only.
0xC02E060B	TLR_E_CLRPC_CLIENT_RESPONSE_SEQUENCE_NUMBER Invalid sequence number in RPC-Message received by RPC-Client instance.
0xC02E060C	TLR_E_CLRPC_CLIENT_CANCEL_TIMED_OUT Canceling a running request timed out. This RPC Client will no longer be usable.
0xC02E060D	TLR_E_CLRPC_CLIENT_NO_REQUEST_PACKET The RPC Client did not have a packet to return.
0xC02E060E	TLR_E_CLRPC_CLIENT_RECV_REQ_WITH_UNEXPECTED_FLAG The RPC Client received a request with an unexpected Flag value.
0xC02E060F	TLR_E_CLRPC_CLIENT_ABORTED_BY_UNBIND_REQ The request was aborted because the RPC client was unbind.
0xC02E0610	TLR_E_CLRPC_MAX_ACTIVITY_RESEND_RETRY_REACHED The maximum resend number was reached by the activity.

Table 154: Error Messages of the RPC-Task

### 6.11.1 Diagnostic Codes of the RPC-Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC02E0010	TLR_DIAG_E_RPC_TASK_CLIENT_RESOURCE_INIT_FAILED Initiating CLRPC-Client failed. (CLRPC_Client_Initialize())
0xC02E0011	TLR_DIAG_E_RPC_TASK_SERVER_RESOURCE_INIT_FAILED Initiating CLRPC-Server failed (CLRPC_Server_Initialize()).
0xC02E0012	TLR_DIAG_E_RPC_TASK_EPMAP_RESOURCE_INIT_FAILED Initiating CLRPC-Endpoint-Mapper failed (CLRPC_Mapper_Initialize()).
0xC02E0013	TLR_DIAG_E_RPC_TASK_INIT_LOCAL_CREATE_QUEUE_FAILED Creating message queue failed.
0xC02E0014	TLR_DIAG_E_RPC_TASK_INIT_REMOTE_IDENT_EDD_FAILED Identifying Drv_EDD failed.
0xC02E0015	TLR_DIAG_E_RPC_TASK_INIT_REMOTE_GET_MAC_FAILED Getting the MAC address failed.

Hexadecimal Value	Definition Description
0xC02E0016	TLR_DIAG_E_RPC_TASK_INIT_REMOTE_IDENT_TCPUDP_QUE_FAILED Getting queue handle to TCPIP-Task failed.

*Table 155: Diagnostic Messages of the RPC-Task*

## 6.12 Other Relevant Error Codes

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC0000003	RCX_E_OUTOFMEMORY Out of memory
0xC0000004	RCX_E_UNKNOWN_COMMAND Unknown command
0xC0000010	RCX_E_PACKET_OUT_OF_MEMORY Packet out of memory
0xC0000012	RCX_E_QUE_SENDBUFFER Queue send packet
0xC000001A	RCX_E_REQUEST_RUNNING Request running
0xC0000180	RCX_E_BUS_OFF Bus off
0xC0000181	RCX_E_CONFIG_LOCKED Configuration locked
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().
0xC0110033	TLR_E_PNIO_ALPMR_HANDLE_INVALID The ALPMR protocol-machine corresponding to the index in request packet is invalid.
0xC0110034	TLR_E_PNIO_ALPMR_STATE_INVALID The ALPMR protocol-machine state is invalid for the current request.
0xC0110035	TLR_E_PNIO_ALPMR_PACKET_SEND_FAILED Sending an Alarm-Indication-packet to another task failed in ALPMR.
0xC0110036	TLR_E_PNIO_ALPMR_PACKET_OUT_OF_MEMORY Creating an Alarm-Indication-packet to be send to another task failed due to insufficient memory.
0xC0110037	TLR_E_PNIO_ALPMR_RESOURCE_INDEX_INVALID The index of ALPMR's protocol machine is invalid.
0xC0110040	TLR_E_PNIO_APMR_PARAM_INVALID_INIT The parameter uiMaxApmr (maximum number of parallel APMR protocol-machines) in APMR_ResourceInit() is invalid.
0xC0110041	TLR_E_PNIO_APMR_RESOURCE_OUT_OF_MEMORY Insufficient memory in APMR_ResourceInit() to create the APMR protocol machines.
0xC0110042	TLR_E_PNIO_APMR_HANDLE_INVALID The APMR protocol machine or its index is invalid.

