Protocol API

PROFINET IO-Device

V3.14.0

Hilscher Gesellschaft für Systemautomation mbH
www.hilscher.com

DOC111110API19EN | Revision 19 | English | 2019-04 | Released | Internal

Table of contents

1 Introduction

1.1 About this document

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

1.2 List of revisions

Revision	Date	Author	Revision
18	2018-07-10	BM, HH	Firmware/stack version V3.13.0
			Section *Parameterization Speedup Support service* [▸ page 178]: description expanded.
			Section *Set OEM Parameters request* [▸ page 72]: Overview table added.
			Section *Set OEM Parameters request* [▸ page 72]: Set OEM Parameter type 9, 14, and 15 added.
			Section *Get XMAC (EDD) Diagnosis confirmation* [▸ page 188]: ulLen depends on used firmware version and hardware variant.
			Section *Get Submodule Configuration service* [▸ page 231] revised.
19	2019-04-29	HHE	Firmware/stack version V3.14.0
			Packet structure references updated.
			Section *Remanent data handling* [▸ page 40] updated.
			Section *Set OEM Parameters service* [▸ page 70] updated.
			Section *Connect Request Done service* [▸ page 111]: Stack waits for Connect Request Done response from application.
			Sections *Plug Submodule request* [▸ page 204] and *Extended Plug Submodule request* [▸ page 207] updated.
			Section *Get Parameter service* [▸ page 238]: Reading of switch statistic counters added.
			Section *Get Parameter confirmation* [▸ page 240]: Elements usPrmType and usPadding added to PNS_IF_PARAM_PORT_STATISTIC_DATA_T.
			Section *Send Alarm service* [▸ page 252]: Process Alarm with channel coding added.
			Section *Common status codes* [▸ page 292] added.
			Section *Name encoding* [▸ page 323] added.

Table 1: List of revisions

1.3 Functional overview

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

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

1.4 System requirements

This software package has following system requirements to its environment:

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

1.5 Intended audience

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

- Knowledge of the programming language C
- Knowledge of the PROFINET communication system
- Knowledge of the netX dual-port memory interface and communication mechanism e.g. communication via packets

1.6 Specification

1.6.1 Supported protocols

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

RTA – Real time Acyclic Protocol

DCP – Discovery and Configuration Protocol

CL-RPC – Connectionless Remote Procedure Call

LLDP – Link Layer Discovery Protocol

SNMP – Simple Network Management Protocol

MRP – MRP Client is supported

1.6.2 Technical data

Feature	Value
Maximum number of total cyclic input data	1440 bytes (including IOPS and IOCS)
Maximum number of total cyclic output data	1440 bytes (including IOPS and IOCS)
Maximum number of submodules	255 submodules per Application Relation at the same time, 1000 submodules can be configured
Multiple Application Relations (AR)	The stack can handle up to 8 IO-ARs, multiple Supervisor AR and one Supervisor-DA AR at the same time. If executed on netX50 this is limited to 2 IO-ARs instead of 8.
Acyclic communication	Read/Write Record: up to 8 KB for loadable firmware (using DPM fragmentation) up to 32 KB for linkable object module
Alarm types	Process Alarm, Diagnostic Alarm, Return of Submodule Alarm, Plug Alarm (implicit), Pull Alarm (implicit), Update Alarm, Status Alarm, Isochronous Problem Alarm, Upload and Retrieval Notification Alarm
Identification & Maintenance (I&M)	I&M0-4 (If the stack is configured to handle I&M by itself (this is the default) the GSDML device description shall be adapted to indicate the support of I&M0-4.) Reading of I&M5 for loadable firmware (depending on the target)
Asset management	Up to 199 assets
PROFIenergy	PROFIenergy ASE implementation with one PE entity per submodule
Topology recognition	LLDP, SNMP V1, MIB2, physical device
Minimum cycle time	1ms for RT_CLASS_1 (all implementations) 1ms for netX50 and RT_CLASS_3 250µs for netX51 and RT_CLASS_3 250µs for netX100/500 and RT_CLASS_3
IRT support	RT_CLASS_3
Media redundancy	MRP client
Additional supported features	DCP, VLAN- and priority tagging, Shared Device (but only 1 RTC3 AR in total)
Baud rate	100 MBit/s
Data transport layer	Ethernet II, IEEE 802.3
PROFINET IO specification	V2.3, legacy startup of specification v2.2 is supported

Table 2: Technical data

1.6.3 Limitations

- RT over UDP not supported
- Multicast communication not supported
- Only one device instance is supported
- DHCP is not supported
- Fast Startup is implemented in the stack. However some additional hardware limitations apply to use it.
- The amount of configured IO-data influences the minimum cycle time that can be reached.
- Only 1 Input-CR and 1 Output-CR per AR are supported
- Using little endian byte order for cyclic process data instead of default big endian byte order may have an negative impact on minimum reachable cycle time
- System Redundancy (SR-AR) and Dynamic Reconfiguration (formerly known as Configuration-in-Run, CiR) are not supported
- Max. 255 submodules can be used simultaneously within one specific Application Relation
- SharedInput is not supported
- MRPD is not supported
- DFP and other HighPerformance-profile related features are not supported
- PDEV functionality is only supported for submodules located in slot 0
- Submodules cannot be configured or used by an AR in subslot 0
- DAP and PDEV submodules only supported in slot 0.
- Only loadable firmware for netX 51 (NXLFW) and netRAPID 51 (NRP51-RE) support Flash Device Label (FDL).

Only for LOM

- The protocol stack cannot be used together with the standard two-port switch of rcX.
- The protocol stack cannot be used together with the standard MAC of rcX.
- Thus the protocol stack can exclusively be used with the PROFINET IO-Device switch. Consequently, in any case 3 XC channels are required at the netX 100 and the netX 500.
- It is not possible to manually configure the integrated Hilscher TCP/IP task with its service `TCPIP_IP_CMD_SET_CONFIG_REQ` when using PROFINET IO Device protocol stack. The service will succeed but has no effect.

1.7 References to documents

[1] PROFIBUS International: Technical Specification for PROFINET IO: Application layer protocol for decentralized periphery, Version 2.3Ed2MU5, Order No. 2.722, English, 2018-03.

[2] PROFIBUS International: Profile Guidelines Part 1: Identification & Maintenance Functions, Version V2.1, Order No. 3.502, English, 2016-05.

[3] PROFIBUS International: PROFINET Field Devices, Recommendations for Design and Implementation, http://www.profibus.com/nc/download/technical-descriptions-books/downloads/profinet-field-devices-recommendations-for-design-and-implementation/display/

[4] Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Interface Manual, netX Dual-Port Memory Interface, Revision 15, DOC060302DPM15EN, English, 2019.

[5] Hilscher Gesellschaft für Systemautomation mbH: Packet API, netX Dual-Port Memory, Packet-based services, Revision 2, DOC161001API02EN, English, 2018.

[6] Hilscher Gesellschaft für Systemautomation mbH: Protocol API, TCP/IP, Packet Interface, V2.4, Revision 14, DOC050201API14EN, English, 2017.

[7] Hilscher Gesellschaft für Systemautomation mbH: Protocol API, Socket Interface, Packet Interface, Revision 5, DOC140401API05EN, English, 2019.

2 Getting started

2.1 Configuration methods

The PROFINET IO Device stack requires configuration parameters. The stack offers the following configuration methods:

- The application can set the configuration parameters using the Set Configuration service to transfer the parameters within one (or serveral) packets to the stack.
- You can use one of the following configuration software to set the configuration parameters: SYCON.net configuration software

2.2 Input and output data conventions

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

PROFINET	netX	PROFINET Device Stack	Description
Inputs (Data)	Application writes data to the output area of the process data memory	Provider Image / Provider Data	Data of Input Submodules. Send from the Device to the Controller
Outputs (Data)	Application reads data from the input area of the process data memory	Consumer Image / Consumer Data	Data of Output Submodules. Send from the Controller to the Device

Table 3: Naming convention of input / output data

3 Exchanging cyclic data

This section describes how the application can access the cyclic I/O data which is to be exchanged with the IO-Controller.

The PROFINET Device stack provides different ways to exchange this data. Depending on the user's application only one of these methods may be used:

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

3.1 General concepts

PROFINET uses the concept of a cyclic process data image. Each controller or device of a PROFINET network has an image of input and output data. Each image is updated via communication partner images using periodic Ethernet telegrams. These frames are sent at intervals configured by the engineering tool. The frames contain the I/O data and the data status associated with it. Furthermore, each frame contains a “global” frame data status field which can be used e.g. to mark the whole frame as invalid.

PROFINET organizes the cyclic data in a Provider Consumer Model, i.e. an IO data consumer exists for every IO data provider. Both indicate their current state to each other in several frames. These states are the IO Provider State (IOPS) and IO Consumer State (IOCS).

The IOPS indicates whether the associated data is valid or invalid. For instance, a faulty submodule in a device will mark its input data as invalid by setting the IOPS appropriately.

To indicate whether the data has been handled, the consumer returns the IOCS to the provider. For instance, a Digital Analog Converter submodule can use this service to indicate to the controller an output value which is out of range.

With PROFINET, each submodule has its I/O data and I/O data states, i.e. the I/O data plus two I/O data states are exchanged for every submodule used by the IO-Controller. Regarding the Ethernet frame structure, the provider data state is located directly behind the I/O data and contains information on whether the I/O data is good and may be evaluated or not. The consumer state is sent in the opposite direction in a different frame and contains information on whether the I/O data can be handled by the consumer.

Important:
Due to the concept of a cyclic process data image, we strongly recommend designing the application in a way that the consumer data can be read cyclically and that the provider data can be written regardless of the state of communication. If no communication is active, the stack will set the consumer data to zero.

Note:
If the application cannot be designed in a way that the process data can be read or written cyclically or if the cycle is slow (larger than 50 ms), the consumer data must be read when the `PNS_IF_IO_EVENT_CONSUMER_UPDATE_REQUIRED` occurs, and the provider data must be written when the `PNS_IF_IO_EVENT_PROVIDER_UPDATE_REQUIRED` event is signaled.

3.2 Behavior regarding IO data and IOPS

An application determines the behavior of physical outputs. The stack transfers the output data received from the controller into the IO image only (e.g. DPM). The application has to decide on whether this data is transferred to physical outputs or not. The stack, if activated, can add status information (IOPS) to the data. IOPS is generated by the provider of the data (IO Controller). Using IOPS, the application can decide on whether the data is to be transferred to the physical outputs or not, depending on the data validity indicated by IOPS.

Section *Set IOXS Config service* [▸ page 90] describes how to activate the IOPS. Depending on the configuration, three possible modes exist which use data with or without status.

In case of an error, the default behavior of the PROFINET stack is the copying of zeros into the IO image of all affected submodules.

Data without IOPS

The stack transfers the data only. In case of an error, the stack sets the data of all affected submodules to zero. If all values of a submodule are zero, the application cannot detect whether the controller sends valid values (which are zero) or whether this data is invalid due to an error.

Note:
Using data without IOPS is not recommended.

Important:
The application has to use the IOXS interface (*Set IOXS Config service* [▸ page 90]) to be able to detect invalid data because zero values can also be valid data.

Data with IOPS - mode Bitwise

IOPS is activated in mode Bitwise. Each submodule has one status bit. The following table lists the coding for a bit.

IOPS	Description
DataState	0 - Bad, data not valid.
	1 - Good, data is valid.

Table 4: IOPS - DataState – mode Bitwise

Data with IOPS - mode Bytewise

IOPS is activated in mode Bytewise.

The application has to verify / set bit 7 of the IOPS. Bits 5-6 has additional information on the instance that has detected invalid data.

© Hilscher 2011 – 2019

IOPS	Description
Bit 5 and 6 - Instance	These bits indicate the instance that has detected the invalid data. If DataState is set to good, bits 5-6 can be ignored.
	00 - Detected by subslot.
	01 - Detected by slot.
	10 - Detected by IO-Device.
	11 - Detected by IO-Controller.
Bit 7 - DataState	0 - Bad, data not valid.
	1 - Good, data is valid.

Table 5: IOPS - DataState – mode Bytewise

3.3 Exchanging cyclic data using callback interface

3.3.1 Overview of the callback interface

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

1. The user application has to allocate two memory blocks. One to hold the consuming (output) and one to hold the providing (input) data. These blocks shall be large enough to hold the IO-data and the IOPS if desired.
2. Pass the pointers to the blocks, their size and optionally user application's event callback function pointer to the stack using the Set IO-Image Service described in section *Set IO Image service* [▸ page 87]. In return this service will provide the user application with function pointers of the callback functions to call for data exchange.
3. Now call the callbacks either cyclically or if necessary:
   - To update the outgoing provider data call the UpdateProviderImage Callback described in section *UpdateProviderImage Callback* [▸ page 19] whenever the outgoing data has changed.
   - To get the newest incoming consumer data, call the UpdateConsumerImage described in section *UpdateConsumerImage callback* [▸ page 18] periodically.

3.3.2 Callback functions

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

Important:
The callback functions may only be called by the application in task context. They shall not be called in interrupt context when the interrupt modes SYSTEM or INTERRUPT are used.
Otherwise the stack may get stuck or the operating system task scheduler will not work properly any longer.

Important:
The callback functions may only be called by the application in one task context. They shall not be called in different contexts (e.g. by different tasks). Otherwise the application may get stuck or the operating system task scheduler will not work properly any longer.

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

3.3.2.1 UpdateConsumerImage callback

Important:
The Update ConsumerImage callback function may only be called by the application in task context. It shall not be called in interrupt context when the interrupt modes SYSTEM or INTERRUPT are used. Otherwise the stack may get stuck or the operating system task scheduler will not work properly any longer.

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

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

The four parameters of the callback are described as follows:

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

Table 6: Parameters of UpdateConsumerImage callback

The callback uses the following return codes:

Return codes	Meaning
0x00000000	TLR_S_OK: No Error.

Table 7: Return Codes of UpdateConsumerImage callback

3.3.2.2 UpdateProviderImage Callback

Important:
The Update ProviderImage callback function may only be called by the application in task context. It shall not be called in interrupt context when the interrupt modes SYSTEM or INTERRUPT are used. Otherwise the stack may get stuck or the operating system task scheduler will not work properly any longer.

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

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

The four parameters of the callback are described as follows:

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

Table 8: Parameters of UpdateProviderImage callback

The callback uses the following return codes:

Return code	Meaning
0x00000000	TLR_S_OK: No error.

Table 9: Return Codes of UpdateProviderImage callback

3.3.2.3 Event handler callback

This callback has to be provided by the user's application to the stack. It will be called by the stack upon events defined in section *Event mechanism* [▸ page 31]. As the callback will be called from stacks task context, the user must consider locking of shared resources.

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

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

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

The two parameters of the callback are described as follows:

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

Table 10: Parameters of event handler callback

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

4 Stack features

4.1 Structure of the PROFINET Device stack

The following figure shows the structure of PROFINET IO Device stack.

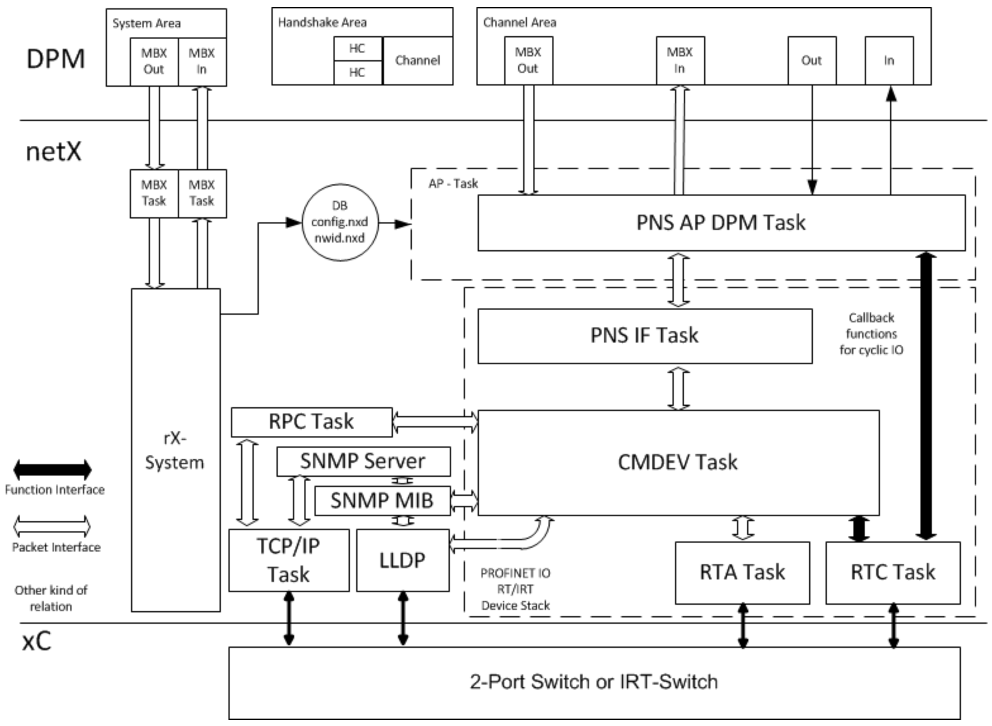


Figure 1: Stack structure

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

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

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

RTA task

The RTA task provides the following functionality:

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

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

RTC task

The RTC task provides the following functionality:

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

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

CMDEV task

The CMDEV task provides the following functionality:

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

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

RPC task

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

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

LLDP task

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

SNMP tasks

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

PNS_IF task

The PNS_IF task provides the following functionality:

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

PNS AP DPM task

The Dual-Port Memory interface of the PROFINET Device stack.

TCP/IP task

The TCP/IP task provides TCP and UDP services.

Ethernet/PROFINET switch and Ethernet driver

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

- Standard Ethernet 2-Port Switch functionality (local send & receive of frames, forwarding of Frames between ports)
- Ethernet PHY handling (LinkUp/Down/State)
- Handling of protocols e.g. PTCP and MRP

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

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

4.2 Configuration

4.2.1 Sequence of configuration evaluation

The stack offers several configuration methods. Exactly one method can be used at a time. At startup, the stack evaluates the configuration in the following order:

1. In case the file CONFIG.NXD and NWID.NXD are available, the stack will use the configuration parameters from these files and starts working.
2. In case the file INIBATCH.NXD is available (legacy), the operating system will send the configuration parameters from this file to the stack and the stack starts working.
3. The stack "waits" for the configuration parameters and remains unconfigured. The application has to use the *Set Configuration* [▸ page 53] packet to configure the PROFINET IO Device and a Channel Init service to activate the configuration parameters.

4.2.2 Configuration lock

The application can lock the configuration, for details see reference [5].

If the configuration of the stack is locked (via Services described in [DPM Manual]), the following behavior is implemented in the stack:

- A Set Configuration packet is not accepted.
- Configuration Reload / Channel Init will be rejected.

However, PROFINET specific services affecting the configuration are still working as defined by PROFINET specification. This includes setting IP parameters and NameOfStation by means of DCP and writing PDEV Parameters using records.

4.2.3 Setting Parameters by means of DCP

The PROFINET specification defines the **Discovery and basic Configuration Protocol** (DCP) to change/set PROFINET device parameters over the network.

A PROFINET IO-Controller, a PROFINET IO-Supervisor or an Engineering System can easily change the IP parameters or the NameOfStation of a PROFINET IO Device at any time. These new parameters can either be marked to be used temporarily or marked to be stored remanent.

The receipt of such a request is indicated to the application with the *Save Station Name service* [▸ page 137], the *Save IP Address service* [▸ page 140] or the *Reset Factory Settings service* [▸ page 146].

The stack will adapt to these new parameters at runtime. New parameters have to be stored remanent. The stack has a configuration parameter whether the stack or the application store the remanent data, for details see section *Remanent data handling* [▸ page 40].

4.3 Ethernet MAC addresses

The stack requires the following MAC addresses for operation:

- one **Interface MAC Address**,
- for each Ethernet port: one **Port MAC address** e.g. two MAC addresses for a 2-port device,
- one **Raw Ethernet MAC Address** (Ethernet interface, NDIS).

The MAC addresses are configured via different paths:

- The Second Stage Boot Loader provides a single MAC address to the firmware on startup of the firmware. This MAC address is used as Interface MAC address. The Port MAC Addresses are calculated from the Interface MAC Address.
- A security memory chip is attached to the netX. The firmware reads the MAC Address from the security memory. This MAC address is used as Interface MAC address. The Port MAC Addresses are calculated from the Interface MAC Address.
- A Flash Device Label is found in the Flash memory chip attached to the netX (netX 51/52 only). The firmware reads the MAC Address from the Flash memory. This MAC address is used as Interface MAC address. The Port MAC Addresses are calculated from the Interface MAC Address.
- Finally, before configuring the protocol stack the application can assign custom MAC addresses. This step is mandatory if the firmware was not able to determine the MAC address of the device using one of the methods above. This step can be performed optionally after startup of the firmware to override any other MAC address settings. In order to assign a MAC address the Set MAC Address service (see reference [5]) and afterwards the *Set Port MAC Address service* [▸ page 68] must be used in exactly this order.

The following figure shows the MAC address determination bootup sequence.

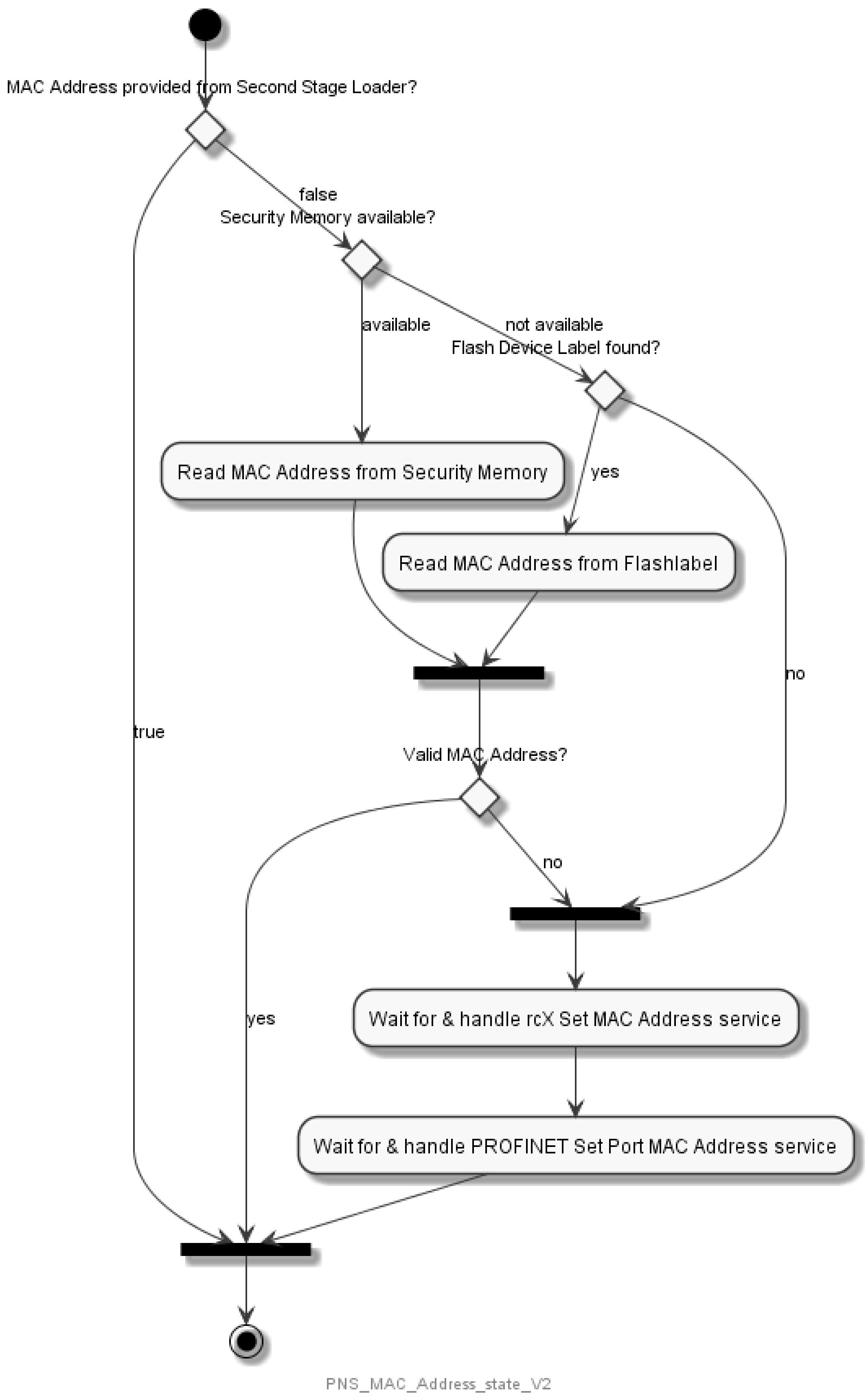

MAC Address provided from Second Stage Loader?
false
Security Memory available?
available
not available
Flash Device Label found?
Read MAC Address from Security Memory
yes
Read MAC Address from Flashlabel
no
true
Valid MAC Address?
no
yes
Wait for & handle rcX Set MAC Address service
Wait for & handle PROFINET Set Port MAC Address service
PNS_MAC_Address_state_V2

Figure 2: Sequence of MAC Address determination

4.4 Identification & Maintenance 5 (I&M5)

While I&M0 to I&M4 are quite old datasets (back from Profibus times) PROFINET introduced a new I&M5 dataset. Its main purpose is visibility of communication interfaces in PROFINET devices.

If a fixed communication module is used (e.g. a PC card) to handle the whole PROFINET communication and the application us running decoupled (e.g. as application program on the PC) it is expected that the I&M0 dataset shows vendor identification of the application.

The information “a PC card with its own dedicated firmware is used” is not visible from the outside world via PROFINET I&M0. The new dataset I&M5 was introduced to solve this issue.

It is expected that this kind of device will support I&M0 dataset as well as I&M5. Inside I&M0 the VendorID and DeviceID of the whole device is used (as set by its application). In I&M5 however the VendorID and DeviceID of the manufacteurer of the PC card is visible.

4.4.1 APIs for usage of I&M5

The following APIs exist to modify the behavior of PROFINET IO Device protocol stack regarding I&M5.

Set OEM Parameter

In case PROFINET IO Device protocol stack handles I&M data using `PNS_IF_SET_OEM_PARAMETERS_TYPE_5` field `ulIMFlags` support of I&M5 can be enabled/disabled.

In case PROFINET IO Device protocol stack handles I&M data using `PNS_IF_SET_OEM_PARAMETERS_TYPE_9` the I&M5 content reported to the network can be changed.

See section *Set OEM Parameters service* [▸ page 70].

Read I&M

In case the application handles I&M data, it is possible to freely choose the values reported to I&M5. It is possible to not support I&M5 by setting the I&MSupported field in I&M0 accordingly. See section *Read I&M service* [▸ page 162]. To activate I&M5 use *Set OEM Parameters service* [▸ page 70].

PNS_StackInit()

In case of NXLOM the Flag `PROFINET_IODEVICE_STARTUP_FLAG_IM5_SUPPORTED` can be used to enable / disable I&M5 support. See section *PNS_StackInit()* [▸ page 266].

Get Parameter

Using the parametertype `PNS_IF_PARAM_IM5_DATA` it is possible to read out the values the PROFINET IO Device protocol stack would respond on the network in case I&M5 is read. See section *Get Parameter service* [▸ page 238].

4.4.2 Usage of OEM VendorId and OEM DeviceId

These parameters have been implemented for the very special use case of ready-for-use PROFINET communication modules (such as Hilscher comX modules, for instance). This approach allows the device to identify itself with two different pairs of Vendor Id and Device Id, namely:

1. The identity of the manufacturer of the entire device (to be stored in I&M 0 vendor and device parameters)
2. The identity of the communication module’s manufacturer (to be stored in I&M 5 vendor and device parameters)

In this way, customers running a plant have the chance to access information about the manufacturer of the communication module within a component of their plant.

There are mainly two possible implementation scenarios for the usage of OEM VendorId and OEM DeviceId:

Minimal applications

Minimal applications work as follows:

- They do not use mailbox-based communication.
- Configuration via database (generated by SYCON.net for instance)
- OEM VendorId and OEM DeviceId are set using the start-up parameters
  - For LFW applications, a special tool is available for adjustment of start-up parameters.

Full-featured mailbox-based applications

Such applications apply:

- Mailbox-based communication
- Configuration via packet interface (configuration parameters)
- Access VendorId and DeviceId using the *Read I&M service* [▸ page 162] or the *Write I&M service* [▸ page 169], respectively.
- The Set OEM Parameter service is used as follows:
  - I&M 5 can be switched on and off via Set OEM Parameter service, type 5.
  - Switching between I&M 5 control by stack or by application is possible, even at runtime (via Set OEM Parameter service, type 8)!

Note:
For general information on the PROFINET feature Information and Maintenance (I&M), see references [2] and [3].

4.5 Status information

4.5.1 Communication state

This section describes how the communication state is used by PROFINET IO Device.

4.5.1.1 Implementation from V3.10

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

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

Table 11: Communication state (V3.10 and later)

4.5.1.2 Legacy Implementation (V3.9 and earlier)

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

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

Table 12: Communication state (V3.9 and earlier)

If needed, legacy handling could be enabled by setting the AP task startup parameter `PNS_AP_DPM_STARTUP_FLAG_LEGACY_COMM_STATE` in firmware/stack V3.10.

4.6 Event mechanism

The PROFINET Device stack uses an event mechanism to indicate some import events to the application. If the stack is accessed using Dual-Port-Memory or Shared Memory API, the Events will be indicated using the *Event Indication service* [▸ page 180].

If the callback interface is used to access the I/O data, the events are indicated using the event callback function.

The following events are currently defined:

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

Table 13: PROFINET Device stack events

4.7 Multiple ARs

4.7.1 Ownership

The PROFINET Device stack supports multiple ARs at the same time. This allows you to use several controllers to access different output or input submodules of one device at the same time. In term of the PROFINET specification this is called **Shared Device** and **Shared Input**. Of course, a Shared Output is not defined as it makes no sense to access an output submodule from more than one controller at the same time. In order to solve the problem of which controller gets the right to write the outputs of a submodule and perform parameterization, the PROFINET specification V2.3 defines a new state machine, the Ownership-State machine (OWNSM). Depending on the submodule-specific Ownership, the following access rights are defined:

- **AR is the owner of the submodule**: The submodule puts its input process data to this AR, takes the output process data from this AR and accepts parameter read and writes on this AR.
- **AR is not the owner of the submodule**: The submodule ignores output process data from this AR and rejects parameter reads and writes on this AR.

The ownership itself is assigned according to the rule *“First Come First Serve”* e.g. the first AR will get the ownership of all requested submodules, the second AR will get the ownership of requested submodules not already owned by the first AR. If the first AR is disconnected, the second AR will get the ownership of all requested submodules not yet assigned to this AR (This is called a Release). There is only one exception from this rule: If a Supervisor AR connects, it takes over the ownership of all requested submodules if the owning ARs allow Ownership Takeover (Depends on AR Properties in Connect.req of the AR). Furthermore, if a Supervisor AR is active and was not allowed to take over the ownership of a submodule and the owning AR disconnects, the Supervisor AR will always get the ownership of the submodule (Supervisor AR has higher Priority than “First Come First Serve” Principle).

Note:
The multi AR feature renders the definition of a “communicating state” ad absurdum. The application usually has no information about which of the submodules is used by any AR. Therefore it is strongly recommended to cyclically update the process data from/to the physical submodules, regardless of any communication state. If the application insists on knowing if a submodule is in data exchange or not, the IOxS status of the submodule should be examined.

4.7.2 Possibilities and Limitations for the Feature Shared Device

The feature Shared Device is extended to up to 8 IO ARs for netX 51 and netX 100. This feature extension is not supported for netX 50.

Note:
When using PROFINET IRT it is not possible to use Shared Device for IRT.

The netX-based IO-Device only supports exactly 1 IRT IO AR. Additional RT IO ARs are possible.

Reachable Cycle Time Depending on the Amount of IO ARs

To give the user of the PROFINET IO Device protocol stack a better understanding of reachable cycle times for this feature, extensions of this chapter will give some performance indicators.

As the variety of combinations is very large obviously this chapter can not show every possible use case. Thus only a small subset of possibilities is shown here.

This does neither mean that a not shown combination is not supported nor that it is supported.

Configurations to be used have to be tested by the one combining the netX PROFINET IO Device with his application.

Note:
As each netX chip type has different calculation power this chapter differs the netX chip type.

For the following tables it is assumed that two 20 byte input submodules and two 20 byte output submodules are used per IO AR.

Please note, that the amount of submodules influences the reachable cycle time.

netX	Amount of ARs	Smalles possible cycle time
netX 100	1-2	1 ms
	3-5	2 ms
	6-8	4 ms
netX 51	1-2	1 ms
	3-4	2 ms
	5-8	4 ms

Table 14: Smalles possible Cycle Time depending using multiple ARs

GSDML, startup Parameterization and Certification

In any case to pass PROFINET certification it is required that the supported amount of ARs documented in GSDML file matches to the capabilities of the product. Thus the value in GSDML (“NumberOfAR”) needs to match the configured amount of ARs of the PROFINET protocol stack.

Loadable firmware offers a tag list entry to modify the amount of ARs supported.

Linkable Object Module offers a stack startup parameter (`ulMaxAr`).

© Hilscher 2011 – 2019

4.8 Second dual-port memory channel: Ethernet Interface

This section is valid for loadable firmware only.

Starting with version 3.8.0.0 of the PROFINET stack, loadable firmware for netX 51 and netX 100 offers a second dual-port memory channel (channel 1) for devices with a 64 KB large dual-port memory only. The PROFINET stack is accessible via dual-port memory channel 0 and the Ethernet Interface via channel 1.

The Ethernet interface stack provides an API for:

- Reusing the API of internal TCP/IP stack running with PROFINET
- Send/receive raw Ethernet frames using a dedicated MAC address (NDIS mode)

Depending on the PROFINET loadable firmware some of this APIs may not be available.

For more information about Ethernet Interface stack refer to reference QV!!!.

Note:
Loadable firmware for netX 51 shows the second DPM channel only, if the NDIS mode is activated.

Note:
Loadable firmware for netX 50 does not support NDIS mode!

Note:
First, the NDIS mode needs to be activated in loadable firmware using the **Tag List Editor** software.

Note:
The “cifX Device Driver Setup” delivers a driver for Windows that allows you to use the Ethernet interface stack running NDIS mode as usual Ethernet adapter and appears as "Virtual cifX Ethernet Adapter".

Note:
Starting with version 3.11.0, the OEM mode (Send/receive raw Ethernet frames to/from PROFINET interface MAC address) is no longer supported. The NDIS mode has replaced the OEM mode.

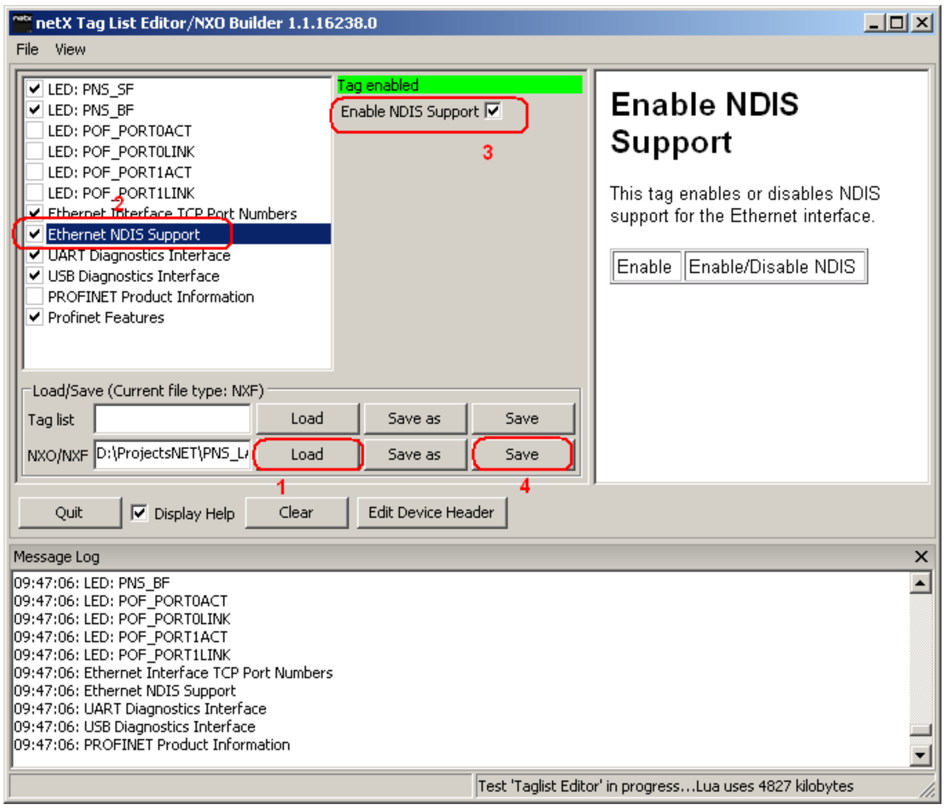
Activation of the NDIS mode

By default the **NDIS mode** for the Ethernet interface is deactivated in loadable firmware.

Note:
Using the “Ethernet Interface” in NDIS mode has a negative effect on netload environment! It may be a drawback in achieving of the desired network load class in the PROFINET IO-Device certification.

Use the Tag List Editor software to activate the NDIS mode.

1. Activate NDIS mode
   - ➢ Load firmware (1).
   - ➢ Go to the “Ethernet NDIS Support” tag (2).
   - ➢ Set check box “Enable NDIS Support” (3).
   - ➢ Save the loadable firmware with activated NDIS support (4).



```
09:47:06: LED: PNS_BF
09:47:06: LED: POF_PORT0ACT
09:47:06: LED: POF_PORT0LINK
09:47:06: LED: POF_PORT1ACT
09:47:06: LED: POF_PORT1LINK
09:47:06: Ethernet Interface TCP Port Numbers
09:47:06: Ethernet NDIS Support
09:47:06: UART Diagnostics Interface
09:47:06: USB Diagnostics Interface
09:47:06: PROFINET Product Information
```

⇨

4.9 Asset Management

Asset Management is a feature defined by PROFINET specification V2.34. It is used to provide information about all installed software and hardware versions within the PROFINET IO-Device or accessible via subsequent field busses / networks in case of a gateway or more complex device (e.g. robot unit with internal busses). Diagnostic tools will use this feature to show detailed version information about a device or unit. The functionality uses a globally defined record object to exchange the information.

The PROFINET IO-Device stack offers the *Get Asset service* [▸ page 172] to retrieve the asset information from the application. The stack will collect the data using multiple Get Asset services and generates a properly encoded read record response.

4.10 PROFIenergy ASE

The protocol stack has implemented the PROFIenergy ASE as defined by PROFINET specification. For that purpose, it maintains an internal database of PE entities. PE entities are added removed or updated in this database by the application using the corresponding packet services. The protocol stack generates all PROFINET Alarms associated with these operations internally. Futhermore the protocol stack implements the PE Filter Data and PE Status Data records using the information from the database. Read or Write access to the PESAP record object (Index 0x80A0) is filtered by the protocol stack using the database. If the database contains a PE entity associated with the accessed submodule, a regular Read/Write record service is used to hand over processing of the record object access to the application. If no PE entity is associated with the accessed submodule, the access is rejected by the protocol stack.

Note:
Applications that already have implemented the PROFIenergy profile without using the PE ASE services need to be modified in order to still support PROFIenergy. Without application modification the required PROFIenergy services will no longer be usable with V3.12.0.0.

5 Requirements to the application

5.1 What the application always has to do

The application must answer each indication packet as fast as possible.

The application must always update the I/O data.

You have to determine whether the stack or your application handles remanent data. For details, see section *Remanent data handling* [▸ page 40].

5.2 Device handle

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

- `hDeviceHandle = 0`: The request is associated with no AR (Only ReadImplicit) or the request is associated with a Supervisor DA AR.
- `hDeviceHandle != 0`: The request is associated with a Supervisor AR or an IO-AR. The Type of the AR is indicated to the application using the *AR Check service* [▸ page 102].

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

- *AR Check indication* [▸ page 102]
- *Check Indication* [▸ page 107]
- *Connect Request Done indication* [▸ page 111]
- *Parameter End indication* [▸ page 114]
- *AR InData indication* [▸ page 119]
- *Read Record indication* [▸ page 126]
- *Write Record indication* [▸ page 130]
- *AR Abort Indication* [▸ page 135]
- *APDU Status Changed indication* [▸ page 152]
- *Alarm Indication* [▸ page 154]
- *Release Request Indication* [▸ page 156]
- *Read I&M indication* [▸ page 163]
- *Write I&M indication* [▸ page 170]
- *Parameterization Speedup Support indication* [▸ page 179]

5.3 Remanent data handling

5.3.1 Remanent data

Remanent data contain device parameters e.g. Name of Station, IP Address Parameters, Ethernet-port related parameters, IRT related parameters, which an IO Device has to store permanently. The stack automatically updates the remanent data when requested via PROFINET by DCP Set NameOfStation and DPC SET IP.

When you design your application, you have to determine whether

- the **protocol stack** stores the remanent data or
- the **application** stores the remanent data.

According to your decision, the application has to configure the stack using the parameter `ulSystemFlags`: Remanent Data Handling bit (D14). This parameter is with the *Set Configuration service* [▷ page 53].

The setting of the parameter Device name and IP Parameters Handling (D17) of `ulSystemFlags` is also relevant for the behavior of the application, see section *Parameters 'Name of Station' and 'IP Address Parameters'* [▷ page 41].

5.3.1.1 Stack stores remanent data

Requirements

- a non-volatile memory has to be connected to the netX
- a Flash-based file system

Configuration

The application has to set parameter Remanent Data Handling (D14) bit `PNS_IF_SYSTEM_DISABLE_STORE_REMANENT_DISABLE` in ulSystemFlags to 0 using the *Set Configuration service* [▷ page 53].

5.3.1.2 Application stores remanent data

Requirement

The application has to use the *Load Remanent Data service* [▷ page 84] and to support the *Store Remanent Data service* [▷ page 121].

Configuration

The application has to send the Load Remanent Data service to the stack, before the application sends the Set Configuration service.

The application has to set parameter Remanent Data Handling (D14) `PNS_IF_SYSTEM_DISABLE_STORE_REMANENT_ENABLED` in ulSystemFlags to 1 using the *Set Configuration service* [▷ page 53].

During runtime

The stack indicates to the application the *Store Remanent Data service* [▸ page 121] and provides the remanent data as a block towards the application. The application has to store the remanent data with each indication.

5.3.2 Parameters ‘Name of Station’ and ‘IP Address Parameters’

The *Set Configuration service* [▸ page 53] includes the setting for the Name of Station and the IP Parameters. These parameters can change during runtime. The stack always indicates to the application a change of the Name of Station (section *Save Station Name service* [▸ page 137]) or a change of the IP Address (section *Save IP Address service* [▸ page 140]). The application has to store these parameters permanently and use them for the next *Set Configuration service* [▸ page 53]. For this purpose, the setting of parameter Device Name and IP Parameters Handling (D17) is relevant for the storing of parameters and for the behavior of the application.

- If set to 0 (`PNS_IF_SYSTEM_NAME_IP_HANDLING_BY_STACK_DISABLED`), the application has to store the parameters Name of Station and the IP Address parameters because they are used in the Set Configuration Service. This means that the application has to store these parameters permanently when the indications `PNS_IF_SAVE_STATION_NAME_IND` and/or `PNS_IF_SAVE_IP_ADDR_IND` are sent from the stack. The application has to use the stored values in the next Set Configuration Service. The application has to reset them on `PNS_IF_RESET_FACTORY_SETTINGS_IND` and use the reset values in the next Set Configuration Service.
- If set to 1 (`PNS_IF_SYSTEM_NAME_IP_HANDLING_BY_STACK_ENABLED`), the application does not need to store the parameters Name of Station and the IP Address parameters. The parameters for Name of Station and the IP Address parameters transferred with the Set Configuration Service are ignored and the values from the remanent data are used instead.
  Note: If Remanent Data Handling (D14) is set to 1 (`PNS_IF_SYSTEM_DISABLE_STORE_REMANENT_ENABLED`) and the application does not use the Load Remanent Service, the Name of Station is not set!

5.4 Isochronous Application

The aim of this chapter is to support the development of the isochronous PROFINET devices based on the netX chip with the current version of the stack.

The most important ability of the isochronous PROFINET device is to deliver the process data (latch inputs and set outputs) exactly to the configured (planned) time in nanosecond accuracy. These strict timing-requirements have to be solved by software structure: all the tasks in OS that are responsible for cyclic data exchange should have the highest priority.

In case of NXLOM see recommendations to the task priorities described above in section *Task Priorities* [▸ page 271].

In case of LFW, the application has the possibility to activate the DPM watchdog to supervise the protocol stack and vice versa (see section *Set Configuration service* [▸ page 53] parameter `ulWdgTime`). The DPM watchdog mechanism is implemented in rcX OS in context of the RX_TIMER task. **If a developed application will support isochronous mode the `RX_TIMER` task has to be necessarily switched to lower priority.**

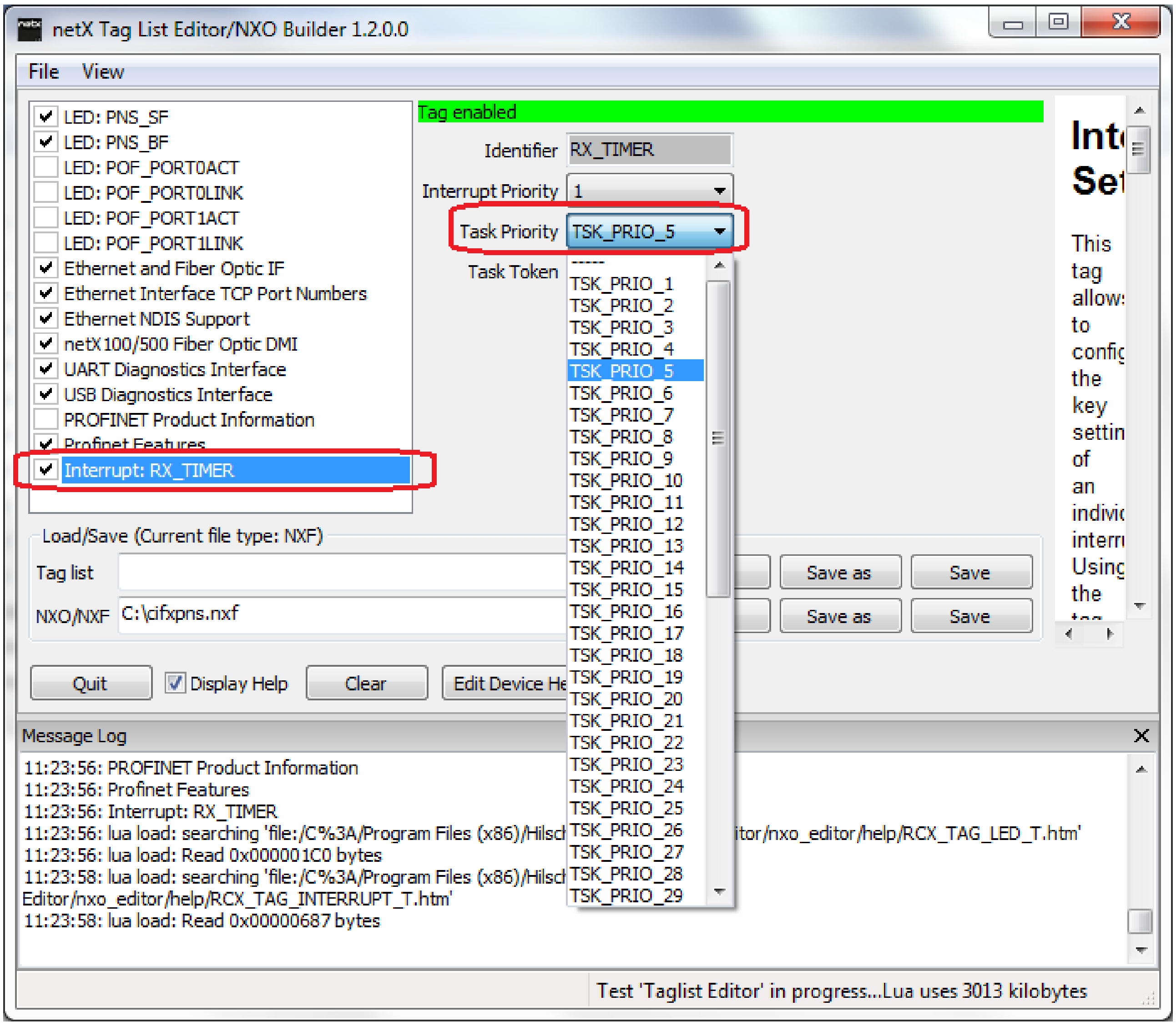
Attention, it has impact on watchdog functionality:

Note:
The DPM watchdog supervising can reach timeout in case of high network load with PROFINET cyclic (RTC) frames (case of network load tests in certification) without any failure of the PROFINET Stack or OS.

Do not activate the DPM watchdog in case of isochronous application!

In case of isochronous application use the “Tag List Editor” software to switch the `RX_TIMER` task to lower priority in the "Interrupt: RX_TIMER" tag (see figure below).

To limit the possible side-effects the priority of the RX_TIMER task will be set forcibly always to TSK_PRIO_5 (according to section *Task Priorities* [▸ page 271]) independently of the priority "TSK_PRIO_**" selection!



- [✓] LED: PNS_SF
- [✓] LED: PNS_BF
- [ ] LED: POF_PORT0ACT
- [ ] LED: POF_PORT0LINK
- [ ] LED: POF_PORT1ACT
- [ ] LED: POF_PORT1LINK
- [✓] Ethernet and Fiber Optic IF
- [✓] Ethernet Interface TCP Port Numbers
- [✓] Ethernet NDIS Support
- [✓] netX100/500 Fiber Optic DMI
- [✓] UART Diagnostics Interface
- [✓] USB Diagnostics Interface
- [ ] PROFINET Product Information
- [✓] Profinet Features
- [✓] Interrupt: RX_TIMER

Tag enabled

Identifier	RX_TIMER
Interrupt Priority	1
Task Priority	TSK_PRIO_5
Task Token	

-----, TSK_PRIO_1, TSK_PRIO_2, TSK_PRIO_3, TSK_PRIO_4, TSK_PRIO_5, TSK_PRIO_6, TSK_PRIO_7, TSK_PRIO_8, TSK_PRIO_9, TSK_PRIO_10, TSK_PRIO_11, TSK_PRIO_12, TSK_PRIO_13, TSK_PRIO_14, TSK_PRIO_15, TSK_PRIO_16, TSK_PRIO_17, TSK_PRIO_18, TSK_PRIO_19, TSK_PRIO_20, TSK_PRIO_21, TSK_PRIO_22, TSK_PRIO_23, TSK_PRIO_24, TSK_PRIO_25, TSK_PRIO_26, TSK_PRIO_27, TSK_PRIO_28, TSK_PRIO_29

Int
Se
This tag allow to confi the key settin of an indivi interr Using the

Load/Save (Current file type: NXF)

Tag list	
NXO/NXF	C:\cifxpns.nxf

[✓] Display Help

```
11:23:56: PROFINET Product Information
11:23:56: Profinet Features
11:23:56: Interrupt: RX_TIMER
11:23:56: lua load: searching 'file:/C%3A/Program Files (x86)/Hilscl                itor/nxo_editor/help/RCX_TAG_LED_T.htm'
11:23:56: lua load: Read 0x000001C0 bytes
11:23:58: lua load: searching 'file:/C%3A/Program Files (x86)/Hilscl
Editor/nxo_editor/help/RCX_TAG_INTERRUPT_T.htm'
11:23:58: lua load: Read 0x00000687 bytes
```

Test 'Taglist Editor' in progress...Lua uses 3013 kilobytes

Figure 3: Priority setup of the RX_TIMER task in the loadable PROFINET firmware for isochronous application

6 Application interface

6.1 Configuring the IO-Device stack

6.1.1 Service overview

The following table gives an overview about the available services provided by the stack:

Service	**Command code (REQ or IND)**	**Command code (CNF or RES)**
Set Configuration service [▸ page 53]	0x1FE2	0x1FE3
Register Application service [▸ page 62]	0x2F10	0x2F11
Unregister Application service [▸ page 64]	0x2F12	0x2F13
Set Port MAC Address service [▸ page 68]	0x1FE0	0x1FE1
Set OEM Parameters service [▸ page 70]	0x1FE8	0x1FE9
Load Remanent Data service [▸ page 84]	0x1FEC	0x1FED
Configuration Delete service [▸ page 86]	0x2F14	0x2F15
Set IOXS Config service [▸ page 90]	0x1FF2	0x1FF3
Register Fatal Error Callback service [▸ page 64]	0x1FDA	0x1FDB
Unregister Fatal Error Callback service [▸ page 67]	0x1FDE	0x1FDF
Set IO Image service [▸ page 87]	0x1FF0	0x1FF1
Configure Signal service [▸ page 92]	0x6100	0x6101

Table 15: Overview: Configuring the IO-Device stack service

6.1.2 Cyclic process data image

The PROFINET IO protocol uses a cyclic process data model to exchange process data between the PROFINET IO Controller and Device. That is, the PROFINET IO Controller's application and the PROFINET IO Device's application periodically update their own copies of the process data. The protocol stacks periodically synchronize these two images vice versa.

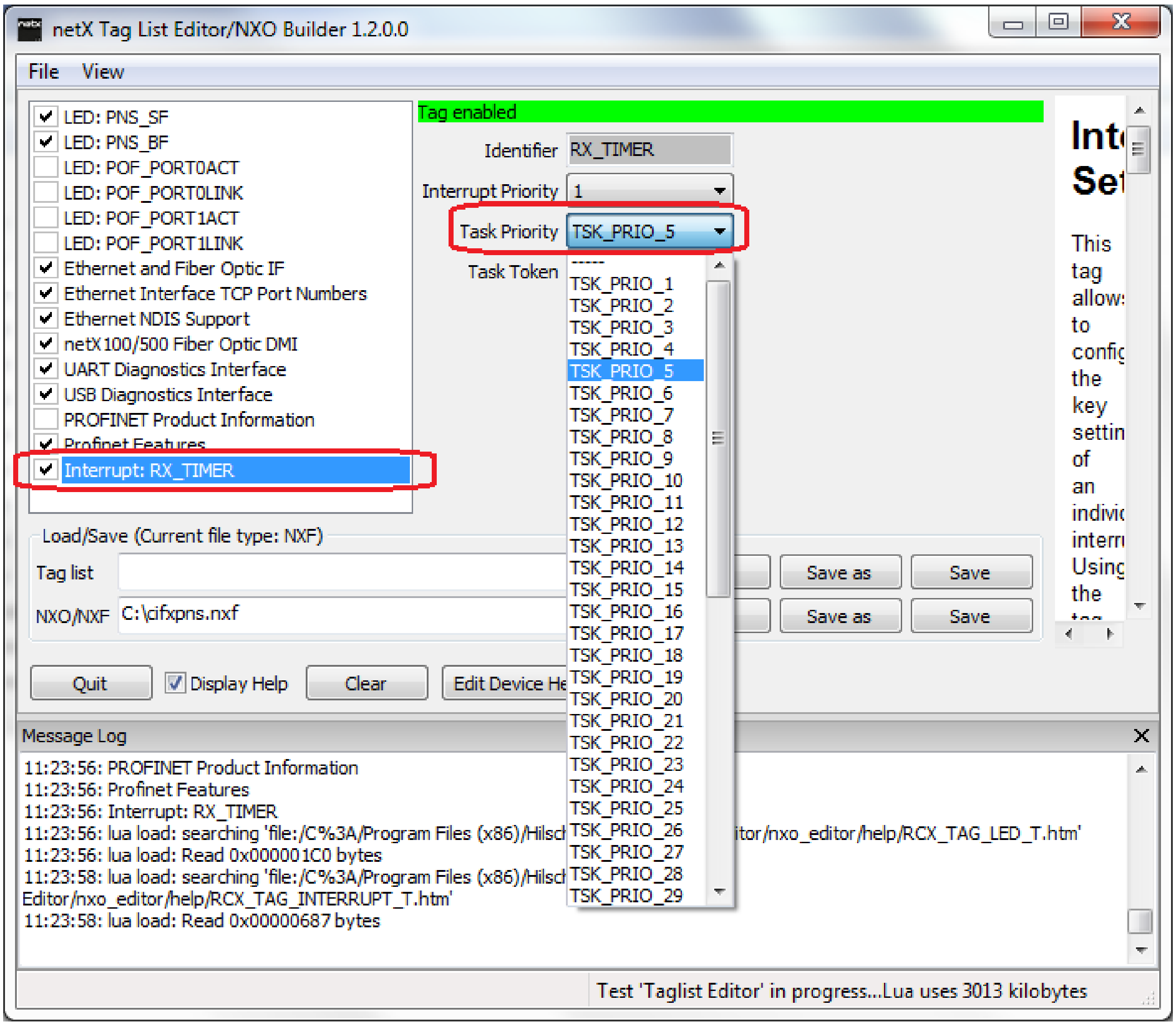
In PROFINET, process data is organized at the level of submodules. Each submodule can be assigned input and/or output process data. Submodules without any data are assigned to the input process data block with a zero length. Each process data block is associated with a provider data status (IOPS) and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

Note:
In order to simplify the host application development, the default configuration of the Hilscher PROFINET Device stack uses an operation mode where it assumes valid data whenever the DPM Output Area/Provider Data Area is updated. If explicit provider and/or consumer status processing is required, the application developer has to configure the stack using the *Configure IOXS Service* [page 90].

6.1.3 Configuration of process data images

From the PROFINET IO Device viewpoint, the Provider process data Image is the data sent from the PROFINET IO Device to the PROFINET IO Controller. This is usually called *Input process data* (Inputs). The application has to place this data in the Provider process data Image. If a loadable firmware or module is used, the Provider Data Image corresponds to the DPM Output Area.

For linkable objects, the application defines the Provider process data Image as a gapless memory block. It has to be provided to the protocol stack.



netX Tag List Editor/NXO Builder 1.2.0.0
File
View
Int
Set
This tag allows to config the key settin of an individ interr Using the
Quit
Display Help
Clear
Edit Device He
Save as
Save
Message Log

☑ LED: PNS_SF
☑ LED: PNS_BF
☐ LED: POF_PORT0ACT
☐ LED: POF_PORT0LINK
☐ LED: POF_PORT1ACT
☐ LED: POF_PORT1LINK
☑ Ethernet and Fiber Optic IF
☑ Ethernet Interface TCP Port Numbers
☑ Ethernet NDIS Support
☑ netX100/500 Fiber Optic DMI
☑ UART Diagnostics Interface
☑ USB Diagnostics Interface
☐ PROFINET Product Information
☑ Profinet Features
☑ Interrupt: RX_TIMER

Tag enabled

Identifier: RX_TIMER
Interrupt Priority: 1
Task Priority: TSK_PRIO_5
Task Token

------, TSK_PRIO_1, TSK_PRIO_2, TSK_PRIO_3, TSK_PRIO_4, TSK_PRIO_5, TSK_PRIO_6, TSK_PRIO_7, TSK_PRIO_8, TSK_PRIO_9, TSK_PRIO_10, TSK_PRIO_11, TSK_PRIO_12, TSK_PRIO_13, TSK_PRIO_14, TSK_PRIO_15, TSK_PRIO_16, TSK_PRIO_17, TSK_PRIO_18, TSK_PRIO_19, TSK_PRIO_20, TSK_PRIO_21, TSK_PRIO_22, TSK_PRIO_23, TSK_PRIO_24, TSK_PRIO_25, TSK_PRIO_26, TSK_PRIO_27, TSK_PRIO_28, TSK_PRIO_29

Load/Save (Current file type: NXF)
Tag list:
NXO/NXF: C:\cifxpns.nxf

```
11:23:56: PROFINET Product Information
11:23:56: Profinet Features
11:23:56: Interrupt: RX_TIMER
11:23:56: lua load: searching 'file:/C%3A/Program Files (x86)/Hilscl TSK_PRIO_26 itor/nxo_editor/help/RCX_TAG_LED_T.htm'
11:23:56: lua load: Read 0x000001C0 bytes
11:23:58: lua load: searching 'file:/C%3A/Program Files (x86)/Hilscl
Editor/nxo_editor/help/RCX_TAG_INTERRUPT_T.htm'
11:23:58: lua load: Read 0x00000687 bytes
```

Test 'Taglist Editor' in progress...Lua uses 3013 kilobytes

Figure 4: Priority setup of the RX_TIMER task in the loadable PROFINET firmware for isochronous application

Note:
Remind that process data sent to the PROFINET IO Controller, which is typically called *Inputs*, is to be placed within the DPM output area. This naming issue might lead to confusion.

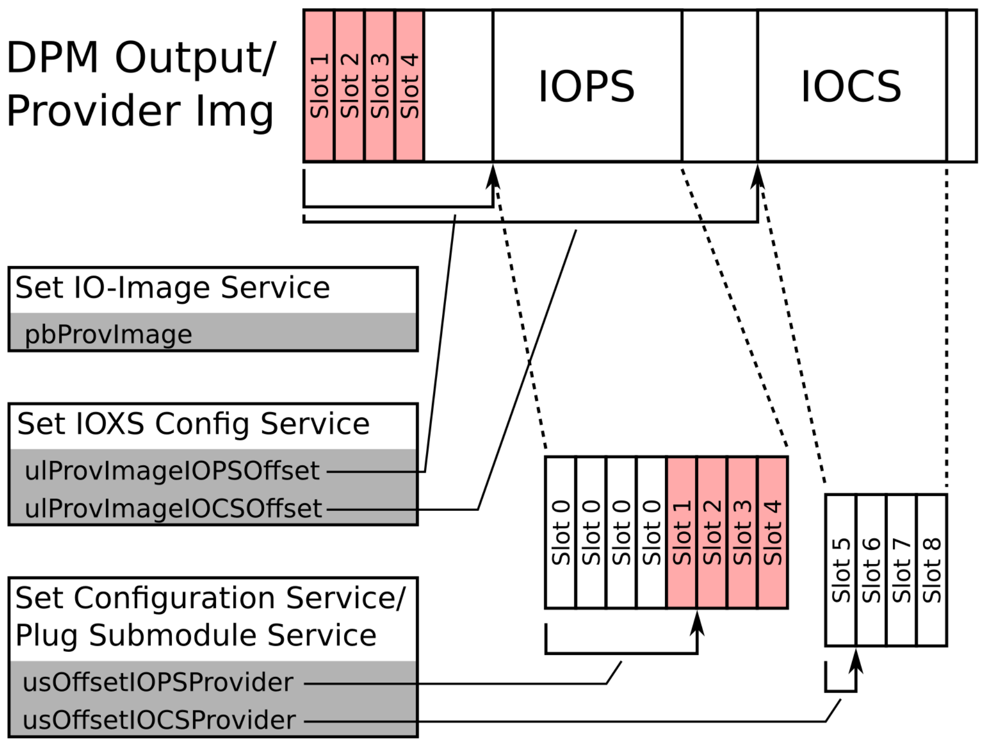


Figure 5: Provider process data structure

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

The process data image consists of up to three blocks:

1. the process data itself,
2. a status area for provider data
3. a status area for consumer data.

Figure *Provider process data structure* [▸ page 47] above illustrates this.

In contrast to that, the consumer process data image is the data sent from the PROFINET IO Controller to the PROFINET IO Device. (If looking from PROFINET IO Device Viewpoint). This data is typically called *Output process data* (Outputs). The application should take this data from the consumer process data image. If a loadable firmware or module is used, the consumer process data image corresponds to the DPM Input Area.

For linkable objects, the application defines the consumer process data image as a gapless memory block. It has to be provided to the protocol stack.

Note:
Remind that process data received from the PROFINET Controller, which is typically called *Outputs* is to be placed within the DPM input area. This naming issue might lead to confusion.

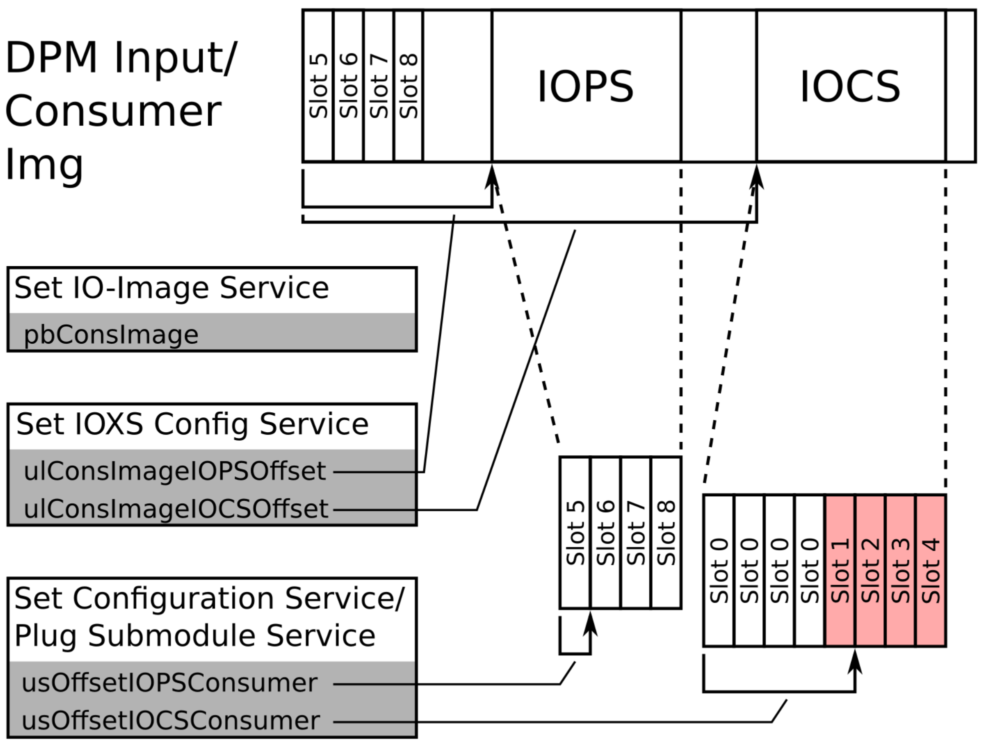


Figure 6: Consumer process data structure

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

Note:
If the host application uses consumer data states, it is important to understand that the consumer data state of a submodule is to be placed in the opposite process data image compared to the process data itself. This is because the consumer data states act as a kind of confirmation for the sender of the data by the receiver.