Hexadecimal Value	Definition Description
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.
0xC0110060	TLR_E_PNIO_CPM_PARAM_INVALID_INIT The parameter uiMaxCpmRtc1 and/or uiMaxCpmRtc2 of CPM_ResourceInit() is invalid.
0xC0110061	TLR_E_PNIO_CPM_PARAM_INVALID_CLASS The requested RTC-class is invalid in CPM_Init_req().
0xC0110062	TLR_E_PNIO_CPM_RESOURCE_LIMIT_EXCEEDED The requested amount of CPM protocol machines is higher than the highest possible value.
0xC0110063	TLR_E_PNIO_CPM_RESOURCE_OUT_OF_MEMORY Insufficient memory for current request in CPM.
0xC0110064	TLR_E_PNIO_CPM_HANDLE_INVALID The handle to CPM protocol machine is invalid.
0xC0110065	TLR_E_PNIO_CPM_STATE_INVALID The state of CPM protocol machine is incorrect for current request.
0xC0110066	TLR_E_PNIO_CPM_PHASE_LIMIT_EXCEEDED Invalid phase found in Init-request-packet in CPM_Init_req() or in ACP_PhaseCpmAdd_req() or ACP_PhaseCpmRemove_req().
0xC0110067	TLR_E_PNIO_CPM_SEND_CLOCK_LIMIT_EXCEEDED The SendClock-factor in Init-request-packet to CPM does not match the one in ACP_Tasks' resources.
0xC0110069	TLR_E_PNIO_CPM_DATALEN_LIMIT_EXCEEDED Packet size to receive is too big. Error is detected in CPM_Init_req().
0xC011006A	TLR_E_PNIO_CPM_PACKET_SEND_FAILED Error while sending a packet to another task in CPM.



Hexadecimal Value	Definition Description
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.
0xC0110084	TLR_E_PNIO_PPM_HANDLE_INVALID The handle to PPM protocol machine is invalid.
0xC0110085	TLR_E_PNIO_PPM_STATE_INVALID The state of PPM protocol machine is incorrect for current request.
0xC0110086	TLR_E_PNIO_PPM_PHASE_LIMIT_EXCEEDED Invalid phase found in Init-request-packet in PPM_Init_req() or in ACP_PhasePPMAdd_req() or ACP_PhasePPMRemove_req().
0xC0110087	TLR_E_PNIO_PPM_SEND_CLOCK_LIMIT_EXCEEDED The SendClock-factor in PPMs Init-request-packet does not match the one in ACP_Tasks' resources.
0xC0110089	TLR_E_PNIO_PPM_DATALEN_LIMIT_EXCEEDED Packet size to send is too big. Error is detected in PPM_Init_req().
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 156: Other relevant Error Messages

## 7 Definition of the LEDs

This definition is only valid using the IO-Controller firmware together with the CIFX 50-RE PCI card.

LED	Color	LED state	Description
SYS yellow/ green	green	ON	Operating System running.
	yellow	Flashing	Device indicates boot error.
	yellow	ON	Bootloader is waiting for booting procedure.
	-	OFF	Power supply for the device is missing or hardware defect.
BF COM1 red	red	Flashing	<b>Configuration fault:</b> not all configured IO-Devices are connected.
	red	ON	<b>No Connection:</b> No Link. or (together with SF „red ON“) <b>No valid Master license</b>
	-	OFF	No Error
SF COM0 red	red	Flashing	<b>System error:</b> Invalid configuration, Watchdog error or internal error
	red	ON	<b>Diagnosis</b> alarm reported by at least one IO-Device or together with BF „red ON“: <b>No valid Master license</b>
	-	OFF	No Error

Table 157: Definition of the LEDs (for the CIFX 50-RE PCI PC Card)