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

1. If a linkable object is used, the first step is to provide the stack with the pointer to the memory blocks for the process data by means of the *Set IO Image* service. **The application must define memory areas large enough to hold the data.** This step is not required if dual-port memory or shared memory API is used. In that case, the process data image of the provider corresponds to the DPM output area and the process data image of the consumer corresponds to the DPM input area.
2. This step is optional and only required if the process data states of the provider and/or consumer are prepared by the host application. This step enables the IOPS and/or IOCS blocks, which are disabled by default. If these blocks are used, **the host application must configure the start offsets of them.** These offsets are specified relative to the beginning of the corresponding process data image. For this purpose, the application can use *Set IOXS Config service.* These offsets are specified in units of bytes.

3. This step of configuring the process data images is to specify the location of the process data and the data states within their corresponding areas. These parameters are part of the *Set Configuration service* and/or *Plug Submodule service.* Always specify the offset of the process data in bytes relative to the beginning of the corresponding process data memory. The offsets of the data states are always relative to the beginning of the corresponding data state block. These offsets are either in units of bytes or in units of bits. This depends on the mode configured in the step before.

6.1.4 Configuration of the submodules

Evaluation of this option only takes place if the IOXS Configuration Service has already been performed before. Otherwise, the stack will ignore the configuration values. If IOXS is configured, it is mandatory to configure correct offsets for each submodule.

The configuration is done by the Set Configuration request for each (see section *Set Configuration Request* [▸ page 54]).

6.1.5 Configuring the PROFINET IO-Device stack

The stack requires configuration parameters. Configuration parameters can be set

- using a configuration software (the configuration software creates a database containing the configuration parameters) or
- by an application using the packet API.

The application configures the stack using the API

In order to configure an IO-Device, the application has to perform the following configuration sequence:

- Configure the IO image with the *Set IOImage* service if dual-port-memory or shared memory interface is not used.
- If necessary, assign a MAC address to the device as described in the netX dual-port-memory manual. Additionally, PROFINET requires assigning of port MAC addresses. It is necessary to assign a MAC address if no local MAC address exists (e.g. no security memory).
- Restore the remanent data using the *Load Remanent Data* service if the application handles remanent data (instead of the stack).
- Set the OEM parameters using the *Set OEM Parameters* service if the Hilscher default values are insufficient.
- Register the application using the *Register Application* service in order to receive indications from the stack if this is required.
- Configure the IOxS handling with the *Set IOXS Config* service if IOxS access is required.
- **Mandatory:** Configure the device using the *Set Configuration* service. This means supplying the device with all parameters needed for operation. These include both basic parameters for identification such as *NameOfStation*, *DeviceID* and *VendorID* as well as the module configuration. This module configuration contains information about the APIs, modules and submodules the stack will use. When the stack returns the Set_Configuration packet back to the application, the given configuration has been evaluated completely and is ready to be applied.
- **Mandatory:** Perform the Channel Initialization (see reference [5]) to activate the new configuration. Now, the stack is ready to start communication with an IO-Controller.

The following figure shows the configuration sequence.

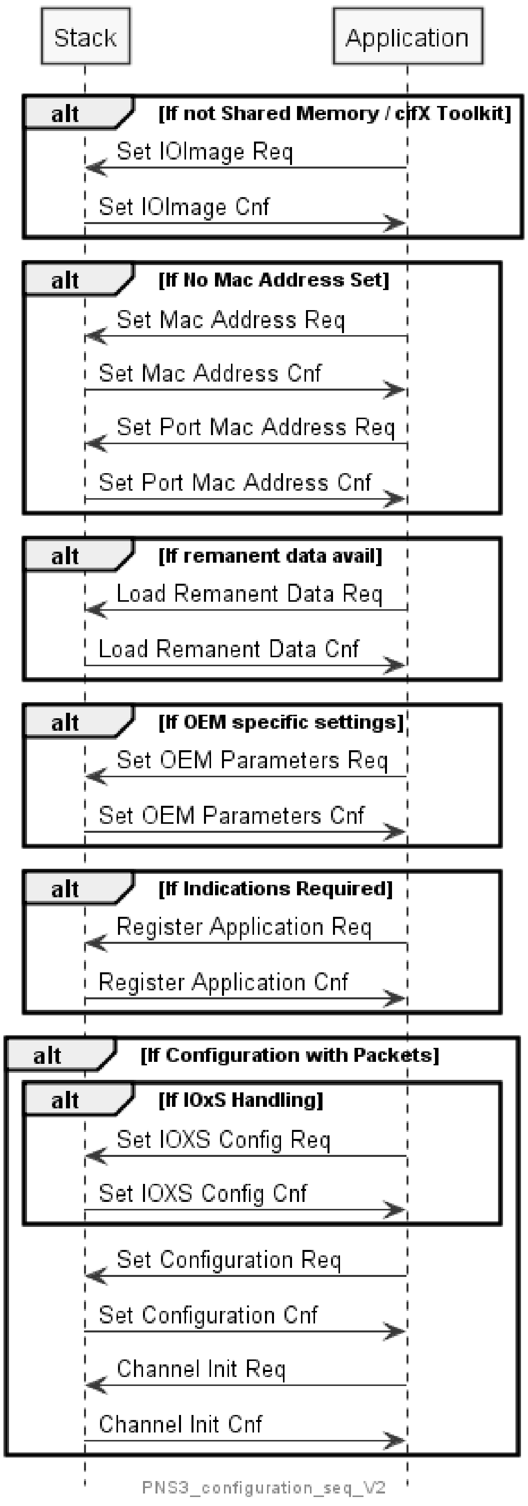


Figure 7: IO-Device configuration sequence

6.1.5.1 Remark on Reconfiguration

It is possible to reconfigure the stack at any time. To do so, simply send a new configuration to the stack followed by a Channel Init Request (see reference [5]). Sending the new configuration without the Channel Init Request will not have an effect on any running communication. The new parameters will simply be stored. Sending the Channel Init Request will stop any communication and take over the new parameters.

6.1.6 Set Configuration service

Set configuration

The application can use the Set Configuration service to configure the IO-Device stack on startup. The application provides information about the API, the modules and the submodules to the stack.

Note:
After the application has sent the Set Configuration request to the stack, the application must send a Channel Initialization request (ChannelInit) to the protocol stack in order to confirm the validity of the new configuration. The stack will use the changed configuration only after it has received the Channel Initialization request.

Note:
In case `PNS_IF_SYSTEM_NAME_IP_HANDLING_BY_STACK_ENABLED` is set to 0: The application must register with the stack in order to receive the relevant indications (See section *Register Application service* [▸ page 62]). The application must handle the Name Of Station and the IP Parameters.

Changing the configuration at runtime

The module configuration may be changed later at runtime using the

- Pull services (see sections *Pull Module service* [▸ page 211] and *Pull Submodule service* [▸ page 214]), and
- Plug services (see sections *Plug Module service* [▸ page 200] and *Plug Submodule service* [▸ page 202]).

© Hilscher 2011 – 2019

6.1.6.1 Set Configuration request

The application has to send this request to the protocol stack. As the application can (almost) freely define the configuration of modules and submodules. Due to the number of used modules and submodules, this packet has a variable length.

If the stack is accessed via Dual Port Memory, due to the limited mailbox size the application may need to use packet fragmentation for the request packet. The handling of packet fragmentation is described in section “General packet fragmentation” in reference [5].

```
PNS_IF_SET_CONFIGURATION_REQ_T
```

`tDeviceParameters` has the following structure:

```
typedef __HIL_PACKED_PRE struct PNS_IF_DEVICE_PARAMETER_Ttag
{
  uint32_t ulSystemFlags; /** flags to use are defined in this file
*/
  uint32_t ulWdgTime; /** DPM watchdog time*/
  uint32_t ulVendorId; /** the Vendor ID */
  uint32_t ulDeviceId; /** the Device ID */
  uint32_t ulMaxAr; /** currently unused */
  uint32_t ulCompleteInputSize; /** max. combined amount of Input
bytes */
  uint32_t ulCompleteOutputSize; /** max. combined amount of Output
bytes */
  uint32_t ulNameOfStationLen; /** length of NameOfStation */
  uint8_t abNameOfStation[PNS_IF_MAX_NAME_OF_STATION]; /** the
NameOfStation */
  uint32_t ulTypeOfStationLen; /** length of TypeOfStation */
  uint8_t abTypeOfStation[PNS_IF_MAX_TYPE_OF_STATION]; /** the
TypeOfStation */
  uint8_t abDeviceType[PNS_IF_MAX_DEVICE_TYPE_LEN +3]; /** the
DeviceType - ignore the last 3 padding bytes ! */
  uint8_t abOrderId[PNS_IF_MAX_ORDER_ID]; /** the OrderID (pad with
spaces (0x20) */

  uint32_t ulIpAddr; /** IP Address, default: 0.0.0.0 */
  uint32_t ulNetMask; /** Netmask, default: 0.0.0.0 */
  uint32_t ulGateway; /** Gateway, default: 0.0.0.0 */
  uint16_t usHwRevision; /** Hardware Revision, default: 0 */
  uint16_t usSwRevision1; /** Software Revision 1, default: 0 */
  uint16_t usSwRevision2; /** Software Revision 2, default: 0 */
  uint16_t usSwRevision3; /** Software Revision 3, default: 0 */
  uint8_t bSwRevisionPrefix; /** Software Prefix, default: 0 */
  uint8_t bReserved; /** reserved, shall be set to 0 */
  uint16_t usMaxDiagRecords; /** max. amaount of parallel Diagnosis
records, default: 256 */
  uint16_t usInstanceId; /** GSDML-parameter ObjectUUID_LocalIndex,
default: 1 */
  uint16_t usReserved; /** reserved, shall be set to 0 */
} __HIL_PACKED_POST PNS_IF_DEVICE_PARAMETER_T;
```

Variable	Type	Value / Range	Description
ulSystemFlags	uint32_t		Flags for system behavior. See below.
ulWdgTime	uint32_t	Default value: 1000 Allowed values: 0, 20…65535	Watchdog time (specified in milliseconds). 0 = Watchdog timer is switched off
ulVendorId	uint32_t	1..65279 (= 0xFEFF)*, Hilscher: 286 (decimal)/ 0x011E (hex)	Vendor ID: Vendor identification number of the manufacturer, which the PROFIBUS Nutzerorganisation e. V. has assigned to the vendor. All Hilscher products use the value 0x011E.
ulDeviceId	uint32_t	1 … (2^{16} - 1)*, e.g. for cifX 50-RE:: 259 (decimal)/ 0x103 (hex)	Device ID This is an identification number of the device, freely eligible by the manufacturer, which is fixed and unique for every device.
ulMaxAr	uint32_t	0	Currently not used. Set to zero.
ulCompleteInputSize	uint32_t	Default value:128 Allowed values: 0 .. 1440 bytes	Maximum amount of allowed input data. The sum of data of all submodules configured by the user must not exceed this value. This field references input data as data received by the IO-Device.
ulCompleteOutputSize	uint32_t	Default value:128 Allowed values: 0 .. 1440 bytes	Maximum amount of allowed output data. The sum of data of all submodules configured by the user must not exceed this value. This field references output data as data sent by the IO-Device.
ulNameOfStationLen	uint32_t	0..240	Length of NameOfStation
abNameOfStation[240]	uint8_t[]		The NameOfStation as ASCII char-array. If bit D17 in the system flags ulSystemFlags is set the stack does not evaluate this parameter, the stack will use the NameOfStation saved in remanent data. For details about allowed characters, see section *Name encoding* [▹ page 323].
ulTypeOfStationLen	uint32_t	1..240	Length of TypeOfStation
abTypeOfStation[240]	uint8_t[]		The TypeOfStation as ASCII char-array.
abDeviceType[28]	uint8_t[]		The DeviceType as ASCII char-array. The last 3 bytes are reserved padding bytes and shall be set to zero.
abOrderId[20]	uint8_t[]		The OrderID as ASCII char-array.
ulIpAddr	uint32_t	Valid IP address, default: 0.0.0.0	IP address. If bit D17 in the system flags ulSystemFlags is set, the stack does not evaluate this parameter. The stack will use the IP address saved in remanent data.
ulNetMask	uint32_t	Valid network mask, default: 0.0.0.0	Network mask. If bit D17 in the system flags ulSystemFlags is set, the stack does not evaluate this parameter. The stack will use the network mask saved in remanent data.
ulGateway	uint32_t	Valid gateway address, default: 0.0.0.0	Gateway address. If bit D17 in the system flags ulSystemFlags is set, the stack does not evaluate this parameter. The stack will use the gateway address saved in remanent data.
usHwRevision	uint16_t	0..0xFFFF, default: 0	Hardware Revision, used e.g. in I & M

Variable	Type	Value / Range	Description
usSwRevision1	uint16_t	0..0xFFFF, default: 0	Software Revision 1, used e.g. in I & M
usSwRevision2	uint16_t	0..0xFFFF, default: 0	Software Revision 2
usSwRevision3	uint16_t	0..0xFFFF, default: 0	Software Revision 3
bSwRevisionPrefix	uint8_t	‘V’, ‘R’, ‘P’, ‘U’, ‘T’	Software Revision Prefix, used e.g. in I & M Possible values and their meanings are: ‘V’: Released version ‘R’: Revision ‘P’: Prototype ‘U’: Under field test ‘T’: Test device
bReserved	uint8_t	0	Reserved, set to zero.
usMaxDiagRecords	uint16_t	0	Set to 0. (The maximum number of diagnosis records can be set via tag list.)
usInstanceId	uint16_t	0..0xFFFF, default: 0	Instance ID. This parameter must match to value `ObjectUUID_LocalIndex` in the GSDML file corresponding to the IO-Device. The value 1 is recommended.
usReserved	uint16_t	0	Reserved for future use, set to zero

Table 16: Structure tDeviceParameters

System flags

Bit No	Behavior to influence	Flag Define/ Desired Behavior
D0	Bus State after applying configuration	0 - PNS_IF_SYSTEM_START_AUTO_START The stack will enable the Bus (Network access) right after Channel Initialization.
		1 - PNS_IF_SYSTEM_START_APPL_CONTROLLED The stack disables the Bus after Channel Initialization. Application shall use Start/Stop Communication Service to enable Bus.
D3	Byte order of Process Data	0 - PNS_IF_SYSTEM_BYTEORDER_BIG_ENDIAN The stack uses big endian byte order for process data.
		1 - PNS_IF_SYSTEM_BYTEORDER_LITTLE_ENDIAN The stack uses little endian byte order for process data. **Attention:** Using this mode causes a lot of CPU usage and may have a bad impact on minimum reachable cycle time.
D8	Identification & Maintenance Handling	0 - PNS_IF_SYSTEM_STACK_HANDLE_I_M_DISABLED I&M requests are only parsed by the stack and will be forwarded to the application using Indication and *Write I&M service* [▸ page 169]. The application must handle I&M 0-4 Data sets.
		1 - PNS_IF_SYSTEM_STACK_HANDLE_I_M_ENABLED The stack internally handles I&M requests. I&M 0-4 is supported on the DAP submodule (Slot 0, subslot 1).
D9	Automatic Application Ready if no application is registered	0 - PNS_IF_SYSTEM_ARDY_WOUT_APPL_REG_DISABLED The Stack will never generate an Application Ready Sequence if no application is registered: Therefore the application must register with the stack in order to establish an AR.
		1 - PNS_IF_SYSTEM_ARDY_WOUT_APPL_REG_ENABLED The stack will generate an Application Ready automatically if no application is registered. Use this flag with care: the stack has no information about the Readiness of the application. **Therefore sending Application Ready automatically can be dangerous**.

Bit No	Behavior to influence	Flag Define/ Desired Behavior
D11	Generate Check Indications for matching submodules	0 - PNS_IF_SYSTEM_CHECK_IND_ALL_MODULES_DISABLED The stack will not generate a *Check Indication service* [▸ page 105] for expected submodules that match the configuration.
		1 - PNS_IF_SYSTEM_CHECK_IND_ALL_MODULES_ENABLED The stack will generate a *Check Indication service* [▸ page 105] for expected submodules that match the configuration. E.g. the application will receive a *Check Indication service* [▸ page 105] for each submodule owned by an AR.
D12	Reserved	
D13	Generate Check Indications for unused modules	0 - PNS_IF_SYSTEM_CHECK_IND_UNUSED_MODULES_DISABLED The stack does not generate a*Check Indication service* [▸ page 105] for modules not used by any AR.
		1 - PNS_IF_SYSTEM_CHECK_IND_UNUSED_MODULES_ENABLED The stack will generate a *Check Indication service* [▸ page 105] for each submodule not used by any AR right before generating the *Connect Request Done service* [▸ page 111]. If another controller starts establishing an AR while these Check Indications are generated, the stack will stop generating these Check Indications and restart after processing another AR expected submodules. The stack generates *Check Indication service* [▸ page 105] also after *AR Abort Indication service* [▸ page 134].
D14	Remanent Data Handling	0 - PNS_IF_SYSTEM_DISABLE_STORE_REMANENT_DISABLE The stack handles the storing of remanent data. This requires using of DPM/SHM and non-volatile storage (Flash memory) is available.
		1 - PNS_IF_SYSTEM_DISABLE_STORE_REMANENT_ENABLED The application handles the storing of remanent data.
D15	Reserved	
D16	Check if IO-data offsets in IO-image	0 - PNS_IF_SYSTEM_ENABLE_IO_OFFSET_CHECKING The stack will check if IO-data offsets in IO-image (or DPM) are configured properly
		1 - PNS_IF_SYSTEM_DISABLE_IO_OFFSET_CHECKING The stack will never check if IO-data offsets in IO-image (or DPM) are configured properly. **ATTENTION: disabling this check may lead to inconsistent IO-data in case of a faulty application configuration. Disabling this check will lead to invalid stack response to the service RCX_GET_DPM_IO_INFO_REQ!**
D17	Device name and IP parameter handling	0 - PNS_IF_SYSTEM_NAME_IP_HANDLING_BY_STACK_DISABLED The parameters for Name of station and the IP address parameters are used from the Set Configuration Service. This means, that the application has to store these parameters, when the indications PNS_IF_SAVE_STATION_NAME_IND and/or PNS_IF_SAVE_IP_ADDR_IND are sent from the stack. The application has to use the stored values and to use them in the next Set Configuration service. The application has to reset them on PNS_IF_RESET_FACTORY_SETTINGS_IND and use the reset values in the next Set Configuration service.
		1 - PNS_IF_SYSTEM_NAME_IP_HANDLING_BY_STACK_ENABLED The parameter for Name of station and the IP address parameters transferred with the Set Configuration Service are ignored and the values from the remanent data are used instead. The stack must have a possibility to save the Remanent Data: either DPM/ SHM is in use and non-volatile storage (flash memory) is available or application (using the linkable object module) supports *Store Remanent Data service* [▸ page 121] and *Load Remanent Data service* [▸ page 84].

Table 17: System flags

`tModuleConfig` has the following structure:

```
typedef __HIL_PACKED_PRE struct PNS_IF_MODULE_CFG_REQ_DATA_Ttag
{
  /** Number of APIs following this structure */
  uint32_t ulNumApi;
} __HIL_PACKED_POST PNS_IF_MODULE_CFG_REQ_DATA_T;
```

As first element, it contains the number of API elements of the `tModuleConfig` structure. This is stored in variable `ulNumApi`. If you want to configure the device for using m APIs, you set `ulNumApi` to the value m.

For every additional API, a structure describing it and its submodules follows the last submodule structure of the preceding API. Assume this API should be configured for using n submodules.

Variable	Type	Value / Range	Description
ulApi	uint32_t	0..m-1	The number of the API profile to be configured. 0 indicates “manufacturer specific”. Currently only one single API is supported, so only the value 0 makes sense.
ulNumSubmoduleItems	uint32_t	1..n	Number of submodule-items this API contains. These items follow directly behind this entry.

Table 18: Structure PNS_IF_API_STRUCT_T

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

Variable	Type	Value / Range	Description
usSlot	uint16_t		The slot this submodule belongs to.
usSubslot	uint16_t		The subslot this submodule belongs to.
ulModuleID	uint32_t		The ModuleID of the module this submodule belongs to.
ulSubmoduleID	uint32_t		The SubmoduleID of this submodule.
ulProvDataLen	uint32_t	0..1440	The length of data provided by this submodule. This length describes the data sent by IO-Device and received by IO-Controller.
ulConsDataLen	uint32_t	0..1440	The length of data consumed by this submodule. This length describes the data sent by IO-Controller and received by IO-Device.
ulDPMOffsetIn	uint32_t		Offset in DPM input area or in Input image (in `pbConsImage`, see *Set IO Image service* [▸ page 87]) where consumed data for the submodule are copied to. This data is received by IO-Device and sent by IO-Controller. If the length of data in this direction is 0 set this value to 0. See section *Configuration of the submodules* [▸ page 49] for further information
ulDPMOffsetOut	uint32_t		Offset in DPM output area or in Output image (in `pbProvImage`, see *Set IO Image service* [▸ page 87]) where provided data of the submodule are taken from. This data is sent by the IO-Device and received by the IO-Controller. If the length of the data in this direction is 0, set this value to 0. See section *Configuration of the submodules* [▸ page 49] for further information
usOffsetIOPSProvider	uint16_t		Offset of the IO provider state (provider) for this submodule. The offset is relative to the beginning of the IOPS block in DPM output area. See section *Configuration of the submodules* [▸ page 49] for further information. **Note:** If the IOPS mode is set to “bitwise mode”, the application has to ensure that the offset addresses do not overlap. The stack will not check them for overlapping in this mode.
usOffsetIOPSConsumer	uint16_t		Offset of the IO provider state (consumer) for this submodule. The offset is relative to the beginning of the IOPS block in the DPM input area. See section *Configuration of the submodules* [▸ page 49] for further information. **Note:** If the IOPS mode is set to “bitwise mode”, the application has to ensure that the offset addresses do not overlap. The stack will not check them for overlapping in this mode.
usOffsetIOCSProvider	uint16_t		Offset of the IO consumer state (provider) for this submodule. The offset is relative to the beginning of IOCS block in DPM output area. See section *Configuration of the submodules* [▸ page 49] for further information. **Note:** If the IOCS mode is set to “bitwise mode”, the application has to ensure that the offset addresses do not overlap. The stack will not check them for overlapping in this mode.
usOffsetIOCSConsumer	uint16_t		Offset of the IO consumer state (consumer) for this submodule relative to the beginning of the IOCS block in DPM input area. See section *Configuration of the submodules* [▸ page 49] for further information. **Note:** If the IOCS mode is set to “bitwise mode”, the application has to ensure that the offset addresses do not overlap. The stack will not check them for overlapping in this mode.
ulReserved	uint32_t	0	Reserved for future use. Set to zero.

Table 19: Structure PNS_IF_SUBMODULE_STRUCT_T

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

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

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t		This is a value depending on the amount of submodules.
ulCmd	uint32_t	0x1FE2	PNS_IF_SET_CONFIGURATION_REQ
Data			
ulExt	uint32_t		Sequence number for use in fragmented packets
ulTotalConfigPckLen	uint32_t		Length (in bytes) of the entire configuration data. If the size of the whole Set_Configuration Request exceeds the DPM mailbox size sequenced mechanisms have to be used. This parameter in the very first packet shall contain the complete size of all sequenced packets (without the packet headers).
tDeviceParameters	PNS_IF_DEVICE_PARAMETER_T		The structure describing the device parameters, see explanation below.
tModuleConfig	PNS_IF_MODULE_CFG_REQ_DATA_T		The structure describing APIs and submodules, see explanation below.
tSignalConfig	PNS_IF_SIGNAL_CFG_REQ_DATA_T		This optional structure is needed if little endian byte order shall be used. It describes the data structure of each submodule

Table 20: PNS_IF_SET_CONFIGURATION_REQ_T - Set Configuration request

6.1.6.2 Set Configuration confirmation

The PROFINET IO-Device protocol stack will return this confirmation to the application.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulState	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1FE3	PNS_IF_SET_CONFIGURATION_CNF

Table 21: PNS_IF_SET_CONFIGURATION_CNF_T - Set Configuration confirmation

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_SET_CONFIGURATION_REQUEST_CNF_T;
```

© Hilscher 2011 – 2019

6.1.6.3 Behavior when receiving a Set Configuration Command

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

The configuration data are checked for consistency and integrity.

In case of failure, no data are accepted.

In case of success, the configuration parameters are stored internally (within the RAM).

Note:
The new configuration is not processed until a channel init is performed!

No automatic registration of the application at the stack happens.

The confirmation packet `PNS_IF_SET_CONFIGURATION_CNF` only transfers simple status information, but does not repeat the whole parameter set.

6.1.7 Register Application service

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

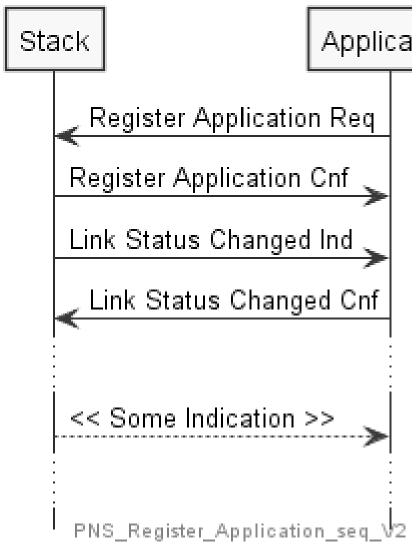


Figure 8: Register Application service packet sequence

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

- *Save Station Name indication* [▷ page 138]
- *Save IP Address indication* [▷ page 141]
- *Reset Factory Settings indication* [▷ page 149]
- *AR Check indication* [▷ page 102]
- *Check Indication* [▷ page 107]
- *Connect Request Done indication* [▷ page 111]
- *Parameterization Speedup Support indication* [▷ page 179]
- *Parameter End indication* [▷ page 114]
- *Store Remanent Data indication* [▷ page 122]
- *AR InData indication* [▷ page 119]
- *APDU Status Changed indication* [▷ page 152]
- *Read Record indication* [▷ page 126]
- *Write Record indication* [▷ page 130]
- *Read I&M indication* [▷ page 163]
- *Write I&M indication* [▷ page 170]
- *Alarm Indication* [▷ page 154]
- *AR Abort Indication* [▷ page 135]
- *Release Request Indication* [▷ page 156]
- *Start LED Blinking indication* [▷ page 143]
- *Stop LED Blinking indication* [▷ page 145]

- *Link Status Changed indication* [▸ page 158]
- *Error Indication* [▸ page 161]
- *Event Indication* [▸ page 180]

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

For a description of this service, see reference [5].

After Register Application, the stack will send a *Link Status Changed indication* [▸ page 158] to the application (see figure *Register Application service packet sequence* [▸ page 62]).

6.1.7.1 Register Application for selective indications only

With the “selective indications only” function. it is possible for an application to handle only the indications the application is interested in. For all other indications, it is possible to implement a default handling in the application. In that case, the stack behaves in the same way as “if no application would be registered”.

To activate this service there is no special handling required. Just the regular *Register Application service* [▸ page 62] is used.

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

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

To meet the conditions of PROFINET certification, the application has to fulfill conventions listes in the following table.

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

Table 22: Conventions concerning the application

© Hilscher 2011 – 2019

6.1.8 Unregister Application service

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

For a description of the service, see reference [4].

6.1.9 Register Fatal Error Callback service

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

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

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

Note:
If the application is not running locally on the netX this functionality is NOT available. In this case, the stack is informed about the error with a packet as described in section *Error Indication service* [▸ page 161] of this document. Therefore, this service is NOT available if dual-port memory is used.

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

6.1.9.1 Register Fatal Error Callback request

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle of PNS_IF task process queue
ulLen	uint32_t	8	Packet data length in bytes
ulCmd	uint32_t	0x1FDA	PNS_REGISTER_FATAL_ERROR_CALLBACK_REQ
pfnClbFnc	PNS_FATAL_ERROR_CLB		The pointer to the callback function. Definition see below.
pvUserParam	void *		This user specific parameter is passed to the callback function if it is called.

Table 23: PNS_REG_FATAL_ERROR_CALLBACK_REQ_T - Register Fatal Error Callback request

Packet structure reference

```
/* Request packet */
typedef __HIL_PACKED_PRE struct
PNS_REG_FATAL_ERROR_CALLBACK_REQ_DATA_Ttag
{
  PNS_FATAL_ERROR_CLB pfnClbFnc;
  void* pvUserParam;
} __HIL_PACKED_POST PNS_REG_FATAL_ERROR_CALLBACK_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct
PNS_REG_FATAL_ERROR_CALLBACK_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_REG_FATAL_ERROR_CALLBACK_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_REG_FATAL_ERROR_CALLBACK_REQ_T;
```

Note:
The definition of the type of callback function is as follows:

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

6.1.9.2 Register Fatal Error Callback confirmation

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle of PNS_IF task process queue
ulLen	uint32_t	0	Packet data length in bytes
ulCmd	uint32_t	0x1FDB	PNS_REGISTER_FATAL_ERROR_CALLBACK_CNF

Table 24: PNS_REG_FATAL_ERROR_CALLBACK_CNF_T - Register Fatal Error Callback confirmation

Packet structure reference

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

6.1.10 Unregister Fatal Error Callback service

Using this service, the user application can unregister a previously registered error callback function from the stack. After unregistering this callback, the application will only report fatal errors using the service described in *Write I&M service* [▷ page 169].

6.1.10.1 Unregister Fatal Error Callback request

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

Packet description

Variable	**Type**	**Value / Range**	**Description**
ulDest	uint32_t		Destination queue handle of PNS_IF task process queue
ulLen	uint32_t	0	Packet data length in bytes
ulCmd	uint32_t	0x1FDE	PNS_UNREGISTER_FATAL_ERROR_CALLBACK_REQ

Table 25: PNS_UNREG_FATAL_ERROR_CALLBACK_REQ_T - Unregister Fatal Error Callback request

Packet structure reference

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

6.1.10.2 Unregister Fatal Error Callback confirmation

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

Packet description

Variable	**Type**	**Value / Range**	**Description**
ulDest	uint32_t		Destination queue handle of PNS_IF task process queue
ulLen	uint32_t	0	Packet data length in bytes
ulCmd	uint32_t	0x1FDF	PNS_UNREGISTER_FATAL_ERROR_CALLBACK_CNF

Table 26: PNS_UNREG_FATAL_ERROR_CALLBACK_CNF_T - Unregister Fatal Error Callback confirmation

Packet structure reference

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

6.1.11 Set Port MAC Address service

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

These protocols are part of the stack and cannot be disabled. They require a different MAC-address for each single Ethernet port of the hardware. Therefore, a hardware with two Ethernet ports needs at least three MAC-addresses.

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

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

Note:
If the user application wants to set the “Chassis MAC-address” the user application has to use the “Set MAC address” (`RCX_SET_MAC_ADDR_REQ`) service (see reference [5]).

6.1.11.1 Set Port MAC Address request

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

Note:
This packet must be sent prior to the *Set Configuration request* [▸ page 54] packet, otherwise it will be rejected and the former choice of the Port MAC address will be fixed.

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	12	Packet data length in bytes
ulCmd	uint32_t	0x1FE0	PNS_IF_SET_PORT_MAC_REQ
Data			
atPortMacAddr[2]	PORT_MAC_ADDR_T		Structure containing the two port MAC addresses. atPortMacAddr[0] stores MAC-address for Port 0, atPortMacAddr[1] stores MAC-address for Port .

Table 27: PNS_IF_SET_PORT_MAC_REQ_T - Set Port MAC Address request

Packet structure reference

```
typedef uint8_t PORT_MAC_ADDR_T[6];

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

typedef __HIL_PACKED_PRE struct PNS_IF_SET_PORT_MAC_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_SET_PORT_MAC_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_SET_PORT_MAC_REQ_T;
```

6.1.11.2 Set Port MAC Address confirmation

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle of PNS_IF task process queue
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▶ page 292].
ulCmd	uint32_t	0x1FE1	PNS_IF_SET_PORT_MAC_CNF

Table 28: PNS_IF_SET_PORT_MAC_CNF_T - Set Port MAC Address confirmation

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_SET_PORT_MAC_CNF_T;
```

© Hilscher 2011 – 2019

6.1.12 Set OEM Parameters service

The application can use this request to

- set or change specific identification parameters (used for I&M for example, for more information on Information and Maintenance (I&M), see references [2] and [3]),
- set or change specific communication parameters, or
- activate or deactivate features.

Normally, well-defined Hilscher default values are used. If this is not sufficient for the application, the application can use this service to change the default values.

Note:
This service has to be used before using the *Set Configuration service* [▶ page 53] to configure the stack if stack is not configured with database.

The following table lists the OEM Parameter services and includes

- whether the OEM parameter can be set in case of a configuration based on Set Config or on a database,
- whether the application can set the OEM parameter multiple time or only once,
- whether the OEM parameter has influence to the device identity (device identity are all parameters that identify the IO Device in the network, e.g. VendorID, DeviceID, SerialNumber, ...),
- whether the OEM parameter activates/deactivates a device feature,
- whether the OEM parameter requires GSDML file modifications.