## 8 Appendix

### 8.1 List of Tables

Table 1: List of Revisions .....	5
Table 2: Terms, Abbreviations and Definitions .....	9
Table 3: References to Documents .....	10
Table 4: Names of Queues in PROFINET Firmware .....	14
Table 5: Meaning of Source- and Destination-related Parameters .....	15
Table 6: Meaning of Destination-Parameter ulDest.Parameters. ....	16
Table 7: Example for correct Use of Source- and Destination-related parameters. ....	18
Table 8: Input Data Image .....	24
Table 9: Output Data Image .....	24
Table 10: General Structure of Packets for non-cyclic Data Exchange .....	26
Table 11: Channel Mailboxes .....	29
Table 12: Common Status Structure Definition .....	31
Table 13: Communication State of Change .....	32
Table 14: Meaning of Communication Change of State Flags .....	33
Table 15: Master Status Structure Definition .....	36
Table 16: Status and Error Codes .....	37
Table 17: Extended Status Block .....	38
Table 18: Extended Status Block for PROFINET IO Controller – Second Part (State Field Definition Block) .....	39
Table 19: Status Type IDs used by the PROFINET IO Controller Protocol Stack .....	40
Table 20: Communication Control Block .....	40
Table 21: Overview about essential Functionality (cyclic and acyclic Data Transfer, Alarm Handling and Registering Application to Firmware) .....	42
Table 22: Packet Structure .....	60
Table 23: Packet Header of Request Packet .....	63
Table 24: Overview over the Configuration Packets of the PROFINET IO Controller Protocol Stack .....	66
Table 25: Configure extended PNM Request Packet (Structure Version == 2) .....	68
Table 26: Configure extended PNM Request Packet (Structure Version == 1) .....	69
Table 27: System Flags .....	71
Table 28: IP Flags .....	72
Table 29: Configure extended PNM Confirmation .....	74
Table 30: Configure PNM Request Packet .....	76
Table 31: Configure PNM Confirmation Packet .....	77
Table 32: Configure PNM_IOD extended Request Packet .....	79
Table 33: AR Property Flags .....	80
Table 34: Alarm CR Property Flags .....	81
Table 35: Configure PNM_IOD extended Confirmation Packet .....	82
Table 36: Configure PNM_IOD Request Packet .....	84
Table 37: Configure PNM_IOD Confirmation Packet .....	85
Table 38: Configure PNM_IOD_IOC request packet .....	87
Table 39: IOC Property Flags .....	88
Table 40: Configure PNM_IOD_IOC Confirmation Packet .....	89
Table 41: Configure PNM_IOD_AP Request Packet .....	90
Table 42: Configure PNM_IOD_AP Confirmation Packet .....	91
Table 43: Configure PNM_IOD_Module Request Packet .....	92
Table 44: Configure PNM_IOD_Module Confirmation Packet .....	93
Table 45: Configure PNM_IOD_Submodule Request Packet .....	94
Table 46: Submodule Property Flags .....	95
Table 47: Configure PNM_IOD_Submodule confirmation packet .....	96
Table 48: Configure PNM_IOD_SubmDescr_Ext Request .....	98
Table 49: Configure PNM_IOD_Submdescr Confirmation .....	99
Table 50: Configure PNM_IOD_SubmDescr request packet .....	100
Table 51: Configure PNM_IOD_SubmDescr Confirmation Packet .....	102
Table 52: Configure PNM_IOD_IO_Signals Request Packet .....	104
Table 53: Configure PNM_IOD_IO_Signals Confirmation Packet .....	104
Table 54: Configure PNM_IOD Interface Submodule Request Packet with Structure Version == 2 .....	105
Table 55: Configure PNM_IOD Interface Submodule Request Packet with Structure Version == 1 .....	106
Table 56: Configure PNM_IOD Interface Submodule Confirmation Packet .....	108
Table 57: Configure PNM_IOD Port Submodule Request Packet .....	109
Table 58: Configure PNM_IOD Port Submodule Confirmation Packet .....	111
Table 59: Configure PNM_IOD_RecData request packet .....	112
Table 60: Configure PNM_IOD_RecData Confirmation Packet .....	114
Table 61: PNM_DWNL_FIN Request Packet .....	115
Table 62: PNM_DWNL_FIN Confirmation Packet .....	116
Table 63: Overview over the Registration and Unregistration Packets of the PROFINET IO Controller Protocol Stack ..	119