No.	Action	Usable with Set Config	Usable with database	Allowed multiple times	Device identity	I&M	Device feature	GSDML impact
1	Set serial number for I&M 0 responses Set ProfileID for I&M responses Set ProfileSpecificType	yes	yes	no	yes	I&M0	-	-
2	Set serial number for I&M 0 responses	yes	yes	no	yes	I&M0	-	-
3	Set timeout value for waiting for application packets (in units of milliseconds)	yes	yes	yes	no	-	-	-
4	-	-	-	-	-	-	-	-
5	Set bit mask for I&M support (I&M 1 to I&M 5)	yes	yes	yes	no	I&M0	-	yes
6	Offset address of the “Current local cycle counter”	yes	yes	yes	-	-	-	-
7	Sets the packet command code used for LinkStatus indications	yes	yes	yes	no	-	-	-
8	Switch between I&M handling by protocol stack or by application	-	yes	yes	no	-	-	-
9	Set I&M5 parameters	yes	yes	no	yes	I&M5	-	-
10	Activate / deactivate PROFIenergy support of the stack	yes	yes	yes	no	-	yes	yes
11	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-
14	Set Softwareversion and HardwareRevision in case of database configuration.	no	yes	yes	yes	I&M0	-	(yes)
15	Set Device Identity Parameters in case of database configuration.	no	yes	yes	yes	I&M0	-	yes

Table 29: OEM Parameters and their related actions

6.1.12.1 Set OEM Parameters request

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

Important:
This packet must be sent prior to the Set Configuration Service packet if stack is not configured with database, otherwise it will be rejected and the well-defined Hilscher default values will be used.

Note:
Each subtype of this packet may be sent to the stack **either once or never at all**. Multiple use of the same subtype for this packet is not allowed.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	4 + n	Packet data length in bytes. n depends on the parameter type contained in the packet.
ulCmd	uint32_t	0x1FE8	PNS_IF_SET_OEM_PARAMETERS_REQ
Data			
ulParamType	uint32_t		PNS_IF_SET_OEM_PARAMETERS_TYPE_1, …, This parameter specifies which structure follows. See below.
union PNS_IF_SET_OEM_PARAMETERS_UNION_T			
			Depending on the chosen Paramtype the corresponding structure shall be used here. See below.

Table 30: PNS_IF_SET_OEM_PARAMETERS_REQ_T - Set OEM Parameters Request

Packet structure reference

```
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_1 0x01
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_2 0x02
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_3 0x03
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_4 0x04
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_5 0x05
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_6 0x06
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_7 0x07
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_8 0x08
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_9 0x09
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_10 0x0A
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_11 0x0B
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_12 0x0C
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_13 0x0D
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_14 0x0E
#define PNS_IF_SET_OEM_PARAMETERS_TYPE_15 0x0F

/* Request packet */
typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_1_Ttag
{
  uint8_t abSerialNumber[16];
  uint16_t usProfileId;
  uint16_t usRevisionCounter;
  uint16_t usProfileSpecificType;
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_1_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_2_Ttag
{
  /* SerialNumber for usage in I&M0, RPC EndPointMapper and SNMP
sysDescr */
  uint8_t abSerialNumber[16];
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_2_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_3_Ttag
{
  uint32_t ulTimeout;
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_3_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_4_Ttag
{
  uint16_t usSystemNameLen;
  uint8_t abSystemName[255];
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_4_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_5_Ttag
{
  uint32_t ulIMFlag;
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_5_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_6_Ttag
{
  uint16_t usIRTCycleCounterOffset;
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_6_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_7_Ttag
{
  uint32_t fUseOldLinkStateCommandCode; /* if set use the old packet
command PNS_IF_LINK_STATE_CHANGE_IND, else the new
RCX_LINK_STATUS_CHANGE_IND */
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_7_T;
```

```
typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_8_Ttag
{
  uint32_t fDisableFirmwareIMHandling; /* if set firmware does not
handle I&M requests in case of database configuration */
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_8_T;

typedef PNS_IF_IM5_DATA_T PNS_IF_SET_OEM_PARAMETERS_TYPE_9_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_10_Ttag
{
  uint32_t fSupportProfiEnergyFunction; /* if set firmware supports
ProfiEnergy functionality */
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_10_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_11_Ttag
{
  /* if set firmware supports IRT communication
   * @Note IRT support is enabled by default */
  uint32_t fSupportIRT;
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_11_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_12_Ttag
{
  /* if set firmware supports IO Supervisor communication
   * @Note IO Supervisor communication support is disabled by
default. */
  uint32_t fSupportIOSupervisorAR;
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_12_T;

#define PNS_IF_SET_OEM_MINDEVICE_INTERVAL_MIN 8
#define PNS_IF_SET_OEM_MINDEVICE_INTERVAL_MAX 4096
#define PNS_IF_SET_OEM_MINDEVICE_INTERVAL_IRT_MAX 32

typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_13_Ttag
{
  /* The minimum supported cycle time in base of 31,25us (allowed
values: Power of two in range [8 - 4096]).
   * It must correspond with MinDeviceInterval from GSDML file
   * @Note By default the firmware supports a usMinDeviceInterval of
8 is supported
   * this means that a cycle time of 250us will be supported (8 *
31,25 = 250us)
   * If IRT supported MinDeviceInterval must not be greater than 32
*/
  uint16_t usMinDeviceInterval;
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_13_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_14_Ttag
{
  /* HardwareRevision for usage in I&M0, RPC EndPointMapper and SNMP
sysDescr */
  uint16_t usHardwareRevision;
  /* SoftwareVersion for usage in I&M0, RPC EndPointMapper and SNMP
sysDescr */
  PNS_IF_IM_SWVERSION_T tSoftwareRevision;
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_14_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_SET_OEM_PARAMETERS_TYPE_15_Ttag
{
  /* VendorID to be used (instead of the one contained in
configuration database file) */
  uint16_t usVendorId;
  /* DeviceID to be used (instead of the one contained in
configuration database file) */
  uint16_t usDeviceId;
  /* OrderID to be used (instead of the one contained in
configuration database file) - (pad with spaces (0x20) */
  uint8_t abOrderId[PNS_IF_MAX_ORDER_ID];
  /* DeviceType to be used (instead of the one contained in
configuration database file) - (ignore the last 3 padding bytes */
  uint8_t abDeviceType[PNS_IF_MAX_DEVICE_TYPE_LEN +3];
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_TYPE_15_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SET_OEM_PARAMETERS_DATA_Ttag
{
  uint32_t ulParameterType;
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_DATA_T;

typedef union PNS_IF_SET_OEM_PARAMETERS_UNION_Ttag
{
  PNS_IF_SET_OEM_PARAMETERS_TYPE_1_T tType1Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_2_T tType2Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_3_T tType3Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_4_T tType4Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_5_T tType5Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_6_T tType6Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_7_T tType7Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_8_T tType8Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_9_T tType9Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_10_T tType10Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_11_T tType11Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_12_T tType12Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_13_T tType13Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_14_T tType14Param;
  PNS_IF_SET_OEM_PARAMETERS_TYPE_15_T tType15Param;
} PNS_IF_SET_OEM_PARAMETERS_UNION_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SET_OEM_PARAMETERS_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_SET_OEM_PARAMETERS_DATA_T tData;
  PNS_IF_SET_OEM_PARAMETERS_UNION_T tParam;
} __HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_REQ_T;
```

6.1.12.2 OEM parameter: type 1

The application can use this OEM parameter type to modify I&M0 parameters.

Note:
The application can use this OEM type parameter only, if the stack handles I&M data.

If ulParamType is set to `PNS_IF_SET_OEM_PARAMETERS_TYPE_1` the structure tType1Param of `PNS_IF_SET_OEM_PARAMETERS_UNION_T` shall be used. It is defined as follows:

Variable	Type	Value / Range	Description
abSerialNumber	uint8_t[16]		The serial number to be used for the I&M 0 responses. The serial number shall be left aligned and space (hex 0x20) padded. This serial number is used for SNMP SystemDescription as well.
usProfileId	uint16_t		The ProfileId to be used for I&M responses.
usRevisionCounter	uint16_t		Revision counter to be used for I&M responses.
usProfileSpecificType	uint16_t		The ProfileSpecificType to be used for I&M responses.

Table 31: PNS_IF_SET_OEM_PARAMETERS_TYPE_1_T - Set OEM Parameter type 1

Note:
The application can use this service only if the firmware is in Bus Off state. The application cannot use this service if the firmware is configured and Bus On state is set.

6.1.12.3 OEM parameter: type 2

The application can use this OEM parameter type to modify the serial number of the I&M0 parameters.

Note:
The application can use this OEM type parameter only, if the stack handles I&M data.

If `ulParamType` is set to `PNS_IF_SET_OEM_PARAMETERS_TYPE_2` use the structure `tType2Param` of `PNS_IF_SET_OEM_PARAMETERS_UNION_T`. It is defined as follows:

Variable	Type	Value / Range	Description
abSerialNumber	uint8_t[16]		The serial number to be used for I&M 0 responses. The serial number shall be left aligned and space (hex 0x20) padded. This serial number is used for SNMP SystemDescription as well.

Table 32: PNS_IF_SET_OEM_PARAMETERS_TYPE_2_T - Set OEM Parameter type 2

Note:
The application can use this service only if the firmware is in Bus Off state. The application cannot use this service if the firmware is configured and Bus On state is set.

6.1.12.4 OEM parameter: type 3

The application can use this OEM parameter type to modify the default timeout for application supervision.

If `ulParamType` is set to `PNS_IF_SET_OEM_PARAMETERS_TYPE_3` use the structure `tType3Param` of `PNS_IF_SET_OEM_PARAMETERS_UNION_T`. It is defined as follows:

Variable	**Type**	**Value / Range**	**Description**
ulTimeout	uint32_t	0x00000BB8 (3000) …	The timeout (in milliseconds) specifies how long the stack waits for application packets. The value can be 3000 ms or higher.

Table 33: PNS_IF_SET_OEM_PARAMETERS_TYPE_3_T - Set OEM Parameter type 3

6.1.12.5 OEM parameter: type 4

Parameter type 4 is deprecated and not supported.

6.1.12.6 OEM parameter: type 5

The application can use this OEM parameter type to configure which I&M data set

Using this OEM parameter type the supported I&M datasets can be modified.

If `ulParamType` is set to `PNS_IF_SET_OEM_PARAMETERS_TYPE_5` use the structure `tType5Param` of `PNS_IF_SET_OEM_PARAMETERS_UNION_T`. It is defined as follows:

Variable	Type	Value / Range	Description
ulIMFlag	uint32_t	0x07, 0x0F (default), 0x17, 0x1F	A bitmask enables which I&M data sets are supported. Each bit corresponds to an I&M data set. I&M1, I&M2, and I&M3 always have to be enabled. If a flag is set, the corresponding I&M data set is enabled and will be either handled by the stack internally or forwarded to the application depending on system flags used in *Set Configuration service* [▸ page 53]. Default value is 0x0F: I&M1, I&M2, I&M3, and I&M4 are enabled.

Table 34: PNS_IF_SET_OEM_PARAMETERS_TYPE_5_T – Set OEM Parameter type 5

Coding of ulIMFlag

Bit No	Flag Define/ Desired Behavior
D0	If this flag is set using PNS_IF_STACK_HANDLE_IM_1, the stack will enable I&M1 data set. I&M1 always has to be enabled.
D1	If this flag is set using PNS_IF_STACK_HANDLE_IM_2, the stack will enable I&M2 data set. I&M2 always has to be enabled.
D2	If this flag is set using PNS_IF_STACK_HANDLE_IM_3, the stack will enable I&M3 data set. I&M3 always has to be enabled.
D3	If this flag is set using PNS_IF_STACK_HANDLE_IM_4, the stack will enable I&M4 data set. 0 = I&M4 data set is disabled. 1 = I&M4 data set is enabled.
D4	If this flag is set using PNS_IF_STACK_HANDLE_IM_5, the stack will enable I&M5 data set. 0 = I&M5 data set is disabled. 1 = I&M5 data set is enabled.
D5 – D15	Reserved, set to zero.

Table 35: Coding of ulIMFlag

6.1.12.7 OEM parameter: type 6

The application can use this OEM parameter type to configure the offset address for the “current local cycle counter”. The stack copies the value of the “current local cycle counter” (16-bit value) to the input data memory (consumer data of the IO Device).

Note:
This service is intended to be used by Isochrone Mode applications for IRT.

IRT mode

In case of IRT operation of the IO Device, the stack uses the “cycle counter” value from the last received frame of the IO Controller for the “current local cycle counter” before copying it into the input data memory. In case of error (e.g. no cyclic frame received), the stack copies an incremented value of the “current local cycle counter” to the input data memory. As soon as the communication returns to normal operation, the stack continues to use the “cycle counter” value from the last received frame of the IO Controller for the “current local cycle counter”.

Note:
In IRT operation mode, the “current local cycle counter” is equal to the “cycle counter” of the IO Controller. Thus in IRT operation mode the cycle counter represents the “current IRT cycle” counter at any time.

RT mode

Note:
In case of RT operation of the IO Device, the value of the “current local cycle counter” is NOT identical to the “cycle counter” of the IO Controller. This service is not useful for RT mode.

Description of the parameter

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

Variable	Type	Value / Range	Description
usIRTCycleCounterOffset	uint16_t	0 … 5758	Offset address in the input data memory the value of the “current local cycle counter” is copied to. The value of the “current local cycle counter” requires 2 bytes in the input data memory.
		0xFFFF	“Current local cycle counter” is not copied to the input data memory (default).

Table 36: PNS_IF_SET_OEM_PARAMETERS_TYPE_6_T - Set OEM Parameter type 6

Note:
In case this service is active, a ChannelInit will not deactivate this service. To deactivate this service, the application has to write offset 0xFFFF.

Note:
If the application wants to change the offset address, the application has to write offset 0xFFFF to deactivate this service first. Afterwards, the application can write the new offset value and reactivates this service.

6.1.12.8 OEM parameter: type 7

The application can use this OEM parameter type to configure the packet command code for LinkStatus indications (for compatibility reasons).

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

Variable	Type	Value / Range	Description
fUseOldLinkStateCommandCode	uint32_t	0 1	Stack has to use HIL_LINK_STATUS_CHANGE_IND Stack has to use PNS_IF_LINK_STATE_CHANGE_IND

Table 37: PNS_IF_SET_OEM_PARAMETERS_TYPE_7_T - Set OEM Parameter type 7

6.1.12.9 OEM parameter: type 8

In case the stack is configured with a database file, the application can disable the handling of I&M data by the stack. The application can use this OEM type parameter service at runtime but only in state “Bus off”.

In case the stack is configured with the *Set Configuration service* [▸ page 53], the packet will be accepted without error code “invalid parameter”, but the stack behavior regarding I&M handling will not be modified. However, in that case I&M handling can be influenced by other means, see `ulSystemFlags` in *Set Configuration request* [▸ page 54] packet.

If `ulParamType` is set to `PNS_IF_SET_OEM_PARAMETERS_TYPE_8` use the structure `tType8Param` of `PNS_IF_SET_OEM_PARAMETERS_UNION_T`. It is defined as follows:

Variable	Type	Value / Range	Description
fDisableFirmwareIMHandling	uint32_t	0 1	Stack handles I&M data Application handles I&M data

Table 38: PNS_IF_OEM_PARAMETERS_TYPE_8_T - Set OEM Parameter type 8

6.1.12.10 OEM parameter: type 9

The application can use this OEM parameter type to modify the I&M5 parameters reported by stack to the network.

If ulParamType is set to `PNS_IF_SET_OEM_PARAMETERS_TYPE_9` the structure tType9Param of `PNS_IF_SET_OEM_PARAMETERS_UNION_T` shall be used. It is defined as follows:

Variable	Type	Value / Range	Description
abAnnotation[64]	uint8_t	String	Annotation
abOrderId[64]	uint8_t	String	Order id
usVendorId	uint16_t	Numeric	Vendor id
abSerialNumber[16]	uint8_t	String	Serial number od the device
usHardwareRevision	uint16_t	Numeric	Hardware revision
tSoftwareRevision	PNS_IF_IM_SWVERSION_T		Software revision including: prefix and version x, y, and z.

Table 39: PNS_IF_OEM_PARAMETERS_TYPE_9_T - Set OEM Parameter type 9

© Hilscher 2011 – 2019

6.1.12.11 OEM parameter: type 10

The application can use this OEM parameter type to enable PROFIenergy feature.

Note:
The application can use this service only if bus communication is deactivated (BUS OFF).

If `ulParamType` is set to `PNS_IF_SET_OEM_PARAMETERS_TYPE_10` use the structure `tType10Param` of `PNS_IF_SET_OEM_PARAMETERS_UNION_T`. It is defined as follows:

Variable	Type	Value / Range	Description
fSupportProfiEnergyFunction	uint32_t	0	PROFIenergy not supported
		1	Stack supports PROFIenergy functionality

Table 40: PNS_IF_OEM_PARAMETERS_TYPE_10_T - Set OEM Parameter type 10

6.1.12.12 OEM parameter: type 11

This parameter type is not supported.

6.1.12.13 OEM parameter: type 12

This parameter type is not supported.

6.1.12.14 OEM parameter: type 13

This parameter type is not supported.

6.1.12.15 OEM parameter: type 14

The application can use this OEM parameter type to modify device identity parameters used for I&M, SNMP and RPC.

Note:
The application can use this service only if the stack is configured with a database file, the start behavior is set in the database “Application Controlled Startup” and bus communication is still deactivated (BUS OFF).

If `ulParamType` is set to `PNS_IF_SET_OEM_PARAMETERS_TYPE_14` use the structure `tType14Param` of `PNS_IF_SET_OEM_PARAMETERS_UNION_T`.

Variable	Type	Value / Range	Description
usHardwareRevision	uint16_t		HardwareRevision of the device
tSoftwareRevision.bPrefix	uint8_t		Character describing the software version Prefix. Allowed values: ‘V’, ‘R’, ‘P’, ‘U’ and ‘T’
tSoftwareRevision.bX	uint8_t		Function enhancement. (Major version number) of the application.
tSoftwareRevision.bY	uint8_t		Bug fix (Minor version number) of the application.
tSoftwareRevision.bZ	uint8_t		Internal Change (Build version number) of the application.

Table 41: PNS_IF_OEM_PARAMETERS_TYPE_14_T - Set OEM Parameter type 14

This parameter type is not supported.

6.1.12.16 OEM parameter: type 15

The application can use this OEM parameter type to modify device identity parameters used for I&M, SNMP and RPC.

Note:
The application can use this service only if the stack is configured with a database file, the start behavior is set in the database “Application Controlled Startup” and bus communication is still deactivated (BUS OFF).

If `ulParamType` is set to `PNS_IF_SET_OEM_PARAMETERS_TYPE_15` use the structure `tType15Param` of `PNS_IF_SET_OEM_PARAMETERS_UNION_T`.

Variable	Type	Value / Range	Description
usVendorId	uint16_t		Vendor ID, see “Set Configuration request” for details
usDeviceId	uint16_t		Device ID, see “Set Configuration request” for details
abOrderId[20]	uint8_t[]		Order ID, see “Set Configuration request” for details
abDeviceType[28]	uint8_t[]		DeviceType, see “Set Configuration request” for details

Table 42: PNS_IF_OEM_PARAMETERS_TYPE_15_T - Set OEM Parameter type 15

© Hilscher 2011 – 2019

6.1.12.17 Set OEM Parameters confirmation

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle of PNS_IF task process queue
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1FE9	PNS_IF_SET_OEM_PARAMETERS_CNF

Table 43: PNS_IF_SET_OEM_PARAMETERS_CNF_T - Set OEM Parameters confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_SET_OEM_PARAMETERS_CNF_Ttag{
  HIL_PACKET_HEADER_T tHead;
}__HIL_PACKED_POST PNS_IF_SET_OEM_PARAMETERS_CNF_T;
```

6.1.13 Load Remanent Data service

Using this service the application can hand over the required remanent data to the stack. This service has to be used if the application stores remanent data, which is the case when bit D14 is set in the system flags of the *Set Configuration request* [▹ page 54].

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

In this case, the application has to continue with startup and ignore the return value.

6.1.13.1 Load Remanent Data request

This service allows the application to hand over the remanent data needed by the stack. The stack will check the data and if it is correct, this data will be used.

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

Important:
This packet must be sent prior to the *Set Configuration request* [▹ page 54] packet, otherwise this packet will be rejected and the default values will be used.

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle of PNS_IF task process queue
ulLen	uint32_t	1 + n	Packet data length in bytes.
ulCmd	uint32_t	0x1FEC	PNS_IF_LOAD_REMANENT_DATA_REQ
ulExt	uint32_t		Sequence number for use in fragmented packets
Data			
abData[1]	uint8_t		The remanent data, which was reported by the stack. This is only the first byte as a placeholder. All remaining bytes have to follow this one.

Table 44: PNS_IF_LOAD_REMANENT_DATA_REQ_T - Load Remanent Data request

```
typedef __HIL_PACKED_PRE struct
{
  /** this is only the first byte, many others may follow */
  uint8_t abData[1];
} __HIL_PACKED_POST PNS_IF_LOAD_REMANENT_DATA_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_LOAD_REMANENT_DATA_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_LOAD_REMANENT_DATA_REQ_T;
```

6.1.13.2 Load Remanent Data confirmation

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle of PNS_IF task process queue
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1FED	PNS_IF_LOAD_REMANENT_DATA_CNF

Table 45: PNS_IF_LOAD_REMANENT_DATA_CNF_T - Load Remanent Data confirmation

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_LOAD_REMANENT_DATA_CNF_T;
```

6.1.14 Configuration Delete service

The application can delete the current configuration with the Configuration Delete service, for a description see `RCX_DELETE_CONFIG_REQ` in reference [5].

In addition to the current configuration, the stack deletes all remanent data and resets these to default values.

This service will **not** delete configuration files downloaded with the configuration software SYCON.net.

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

6.1.15 Set IO Image service

This service has to be used if the user application uses the callback interface for accessing the cyclic I/O-data. The service request will provide the PROFINET IO-Device stack with pointers to the consumer and provider data images and the user application's event callback. In return, the service's confirmation will provide the user application with pointers to update input, update output and update extended status block callbacks. This request is essential for any user application running locally on the netX and not using the shared memory interface. It has to be issued before using the *Set Configuration service* [▸ page 53]. See also section *Cyclic process data image* [▸ page 45] which contains an example.

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

6.1.15.1 Set IO-Image request

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	24	Packet data length in bytes.
ulCmd	uint32_t	0x1FF0	PNS_IF_SET_IOIMAGE_REQ
Data			
ulConsImageSize	uint32_t	0 ... 2^{16} -1	The size of the consumer data image.
ulProvImageSize	uint32_t	0 ... 2^{16} -1	The size of the provider data image.
pbConsImage	uint8_t*		Pointer to consumer data image. Required to point to a memory area provided by the user's application where the incoming consumer data can be copied to by the stack. See section Cyclic Process Data Image for further information
pbProvImage	uint8_t*		Pointer to provider data image. Required to point to a memory area provided by the user's application where the outgoing provider data can be taken from by the stack. See section Cyclic Process Data Image for further information
pfnEventHandler	PNS_IF_IOEVENT_HANDLER_CLB_T		Pointer to the user's callback function for application events to be called by the stack for various events.
hUserParam	HANDLE		This parameter will be passed as first parameter to the user's event callback function. Typically the user applications resource parameter.

Table 46: PNS_IF_SET_IOIMAGE_REQ_T – Set IO-Image request

Packet sStructure reference

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

typedef uint32_t (*PNS_IF_UPDATE_IOIMAGE_CLB_T) (void* hUserParam,
unsigned int uiTimeout, unsigned int uiReserved1, unsigned int
uiReserved2);
typedef uint32_t (*PNS_IF_IOEVENT_HANDLER_CLB_T) (void* hUserParam,
unsigned int uiEvent);

typedef __HIL_PACKED_PRE struct PNS_IF_SET_IOIMAGE_REQ_DATA_Ttag
{
  uint32_t ulConsImageSize;
  uint32_t ulProvImageSize;
  uint8_t* pbConsImage;
  uint8_t* pbProvImage;

  PNS_IF_IOEVENT_HANDLER_CLB_T pfnEventHandler;
  void* hUserParam;

} __HIL_PACKED_POST PNS_IF_SET_IOIMAGE_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SET_IOIMAGE_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_SET_IOIMAGE_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_SET_IOIMAGE_REQ_T;
```

6.1.15.2 Set IO-Image confirmation

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle of PNS_IF task process queue
ulLen	uint32_t	16	Packet data length in bytes.
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1FF1	PNS_IF_SET_IOIMAGE_CNF-Command
Data			
hCallbackParam	HANDLE		This handle shall be passed by the user's application as first parameter to the callback functions.
pfnUpdateConsumerImage	PNS_IF_UPDATE_IOIMAGE_CLB_T		Pointer to the stack's update consumer callback function. This callback shall be used by the user's application to update its consumer data image with newest cyclic data.
pfnUpdateProviderImage	PNS_IF_UPDATE_IOIMAGE_CLB_T		Pointer to the stack's update provider callback function. This callback shall be used by the user's application to instruct the stack to update the cyclic data from the provider data image.

Table 47: PNS_IF_SET_IOIMAGE_CNF_T – Set IO-Image confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_SET_IOIMAGE_CNF_DATA_Ttag
{
  void* hCallbackParam;
  PNS_IF_UPDATE_IOIMAGE_CLB_T pfnUpdateConsumerImage;
  PNS_IF_UPDATE_IOIMAGE_CLB_T pfnUpdateProviderImage;
} __HIL_PACKED_POST PNS_IF_SET_IOIMAGE_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SET_IOIMAGE_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_SET_IOIMAGE_CNF_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_SET_IOIMAGE_CNF_T;
```

6.1.16 Set IOXS Config service

The user application must use this service in order to enable reading/writing data states from/to the consumer/provider data image. When enabling this functionality, the stack will copy the provider data states of consumer data into the consumer data image and take the provider data states of provider data from provider data image. In this case, the user application is responsible for setting these states appropriate. For a detailed description of the structure of the data state block in the input/output image, refer to the *Dual Port Memory interface manual*. See also section *Configuration of process data images* [▸ page 46] which contains an example.

6.1.16.1 Set IOXS Config request

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	32	Packet data length in bytes.
ulCmd	uint32_t	0x1FF2	PNS_IF_SET_IOXS_CONFIG_REQ- Command
Data			
ulIOPSMode	uint32_t	0 ... 2	Desired encoding of the IOPS states. Either disabled, bit list (valid invalid) or byte list (complete IOPS)
ulConsImageIOPSOffset	uint32_t	0 ... 2^{16} -1	The offset, where the provider state block for consumer data shall start in the consumer data image. The offset is relative to the beginning of the consumer data image. See section *Configuration of process data images* [▸ page 46] for further information
ulReserved1	uint32_t	0	Reserved. Set to zero for compatibility reasons.
ulProvImageIOPSOffset	uint32_t	0 ... 2^{16} -1	The offset, where the provider state block for provider data shall start in the provider data image. The offset is relative to the beginning of the provider data image. See section *Configuration of process data images* [▸ page 46] for further information
ulIOCSMode	uint32_t	0 …. 2	Desired encoding of the IOCS states. Either disabled, bit list (valid invalid) or byte list (complete IOCS)
ulConsImageIOCSOffset	uint32_t	0 ... 2^{16} -10	The offset, where the consumer state block for consumer data shall start in the consumer data image. The offset is relative to the beginning of the consumer data image. See section *Configuration of process data images* [▸ page 46] for further information
ulReserved2	uint32_t	0	Reserved. Set to zero for compatibility reasons.
ulProvImageIOCSOffset	uint32_t	0 ... 2^{16} -1	The offset, where the consumer state block for provider data shall start in the provider data image. The offset is relative to the beginning of the provider data image. See section *Configuration of process data images* [▸ page 46] for further information

Table 48: PNS_IF_SET_IOXS_CONFIG_REQ_T – Set IOXS Config request

Packet structure reference

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

typedef enum PNS_IF_IOCS_MODE_Etag
{
  PNS_IF_IOCS_DISABLED = 0,
  PNS_IF_IOCS_BITWISE,
  PNS_IF_IOCS_BYTEWISE,
} PNS_IF_IOCS_MODE_E;

typedef __HIL_PACKED_PRE struct PNS_IF_SET_IOXS_CONFIG_DATA_Ttag
{
  uint32_t ulIOPSMode;
  uint32_t ulConsImageIOPSOffset;
  uint32_t ulReserved1;
  uint32_t ulProvImageIOPSOffset;

  uint32_t ulIOCSMode;
  uint32_t ulConsImageIOCSOffset;
  uint32_t ulReserved2;
  uint32_t ulProvImageIOCSOffset;
} __HIL_PACKED_POST PNS_IF_SET_IOXS_CONFIG_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SET_IOXS_CONFIG_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_SET_IOXS_CONFIG_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_SET_IOXS_CONFIG_REQ_T ;
```

6.1.16.2 Set IOXS Config confirmation

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle of PNS_IF task process queue
ulLen	uint32_t	0	Packet data length in bytes.
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1FF3	PNS_IF_SET_IOXS_CONFIG_CNF

Table 49: PNS_IF_SET_IOXS_CONFIG_CNF_T – Set IOXS Config confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_SET_IOXS_CONFIG_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} __HIL_PACKED_POST PNS_IF_SET_IOXS_CONFIG_CNF_T ;
```

6.1.17 Configure Signal service

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

6.1.17.1 Configure Signal request

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle of PNS_IF task process queue
ulLen	uint32_t	40 * 2 * Number of Signals	Packet data length in bytes.
ulCmd	uint32_t	0x6100	IO_SIGNALS_CONFIGURE_SIGNAL_REQ- Command
ulFieldbusSpecific1	uint32_t	0	Unused, set to zero.
ulFieldbusSpecific2	uint32_t	0	The API of the submodule. Currently only 0 supported.
ulFieldbusSpecific3	uint32_t	0..0x7FFF	The slot of the submodule.
ulFieldbusSpecific4	uint32_t	1..0x7FFF	The subslot of the submodule
ulFieldbusSpecific5	uint32_t	0	Unused. Set to zero
ulFieldbusSpecific6	uint32_t	0	Unused. Set to zero
ulFieldbusSpecific7	uint32_t	0	Unused. Set to zero
ulFieldbusSpecific8	uint32_t	0	Unused. Set to zero
ulSignalsDirection	uint32_t	1..2	Either IO_SIGNALS_DIRECTION_CONSUMER or IO_SIGNALS_DIRECTION_PROVIDER
ulTotalSignalCount	uint32_t	1..1024	Count of Signals elements. (n)
atSignals	{uint8_t ; uint8_t} * n		Array of pairs of bytes which describe the signal. The first byte is the signal type, the second byte is the number of elements of the specified kind of the signal. (e.g. (20,4) is an array of four signed 64 bit integers)

Table 50: Structure IO_SIGNALS_CONFIGURE_SIGNAL_REQ_T

Packet structure reference

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

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

typedef __HIL_PACKED_PRE struct
IO_SIGNALS_CONFIGURE_SIGNAL_REQ_DATA_Ttag
{
  /* see fieldbus specific API Manual for a definition how this */
  /* fieldbus specific fields shall be filled. */
  uint32_t ulFieldbusSpecific1; /* e.g. Slave Handle */
  uint32_t ulFieldbusSpecific2;
  uint32_t ulFieldbusSpecific3; /* e.g. Slot */
  uint32_t ulFieldbusSpecific4; /* e.g. SubSlot */
  uint32_t ulFieldbusSpecific5;
  uint32_t ulFieldbusSpecific6;
  uint32_t ulFieldbusSpecific7;
  uint32_t ulFieldbusSpecific8;
  /* signal direction described in this packet */
  uint32_t ulSignalsDirection;
  /* amount of signals contained in abSignals */
  uint32_t ulTotalSignalCount;
  /* array of signals - packet definition only contains the first
signal, all other follow */
  struct
  {
    /* type of signal - see IO_SIGNALS_TYPES_E */
    uint8_t bSignalType;
    /* amount of signal (e.g. 16 for a "16 Byte Input module") */
    uint8_t bSignalAmount;
  } atSignals[1];
} IO_SIGNALS_CONFIGURE_SIGNAL_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct IO_SIGNALS_CONFIGURE_SIGNAL_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  IO_SIGNALS_CONFIGURE_SIGNAL_REQ_DATA_T tData;
} IO_SIGNALS_CONFIGURE_SIGNAL_REQ_T;
```

6.1.17.2 Configure Signal confirmation

The confirmation packet has no data part.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle
ulLen	uint32_t	0	Packet Data Length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x6101	IO_SIGNALS_CONFIGURE_SIGNAL_CNF- Command

Table 51: IO_SIGNALS_CONFIGURE_SIGNAL_CNF_T – Configure Signal confirmation

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T IO_SIGNALS_CONFIGURE_SIGNAL_CNF_T;
```

6.1.17.3 Example: Configure Signal request packet

For an imaginary 12 byte input submodule using the data structure

```
struct {
  uint32_t aulData1[2];
  uint8_t abData2[2];
  uint16_t usData3;
}
Which corresponds to the following IO data subsection in a GSDML
file
<IOData IOPS_Length="1" IOCS_Length="1">
<Input>
<DataItem DataType="Unsigned32" TextId="aulData1a"/>
<DataItem DataType="Unsigned32" TextId="aulData1b"/>
<DataItem DataType="OctetString" Length="2" TextId="abData2"/>
<DataItem DataType="Unsigned16" TextId="abData3"/>
</Input>
</IOData>
The Configure Signal Request packet has to be filled in as follows:
IO_SIGNALS_CONFIGURE_SIGNAL_REQ_T* ptRequest =
malloc(sizeof(IO_SIGNALS_CONFIGURE_SIGNAL_REQ) + 2 * (4-1));

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

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

ptRequest->tData.ulFieldbusSpecific2 = 0; /* api */
ptRequest->tData.ulFieldbusSpecific3 = 1; /* slot */
ptRequest->tData.ulfieldbusSpecific4 = 1; /* subslot */

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

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

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

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

6.2 Connection Establishment

After the stack has been configured, the device is ready for operation.

Bus On State

If auto start is disabled, the application has to switch the device to Bus On. If auto start is enabled, the device automatically reaches the Bus On state. In Bus On state, the device waits for incoming events from the network.

NameOfStation

If the device is activated for the first time, usually NameOfStation has not been configured yet in the device. The NameOfStation stored in the device must match to the configuration of the controller in order that the controller can communicate with the device. Therfore the NameOfStation needs to be set by the engineering tool to the device or will be assigned by the controller according to the topology. For details about allowed characters, see section *Name encoding* [▸ page 323].

The following figure shows the sequence of DCP Set NameOfStation if using an engineering tool.

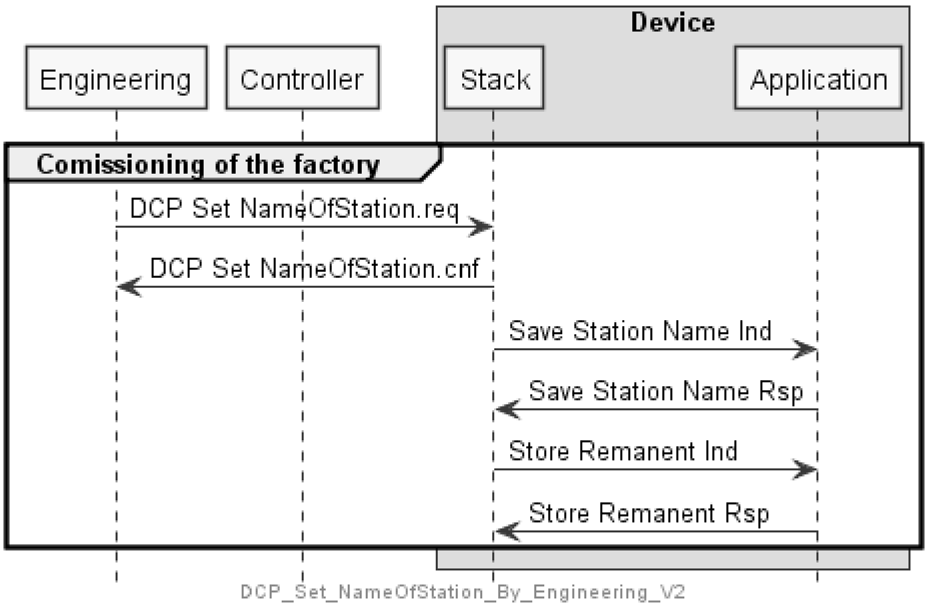


Figure 9: Initial configuration: DCP Set NameOfStation (by engineering tool)

© Hilscher 2011 – 2019

In case, the controller has the topology information of the network, the controller can set the NameOfStation. The following figure shows the sequence of DCP Set NameOfStation by the controller.

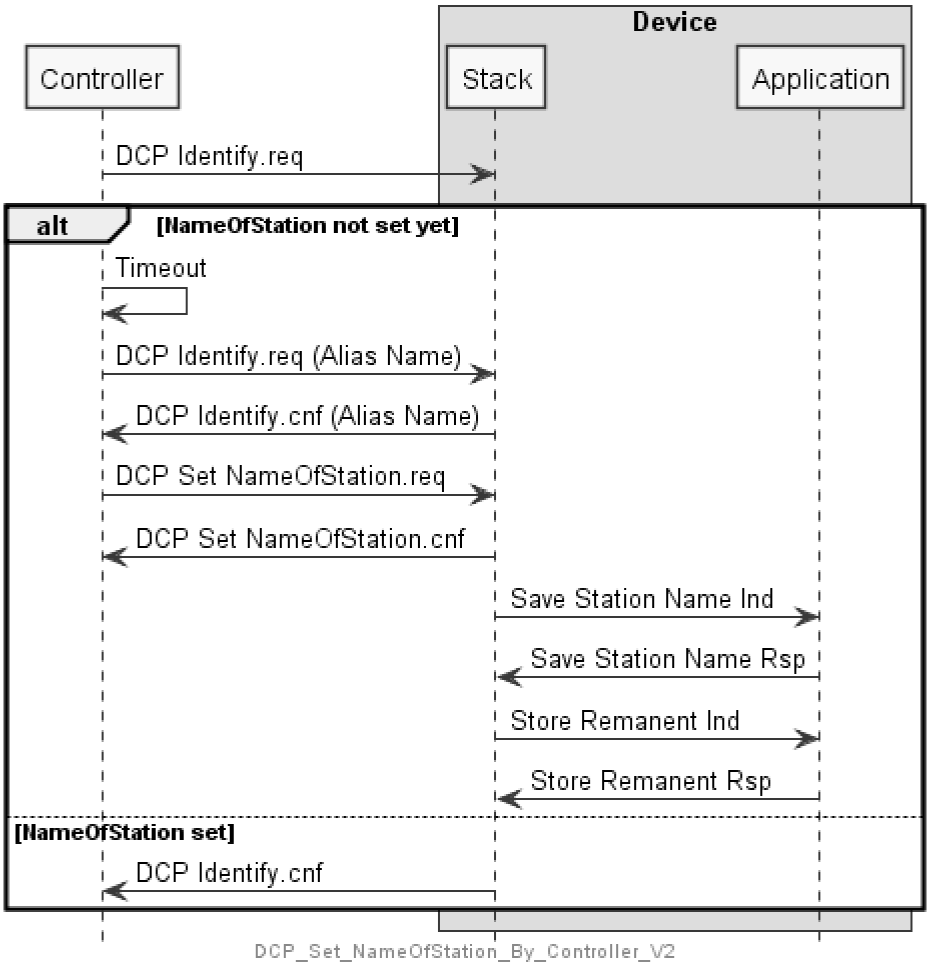

Device
Controller
Stack
Application
DCP Identify.req
alt
[NameOfStation not set yet]
Timeout
DCP Identify.req (Alias Name)
DCP Identify.cnf (Alias Name)
DCP Set NameOfStation.req
DCP Set NameOfStation.cnf
Save Station Name Ind
Save Station Name Rsp
Store Remanent Ind
Store Remanent Rsp
[NameOfStation set]
DCP Identify.cnf
DCP_Set_NameOfStation_By_Controller_V2

Figure 10: Initial configuration: DCP Set NameOfStation (by controller)

IP parameters

In the next step, the controller will verify the IP parameters of the device and adapt them according to its configuration.

The next figure shows the DCP protocol to set IP parameters.

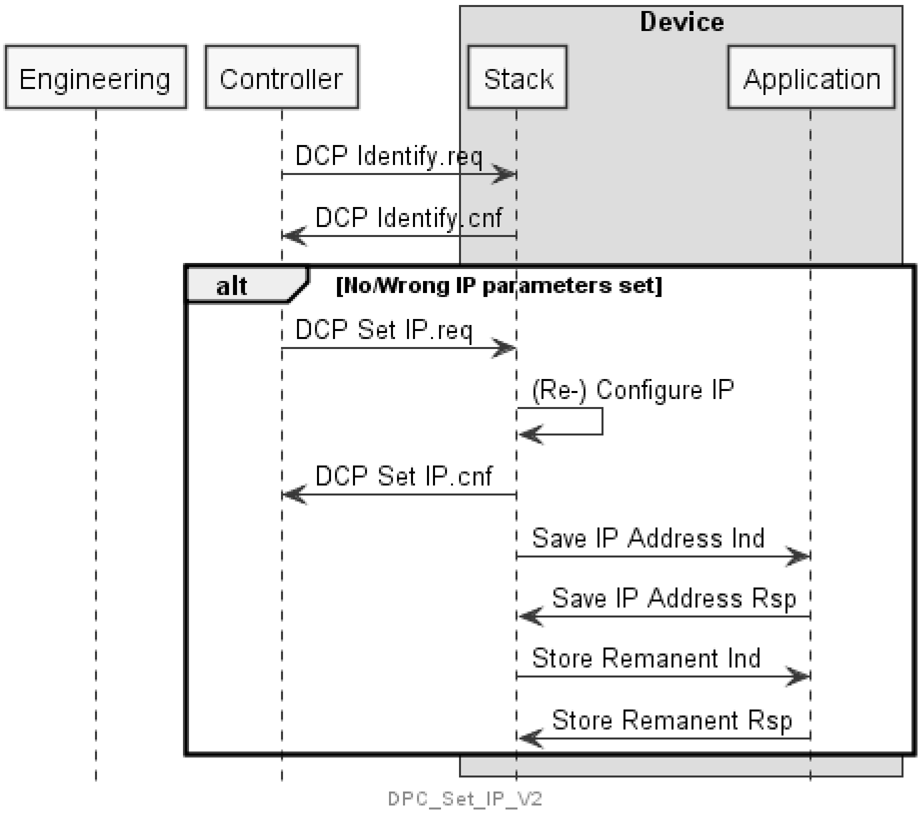


Figure 11: Initial configuration: DCP Set IP

AR establishment

After the IP parameters have been set in the device, the IP stack of the device is active and the RPC layer accepts incoming requests from the network. The controller will now establish an AR. If the application registered at the stack previously, several indications will be generated. The application has to handle these indications properly in order to establish the connection successfully.

AR establishment: Connect

The very first indication will be the AR Check indication (see section *AR Check service* [▸ page 102]). This informative indication signifies the beginning of the AR establishment phase.

In the next step, the stack compares the module configuration requested by the controller with the current module configuration of the stack. Depending on the stack configuration parameters, the stack will now generate a Check Indication (see section *Check Indication service* [▸ page 105]) for each used, wrong, unused or missing (sub) module. If desired, the application has the opportunity to reconfigure the stack. In order to do so, the application should use the Plug/Pull services before returning the Check Response back to the stack. Additionally, the Check Response provides the ability to report a substitute (sub) module.

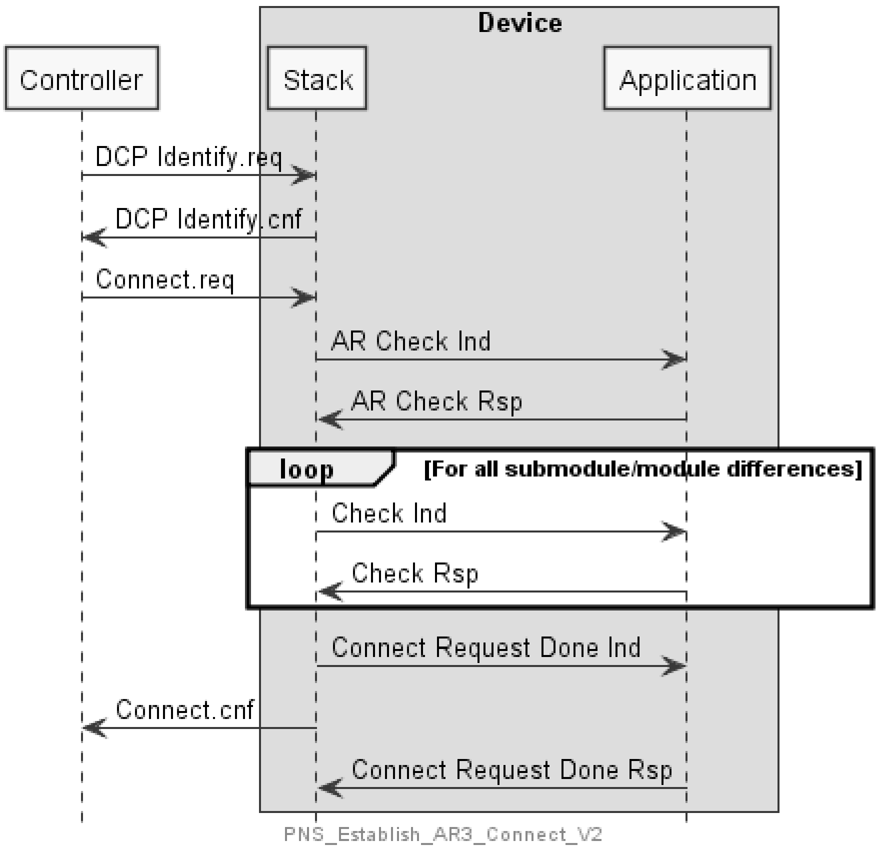


Figure 12: Connection Establishment: Connect

After all Check Indications have been processed, the stack will generate the Connect.cnf to the PROFINET Controller and issue a Connect Request Done indication (see section *Connect Request Done service* [▸ page 111]) to the application. This informative indication signifies the end of the first phase.

AR establishment: Write Parameters

The PROFINET Controller will now parameterize the device by writing the parameter records of the valid (sub-) modules. If parameter records have been specified in the device description file (GSDML), Write Record indications (see section *Write Record service* [▸ page 129]) will be generated for each record written by the controller. This data has to be validated and evaluated by the application.

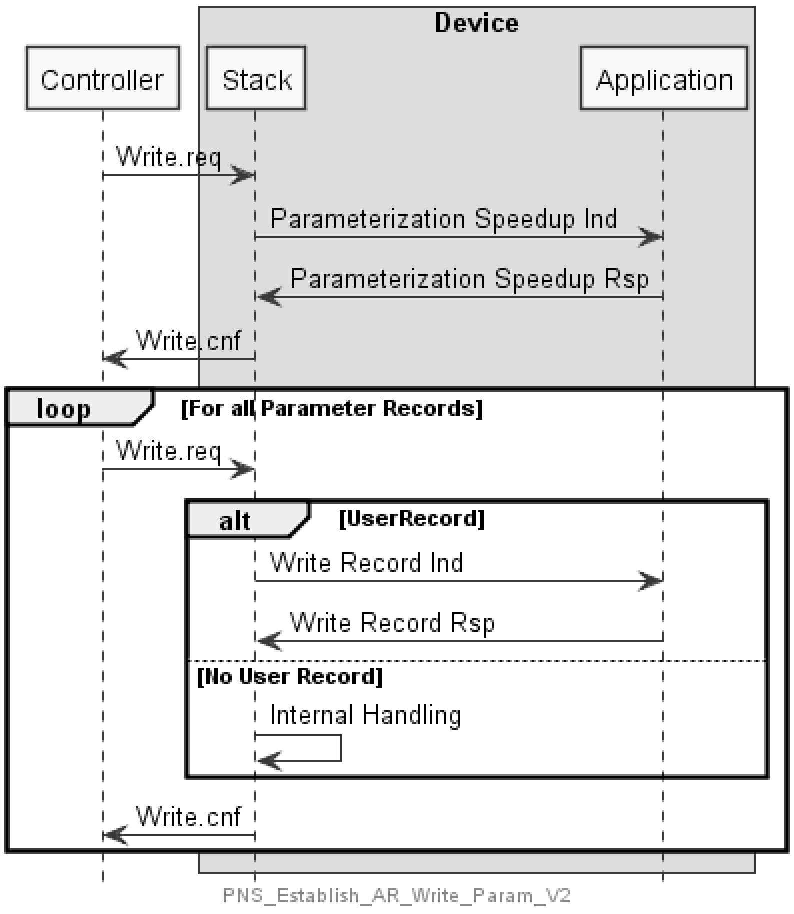


Figure 13: Connection Establishment: Write Parameter

For **advanced start-up**: The stack sends a Parameterization Speedup indication to the application before it has received a **ParamEnd** control message from the IO-Controller if this AR uses FastStartUp.

For **legacy start-up**: As soon as the PROFINET Controller has finished writing the parameters, it will generate a **ParamEnd** control message. At first, the stack will check if this AR uses FastStartUp (FSU). This will be indicated to the application by issuing a Parameterization Speedup indication with non-zero UUID.

If the application has received a Parameterization Speedup indication with non-zero UUID, the application must store the UUID and the parameter records into a non-volatile memory in order to restore them on next power up. Use the UUID value to detect parameter changes in that case.

AR establishment: Parameter End and Application Ready

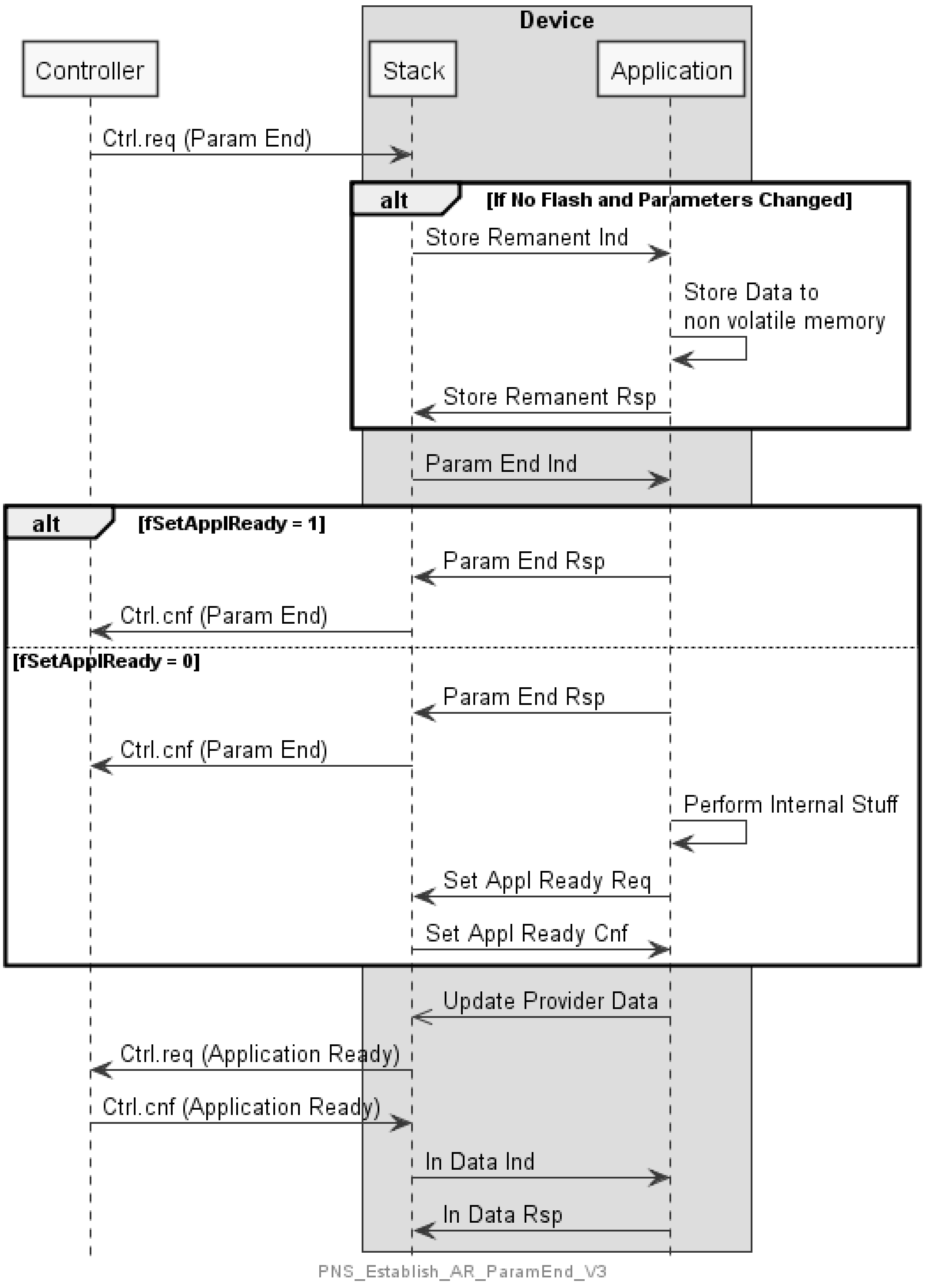


Figure 14: Connection Establishment: Parameter End and Application Ready

The stack generates a Parameter End indication (see section *Parameter End service* [▹ page 113]). At this time, the application has to apply all written parameter data to its internal logic/hardware. Only if parameters have changed or non-FSU mode is in use. Otherwise, the parameter should have been restored from non-volatile memory at power on already. When finished, the application has to generate the Parameter End response. If the application will need some further initialization time, the

© Hilscher 2011 – 2019

application may explicitly request Application Ready by setting fSendApplReady to False and generating an Set_ApplReady_Req later on (see section *Application Ready service* [▸ page 116]).

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

Finally, the AR InData indication (see section *AR InData service* [▸ page 119]) informs the application about the fact that the first cyclic frame from the IO-Controller has been received after the Application Ready confirmation. The AR is now established and valid data exchange takes place.

Note:
As the stack supports multiple ARs, multiple connection establishment sequences may occur in the same time. In order to differentiate between these ARs, use the device handle parameter of the indications.

6.2.1 Service overview

Service	Command code (REQ or IND)	Command code (CNF or RES)
AR Check service [▸ page 102]	0x1F14	0x1F15
Check Indication service [▸ page 105]	0x1F16	0x1F17
Connect Request Done service [▸ page 111]	0x1FD4	0x1FD5
Parameter End service [▸ page 113]	0x1F0E	0x1F0F
Application Ready service [▸ page 116]	0x1F10	0x1F11
AR InData service [▸ page 119]	0x1F28	0x1F29
Store Remanent Data service [▸ page 121]	0x1FEA	0x1FEB

Table 52: Connection establishment service overview

6.2.2 AR Check service

The AR Check Service is used by the protocol stack to indicate that a new AR is going to be established. The protocol stack will generate it right after receiving and checking the PROFINET connect request.

6.2.2.1 AR Check indication

This packet contains information about the AR to be established. The stack sends this packet to the application.

Only evaluate the three last parameters of this packet if `usARType == 0x20`.

Note:
The PROFINET IO Device stack V3 does not support these three last parameters. They will always be set to zero.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	272	Packet data length in bytes
ulCmd	uint32_t	0x1F14	PNS_IF_AR_CHECK_IND
Data			
hDeviceHandle	uint32_t		The device handle.
usARType	uint16_t		Connect request's AR-Type
ulARProperties	uint32_t		Connect request's AR-Properties.
ulRemoteIpAddr	uint32_t		IO-Controller's IP address.
usRemoteNameOfStationLen	uint16_t	1..240	Length of IO-Controller's NameOfStation.
abRemoteNameOfStation[240]	uint8_t		IO-Controller's NameOfStation as ASCII byte-array.
tCmInitiatorObjUUID	HIL_UUID_T		The ContextManagement Initiator Object UUID used by IO-Controller for this AR.
tARUUID	HIL_UUID_T		AR ARUUID
usRdhtValue	uint16_t		RedundancyDataHoldTimer of the whole AR-Set
bARnumber	uint8_t		Maximum number of ARs in the SR-AR-set

Table 53: PNS_IF_AR_CHECK_IND_T - AR Check indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_AR_CHECK_IND_DATA_Ttag
{
  uint32_t hDeviceHandle;
  uint16_t usARType;
  uint32_t ulARProperties;
  uint32_t ulRemoteIpAddr;
  uint16_t usRemoteNameOfStationLen;
  uint8_t abRemoteNameOfStation[PNS_IF_MAX_NAME_OF_STATION];
  HIL_UUID_T tCmInitiatorObjUUID;

  /** The following fields are not supported by PNSv3 implementation
and only valid for PNSv4 implementation */
  /** The following fields shall only be evaluated if usARType ==
0x20 (IOCARSR: System redundancy or dynamic reconfiguration). */
  /* AR ARUUID */
  HIL_UUID_T tARUUID;
  /* RedundancyDataHoldTimer of the whole AR-Set */
  uint16_t usRdhtValue;
  /** Maximum number of ARs in the SR-AR-set */
  uint8_t bARnumber;
} __HIL_PACKED_POST PNS_IF_AR_CHECK_IND_DATA_T;

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

usARType

Value	AR Type
0x01	IO Controller AR with RT
0x06	IO Supervisor AR • Supervisor DA AR (ulARProperties.DeviceAccess = 1) • Supervisor AR (ulARProperties.DeviceAccess = 0)
0x10	IO Controller AR with IRT
0x20	SystemRedundancy AR

Table 54: AR Types

ulARProperties

Bit	Name	AR Properties
3	Supervisor Takeover allowed	0: IO-Controller does not allow take over 1: IO-Controller allowed take over
4	ParameterizationServer	0: External ParameterizationServer is used 1: IO-Controller is ParameterizationServer
8	DeviceAccess	1: IO Supervisor AR is of Type DeviceAccess
30	Startup Mode	0: Legacy Startup is used 1: Advanced Startup is used
all others	-	Not to be evaluated by the application. Values: 0 or 1.

Table 55: AR Properties

6.2.2.2 AR Check response

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet data length in bytes
ulState	uint32_t	0	Status has to be ok for this service.
ulCmd	uint32_t	0x1F15	PNS_IF_AR_CHECK_RES
Data			
hDeviceHandle	uint32_t		The device handle.

Table 56: PNS_IF_AR_CHECK_RSP_T - AR Check response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_DATA_Ttag
{
  /** The device handle */
  uint32_t hDeviceHandle;
} __HIL_PACKED_POST PNS_IF_HANDLE_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_PACKET_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_HANDLE_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T PNS_IF_AR_CHECK_RSP_T;
```

6.2.3 Check Indication service

If the PROFINET IO-Controller expects different submodules as configured in the PROFINET IO-Device stack, the stack sends the *Check Indication* [▸ page 107] service to the application. The indication contains all parameters the controller expected along with the current module and submodule state. If the submodule state is indicated as `PNIO_SUBSTATE_WRONG_SUBMODULE` the application may use the submodule state `PNIO_SUBSTATE_SUBSTITUTE_SUBMODULE` in the response to tell the stack that the configured submodule is able to perform like the requested submodule. Alternatively, if the application is designed to adapt to the requested configuration, the application may reconfigure the stack on check indication.

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

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

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

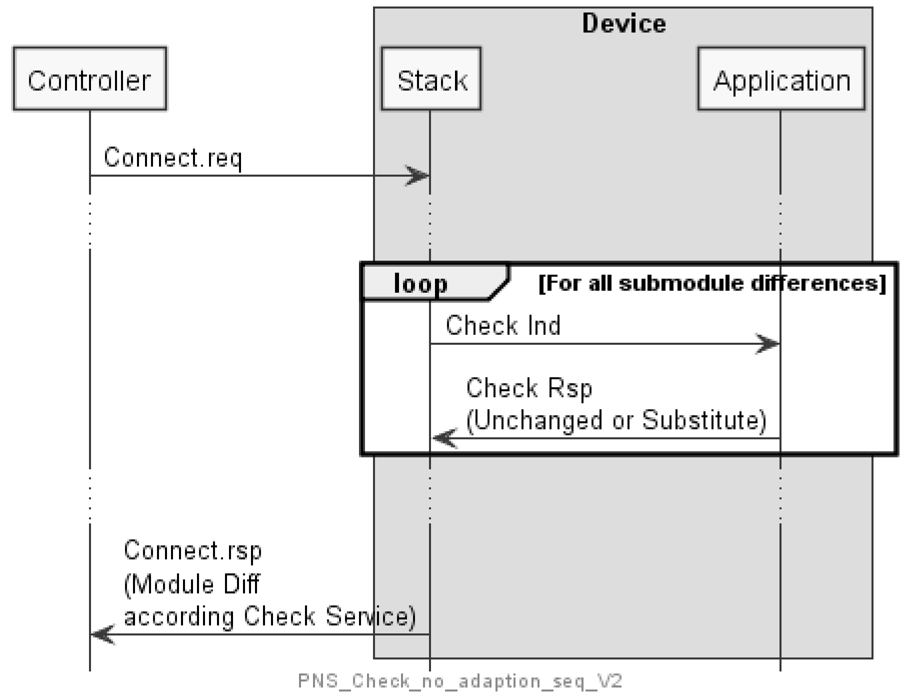


Figure 15: Check Service Packet sequence for non-adapting applications

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

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

The packet sequence then is

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

The next figure shows the sequence for adaption:

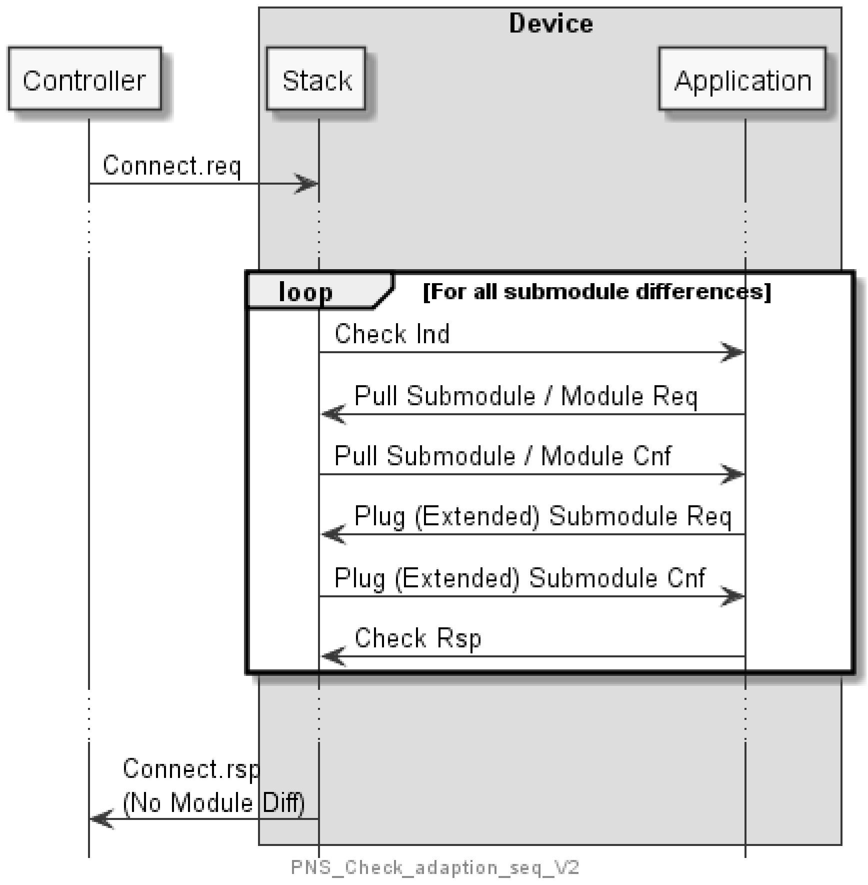

Device
Controller
Stack
Application
Connect.req
loop
[For all submodule differences]
Check Ind
Pull Submodule / Module Req
Pull Submodule / Module Cnf
Plug (Extended) Submodule Req
Plug (Extended) Submodule Cnf
Check Rsp
Connect.rsp
(No Module Diff)
PNS_Check_adaption_seq_V2

Figure 16: Check Service Packet sequence for adapting applications

6.2.3.1 Check Indication

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	32	Packet data length in bytes
ulCmd	uint32_t	0x1F16	PNS_IF_CHECK_IND
Data			
hDeviceHandle	uint32_t		The device handle.
ulApi	uint32_t		The API of the wrong submodule.
ulSlot	uint32_t		The slot of the wrong submodule.
ulSubslot	uint32_t		The subslot of the wrong submodule.
ulModuleId	uint32_t		The Module ID the IO-Controller expected
usModuleState	uint16_t		The Module State suggested by the stack (see table *Values of usModuleState in check indication* [▸ page 108]).
ulSubmodId	uint32_t		The Submodule ID the IO-Controller expected.
usSubmodState	uint16_t		The Submodule state suggested by the stack (see table *Values of usSubmodState in check indication* [▸ page 108]).
usExpInDataLen	uint16_t		The length of input data IO-Controller expects for the submodule. This is only informative for application.
usExpOutDataLen	uint16_t		The length of output data IO-Controller expects for the submodule. This is only informative for application.