Table 64: APIOC_APPLICATION_REGISTER_REQ – Request Packet for Registering an Application to IO Controller..	120
Table 65: APIOC_APPLICATION_REGISTER_CNF – Confirmation Packet for Registering an Application to IO Controller .....	121
Table 66: APIOC_APPLICATION_REGISTER_CNF - Status Codes of Registering an Application to IO Controller.....	121
Table 67: APIOC_APPLICATION_DEREGISTER_REQ – Request Packet for Deregistering the registered Application..	122
Table 68: Deregister Application Flags .....	123
Table 69: APIOC_APPLICATION_DEREGISTER_CNF – Confirmation Packet for deregistering the registered Application .....	124
Table 70: APIOC_APPLICATION_DEREGISTER_CNF - Status Codes of deregistering the registered Application .....	124
Table 71: Overview over the Acyclic Request Packets of the PROFINET IO Controller Protocol Stack .....	125
Table 72: APIOC_READ_REQ – Request Packet for Reading Data from IO Device .....	126
Table 73: APIOC_READ_CNF – Confirmation Packet for Reading Data from IO Device .....	127
Table 74: APIOC_READ_CNF - Packet Sstatus/Error .....	128
Table 75: APIOC_WRITE_REQ – Request Packet for Writing Data to IO Device .....	130
Table 76: APIOC_WRITE_CNF – Confirmation Packet for Writing Data to IO Device .....	130
Table 77: APIOC_WRITE_CNF - Packet Status/Error.....	131
Table 78: APIOC_READ_IMPL_REQ – Request Packet for implicitly Reading Data from IO Device .....	132
Table 79: APIOC_READ_IMPL_CNF – Confirmation Packet for implicitly Reading Data from IO Device .....	133
Table 80: APIOC_READ_IMPL_CNF - Packet Status/Error .....	134
Table 81: APIOC_DEVICE_DIAG_REQ – Request Packet for getting diagnosis Data for an IO Device.....	135
Table 82: Diagnosis Data Flags .....	135
Table 83: APIOC_DEVICE_DIAG_CNF – Confirmation Packet with Diagnosis Data of an IO Device.....	136
Table 84: TCP/IP_TCP_UDP_CMD_GET_SOCKET_OPTION_CNF - Packet Status/Error.....	136
Table 85: Diagnosis Data Flags .....	137
Table 86: Get Slave Connection Information Request .....	138
Table 87: Device Connection Information Confirmation for active IO Devices .....	139
Table 88: Device Connection Information Confirmation for inactive IO Devices .....	140
Table 89: DIAG_INFO_GET_COMMON_STATE_REQ - Get Common Status Block Request .....	141
Table 90: DIAG_INFO_GET_COMMON_STATE_CNF - Get Common Status Block Confirmation .....	142
Table 91: APIOC_GET_MOD_DIFF_BLOCK_REQ – Get ModuleDiffBlock Request .....	143
Table 92: APIOC_GET_MOD_DIFF_BLOCK_CNF – Get ModuleDiffBlock Confirmation .....	144
Table 93: Flags used in Get ModuleDiffBlock confirmation packet .....	145
Table 94: GET_MOD_DIFF_BLOCK_CNF - Status/Error Codes .....	145
Table 95: APIOC_DCP_SET_SIGNAL_REQ – DCP SIGNAL Request Packet.....	146
Table 96: Flags to use in DCP SIGNAL Request Packet.....	147
Table 97: APIOC_DCP_SET_CNF – DCP SIGNAL Confirmation Packet.....	148
Table 98: DCP_SIGNAL_CNF - Status/Error Codes.....	149
Table 99: APIOC_DCP_SET_NAME_REQ – DCP SET Name Request Packet.....	150
Table 100: Flags to use in DCP SET Name Request Packet.....	151
Table 101: APIOC_DCP_SET_CNF – DCP SET NAME Confirmation Packet.....	152
Table 102: DCP_SET_NAME_CNF - Status/Error Codes .....	153
Table 103: APIOC_DCP_SET_IP_REQ – DCP SET IP Request Packet .....	154
Table 104: Flags to use in DCP SET IP Request Packet.....	155
Table 105: APIOC_DCP_SET_CNF – DCP SET IP Confirmation Packet.....	156
Table 106: DCP_SET_IP_CNF - Status/Error Codes.....	157
Table 107: APIOC_DCP_RESET_FACTORY_REQ – DCP RESET FACTORY Request Packet .....	158
Table 108: APIOC_DCP_RESET_FACTORY_CNF – DCP RESET FACTORY Confirmation Packet.....	159
Table 109: DCP_RESET_FACTORY_CNF - Status/Error Codes.....	160
Table 110: APIOC_DCP_IDENT_ALL_REQ – DCP IDENT ALL Request Packet.....	161
Table 111: APIOC_DCP_IDENT_ALL_CNF – DCP IDENT ALL Confirmation Packet .....	162
Table 112: DCP_IDENT_ALL_CNF - Status/Error Codes .....	162
Table 113: APIOC_DCP_GET_REQ_T – DCP GET Request Packet .....	163
Table 114: DCP GET Options Flags .....	164
Table 115: APIOC_DCP_GET_CNF_T – DCP GET Confirmation Packet .....	165
Table 116: DCP_GET_CNF - Status/Error Codes .....	166
Table 117: APIOC_ALARM_ACK_REQ – Alarm Acknowledge Request Packet .....	167
Table 118: APIOC_ALARM_ACK_REQ_V1 - Alarm Acknowledge Request Packet (legacy) .....	168
Table 119: APIOC_ALARM_ACK_CNF – Alarm Acknowledge Confirmation Packet.....	169
Table 120: Alarm Acknowledge Confirmation - Status Codes.....	170
Table 121: Release IO-Device Request Packet Definition .....	171
Table 122: Release IO-Device Confirmation Packet Definition .....	172
Table 123: Release IO-Device Confirmation - Status Codes .....	173
Table 124: Overview over the Acyclic Indication Packets of the PROFINET IO Controller Protocol Stack .....	174
Table 125: APIOC_DCP_IDENT_ENTRY_REQ – DCP IDENT ENTRY Indication Packet .....	175
Table 126: APIOC_DCP_IDENT_ENTRY_RSP – DCP IDENT ENTRY Response Packet .....	176
Table 127: DCP_IDENT_ENTRY_RSP - Status/Error Codes.....	176