Table 57: PNS_IF_CHECK_IND_T - Check Indication

Packet structure reference

```
/** Module state code */
typedef enum PNIO_MODSTATE_Etag /* Module state */
{
  PNIO_MODSTATE_NO_MODULE = 0, /**< no module plugged in slot */
  PNIO_MODSTATE_WRONG_MODULE, /**< wrong module plugged in slot */
  PNIO_MODSTATE_PROPER_MODULE, /**< proper module */
  PNIO_MODSTATE_SUBSTITUTE_MODULE, /**< substitute, the wrong module
can fulfill requirements */
  PNIO_MODSTATE_UNUSED_MODULE, /**< module not expected by
controller */
  PNIO_MODSTATE_CORRECT_MODULE = 0xFFFF, /**< correct module was
plugged by application */
} PNIO_MODSTATE_E;

/** submodule state code */
typedef enum PNIO_SUBSTATE_Etag /* Submodule state */
{
  PNIO_SUBSTATE_NO_SUBMODULE = 0, /**< no submodule */
  PNIO_SUBSTATE_WRONG_SUBMODULE, /**< Wrong submodule */
  PNIO_SUBSTATE_PROPER_SUBMODULE, /**< locked by IO controller */
  PNIO_SUBSTATE_RESERVED_1, /**< reserved */
  PNIO_SUBSTATE_APPL_READY_PENDING, /**< application ready pending
*/
  PNIO_SUBSTATE_RESERVED_2, /**< reserved */
  PNIO_SUBSTATE_RESERVED_3, /**< reserved */
  PNIO_SUBSTATE_SUBSTITUTE_SUBMODULE, /**< substitute, the wrong
submodule can fulfill requirements */
  PNIO_SUBSTATE_UNUSED_SUBMODULE, /**< submodule not expected by
controller */
  PNIO_SUBSTATE_CORRECT_SUBMODULE = 0xFFFF, /**< correct submodule
was plugged by application*/
} PNIO_SUBSTATE_E;

typedef __HIL_PACKED_PRE struct PNS_IF_CHECK_IND_IND_DATA_Ttag
{
  uint32_t hDeviceHandle;
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t ulModuleId;
  uint16_t usModuleState;
  uint32_t ulSubmodId;
  uint16_t usSubmodState;
  uint16_t usExpInDataLen;
  uint16_t usExpOutDataLen;
} __HIL_PACKED_POST PNS_IF_CHECK_IND_IND_DATA_T;

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

Definitions used for the field `usModuleState` for indication:

Definition / (Value)	Description
PNIO_MODSTATE_NO_MODULE (0x0)	No module plugged into the slot specified.
PNIO_MODSTATE_WRONG_MODULE (0x1)	A wrong module is plugged into the specified slot.
PNIO_MODSTATE_PROPER_MODULE (0x2)	A proper (the correct) module is plugged into the specified slot but is already used by another IO-Controller and is therefore not accessible.
PNIO_MODSTATE_UNUSED_MODULE (0x4)	A module is plugged into the specified slot but is not requested/used by the IO-Controller.

Table 58: Values of usModuleState in check indication

Definitions used for the field `usSubmodState` field for indication:

Definition / (Value)	Description
PNIO_SUBSTATE_NO_SUBMODULE (0x0)	No submodule plugged into the slot/subslot specified.
PNIO_SUBSTATE_WRONG_SUBMODULE (0x1)	A wrong submodule is plugged into the specified slot/subslot.
PNIO_SUBSTATE_PROPER_SUBMODULE (0x2)	A proper (the correct) submodule is plugged into the specified slot/ subslot but is already used by another IO-Controller. It cannot be used now.
PNIO_SUBSTATE_APPL_READY_PENDING (0x4)	The correct submodule is plugged into the specified slot/subslot but it is not ready for data exchange yet.
PNIO_SUBSTATE_UNUSED_SUBMODULE (0x8)	A submodule is plugged into the specified slot/subslot but it is not requested/used by the IO-Controller.

Table 59: Values of usSubmodState in check indication

6.2.3.2 Check Response

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

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	28	Packet data length in bytes
ulState	uint32_t	0	The status has to be okay for this service.
ulCmd	uint32_t	0x1F17	PNS_IF_CHECK_RES
Data			
hDeviceHandle	uint32_t		The device handle
ulApi	uint32_t		The API of the wrong submodule.
ulSlot	uint32_t		The slot of the wrong submodule.
ulSubslot	uint32_t		The subslot of the wrong submodule.
ulModuleId	uint32_t		The ModuleID the IO-Controller expected
usModuleState	uint16_t	See below	The ModuleState (see table *Values of usModuleState in response packet* [▸ page 110]).
ulSubmodId	uint32_t		The SubmoduleID the IO-Controller expected.
usSubmodState	uint16_t	See below	The SubmoduleState (see table *Values of usSubmodState in response packet* [▸ page 110]). Only the value PNIO_SUBSTATE_SUBSTITUTE_MODULE is honored by the stack. Any other value will lead to default behavior.

Table 60: PNS_IF_CHECK_RSP_T - Check Response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_CHECK_IND_RSP_DATA_Ttag
{
  uint32_t hDeviceHandle;
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t ulModuleId;
  uint16_t usModuleState;
  uint32_t ulSubmodId;
  uint16_t usSubmodState;
} __HIL_PACKED_POST PNS_IF_CHECK_IND_RSP_DATA_T;

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

Definitions to use for the field `usModuleState` field in response:

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

Table 61: Values of usModuleState in response packet

Definitions to use for the field `usSubmodState` field in response:

Definition / (Value)	**Description**
PNIO_SUBSTATE_NO_SUBMODULE (0x0)	No submodule plugged into the slot/subslot specified.
PNIO_SUBSTATE_WRONG_SUBMODULE (0x1)	A wrong submodule is plugged into the specified slot/subslot.
PNIO_SUBSTATE_PROPER_SUBMODULE (0x2)	A proper (the correct) submodule is plugged into the specified slot/ subslot and therefore no configuration adaptation happens.
PNIO_SUBSTATE_SUBSTITUTE_SUBMODULE (0x7)	A substitute submodule is plugged into the specified slot/subslot. Substitute submodules can offer the same functionality as the originally requested submodule.
PNIO_SUBSTATE_UNUSED_SUBMODULE (0x8)	A submodule is plugged into the specified slot/subslot but it is not requested/used by the IO-Controller.
PNIO_MODSTATE_CORRECT_SUBMODULE (0xFFFF)	A correct submodule is plugged into the specified slot/subslot on adaptation of the submodule configuration by application.

Table 62: Values of usSubmodState in response packet

6.2.4 Connect Request Done service

The stack sends the Connect Request Done indication to the application to indicate that no more Check Indications will be sent from the stack to the application.

6.2.4.1 Connect Request Done indication

This packet indicates that the Connect Request from the IO-Controller was parsed by stack. The stack is waiting for the corresponding application generated response. Without this response the Connect Response on the network is not generated.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet data length in bytes
ulCmd	uint32_t	0x1FD4	PNS_IF_CONNECT_REQ_DONE_IND
Data			
hDeviceHandle	uint32_t		The device handle

Table 63: PNS_IF_CONNECTREQ_DONE_IND_T - Connect Request Done indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_DATA_Ttag
{
  /** The device handle */
  uint32_t hDeviceHandle;
} __HIL_PACKED_POST PNS_IF_HANDLE_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_PACKET_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_HANDLE_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T PNS_IF_CONNECTREQ_DONE_IND_T;
```

6.2.4.2 Connect Request Done response

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet data length in bytes
ulSta	uint32_t	0	The status has to be okay for this service.
ulCmd	uint32_t	0x1FD5	PNS_IF_CONNECT_REQ_DONE_RES
Data			
hDeviceHandle	uint32_t		The device handle

Table 64: PNS_IF_CONNECTREQ_DONE_RSP_T - Connect Request Done response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_DATA_Ttag
{
  /** The device handle */
  uint32_t hDeviceHandle;
} __HIL_PACKED_POST PNS_IF_HANDLE_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_PACKET_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_HANDLE_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T PNS_IF_CONNECTREQ_DONE_RSP_T;
```

6.2.5 Parameter End service

As soon as the IO-Controller has finished parameterizing the IO-Device, it sends the Parameter End command. The stack indicates this to the application.

1. According to the PROFINET specification, with reception of this indication the application should first check the parameters against each other. Here, the application must quickly decide whether applying the parameters will last more than 300 seconds.
2. If for any reason the application requires additional time to get ready, the application shall set `fSendApplicationReady` to *false* and respond to the indication immediately.
3. Next the application should generate the Parameter End response. Finally, the application shall start configuring its submodules with the parameters from the write indications having been received. If no write indication was received, nothing has to be done.
4. If the application is ready for cyclic process data exchange and it did not request additional time, answer with `fSendApplicationReady` set to *true* in order to indicate this to the stack.

If the application requested additional time within step 2:
When later on the application gets ready, it has to use the *Application Ready service* [▸ page 116] to indicate this to the stack.

Note:
Before the stack will signal Application Ready to the IO-Controller, the application must provide valid I/O data to the stack. Therefore the application is required to write the I/O data after returning the Parameter End response with `fSendApplicationReady` set to true.

Note:
For this service, a timeout is implemented in the stack. If the application does not answer within the timeout the stack will automatically generate a negative response the IO-Controller.

6.2.5.1 Parameter End indication

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	12	Packet data length in bytes
ulCmd	uint32_t	0x1F0E	PNS_IF_PARAM_END_IND
Data			
hDeviceHandle	uint32_t		The device handle
ulApi	uint32_t		The API of the module whose parameterization is finished. Only valid if usSubslot != 0.
usSlot	uint16_t		The slot of the module whose parameterization is finished. Only valid if usSubslot != 0.
usSubslot	uint16_t	0	Parameterization is finished for either • all submodules being part of the connection currently established (indicated via PNS_IF_AR_CHECK_IND before) or • all submodules being part of a previous Parameter Begin Indication (in case of SystemRedundancy).
		!= 0	Parameterization is finished for the module specified by ulApi, usSlot and usSubslot.

Table 65: PNS_IF_PARAM_END_IND_T - Parameter End indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_PARAM_END_IND_DATA_Ttag
{
  uint32_t hDeviceHandle;
  uint32_t ulApi; /* valid only if usSlot != 0 */
  uint16_t usSlot; /* valid only if usSubslot != 0 */
  uint16_t usSubslot; /* 0: for all (sub)modules, != 0: for this
specific submodule */
} __HIL_PACKED_POST PNS_IF_PARAM_END_IND_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_PARAM_END_IND_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_PARAM_END_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_PARAM_END_IND_T;
```

6.2.5.2 Parameter End response

This packet has to be returned to the stack. Depending on `fSendApplicationReady` the stack will automatically send the command Application Ready to the IO-Controller.

Note:
It is not allowed to send Application ready on the network until the IOPS and IOCS of all submodules are set to good or a diagnosis has been added for bad submodules.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	8	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▹ page 292].
ulCmd	uint32_t	0x1F0F	PNS_IF_PARAM_END_RES
Data			
hDeviceHandle	uint32_t		Handle to the IO-Device.
fSendApplicationReady	uint32_t	FALSE (0)	The stack shall not automatically send ApplicationReady. This will be initiated by Application using the Service described in section *Application Ready service* [▹ page 116].
		TRUE	The stack shall automatically send ApplicationReady.

Table 66: PNS_IF_PARAM_END_RSP_T - Parameter End response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_PARAM_END_RSP_DATA_Ttag
{
  uint32_t hDeviceHandle;
  uint32_t fSendApplicationReady; /* set to TRUE to send ApplReady
automatically */
} __HIL_PACKED_POST PNS_IF_PARAM_END_RSP_DATA_T;

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

6.2.6 Application Ready service

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

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

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

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

6.2.6.1 Application Ready request

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet data length in bytes
ulCmd	uint32_t	0x1F10	PNS_IF_SET_APPL_READY_REQ
Data			
hDeviceHandle	uint32_t		The device handle representing the AR for which Application Ready should be signaled. Use the device handle from associated Parameter End Indication.

Table 67: PNS_IF_APPL_READY_REQ_T - Application Ready request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_DATA_Ttag
{
  /** The device handle */
  uint32_t hDeviceHandle;
} __HIL_PACKED_POST PNS_IF_HANDLE_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_PACKET_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_HANDLE_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T PNS_IF_APPL_READY_REQ_T;
```

6.2.6.2 Application Ready confirmation

The stack will respond with this packet.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F11	PNS_IF_SET_APPL_READY_CNF
Data			
hDeviceHandle	uint32_t	0	The device handle.

Table 68: PNS_IF_APPL_READY_CNF_T - Application Ready confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_DATA_Ttag
{
  /** The device handle */
  uint32_t hDeviceHandle;
} __HIL_PACKED_POST PNS_IF_HANDLE_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_PACKET_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_HANDLE_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T PNS_IF_APPL_READY_CNF_T;
```

6.2.7 AR InData service

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

6.2.7.1 AR InData indication

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet data length in bytes
ulCmd	uint32_t	0x1F28	PNS_IF_AR_INDATA_IND
Data			
hDeviceHandle	uint32_t		The device handle

Table 69: PNS_IF_AR_IN_DATA_IND_T - AR InData indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_DATA_Ttag
{
  /** The device handle */
  uint32_t hDeviceHandle;
} __HIL_PACKED_POST PNS_IF_HANDLE_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_PACKET_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_HANDLE_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T PNS_IF_AR_IN_DATA_IND_T;
```

6.2.7.2 AR InData response

The application has to return this packet.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet data length in bytes
ulSta	uint32_t	0	The status has to be okay for this service.
ulCmd	uint32_t	0x1F29	PNS_IF_AR_INDATA_RES
Data			
hDeviceHandle	uint32_t		The device handle

Table 70: PNS_IF_AR_IN_DATA_RSP_T - AR InData response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_DATA_Ttag
{
  /** The device handle */
  uint32_t hDeviceHandle;
} __HIL_PACKED_POST PNS_IF_HANDLE_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_PACKET_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_HANDLE_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T PNS_IF_AR_IN_DATA_RSP_T;
```

6.2.8 Store Remanent Data service

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

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

This service is tightly coupled with the following services:

1. RPC Parameter End Service, see *Parameter End service* [▸ page 113].
2. DCP Set Station Name, see *Save Station Name service* [▸ page 137].
3. DCP Set IP Address, see *Save IP Address service* [▸ page 140].
4. DCP Reset to Factory, see *Reset Factory Settings service* [▸ page 146].

The application always must respond. Otherwise, these services will not work correctly.

6.2.8.1 Store Remanent Data indication

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

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

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

In case, large amounts of data are transferred, this data will be divided to multiple response packets, which have to be evaluated one by another.

If the application receives this indication, it has to compare old and new data.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	1 + n	Packet data length in bytes
ulSta	uint32_t	0	The status has to be ok for this service.
ulCmd	uint32_t	0x1FEA	PNS_IF_STORE_REMANENT_DATA_IND
Data			
abData[1]	uint8_t[]		The remanent data to be stored by application. Only the first byte is shown in the packet definition. The application has to store the amount of bytes reported by this packet header's field ulLen.

Table 71: PNS_IF_STORE_REMANENT_DATA_IND_T - Store Remanent Data indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_STORE_REMANENT_DATA_IND_DATA_Ttag
{
  /** this is only the first byte, many others may follow */
  uint8_t abData[1];
} __HIL_PACKED_POST PNS_IF_STORE_REMANENT_DATA_IND_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_STORE_REMANENT_DATA_IND_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_STORE_REMANENT_DATA_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_STORE_REMANENT_DATA_IND_T;
```

6.2.8.2 Store Remanent Data response

The application has to return this packet on reception of the PNS_IF_STORE_REMANENT_DATA_IND_T indication.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t	0	The status has to be okay for this service.
ulCmd	uint32_t	0x1FEB	PNS_IF_STORE_REMANENT_DATA_RES

Table 72: PNS_IF_STORE_REMANENT_DATA_RES_T - Store Remanent Data response

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_STORE_REMANENT_DATA_RES_T;
```

6.3 Acyclic Events indicated by the Stack

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

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

Note:
If the netX COM LEDs are present, these are managed by the protocol stack and this service is informative in this case. In all other cases the implementation is mandatory and relevant for certification.

6.3.1 Service overview

The following table lists all packets of acyclic events indicated by the stack.

Service	Command code (REQ or IND)	Command code (CNF or RES)
Read Record service [▸ page 125]	0x1F36	0x1F37
Write Record service [▸ page 129]	0x1F3A	0x1F3B
AR Abort Indication service [▸ page 134]	0x1F2A	0x1F2B
Save Station Name service [▸ page 137]	0x1F1A	0x1F1B
Save IP Address service [▸ page 140]	0x1FB8	0x1FB9
Start LED Blinking service [▸ page 143]	0x1F1E	0x1F1F
Stop LED Blinking service [▸ page 145]	0x1F20	0x1F21
Reset Factory Settings service [▸ page 146]	0x1F18	0x1F19
APDU Status Changed service [▸ page 152]	0x1F2E	0x1F2F
Alarm Indication service [▸ page 154]	0x1F30	0x1F31
Link Status Changed service [▸ page 158]	0x1F70	0x1F71
Error Indication service [▸ page 161]	0x1FDC	0x1FDD
Read I&M service [▸ page 162]	0x1F32	0x1F33
Write I&M service [▸ page 169]	0x1F34	0x1F35
Parameterization Speedup Support service [▸ page 178]	0x1FF8	0x1FF9
Event Indication service [▸ page 180]	0x1FFE	0x1FFF

Table 73: Service overview of acyclic events indicated by the PROFINET IO Device stack

© Hilscher 2011 – 2019

6.3.2 Read Record service

With the Read Record service, the stack indicates the reception of a Read Record request from the IO-Controller. The stack provides all parameters to the application, which are necessary to answer the request such as slot, subslot and index. The application has to return the Read Record response packet to the stack so that the stack can send the Read response to the PROFINET IO-Controller.

Note:
For this service, the stack uses a timeout. If the application does not answer within the timeout period, the stack automatically generates a negative response the IO-Controller.

Note:
The LOM target supports up to 32 KB of response data payload for read records. The maximum amount of the data the packet can actually hold is indicated by the indication's `ulLenToRead` field. In order to avoid memory corruption, the aplication must not exceed this upper size limit.

If the stack is accessed via Dual-Port Memory, due to the limited mailbox size the application has to use packet fragmentation for the response packet. The handling of packet fragmentation is described in section "General packet fragmentation" in reference [5].

6.3.2.1 Read Record indication

This packet indicates the reception of a *Read Record* request by the stack to the application.

Packet Description

Variable	Type	Value / Range	Description
ulLen	uint32_t	32	Packet data length in bytes
ulCmd	uint32_t	0x1F36	PNS_IF_READ_RECORD_IND
Data			
hRecordHandle	uint32_t		A stack internal identifier, which belongs to this indication.
hDeviceHandle	uint32_t		The device handle.
ulSequenceNum	uint32_t		The sequence number used by IO-Controller for this Read Record Request.
ulApi	uint32_t		The API the IO-Controller wants to read.
ulSlot	uint32_t		The slot the IO-Controller wants to read.
ulSubslot	uint32_t		The subslot the IO-Controller wants to read.
ulIndex	uint32_t		The index the IO-Controller wants to read.
ulLenToRead	uint32_t	1..n	The number of bytes the IO-Controller requested. If the stack is accessed using DPM, n may be up to 1024 Bytes. If the Protocol Stacks AP Queue is directly programmed, n may be up to 32768.

Table 74: PNS_IF_READ_RECORD_IND_T - Read Record indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_READ_RECORD_IND_DATA_Ttag
{
  uint32_t hRecordHandle;
  uint32_t hDeviceHandle;
  uint32_t ulSequenceNum;
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t ulIndex;
  uint32_t ulLenToRead;
} __HIL_PACKED_POST PNS_IF_READ_RECORD_IND_DATA_T;

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

6.3.2.2 Read Record response

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

The application will not receive any feedback whether the response has reached the IO-Controller or not. However, if the Read Record response packet received by the stack contains wrong values, the stack will indicate this to the IO-Controller using a User Error indication.

Note:
The data provided by device to the PROFINET network via the read record response (`abRecordData[1024]`) are delivered in **big-endian** representation. Big-endian means, that the highest value byte is stored first, i.e. at the lowest memory address (Motorola format).

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	40 + n	Packet data length in bytes. n is the value of ulReadLen.
ulCmd	uint32_t	0x1F37	PNS_IF_READ_RECORD_RES
Data			
hRecordHandle	uint32_t		A stack internal identifier which belongs to this indication.
hDeviceHandle	uint32_t		The device handle
ulSequenceNum	uint32_t		The sequence number passed in the indication.
ulApi	uint32_t		The API the IO-Controller wants to read.
ulSlot	uint32_t		The slot the IO-Controller wants to read.
ulSubslot	uint32_t		The subslot the IO-Controller wants to read.
ulIndex	uint32_t		The index the IO-Controller wants to read.
ulReadLen	uint32_t	0..n	The record data length read. N shall be smaller or equal than value of indication's ulLenToRead field.
ulPnio	uint32_t		PROFINET error code, consists of ErrorCode, ErrorDecode, ErrorCode1 and ErrorCode2. See section "PROFINET Status Code".
usAddValue1	uint16_t		Additional Value 1. Value has to be 0 if ulPnio is 0. In case of error, ulPnio is not 0, this value can be used to report customer-specific error information.
usAddValue2	uint16_t		Additional Value 2. Value has to be 0 if ulPnio is 0. In case of error, ulPnio is not 0, this value can be used to report customer-specific error information.
abRecordData[]	uint8_t[]		Read record data. The array must not be larger than the value in indication's ulLenToRead field.

Table 75: PNS_IF_READ_RECORD_RSP_T - Read Record response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_READ_RECORD_RSP_DATA_Ttag
{
  uint32_t hRecordHandle;
  uint32_t hDeviceHandle;
  uint32_t ulSequenceNum;
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t ulIndex;
  uint32_t ulReadLen;
  /* Profinet error code, consists of ErrCode, ErrDecode, ErrCode1
and ErrCode2 */
  uint32_t ulPnio;
  uint16_t usAddValue1;
  uint16_t usAddValue2;
  uint8_t abRecordData[]; /* ATTENTION: in case of LOM the length of
abRecordData may be up to PNS_IF_MAX_RECORD_DATA_LEN bytes (BUT
never more than ulLenToRead)*/
} __HIL_PACKED_POST PNS_IF_READ_RECORD_RSP_DATA_T;

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

© Hilscher 2011 – 2019

6.3.3 Write Record service

With the Write Record service, the stack indicates the reception of a Write Record request from the IO-Controller. The stack provides the application with all parameters contained in the request (such as slot, subslot, index and the data. It has to handle the data (e.g. send it to configure modules). It has to return the Write Record response packet to the stack so that the stack can answer the request from the IO-Controller.

If the application receives the Write Record during connection establishment, the application should wait for reception of the Parameter End indication instead of trying to configure the submodule instantly.

Note:
The LFW target supports by default up 4 KB of indication data payload for write records. For this, the data must be transferred using the fragmentation transfer. The application might pre-calculate the size of the unfragmented packet using the data field `ulLentoWrite` which is transferred in the first fragment. This also requires an entry within the tag list.

If the stack is accessed via Dual Port Memory, due to the limited mailbox size the application may need to use packet fragmentation for the indication packet. The handling of packet fragmentation is described in section “General packet fragmentation” in reference [5].

6.3.3.1 Write Record indication

Using this packet, the stack indicates the reception of a Write Request to the application. It contains the data sent by the IO-Controller.

Note:
The data received from the PROFINET network via the write record indication (`abRecordData[1024]`) are delivered in **big-endian** representation. Big-endian means, that the highest value byte is stored first, i.e. at the lowest memory address (Motorola format).

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	32 +n	Packet data length in bytes. n is the value of ulLenToWrite.
ulCmd	uint32_t	0x1F3A	PNS_IF_WRITE_RECORD_IND
Data			
hRecordHandle	uint32_t		A stack internal identifier, which belongs to this indication.
hDeviceHandle	uint32_t		The device handle
ulSequenceNum	uint32_t		The sequence number used by IO-Controller for this Write Record Request.
ulApi	uint32_t		The API the IO-Controller wants to write to.
ulSlot	uint32_t		The slot the IO-Controller wants to write to.
ulSubslot	uint32_t		The subslot the IO-Controller wants to write to.
ulIndex	uint32_t		The index the IO-Controller wants to write to.
ulLenToWrite	uint32_t	1..n	The length of write record data. If the stack is accessed using DPM, n may be up to 1024 Bytes. If the protocol stack's AP Queue is programmed directly, n may be up to 32768.
abRecordData[]	uint8_t[]		The write record data. The actual length of the array depends on ulLenToWrite field.

Table 76: PNS_IF_WRITE_RECORD_IND_T - Write Record indication

© Hilscher 2011 – 2019

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_WRITE_RECORD_IND_DATA_Ttag
{
  /** Stack internal identifier. Do not evaluate */
  uint32_t hRecordHandle;
  /** The requests associated device handle */
  uint32_t hDeviceHandle;
  /** The requests sequence number used by the IO controller */
  uint32_t ulSequenceNum;
  /** The api the record shall be written to */
  uint32_t ulApi;
  /** The slot the record shall be written to */
  uint32_t ulSlot;
  /** The subslot the record shall be written to */
  uint32_t ulSubslot;
  /** The index the record shall be written to */
  uint32_t ulIndex;
  /** The length of the record data */
  uint32_t ulLenToWrite;
  /** The records data */
  uint8_t abRecordData[]; /* ATTENTION: in case of LOM the length of
abRecordData may be up to PNS_IF_MAX_RECORD_DATA_LEN bytes */
} __HIL_PACKED_POST PNS_IF_WRITE_RECORD_IND_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_WRITE_RECORD_IND_Ttag
{
  /** Packet header */
  HIL_PACKET_HEADER_T tHead;
  /** Packet data */
  PNS_IF_WRITE_RECORD_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_WRITE_RECORD_IND_T;
```

6.3.3.2 Write Record response

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

The application will not receive any feedback whether the response has reached the IO-Controller or not. However, if the Write Record response packet received by the stack contains wrong values, the stack will indicate this to the IO-Controller using a User Error indication.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	40	Packet data length in bytes
ulSta	uint32_t		
ulCmd	uint32_t	0x1F3B	PNS_IF_WRITE_RECORD_RES
Data			
hRecordHandle	uint32_t		A stack internal identifier which belongs to this indication.
hDeviceHandle	uint32_t		The device handle
ulSequenceNum	uint32_t		The sequence number passed in the indication.
ulApi	uint32_t		The API the IO-Controller wants to write to.
ulSlot	uint32_t		The slot the IO-Controller wants to write to.
ulSubslot	uint32_t		The subslot the IO-Controller wants to write to.
ulIndex	uint32_t		The index the IO-Controller wants to write to.
ulWriteLen	uint32_t		The record data length written.
ulPnio	uint32_t		PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See section “PROFINET Status Code”.
usAddValue1	uint16_t	0	Additional Value 1. Value has to be 0 if ulPnio is 0. In case of error, ulPnio is not 0, this value can be used to report customer-specific error information.
usAddValue2	uint16_t	0	Additional Value 2. Value has to be 0 if ulPnio is 0. In case of error, ulPnio is not 0, this value can be used to report customer-specific error information.

Table 77: PNS_IF_WRITE_RECORD_RSP_T - Write Record response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_WRITE_RECORD_RSP_DATA_Ttag
{
  /** Stack internal identifier. Use value from indication. */
  uint32_t hRecordHandle;
  /** The requests associated device handle. Use value from
indication. */
  uint32_t hDeviceHandle;
  /** The requests sequence number used by the IO controller. Use
value from indication. */
  uint32_t ulSequenceNum;
  /** The api the record shall be written to. Use value from
indication. */
  uint32_t ulApi;
  /** The slot the record shall be written to. Use value from
indication. */
  uint32_t ulSlot;
  /** The subslot the record shall be written to. Use value from
indication. */
  uint32_t ulSubslot;
  /** The index the record shall be written to. Use value from
indication. */
  uint32_t ulIndex;
  /** The length of the written record data. Use value from
indication. */
  uint32_t ulWriteLen;
  /** The profinet status code. This is the status of the write.
   *
   * Use the following values:
   * - PNIO_S_OK: The write was successfull
   * - PNIO_E_IOD_WRITE_ACCESS_INVALIDINDEX: The addressed submodule
does not support this index
   */
  uint32_t ulPnio;
  /** Additional value for write response */
  uint16_t usAddValue1;
  /** Additional value for write response */
  uint16_t usAddValue2;
} __HIL_PACKED_POST PNS_IF_WRITE_RECORD_RSP_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_WRITE_RECORD_RSP_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_WRITE_RECORD_RSP_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_WRITE_RECORD_RSP_T;
```

6.3.4 AR Abort Indication service

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

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

Note:
An application should NOT use the receipt of this indication as cause to trigger actions like „Channel int“, „Pull (sub)module“ or „System reset“. Otherwise certification oft he device will not be possible.

Note:
The application must send a response.

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

Note:
If bit “Generate Check Indications for unused modules” (D13) in ulSystemFlags of service *Set Configuration request* [▸ page 54] is set and the Shared Device function is used then the stack sends Check Indications after an AR Abort Indication.

Note:
The stack sends an Event Indication with PNS_IF_IO_EVENT_CONSUMER_UPDATE_REQUIRED = 0x00000002 to the application after an AR Abort Indication. The application must read the input data once when receiving this Event Indication.

Note:
Typically, the IO data is already marked as invalid (set to 0 if IOxS is not used) by protocol stack.

6.3.4.1 AR Abort Indication

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	8	Packet data length in bytes
ulCmd	uint32_t	0x1F2A	PNS_IF_AR_ABORT_IND
Data			
hDeviceHandle	uint32_t		The device handle.
ulPnio	uint32_t		PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See section "*PROFINET status codes* [▸ page 311]".

Table 78: PNS_IF_AR_ABORT_IND_IND_T - AR Abort Indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_AR_ABORT_IND_IND_DATA_Ttag
{
  uint32_t hDeviceHandle;
  /* Profinet error code, consists of ErrCode, ErrDecode, ErrCode1
and ErrCode2 */
  uint32_t ulPnio;
} __HIL_PACKED_POST PNS_IF_AR_ABORT_IND_IND_DATA_T;

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

6.3.4.2 AR Abort Indication response

The application has to return this packet to the stack.

Packet description

Variable	**Type**	**Value / Range**	**Description**
ulLen	uint32_t	4	Packet data length in bytes
ulSta	uint32_t	0	The status has to be okay for this service.
ulCmd	uint32_t	0x1F2B	PNS_IF_AR_ABORT_RES
Data			
hDeviceHandle	uint32_t		The device handle.

Table 79: PNS_IF_AR_ABORT_IND_RSP_T - AR Abort Indication response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_DATA_Ttag
{
  /** The device handle */
  uint32_t hDeviceHandle;
} __HIL_PACKED_POST PNS_IF_HANDLE_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_PACKET_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_HANDLE_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T PNS_IF_AR_ABORT_IND_RSP_T;
```

6.3.5 Save Station Name service

With this service, the stack indicates the reception of a DCP Set NameOfStation request to the application. The stack automatically sets the new NameOfStation.

If the application configured the stack using packets, the application must be able to store the station name into non-volatile memory. On power-up, the application has to restore the station name using the *Set Configuration Request* packet.

The following figure explains the correct application behavior:

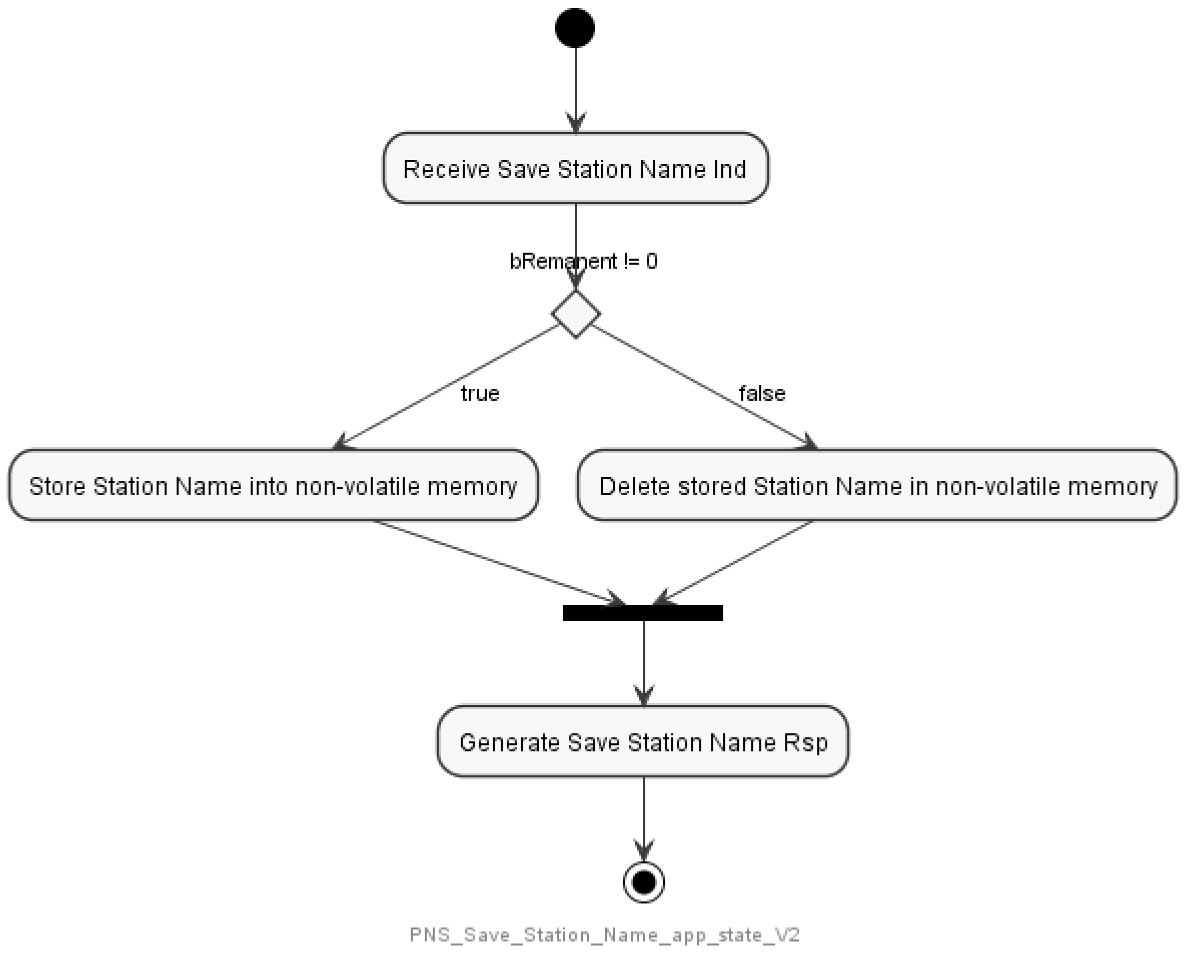

Receive Save Station Name Ind
bRemanent != 0
true
false
Store Station Name into non-volatile memory
Delete stored Station Name in non-volatile memory
Generate Save Station Name Rsp
PNS_Save_Station_Name_app_state_V2

Figure 17: Save Station Name: Application state diagram

Note:
The application can configure the stack to save the NameOfStation on its own. See bit D17 in *System flags* [▷ page 56]. If the application does not use this elegant solution to deal with the NameOfStation, the following applies:
If the flag `bRemanent` is not set, the application must delete the permanently stored *Station Name* and use an empty *Station Name* after the next power-up cycle.

Note:
The DCP Set Station Name response is sent to the network after the application responds to the *Save Station Name* service. The protocol stack will automatically generate a positive response to the network in case the application does not answer to the service in time. However, it is required that the application responds to the indication.

6.3.5.1 Save Station Name indication

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	243	Packet data length in bytes
ulCmd	uint32_t	0x1F1A	PNS_IF_SAVE_STATION_NAME_IND
Data			
usNameLen	uint16_t	0..240	Length of the new NameOfStation.
bRemanent	uint8_t	0 1	Do not save the new NameOfStation remanent. Save the NameOfStation remanent.
abNameOfStation[240]	uint8_t[]		The new NameOfStation as ASCII byte-array. For the station name, only use lower case characters.

Table 80: PNS_IF_SAVE_STATION_NAME_IND_T - Save Station Name indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_SAVE_STATION_NAME_IND_DATA_Ttag
{
  uint16_t usNameLen;
  uint8_t bRemanent;
  uint8_t abNameOfStation[PNS_IF_MAX_NAME_OF_STATION];
} __HIL_PACKED_POST PNS_IF_SAVE_STATION_NAME_IND_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SAVE_STATION_NAME_IND_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_SAVE_STATION_NAME_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_SAVE_STATION_NAME_IND_T;
```

6.3.5.2 Save Station Name response

The application acknowledges the indication with this packet.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t	0	The application has to return SUCCESS_HIL_OK.
ulCmd	uint32_t	0x1F1B	PNS_IF_SAVE_STATION_NAME_RES

Table 81: PNS_IF_SAVE_STATION_NAME_RSP_T - Save Station Name response

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_SAVE_STATION_NAME_RSP_T;
```

6.3.6 Save IP Address service

With this service, the stack indicates to the application that a DCP Set IP request has been received. The stack automatically sets the new IP address and informs the application.

If the application configured the stack using packets, the application must be able to store the IP address parameters into non-volatile memory. On power up, the application has to restore the IP address parameters using the *Set Configuration Request* packet.

The following figure explains the correct application behavior.

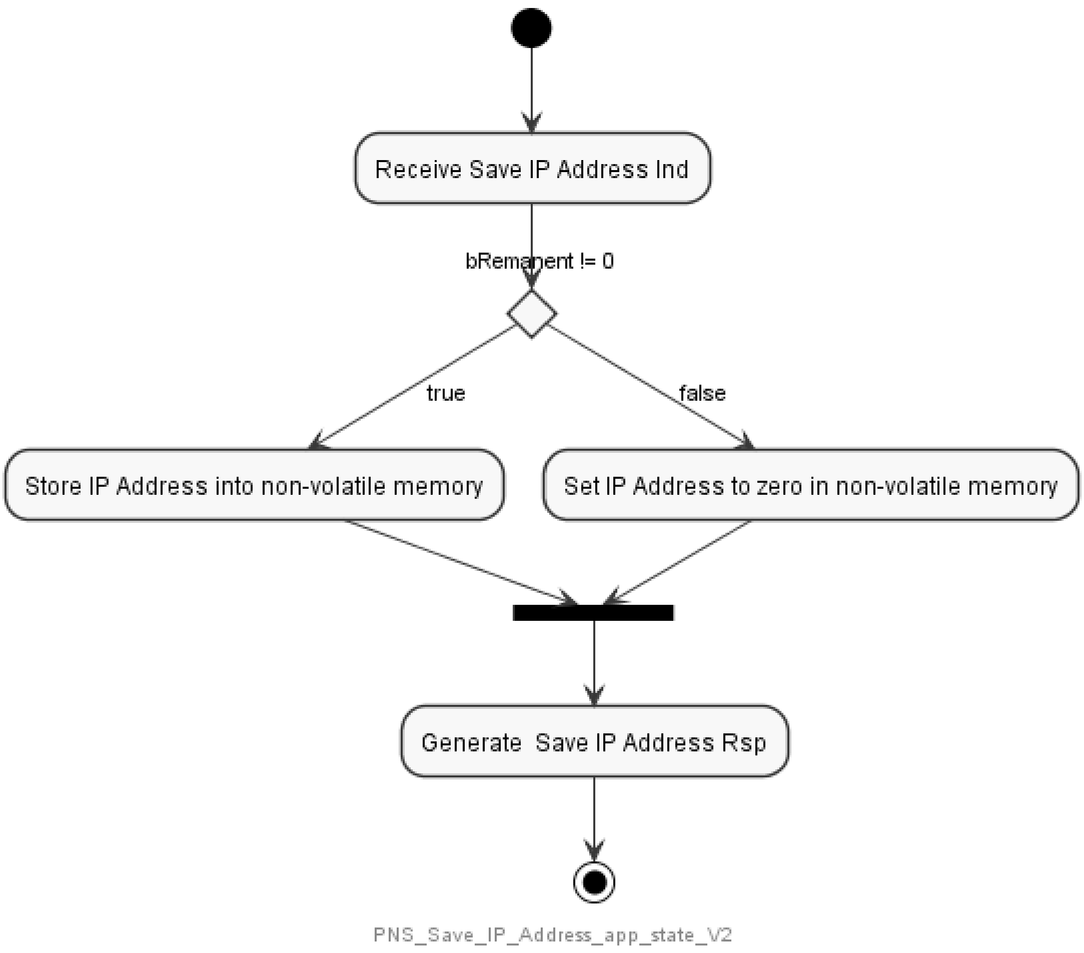

Receive Save IP Address Ind
bRemanent != 0
true
false
Store IP Address into non-volatile memory
Set IP Address to zero in non-volatile memory
Generate Save IP Address Rsp
PNS_Save_IP_Address_app_state_V2

Figure 18: Save IP Address: Application state diagram

Note:
The application can configure the stack to save the IP parameters by itself,
see bit D17 in *System flags* [▸ page 56].If the application does not use this elegant solution to deal with the NameOfStation, the following applies:

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

Note:
The DCP Set IP Address response is sent to the network after the application responds to the Save IP Address Service. The protocol stack will automatically generate a positive response to the network in case the application does not answer to the service in time. However, it is required that the application responds to the indication.

6.3.6.1 Save IP Address indication

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	13	Packet data length in bytes
ulCmd	uint32_t	0x1FB8	PNS_IF_SAVE_IP_ADDR_IND
Data			
ulIpAddr	uint32_t		The new IP address.
ulNetMask	uint32_t		The new network mask.
ulGateway	uint32_t		The new gateway address.
bRemanent	uint8_t	0 1	Do not save the new IP parameters remanent. Save the IP parameters remanent.

Table 82: PNS_IF_SAVE_IP_ADDRESS_IND_T - Save IP Address indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_SAVE_IP_ADDR_IND_DATA_Ttag
{
  uint32_t ulIpAddr;
  uint32_t ulNetMask;
  uint32_t ulGateway;
  uint8_t bRemanent;
} __HIL_PACKED_POST PNS_IF_SAVE_IP_ADDR_IND_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SAVE_IP_ADDR_IND_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_SAVE_IP_ADDR_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_SAVE_IP_ADDR_IND_T;
```

6.3.6.2 Save IP Address response

The application has to return this packet.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t	0	The application has to return SUCCESS_HIL_OK.
ulCmd	uint32_t	0x1FB9	PNS_IF_SAVE_IP_ADDRESS_RSP

Table 83: PNS_IF_SAVE_IP_ADDRESS_RSP_T - Save IP Address response

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_SAVE_IP_ADDR_RSP_T;
```

6.3.7 Start LED Blinking service

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

If all of the following conditions are fulfilled, the stack will automatically cause the LED to blink:

- The stack is used as LOM.
- It is configured to use LEDs.
- A valid blinking LED-Name (see section) is given,

Note:
If the netX COM LEDs are present, these are managed by the protocol stack and this service is informative in this case. In all other cases the implementation is mandatory and relevant for certification.

6.3.7.1 Start LED Blinking indication

With this indication packet the stack informs the application about the reception of a DCP SET Signal request.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet data length in bytes
ulCmd	uint32_t	0x1F1E	PNS_IF_START_LED_BLINKING_IND
Data			
ulFrequency	uint32_t	1	Blinking frequency of the LED (1 Hz).

Table 84: PNS_IF_START_LED_BLINKING_IND_T - Start LED Blinking indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_START_LED_BLINKING_IND_DATA_Ttag
{
  uint32_t ulFrequency;
} __HIL_PACKED_POST PNS_IF_START_LED_BLINKING_IND_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_START_LED_BLINKING_IND_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_START_LED_BLINKING_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_START_LED_BLINKING_IND_T;
```

© Hilscher 2011 – 2019

6.3.7.2 Start LED Blinking response

The application has to return this response packet. If blinking with the LED cannot be accomplished, the packet must contain a negative status.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t	0	The application has to return SUCCESS_HIL_OK.
ulCmd	uint32_t	0x1F1F	PNS_IF_START_LED_BLINKING_RES

Table 85: PNS_IF_START_LED_BLINKING_RSP_T - Start LED Blinking response

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_START_LED_BLINKING_RSP_T;
```

6.3.8 Stop LED Blinking service

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

Note:
If the netX COM LEDs are present, these are managed by the protocol stack and this service is informative in this case. In all other cases the implementation is mandatory and relevant for certification.

Note:
If the stack is used as LOM and is configured to use LEDs and if a valid Blinking LED-Name (see section *Start LED Blinking service* [▸ page 143]) is given the stack will automatically blink with the LED and this indication is only informative.

6.3.8.1 Stop LED Blinking indication

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulCmd	uint32_t	0x1F20	PNS_IF_STOP_LED_BLINKING_IND

Table 86: PNS_IF_STOP_LED_BLINKING_IND_T - Stop LED Blinking indication

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_STOP_LED_BLINKING_IND_T;
```

6.3.8.2 Stop LED Blinking response

The application shall return this response packet.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t	0	The application has to return SUCCESS_HIL_OK.
ulCmd	uint32_t	0x1F21	PNS_IF_STOP_LED_BLINKING_RES

Table 87: PNS_IF_STOP_LED_BLINKING_RSP_T - Stop LED Blinking response

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_STOP_LED_BLINKING_RSP_T;
```

© Hilscher 2011 – 2019

6.3.9 Reset Factory Settings service

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

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

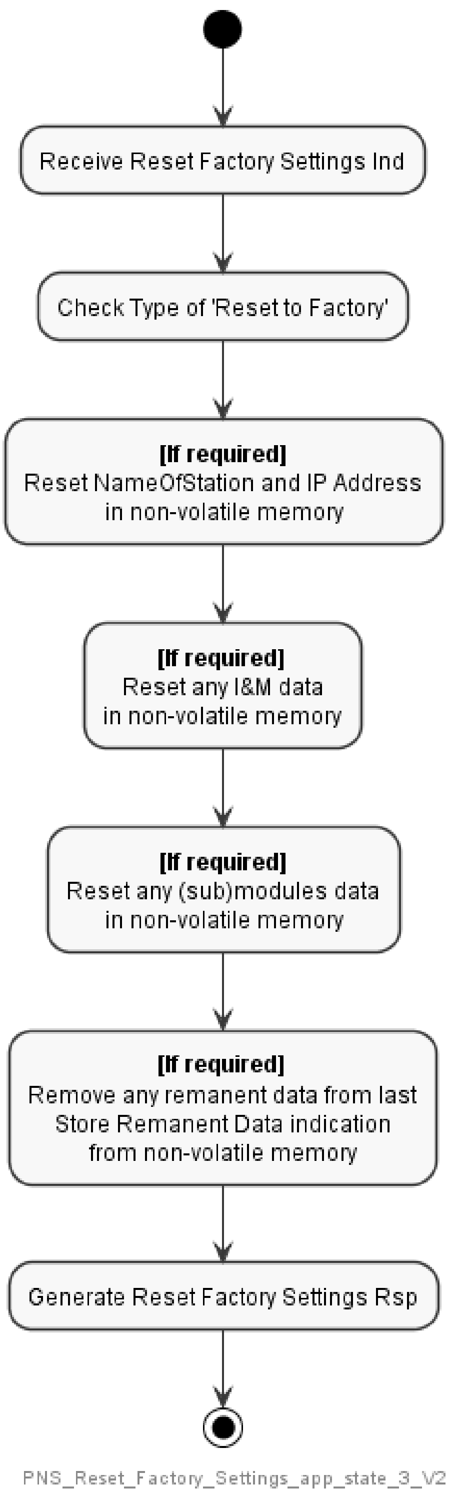

Receive Reset Factory Settings Ind
Check Type of 'Reset to Factory'
[If required]
Reset NameOfStation and IP Address
in non-volatile memory
[If required]
Reset any I&M data
in non-volatile memory
[If required]
Reset any (sub)modules data
in non-volatile memory
[If required]
Remove any remanent data from last
Store Remanent Data indication
from non-volatile memory
Generate Reset Factory Settings Rsp
PNS_Reset_Factory_Settings_app_state_3_V2

Figure 19: Reset Factory Settings: Application state diagram

As default, the application has to use the following reset values:

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

Table 88: Default reset values for the application

Anyway, the host is responsible to process the application parameters.

Note:
PROFINET specifies that the reception of a Reset to factory settings Request shall automatically stop any running cyclic communication. The stack does this automatically and indicates this to the application by the *Abort Indication* service.

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

Note:
If the stack is configured using the *Set Configuration* service the internally stored parameters will also be changed by the stack in the way that e.g. after a ChannelInit the default values will be used.

Note:
The DCP Set Reset Factory response is sent to the network after the application responds to the Reset Factory Settings Service. The protocol stack will automatically generate a positive response to the network in case the application does not answer to the service in time. However, it is required that the application responds to the indication.

6.3.9.1 Reset Factory Settings indication

The indication packet sent by the stack.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	2	Packet data length in bytes
ulCmd	uint32_t	0x1F18	PNS_IF_RESET_FACTORY_SETTINGS_IND
Data			
usResetCode	uint16_t		The reset code, see the table.

Table 89: PNS_IF_RESET_FACTORY_SETTINGS_IND_T - Reset Factory Settings indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
{
  /** This field has the same coding as the DCP BlockQualifier with
option ControlOption and suboption SuboptionResetToFactory.*/
  uint16_t usResetCode;
} __HIL_PACKED_POST PNS_IF_RESET_FACTORY_SETTINGS_IND_DATA_T;

typedef __HIL_PACKED_PRE struct
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_RESET_FACTORY_SETTINGS_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_RESET_FACTORY_SETTINGS_IND_T;
```

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

Name	Value	Description
PNS_IF_RESET_FACTORY_SETTI NGS_IF_APPLICATION	2	**Reset application data in interface** Reset data, which have been stored permanently in submodules and modules to factory values. - Manufacturer specific record data shall be set to factory values. - All I&M data shall be set also to factory values. **The stack does not support this sub-option. Reserved for future implementation.**

Name	Value	Description
PNS_IF_RESET_FACTORY_SETTINGS_IF_COMMUNICATION	4	**Reset communication parameter in interface** All parameters active for the interface or the ports and the ARs shall be set to the default value and reset if permanently stored. This mode is intended to set the addressed communication interface of a device into a state, which is almost similar to the "out of the box" state. In particular these are: - NameOfStation shall be set to "" (empty string) - IP suite parameter shall be set to `0.0.0.0` - DHCP parameters (if available) shall be reset - All PDev parameters shall be set to factory values (The stack resets these values internally) - Parameters adjusted by SNMP, like sysContact, sysName, and sysLocation from MIB-II (The stack resets these values internally) **Observe that all I&M data are not set to factory values!** In case, the application takes care of the remanent data (i.e. uses the *Load Remanent Data service* [▸ page 84] and *Store Remanent Data service* [▸ page 121]) but I&M requests are handled internally by the stack (which means that flag PNS_IF_SYSTEM_STACK_HANDLE_I_M_ENABLED is set by the *Set Configuration service* [▸ page 53]), the application **should not remove the remanent data**, because the stack preserves I&M data inside of the remanent data. **Else the application should reset (remove) remanent data.**
PNS_IF_RESET_FACTORY_SETTINGS_IF_ENGEINEERING	6	**Reset engineering parameter in interface** Reset engineering parameters, which have been stored permanently in the IOC or IOD to factory values. - If a node is able to switch its functionality from IOC to IOD and vice versa via engineering system, the factory functionality must be activated. - An application program loaded by an engineering system shall be reset (or removed). - A configuration loaded by an engineering system must be reset (or removed). **The stack does not support this sub-option, reserved for future implementation.**
PNS_IF_RESET_FACTORY_SETTINGS_IF_ALL	8	**Reset all stored data in interface** Reset all stored data in the IOD or IOC to its factory default values. This covers all data related to application data (2), communication (4) and engineering parameters (6). **The stack does not support this sub-option on the network** but uses this code to indicate old DCP-reset service (SuboptionFactoryReset) to the application.
PNS_IF_RESET_FACTORY_SETTINGS_DEVICE_ALL	16	**Reset all stored data in device** Reset all data stored in the IOD or IOC to its factory values. This service shall reset the communication parameters of all interfaces of the device and should reset all parameters of the device. This mode is intended to set the device into a state, which is similar to the "out of the box" state. It includes parameters adjusted by SNMP. **The stack does not support this sub-option,** because it supports only one interface.
PNS_IF_RESET_FACTORY_SETTINGS_DEVICE_RESTORE	18	**Reset and restore data in device** Reset installed software revisions to factory values. **The stack does not support this sub-option.**

Table 90: Possible values of the reset code

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

`ResetToFactoryModes = usResetCode / 2`

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

6.3.9.2 Reset Factory Settings response

The application must return this packet.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t	0	The application has to return SUCCESS_HIL_OK.
ulCmd	uint32_t	0x1F19	PNS_IF_RESET_FACTORY_SETTINGS_RES

Table 91: PNS_IF_RESET_FACTORY_SETTINGS_RSP_T - Reset Factory Settings response

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_RESET_FACTORY_SETTINGS_RSP_T;
```

6.3.10 APDU Status Changed service

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

6.3.10.1 APDU Status Changed indication

This packet indicates the APDU status Change to application.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	8	Packet data length in bytes
ulCmd	uint32_t	0x1F2E	PNS_IF_APDU_STATUS_IND
Data			
hDeviceHandle	uint32_t		The device handle
ulStatus	uint32_t		See the table below.. A change of the Primary/Backup is not indicated to the application

Table 92: PNS_IF_APDU_STATUS_CHANGED_IND_T - APDU Status Changed indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_APDU_STATUS_CHANGED_IND_DATA_Ttag
{
  uint32_t hDeviceHandle;
  uint32_t ulStatus;
} __HIL_PACKED_POST PNS_IF_APDU_STATUS_CHANGED_IND_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_APDU_STATUS_CHANGED_IND_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_APDU_STATUS_CHANGED_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_APDU_STATUS_CHANGED_IND_T;
```

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

Table 93: Meaning of bits of APDU status field

6.3.10.2 APDU Status Changed response

The application shall return this packet.

Packet description

Variable	**Type**	**Value / Range**	**Description**
ulLen	uint32_t	4	Packet data length in bytes
ulSta	uint32_t	0	The status has to be okay for this service.
ulCmd	uint32_t	0x1F2F	PNS_IF_APDU_STATUS_RES
Data			
hDeviceHandle	uint32_t		Handle to the IO-Device.

Table 94: PNS_IF_APDU_STATUS_CHANGED_RSP_T - APDU Status Changed response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_DATA_Ttag
{
  /** The device handle */
  uint32_t hDeviceHandle;
} __HIL_PACKED_POST PNS_IF_HANDLE_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_PACKET_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_HANDLE_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T PNS_IF_APDU_STATUS_CHANGED_RSP_T;
```

© Hilscher 2011 – 2019

6.3.11 Alarm Indication service

With this service, the stack indicates the reception of an Alarm PDU from the IO-Controller to the application.

This indication is informative for the application only.

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

6.3.11.1 Alarm Indication

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

Note:
The data received from the PROFINET network via the alarm indication (`abAlarmData[1024]`) are delivered in **big-endian** representation. Big-endian means, that the highest value byte is stored first, i.e. at the lowest memory address (Motorola format).

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	54 + n	Packet data length in bytes. n is the value of usLenAlarmData.
ulCmd	uint32_t	0x1F30	PNS_IF_ALARM_IND
Data			
hDeviceHandle	uint32_t		The device handle
ulApi	uint32_t		The API the alarm belongs to.
ulSlot	uint32_t		The slot the alarm belongs to.
ulSubslot	uint32_t		The subslot the alarm belongs to.
ulModuleId	uint32_t		The ModuleID the alarm belongs to.
ulSubmodId	uint32_t		The SubmoduleID the alarm belongs to.
usAlarmPriority	uint16_t		The Alarm priority.
usAlarmType	uint16_t		The alarm type.
usAlarmSequence	uint16_t		The alarm sequence number.
fDiagChannelAvailable	uint32_t		
fDiagGenericAvailable	uint32_t		
fDiagSubmodAvailable	uint32_t		
fReserved	uint32_t	0	Reserved, will be set to zero.
fArDiagnosisState	uint32_t		
usUserStructId	uint16_t		The User Structure Identifier.
usAlarmDataLen	uint16_t		The length of alarm data.
abAlarmData[1024]	uint8_t[]		Alarm data.

Table 95: PNS_IF_ALARM_IND_T - Alarm Indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_ALARM_IND_DATA_Ttag
{
  uint32_t hDeviceHandle;
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t ulModuleId;
  uint32_t ulSubmodId;
  uint16_t usAlarmPriority;
  uint16_t usAlarmType;
  uint16_t usAlarmSequence;
  uint32_t fDiagChannelAvailable;
  uint32_t fDiagGenericAvailable;
  uint32_t fDiagSubmodAvailable;
  uint32_t fReserved;
  uint32_t fArDiagnosisState;
  uint16_t usUserStructId;
  uint16_t usAlarmDataLen;
  uint8_t abAlarmData[PNS_IF_MAX_ALARM_DATA_LEN];
} __HIL_PACKED_POST PNS_IF_ALARM_IND_DATA_T;

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

6.3.11.2 Alarm Indication response

The application has to return this packet on reception of the alarm indication `PNS_IF_ALARM_IND_T`.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet data length in bytes
ulSta	uint32_t	0	The status has to be okay for this service.
ulCmd	uint32_t	0x1F31	PNS_IF_ALARM_RES
Data			
hDeviceHandle	uint32_t		The device handle

Table 96: PNS_IF_ALARM_RSP_T - Alarm Indication response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_DATA_Ttag
{
  /** The device handle */
  uint32_t hDeviceHandle;
} __HIL_PACKED_POST PNS_IF_HANDLE_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_PACKET_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_HANDLE_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_HANDLE_PACKET_T;
typedef PNS_IF_HANDLE_PACKET_T PNS_IF_ALARM_RSP_T;
```

6.3.12 Release Request Indication service

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

Note:

In order to distinguish an AR Release by the Controller from an AR abort due to an error, the PNIO Status field of the Abort Service is to be used. In case of AR Release by the Controller the PNIO status field will be set to value `PNIO_E_RTA_PROTOCOL_RELEASE_IND_RECEIVED` (`0xCF81FD0F`).

6.3.12.1 Release Request Indication

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	6	Packet data length in bytes
ulCmd	uint32_t	0x1FD6	PNS_IF_RELEASE_RECV_IND
Data			
hDeviceHandle	uint32_t		The device handle
usSessionKey	uint16_t		The session key.

Table 97: PNS_IF_RELEASE_REQ_IND_T - Release Request Indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_RELEASE_REQ_IND_DATA_Ttag
{
  uint32_t hDeviceHandle;
  uint16_t usSessionKey;
} __HIL_PACKED_POST PNS_IF_RELEASE_REQ_IND_DATA_T;

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

6.3.12.2 Release Request Indication response

The application shall answer to the stack with this packet.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet data length in bytes
ulSta	uint32_t	0	The status has to be okay for this service.
ulCmd	uint32_t	0x1FD7	PNS_IF_RELEASE_RECV_RES
Data			
hDeviceHandle	uint32_t		The device handle

Table 98: PNS_IF_RELEASE_REQ_RSP_T - Release Request Indication response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_RELEASE_REQ_RSP_DATA_Ttag
{
  uint32_t hDeviceHandle;
} __HIL_PACKED_POST PNS_IF_RELEASE_REQ_RSP_DATA_T;

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

6.3.13 Link Status Changed service

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

Note:
Depending on the requirements of the user application, the stack is able to use different packet commands to indicate a change of the link state. By default, the generic command `RCX_LINK_STATUS_CHANGE_IND` is used. However, this can be changed using the *Set OEM Parameters Request* [▸ page 72] (`PNS_IF_SET_OEM_PARAMETERS_TYPE_7`) to force the stack to use command `PNS_IF_LINK_STATUS_CHANGE_IND` instead for compatibility reasons.
This command was used by previous stack versions prior to V3.5.x.

6.3.13.1 Link Status Changed indication

This packet indicates the new Link status to the application.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle of CMDEV-task process queue
ulLen	uint32_t	32	Packet data length in bytes
ulCmd	uint32_t	0x2F8A or 0x1F70	HIL_LINK_STATUS_CHANGE_IND PNS_IF_LINK_STATE_CHANGE_IND
atLinkData[2]	PNS_IF_LINK_STATUS_DATA_T		Link Status Information for the 2 ports.

Table 99: PNS_IF_LINK_STATUS_CHANGED_IND_T - Link Status Changed indication

© Hilscher 2011 – 2019

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_LINK_STATUS_DATA_Ttag
{
  /** port the link status is for */
  uint32_t ulPort;
  /** if a full duplex link is available on this port */
  uint32_t fIsFullDuplex;
  /** if a link is available on this port */
  uint32_t fIsLinkUp;
  /** the speed of the link
   *
   * - 0 No Link
   * - 10 for 10MBit
   * - 100 for 100MBit
   */
  uint32_t ulSpeed;
} __HIL_PACKED_POST PNS_IF_LINK_STATUS_DATA_T;

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

typedef __HIL_PACKED_PRE struct PNS_IF_LINK_STATUS_CHANGED_IND_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_LINK_STATUS_CHANGED_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_LINK_STATUS_CHANGED_IND_T;
```

Typically, the indication will contain 2 elements of type PNS_IF_LINK_STATUS_DATA_T.

Variable	Type	Value / Range	Description
ulPort	uint32_t		The port-number this information belongs to.
fIsFullDuplex	uint32_t	FALSE (0) TRUE	Is the established link FullDuplex? Only valid if fIsLinkUp is set.
fIsLinkUp	uint32_t	FALSE (0) TRUE	Is the link up for this port?
ulSpeed	uint32_t	10 or 100	If the link is up this field contains the speed of the established link. Possible values are 10 (10 MBit/s) and 100 (100MBit/s). Only valid if fIsLinkUp is set.

Table 100: Structure PNS_IF_LINK_STATUS_DATA_T

© Hilscher 2011 – 2019

6.3.13.2 Link Status Changed response

The application shall return this packet.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle of CMDEV-task process queue
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t	0	The status has to be okay for this service.
ulCmd	uint32_t	0x2F8B or 0x1F71	HIL_LINK_STATUS_CHANGE_RES PNS_IF_LINK_STATE_CHANGE_RES

Table 101: PNS_IF_LINK_STATUS_CHANGED_RSP_T - Link Status Changed response

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_LINK_STATUS_CHANGED_RSP_T;
```

© Hilscher 2011 – 2019

6.3.14 Error Indication service

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

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

6.3.14.1 Error Indication

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	8	Packet data length in bytes
ulCmd	uint32_t	0x1FDC	PNS_IF_USER_ERROR_IND
Data			
ulErrorCode	uint32_t		The error code.
ulCommand	uint32_t	0 1 … 2^{32} -1	This is an error indication for internal problems inside the stack. The command the application made the error.

Table 102: PNS_IF_USER_ERROR_IND_T - Error Indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_USER_ERROR_IND_DATA_Ttag
{
  uint32_t ulErrorCode;
  uint32_t ulCommand;
} __HIL_PACKED_POST PNS_IF_USER_ERROR_IND_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_USER_ERROR_IND_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_USER_ERROR_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_USER_ERROR_IND_T;
```

6.3.14.2 Error Indication response

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulCmd	uint32_t	0x1FDD	PNS_IF_USER_ERROR_RSP

Table 103: PNS_IF_USER_ERROR_RSP_T - Error Indication response

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_USER_ERROR_RSP_T;
```

© Hilscher 2011 – 2019

6.3.15 Read I&M service

If needed, the stack uses this service to obtain I&M information from the user application. I&M0-5 Information records are always stored into the physical (sub)module. For instance, in case of a modular I/O system, removing a (sub) module from one modular I/O block and plugging it into another one will result in the same I&M0-5 data when reading the I&M0-5 data from the new location.

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

In addition, I&M0 shall be readable on every submodule. If I&M1-5 are supported, they shall be readable on every submodule, even if they are only writable on a specific device-representing submodule.

Note:
In order to fulfill PROFINET conformance needs, the user has to set the flag `PNS_IF_IM0_FILTER_DATA_DEVICE_REF` for one submodule when handling the type `PNS_IF_IM_TYPE_IM0FILTER`.