Table 128: APIOC_DCP_IDENT_FINISHED_IND – DCP IDENT ALL Finished Indication Packet .....	177
Table 129: DCP_IDENT_FINISHED_IND - Status/Error Codes .....	178
Table 130: APIOC_DCP_IDENT_FINISHED_RSP – DCP IDENT ALL Finished Response Packet .....	179
Table 131: DCP_IDENT_FINISHED_IND - Status/Error Codes .....	179
Table 132: APIOC_ALARM_IND - Alarm Indication Packet .....	181
Table 133: Alarm Specifier .....	181
Table 134: APIOC_ALARM_RSP - Alarm Response Packet .....	182
Table 135: APIOC_ALARM_RSP_V1 - Alarm Response Packet (legacy) .....	183
Table 136: APIOC_DIAG_DATA_IND_T - New Diagnosis Data Indication Packet .....	184
Table 137: APIOC_DIAG_DATA_RSP_T - New Diagnosis Data Response Packet .....	186
Table 138: Error Messages of the ACP-Task .....	187
Table 139: Diagnostic Messages of the ACP-Task .....	188
Table 140: Error Messages of the APCFG-Task .....	194
Table 141: Error Messages of the APCTL-Task .....	198
Table 142: Error Messages of the CMCTL-Task .....	202
Table 143: Diagnostic Messages of the CMCTL-Task .....	203
Table 144: Error Messages of the CMDEV-Task .....	206
Table 145: Diagnostic Messages of the CMDEV-Task .....	207
Table 146: Error Messages of the DCP-Task .....	209
Table 147: Diagnostic Messages of the DCP-Task .....	210
Table 148: Error Messages of the EDD-Task .....	210
Table 149: Diagnostic Messages of the EDD-Task .....	210
Table 150: Error Messages of the IO Signal –Task .....	211
Table 151: Error Messages of the LLDP-Task .....	215
Table 152: Error Messages of the MGT-Task .....	218
Table 153: Diagnostic Messages of the MGT-Task .....	218
Table 154: Error Messages of the RPC-Task .....	221
Table 155: Diagnostic Messages of the RPC-Task .....	222
Table 156: Other relevant Error Messages .....	225
Table 157: Definition of the LEDs (for the CIFX 50-RE PCI PC Card) .....	226

## 8.2 List of Figures

Figure 1: The 3 different Ways to access a Protocol Stack running on a netX System .....	13
Figure 2: Use of ulDest in Channel and System Mailbox .....	16
Figure 3: Using ulSrc and ulSrcId .....	17
Figure 4: Transition Chart Application as Client .....	21
Figure 5: Transition Chart Application as Server .....	22
Figure 6: Example: DPM Layout of Input Area .....	41
Figure 7: Bus and Device Structure in PROFINET IO .....	43
Figure 8: Example of Structure and Sequence of the Download Packets .....	45
Figure 9: Example of Structure and Sequence of the Download Packets (with optional Packets) .....	46
Figure 10: Sequence and Nesting of the Configuration Data (Part 1) .....	47
Figure 11: Sequence and Nesting of the Configuration Data (Part 2) .....	48
Figure 12: Task Structure of the PROFINET IO Controller Stack .....	51
Figure 13: Sequence Diagram to get Slave Handle and Connection Information .....	56
Figure 14: Flowchart of the entire Configuration Data download .....	62

## 8.3 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)