6.3.15.1 Read I&M indication

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	12	Packet data length in bytes. n depends on the parameter type contained in the packet.
ulCmd	uint32_t	0x1F32	PNS_IF_READ_IM_IND
Data			
ulApi	uint32_t	0-0xFFFFFFFF	The API of the (sub) module to read the I&M data for. Ignore on I&M0 Filter data read.
usSlot	uint16_t	0-0xFFFF	The Slot of the (sub)module to read the I&M data for. Ignore on I&M0 Filter data read.
usSubslot	uint16_t	0-0xFFFF	The Subslot of the (sub)module to read the I&M data for. Ignore on I&M0 Filter data read.
bIMType	uint8_t	0-5, 255	The I&M record to read. (0-5: I&M0-5, 255: I&M0 Filter Data)
abReserved[3]	uint8_t		Reserved / padding

Table 104: PNS_IF_READ_IM_IND_T – Read I&M Indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_READ_IM_IND_DATA_Ttag
{
  /** api of the submodule the i&m data shall be read from */
  uint32_t ulApi;
  /** slot of the submodule the i&m data shall be read from */
  uint16_t usSlot;
  /** subslot of the submodule the i&m data shall be read from */
  uint16_t usSubslot;
  /** type of i&m to read */
  uint8_t bIMType;
  /** unused */
  uint8_t abReserved[3];
} __HIL_PACKED_POST PNS_IF_READ_IM_IND_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_READ_IM_IND_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_READ_IM_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_READ_IM_IND_T;
```

6.3.15.2 Read I&M response

The application must respond to each Read I&M indication using the Read I&M response. The response must contain the requested I&M data.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	12 + n	Packet data length in bytes. n depends on the returned data
ulSta	uint32_t		Error code. Either SUCCESS_HIL_OK or ERR_HIL_FAIL
ulCmd	uint32_t	0x1F33	PNS_IF_READ_IM_RES
Data			
ulApi	uint32_t	0-0xFFFFFFFF	Use value from indication
usSlot	uint16_t	0-0xFFFF	Use value from indication
usSubslot	uint16_t	0-0xFFFF	Use value from indication
bIMType	uint8_t	0-5, 255	Use value from indication
abReserved[3]	uint8_t		Reserved
tData	union		union of different structures. To be filled with the requested I&M information according bIMType and the related information structure (See below).

Table 105: PNS_IF_READ_IM_RES_T – Read I&M response

Packet structure reference

```
typedef union
{
  /** Array of submodules describing I&M state of submodules
   * ( grow Array as required)*/
  PNS_IF_IM0_FILTER_DATA_T atIM0FilterData[1];
  /** I&M0 Data */
  PNS_IF_IM0_DATA_T tIM0;
  /** I&M1 Data */
  PNS_IF_IM1_DATA_T tIM1;
  /** I&M2 Data */
  PNS_IF_IM2_DATA_T tIM2;
  /** I&M3 Data */
  PNS_IF_IM3_DATA_T tIM3;
  /** I&M4 Data */
  PNS_IF_IM4_DATA_T tIM4;
} PNS_IF_READ_IM_RES_DATA_ITEMS_T;

typedef __HIL_PACKED_PRE struct PNS_IF_READ_IM_RES_DATA_Ttag
{
  /** api of the submodule the i&m data shall be read from */
  uint32_t ulApi;
  /** slot of the submodule the i&m data shall be read from */
  uint16_t usSlot;
  /** subslot of the submodule the i&m data shall be read from */
  uint16_t usSubslot;
  /** type of i&m read */
  uint8_t bIMType;
  /* unused, set to zero */
  uint8_t abReserved[3];
  /** union of I&M Data */
  PNS_IF_READ_IM_RES_DATA_ITEMS_T tData;
} __HIL_PACKED_POST PNS_IF_READ_IM_RES_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_READ_IM_RES_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_READ_IM_RES_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_READ_IM_RES_T;
```

Depending on the value of the field `tData.bIMType`, the correct substructure of the union has to be filled by the user application. In case of I&M 0-5 the substructures `tIM0`-`tIM5` have to be used, in case of I&M0 Filter Data the array `atIM0FilterData` has to be filled with all (sub)modules which have discrete I&M data.

Variable	Type	Value / Range	Description
abManufacturerSpecific	uint8_t [10]	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
usManufacturerId	uint16_t		The Vendor ID of the device. Usually similar to the specified in Set Configuration Request
abOrderId	uint8_t[20]		The Order ID of the (sub) module. (padded with spaces (0x20))
abSerialNumber	uint8_t[16]		The Serial number of the (sub) module. (padded with spaces)
usHardwareRevision	uint16_t		Hardware revision of the (sub) module
tSoftwareRevision.bPrefix	uint8_t		Character describing the software of the (sub) module. Allowed values: 'V', 'R', 'P', 'U' and 'T'
tSoftwareRevision.bX	uint8_t		Function enhancement. (Major version number) of the (sub) module.
tSoftwareRevision.bY	uint8_t		Bug fix (Minor version number) of the (sub)module
tSoftwareRevision.bZ	uint8_t		Internal Change (Build version number) of the (sub)module
usRevisionCounter	uint16_t		Starting from 0, shall increment on each parameter change.
usProfileId	uint16_t		The profile of the (sub) module.
usProfileSpecificType	uint16_t		Additional value depending on profile of the (sub)module
usIMVersion	uint16_t		The I&M version. (Default value 0x0101)
usIMSupported	uint16_t		Bit list describing the I&M variants supported by the (sub)module: 0x02 -> I&M1 Supported 0x04 -> I&M2 Supported 0x08 -> I&M3 Supported 0x10 -> I&M4 Supported 0x20 -> I&M5 Supported

Table 106: PNS_IF_IM0_DATA_T – Structure of I&M0 Information

Variable	Type	Value / Range	Description
abManufacturerSpecific[10]	uint8_t	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
abTagFunction[32]	uint8_t		Function tag of the (sub) module. (padded with spaces)
abTagLocation[22]	uint8_t		Location tag of the (sub) module. (padded with spaces)

Table 107: PNS_IF_IM1_DATA_T – Structure of I&M1 Information

Variable	Type	Value / Range	Description
abManufacturerSpecific[10]	uint8_t	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
abInstallationDate[16]	uint8_t		Installation date of the (sub) module. (padded with spaces)
abReserved[38]	uint8_t		Reserved. Set to zero. Not evaluated by stack.

Table 108: PNS_IF_IM2_DATA_T – Structure of I&M2 Information

Variable	Type	Value / Range	Description
abManufacturerSpecific[10]	uint8_t	0	Not evaluated on PROFINET. (For compatibility with PROFIBUS)
abDescriptor[54]	uint8_t		Description text of the (sub) module. (padded with spaces)

Table 109: PNS_IF_IM3_DATA_T – Structure of I&M3 Information

Variable	Type	Value / Range	Description
abManufacturerSpecific[10]	uint8_t	0	Not evaluated on PROFINET (for compatibility with PROFIBUS)
abSignature[54]	uint8_t		Signature generated by engineering system. (padded with spaces, default value: ZERO)

Table 110: PNS_IF_IM4_DATA_T – Structure of I&M4 Information

Note:
By default read of I&M5 is not forwarded to application but rejected by protocol stack with error code “invalid index”. If the application requires to support I&M5, the application has to enable the support using *Set OEM Parameters service* [page 70].

Variable	Type	Value / Range	Description
abAnnotation	uint8_t[64]	0	Manufacturer specific annotation string describing the OEM part. Padded with spaces (0x20) to 64 byte length.
abOrderId	uint8_t[64]		Order Id of OEM Part, padded with spaces (0x20) to 64 byte length.
usVendorId	uint16_t		PNO VendorId of OEM Part.
abSerialNumber	uint8_t[16]		Serial number of OEM Part. Padded with spaces (0x20) to 16 byte length.
usHardwareRevision	uint16_t		Hardware revision of the OEM part.
tSoftwareRevision.bPrefix	uint8_t		Character describing the software of the OEM part. Allowed values: ‘V’, ‘R’, ‘P’, ‘U’ and ‘T’
tSoftwareRevision.bX	uint8_t		Function enhancement. (Major version number) of the OEM part.
tSoftwareRevision.bY	uint8_t		Bug fix (Minor version number) of the OEM part.
tSoftwareRevision.bZ	uint8_t		Internal Change (Build version number) of the OEM part.

Table 111: PNS_IF_IM5_DATA_T – Structure of I&M5 Information

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

The I&M0FilterData response of application may contain one or more submodules in PNS_IF_READ_IM_RES_DATA_T element atIM0FilterData. The confirmation packet lengths need to be set according to the amount of submodules contained in the packet.

Variable	Type	Value / Range	Description
ulApi	uint32_t		The API of the submodule.
usSlot	uint16_t		The slot of the submodule
usSubslot	uint16_t		The subslot of the submodule
ulFlags	uint32_t		The properties of the submodule. If the submodule has I&M data, the flag PNS_IF_IM0_FILTER_DATA_HAS_IM_DATA shall be set. Otherwise, the submodule will be ignored. (=The entry can be omitted from the PNS_IF_READ_IM_RES_T) The Flag PNS_IF_IM0_FILTER_DATA_MODULE_REF shall be set if the submodule I&M data represents the module I&M data. (Hint: Use this for physical modules with virtual submodules) PNS_IF_IM0_FILTER_DATA_DEVICE_REF shall be set if the submodule I&M data represents the device I&M data. (Hint: Use this for devices which have no pluggable modules)

Table 112: PNS_IF_IM0_FILTER_DATA_T – Structure of I&M0 Filter Information

6.3.16 Write I&M service

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

Note:

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

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

Handling the I&M4 record is optional.

Writing I&M5 is never allowed, but the protocol stack needs to know the proper error code (Access denied or invalid index). Therefore, in case of write for I&M5 the indication is generated without write data by the protocol stack. The application has always to respond using the appropriate error code.

Choosing the proper error code is required to pass certification tests:

If the I&M record object is written for a submodule which implements no own I&M dataset: Either `ERR_PNS_IF_APPL_IM_INVALID_INDEX` or `ERR_PNS_IF_APPL_IM_ACCESS_DENIED` shall be used. This depends on if the representative I&M submodule implements that record object or not.

If the I&M record object is written for a submodule which implements an own I&M dataset but not the requested record object, the error code `ERR_PNS_IF_APPL_IM_INVALID_INDEX` must be used.

6.3.16.1 Write I&M indication

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	12, 76	Packet data length in bytes.
ulCmd	uint32_t	0x1F34	PNS_IF_WRITE_IM_IND
Data			
ulApi	uint32_t	0-0xFFFFFFFF	The API of the submodule to write the I&M data.
usSlot	uint16_t	0-0xFFFF	The slot to write the I&M data.
usSubslot	uint16_t	1-0xFFFF	The subslot to write the I&M data.
bIMType	uint8_t	1-5	The I&M record to be written.
abReserved[3]	uint8_t	0	Reserved / padding
tData	union		Union of different structures. It contains the I&M data to be written. Select the substructure according to the bIMType field. For a description of the substructures, see section *Get Asset response* [▸ page 174].

Table 113: PNS_IF_WRITE_IM_IND_T – Write I&M indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_WRITE_IM_IND_DATA_Ttag
{
  /** api of the submodule the i&m data shall be read from */
  uint32_t ulApi;
  /** slot of the submodule the i&m data shall be read from */
  uint16_t usSlot;
  /** subslot of the submodule the i&m data shall be read from */
  uint16_t usSubslot;
  /** type of i&m to write */
  uint8_t bIMType;
  /* unused, set to zero */
  uint8_t abReserved[3];
  /** union of I&M Data */
  union {
    /** I&M1 Data */
    PNS_IF_IM1_DATA_T tIM1;
    /** I&M2 Data */
    PNS_IF_IM2_DATA_T tIM2;
    /** I&M3 Data */
    PNS_IF_IM3_DATA_T tIM3;
    /** I&M4 Data */
    PNS_IF_IM4_DATA_T tIM4;
  } tData;
} __HIL_PACKED_POST PNS_IF_WRITE_IM_IND_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_WRITE_IM_IND_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_WRITE_IM_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_WRITE_IM_IND_T;
```

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

Note:
When receiving the `PNS_IF_RESET_FACTORY_SETTINGS_IND` packet with a reset mode indicating that I&M Data shall be reset, the I&M1-4 data shall be set to default values. Except for I&M4 signature the default value is a string filled with space characters (`0x20`). The default signature is a string filled with zero characters (`0x00`).

6.3.16.2 Write I&M response

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	12	Packet data length in bytes.
ulSta	uint32_t		Either • ERR_S_OK or • ERR_E_PNS_IF_APPL_IM_INVALID_INDEX or • ERR_E_PNS_IF_APPL_IM_ACCESS_DENIED.
ulCmd	uint32_t	0x1F35	PNS_IF_WRITE_IM_RES
Data			
ulApi	uint32_t	0-0xFFFFFFFF	The application has to use the same value as in the indication.
usSlot	uint16_t	0-0xFFFF	The application has to use the same value as in the indication.
usSubslot	uint16_t	1-0xFFFF	The application has to use the same value as in the indication.
bIMType	uint8_t	1-5	The application has to use the same value as in the indication.
abReserved[3]	uint8_t		Set to zero

Table 114: PNS_IF_WRITE_IM_RES_T – Write I&M response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_WRITE_IM_RES_DATA_Ttag
{
  /** api of the submodule the i&m data shall be read from */
  uint32_t ulApi;
  /** slot of the submodule the i&m data shall be read from */
  uint16_t usSlot;
  /** subslot of the submodule the i&m data shall be read from */
  uint16_t usSubslot;
  /** type of i&m written */
  uint8_t bIMType;
  /* unused, set to zero */
  uint8_t abReserved[3];
} __HIL_PACKED_POST PNS_IF_WRITE_IM_RES_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_WRITE_IM_RES_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_WRITE_IM_RES_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_WRITE_IM_RES_T;
```

6.3.17 Get Asset service

The stack uses this service to request asset information from the application. This asset information is used to process a PROFINET read record service for asset record object. As this record can become very large, the stack will use this service multiple times in order to request all required information from the application.

Note:
See chapter *Technical data* [▸ page 8] for limitations.

The following figure shows the stack will use multiple Get Asset indications to retrieve the information from the application. As soon as no further asset management data is available, the application has to return the Get Asset response without asset management data content. This will indicate to the stack that all asset management data has been delivered and the read response can be generated.

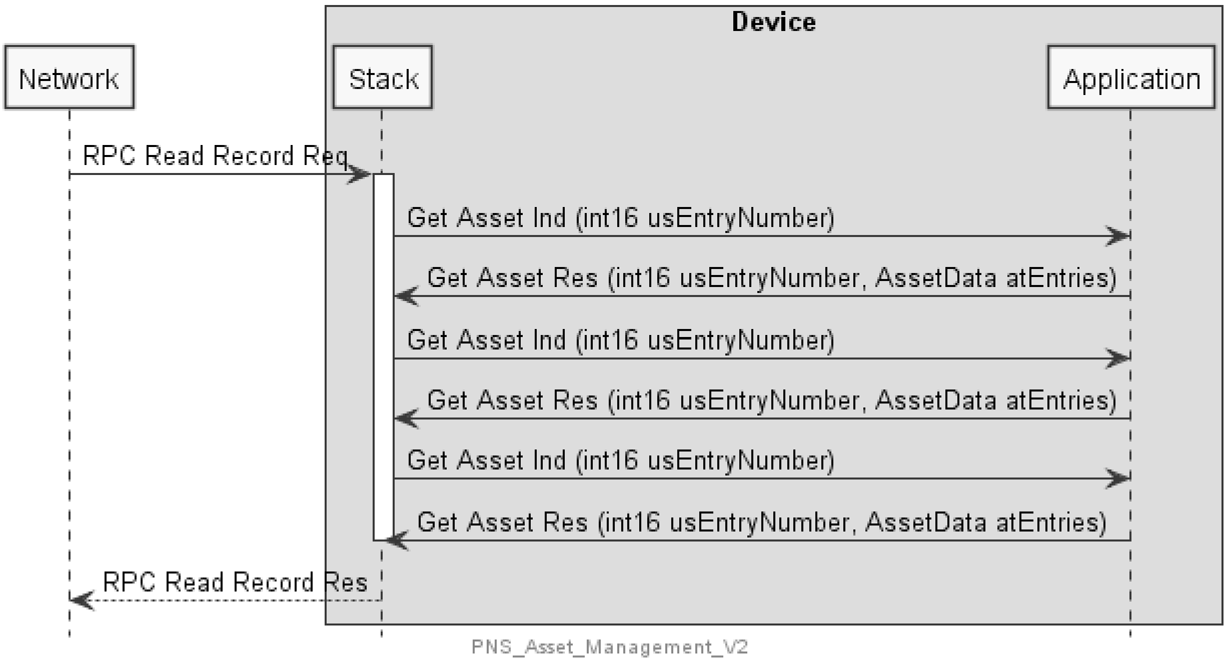

Device
Network
Stack
Application
RPC Read Record Req
Get Asset Ind (int16 usEntryNumber)
Get Asset Res (int16 usEntryNumber, AssetData atEntries)
Get Asset Ind (int16 usEntryNumber)
Get Asset Res (int16 usEntryNumber, AssetData atEntries)
Get Asset Ind (int16 usEntryNumber)
Get Asset Res (int16 usEntryNumber, AssetData atEntries)
RPC Read Record Res
PNS_Asset_Management_V2

Figure 20: Get Asset service

© Hilscher 2011 – 2019

6.3.17.1 Get Asset indication

This indication is sent by the protocol stack to request asset information from the application. The service is used multiple times by the protocol to retrieve all information sequentially.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	2	Packet data length in bytes.
ulCmd	uint32_t	0x1F3E	PNS_IF_GET_ASSET_IND
Data			
usEntryNumber	uint32_t	0-65534	The index of the first requested asset

Table 115: PNS_IF_GET_ASSET_IND_T – Get Asset indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_GET_ASSET_IND_DATA_Ttag
{
  /** Number of first requested asset management entry */
  uint16_t usEntryNumber;
} __HIL_PACKED_POST PNS_IF_GET_ASSET_IND_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_GET_ASSET_IND_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** Indication data */
  PNS_IF_GET_ASSET_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_GET_ASSET_IND_T;
```

Parameter usEntryNumber

This parameter specifies the number of the first asset to return in the response packet. The data model used to represent the asset information is a simple list of assets. This field specifies the index of the first list entry the protocol stack expects in the response. The application is expecting to deliver up to four asset entries from the asset list starting with the entry at index `usEntryNumber`. The index counting starts from zero.

6.3.17.2 Get Asset response

The application always must respond the Get Asset Indication with a Get Asset Response packet.

In case, the application has more than 4 assets available to be reported to the stack, the application must send 4 assets in a response packet. As soon as the application has no more assets to deliver, the length of the response packet, which is the last response packet, must be set to zero.

Example: In case, the application has 10 assets, the application has to send 4 assets with the first response packet, the next 4 assets with the second response packet, the next 2 (last) assets with the third response packet. Finally, the application has to send 0 assets (`ulLen` = 0) with the last (fourth) response packet which terminates the internal handling of the stack asking the application for more assets.

In case, the stack requests asset 197 (or higher), the application has to send all remaining assets in one response packet, which is the **last** response packet and terminates the internal handling of the stack asking the application for more asset (assets higher than 196). A response packet with 0 assets (`ulLen` = 0) is not needed.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0, 336, 672, 1008, 1344	Packet data length in bytes. Set according number of assets in packet.
ulSta	uint32_t		SUCCESS_HIL_OK for a positive response from the application. ERR_HIL_UNKNOWN_COMMAND for a negative response from the application.
ulCmd	uint32_t	0x1F3F	PNS_IF_GET_ASSET_RSP
Data			
atEntries	PNS_IF_ASSET_ENTRY_T[4]		Array of up to four assets.

Table 116: PNS_IF_GET_ASSET_RSP_T – Get Asset response

Structure PNS_IF_ASSET_ENTRY_T

Variable	Type	Value / Range	Description
usEntryNumber	uint16_t	0-65534	Index of this asset
bEntryType	uint8_t	1, 2, 3	Asset contains full information Asset contains only firmware information Asset contains only hardware information
bPadding	uint8_t	0	Padding for alignment. Set to zero.
tIM_UniqueIdentifier	HIL_UUID_T		Unique id of this asset.
tAM_Location	struct/union		Location of this asset. PROFINET defines two kinds of locations: a level-based location information and a slot/subslot-based location.
abIM_Annotation	uint8_t[64]		Annotation of asset in IM encoding
abIM_OrderID	uint8_t[64]		Software revision encoded as UTF-8. This field must only set to non-zero values if field abIM_Software_Revision cannot be used. This field is not used for hardware only information assets.
abAM_HardwareRevision	uint8_t[64]		Hardware revision encoded as UTF-8. This field shall only set to non-zero value if field usIM_Hardware_Revision cannot be used. This field is not used for firmware only information assets.
abIM_Serial_Number	uint8_t[16]		Serial number associated with asset using I&M encoding rules.
abIM_Software_Revision	uint8_t[4]		Software revision associated with asset using I&M encoding rules. (Revision Prefix, Major, Minor, Bugfix) This is the preferred encoding. This field is not used for hardware only information assets.
tAM_DeviceIdentification	struct		Device identification associated with asset
usAM_TypeIdentification	uint16_t		
usIM_Hardware_Revision	uint16_t		Hardware revision associated with asset using I&M encoding rules. This is the preferred encoding. Set to nonzero value if field abAM_HardwareRevision is not used. This field is not used for firmware only information assets.

Table 117: PNS_IF_ASSET_ENTRY_T

Variable	Type	Value / Range	Description
bStructureId	uint8_t	1	Structure Id identifying level-based asset location
abPadding	uint8_t[3]	0	Padding for alignment. Set to zero for future compatibility.
ausLevel	uint16_t[12]	0 to 0x3FF	Level-based asset location. The first unused level and all subsequent levels must be set to 0x3FF.

Table 118: PNS_IF_ASSET_LOCATION_T: PNS_IF_ASSET_LOCATION_LEVEL_T

Variable	Type	Value / Range	Description
bStructureId	uint8_t	2	Structure Id identifying slot/subslot-based asset location.
abPadding	uint8_t[7]	0	Padding for alignment. Set to zero for future compatibility.
usSlotBegin	uint16_t	0 to 0xFFFF	First slot covered by asset.
usSubslotBegin	uint16_t	0 to 0x9FFF	First subslot covered by asset.
usSlotEnd	uint16_t	0 to 0xFFFF	Last slot covered by asset.
usSubslotEnd	uint16_t	0 to 0x9FFF	Last subslot covered by asset.

Table 119: PNS_IF_ASSET_LOCATION_T: PNS_IF_ASSET_LOCATION_SLOTSUBSLOT_T

Packet structure reference

```
enum
{
  PNS_IF_ASSET_TYPE_FULLINFORMATION = (0x0001),
  PNS_IF_ASSET_TYPE_FIRMWAREONLYINFORMATION = (0x0002),
  PNS_IF_ASSET_TYPE_HARDWAREONLYINFORMATION = (0x0003),
};

enum
{
  PNS_IF_ASSET_LOCATION_STRUCTURE_LEVEL = 0x01,
  PNS_IF_ASSET_LOCATION_STRUCTURE_SLOTSUBSLOT = 0x02,
};

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
PNS_IF_ASSET_LOCATION_LEVEL_Ttag
{
  uint8_t bStructureId;
  uint8_t abPadding[3];
  /** Set to 0x3FF for level and subsequent levels if not used*/
  uint16_t ausLevel[12];
} __HIL_PACKED_POST PNS_IF_ASSET_LOCATION_LEVEL_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_ASSET_LOCATION_SLOTSUBSLOT_Ttag
{
  uint8_t bStructureId;
  uint8_t abPadding[7];
  uint16_t usSlotBegin;
  uint16_t usSubslotBegin;
  uint16_t usSlotEnd;
  uint16_t usSubslotEnd;
} __HIL_PACKED_POST PNS_IF_ASSET_LOCATION_SLOTSUBSLOT_T;

typedef union PNS_IF_ASSET_LOCATION_Ttag PNS_IF_ASSET_LOCATION_T;

union PNS_IF_ASSET_LOCATION_Ttag
{
  PNS_IF_ASSET_LOCATION_LEVEL_T tLevel;
  PNS_IF_ASSET_LOCATION_SLOTSUBSLOT_T tSlotSubslot;
};

typedef __HIL_PACKED_PRE struct
PNS_IF_ASSET_DEVICEIDENTIFICATION_Ttag
{
  /** Reserved for future usage. Set to 0x0000 */
  uint16_t usDeviceSubID;
  /** Device ID associated with asset */
  uint16_t usDeviceID;
  /** VendorID associated with asset */
  uint16_t usVendorID;
  /** Organization associated with asset */
  uint16_t usOrganization;
} __HIL_PACKED_POST PNS_IF_ASSET_DEVICEIDENTIFICATION_T;

typedef __HIL_PACKED_PRE struct PNS_IF_ASSET_ENTRY_Ttag
{
  uint16_t usEntryNumber;
  uint8_t bEntryType;
  uint8_t bPadding;
  HIL_UUID_T tIM_UniqueIdentifier;
  PNS_IF_ASSET_LOCATION_T tAM_Location;
  uint8_t abIM_Annotation[64];
  uint8_t abIM_OrderID[64];
  uint8_t abAM_SoftwareRevision[64];
  uint8_t abAM_HardwareRevision[64];
  uint8_t abIM_Serial_Number[16];
  uint8_t abIM_Software_Revision[4];
  PNS_IF_ASSET_DEVICEIDENTIFICATION_T tAM_DeviceIdentification;
  uint16_t usAM_TypeIdentification;
  uint16_t usIM_Hardware_Revision;
} __HIL_PACKED_POST PNS_IF_ASSET_ENTRY_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
PNS_IF_GET_ASSET_RSP_DATA_Ttag
{
  PNS_IF_ASSET_ENTRY_T atEntries[4];
} __HIL_PACKED_POST PNS_IF_GET_ASSET_RSP_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
PNS_IF_GET_ASSET_RSP_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** Indication data */
  PNS_IF_GET_ASSET_RSP_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_GET_ASSET_RSP_T;
```

6.3.18 Parameterization Speedup Support service

The stack uses this service to indicate to the user application whether Parameterization Speedup is enabled. This is the case if the device uses the fast startup mode (FSU). In FSU mode, the application must store all user record data and the associated parameter UUID (this service) into non-volatile memory. In case SharedDevice is supported by the device, the application has to store the speed-up UUID for each submodule individualy. On the next power up, the application must restore the user record data from the non-volatile memory into the submodules before any AR is established. Afterwards, when the AR is being established, the application may compare the stored parameter UUID with the parameter UUID received during last connection establishment to detect whether an update of the parameters is required. This service allows the application that it can very early parameterize the submodules and decreases the time until the device becomes ready for data exchange. If the parameter UUID is zero, the FSU mode is disabled. In this case, no parameters should be restored from non-volatile memory.

In case a connection is established for a specific submodule and FSU is not longer used for this submodule, the stack will not indicate this to the application directly. Nevertheless, in this case, the application has to delete the stored UUID for the submodule and has to set submodule parameters to default. The application can detect this situation by activating *Check Indication service* [▸ page 105] for all submodules and by checking the device handle. If a connection is established and the submodule is part of the AR but the speed-up indication was not generated, then the stored UUID has to be deleted.

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

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

6.3.18.1 Parameterization Speedup Support indication

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	16	Packet data length in bytes.
ulCmd	uint32_t	0x1FF8	PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND
Data			
tFSUuid	HIL_UUID_T	0-0xFFFFFFFF	The UUID send to the application. uint32_t ulData1 uint16_t usData2 uint16_t usData3 uint8_t abData4[8]

Table 120: PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_T – Parameterization Speedup Supported indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct HIL_UUID_Ttag
{
  uint32_t ulData1;
  uint16_t usData2;
  uint16_t usData3;
  uint8_t abData4[8];
} __HIL_PACKED_POST HIL_UUID_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_DATA_Ttag
{
  /** received UUID send to application */
  HIL_UUID_T tFSUuid;
} __HIL_PACKED_POST PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_DATA_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_PARAMET_SPEEDUP_SUPPORTED_IND_T;
```

6.3.18.2 Parameterization Speedup Supported response

The application must respond.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t		SUCCESS_HIL_OK for a positive response from the application. ERR_HIL_UNKNOWN_COMMAND or ERR_HIL_NO_APPLICATION_REGISTERED for a negative response from the application.
ulCmd	uint32_t	0x1FF9	PNS_IF_PARAMET_SPEEDUP_SUPPORTED_RSP

Table 121: PNS_IF_PARAMET_SPEEDUP_SUPPORTED_RES_T- Parameterization Speedup Supported response

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_PARAMET_SPEEDUP_SUPPORTED_RES_T;
```

6.3.19 Event Indication service

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

This service is only available if the Dual Port Memory Interface is used. It corresponds to the Callback service described in section *Event handler callback* [page 20] (see more details there as well).

6.3.19.1 Event Indication

The protocol stack sends the following indication packet.

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	14	Packet data length in bytes.
ulCmd	uint32_t	0x1FFE	PNS_IF_EVENT_IND
Data			
ausEventCnt	uint16_t[7]		Array of event counters. For each event the amount if occurrence is counted since the last event indication.

Table 122: Event Indication

Packet structure reference

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

typedef __HIL_PACKED_PRE struct
{
  /** For each event the number of events occurred since the last
event
   * indication. */
  uint16_t ausEventCnt[PNS_IF_IO_EVENT_MAX];
} __HIL_PACKED_POST PNS_IF_EVENT_IND_DATA_T;

typedef __HIL_PACKED_PRE struct
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_EVENT_IND_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_EVENT_IND_T;
```

6.3.19.2 Event Indication response

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes.
ulCmd	uint32_t	0x1FFF	PNS_IF_EVENT_RSP

Table 123: Event Indication response

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_EVENT_RSP_T;
```

6.4 Acyclic Events requested by the Application

6.4.1 Service overview

Service	**Command code (REQ)**	**Command code (CNF)**
Get Diagnosis service [▸ page 183]	0x1FB2	0x1FB3
Get XMAC (EDD) Diagnosis service [▸ page 187]	0x1FE4	0x1FE5
Process Alarm service [▸ page 189]	0x1F52	0x1F53
Diagnosis Alarm service [▸ page 193]	0x1F4C	0x1F4D
Return of Submodule Alarm service [▸ page 195]	0x1F50	0x1F51
AR Abort Request service [▸ page 198]	0x1FD8	0x1FD9
Plug Module service [▸ page 200]	0x1F04	0x1F05
Plug Submodule service [▸ page 202]	0x1F08	0x1F09
Pull Module service [▸ page 211]	0x1F06	0x1F07
Pull Submodule service [▸ page 214]	0x1F0A	0x1F0B
Get Station Name service [▸ page 217]	0x1F8E	0x1F8F
Get IP Address service [▸ page 219]	0x1FBC	0x1FBD
Add Channel Diagnosis service [▸ page 221]	0x1F46	0x1F47
Add Extended Channel Diagnosis service [▸ page 223]	0x1F54	0x1F55
Add Generic Diagnosis service [▸ page 225]	0x1F58	0x1F59
Remove Diagnosis service [▸ page 229]	0x1FE6	0x1FE7
Get Submodule Configuration service [▸ page 231]	0x1F22	0x1F23
Set Submodule State service [▸ page 233]	0x1F92	0x1F93
Get Parameter service [▸ page 238]	0x1F64	0x1F65
Add PE Entity service [▸ page 246]	0x1F94	0x1F95
Remove PE Entity service [▸ page 248]	0x1F96	0x1F97
Update PE Entity service [▸ page 250]	0x1F98	0x1F99
Send Alarm service [▸ page 252]	0x1F5C	0x1F5E

Table 124: Acyclic Events requested by the Application - Overview

6.4.2 Get Diagnosis service

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

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

6.4.2.1 Get Diagnosis request

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	0	Packet data length in bytes
ulCmd	uint32_t	0x1FB2	PNS_IF_GET_DIAGNOSIS_REQ

Table 125: PNS_IF_GET_DIAGNOSIS_REQ_T - Get Diagnosis request

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_GET_DIAGNOSIS_REQ_T;
```

6.4.2.2 Get Diagnosis confirmation

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

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	32	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1FB3	PNS_IF_GET_DIAGNOSIS_CNF
Data			
ulPnsState	uint32_t	Bit mask	State of Protocol Stack. See below.
ulLastRslt	uint32_t	Error code	Last Result
ulLinkState	uint32_t	0-3	Link State. See below.
ulConfigState	uint32_t	0-8	Configuration State. See below.
ulCommunicationState	uint32_t	0-4	Communication State. See below.
ulCommunicationError	uint32_t	Error code	Communication Error. See below.
aulLineDelay[2]	uint32_t		Structure containing cable delay for port 0 and port 1. aulLineDelay[0] stores cable delay for Port 0, aulLineDelay[1] stores cable delay for Port 1.

Table 126: PNS_IF_GET_DIAGNOSIS_CNF_T - Get Diagnosis confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_STATUS_Ttag
{
  uint32_t ulPnsState;
  uint32_t ulLastRslt;
  uint32_t ulLinkState;
  uint32_t ulConfigState;
  uint32_t ulCommunicationState;
  uint32_t ulCommunicationError;
  uint32_t aulLineDelay[2];
} __HIL_PACKED_POST PNS_IF_STATUS_T;

typedef __HIL_PACKED_PRE struct PNS_IF_GET_DIAGNOSIS_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_STATUS_T tData;
} __HIL_PACKED_POST PNS_IF_GET_DIAGNOSIS_CNF_T;
```

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

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

ulPnsState

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

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

Table 127: Meaning of single Bits in ulPnsState

eLastRslt

This parameter denotes the error code of the last encountered error. Section *Error codes and status codes* [▸ page 292] lists the error codes.

ulLinkState

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

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

Table 128: Values and their corresponding Meanings of ulLinkState

ulConfigState

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

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

Table 129: Values and their corresponding Meanings of ulLinkState

ulCommunicationState

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

Allowed value	Communication status
0x0000	`UNKNOWN`
0x0001	`OFFLINE`
0x0002	`STOP`
0x0003	`IDLE`
0x0004	`OPERATE`

ulCommunicationError

This parameter holds the current error code of the communication channel. If the cause of error is resolved, the communication error field is set to zero (= RCX_S_OK) again. For possible error codes, see section Status/Error Codes Overview.

aulLineDelay

This parameter represents the propagation delay for the ports 0 and 1 in nanoseconds. To get the cable length the time needs to be multiplied with the speed of light.

6.4.3 Get XMAC (EDD) Diagnosis service

The application can request statistic information from the integrated switch.

Note:
The information provided with this service is not intended to be used by the application. This packet is intended for support purposes only, is used to help debugging Ethernet problems, and analyzed by Hilscher.

6.4.3.1 Get XMAC (EDD) Diagnosis request

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	0	Packet Data Length in bytes
ulCmd	uint32_t	0x1FE4	PNS_IF_GET_XMAC_DIAGNOSIS_REQ

Table 130: PNS_IF_GET_XMAC_DIAGNOSIS_REQ_T - Get XMAC (EDD) Diagnosis request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_GET_XMAC_DIAGNOSIS_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  uint8_t abReserved[sizeof(PNS_IF_GET_XMAC_DIAGNOSIS_DATA_T)];
} __HIL_PACKED_POST PNS_IF_GET_XMAC_DIAGNOSIS_REQ_T;
```

6.4.3.2 Get XMAC (EDD) Diagnosis confirmation

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	> 4	Packet Data Length in bytes depending on the used firmware version and hardware variant.
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x00001FE5	PNS_IF_GET_XMAC_DIAGNOSIS_CNF
Data			
bMajor	uint8_t		Version of the used firmware/stack.
bMinor	uint8_t		
bBuild	uint8_t		
bRevision	uint8_t		
aulBuffer	uint32_t[]		Debug information that can only be decoded by Hilscher.

Table 131: PNS_IF_GET_XMAC_DIAGNOSIS_CNF_T - Get XMAC (EDD) Diagnosis confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_GET_XMAC_DIAGNOSIS_DATA_Ttag
{
  uint8_t bMajor;
  uint8_t bMinor;
  uint8_t bBuild;
  uint8_t bRevision;
  /* placeholder for real data */
  uint32_t aulBuffer[];
} __HIL_PACKED_POST PNS_IF_GET_XMAC_DIAGNOSIS_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_GET_XMAC_DIAGNOSIS_CNF_Ttag
{
  PNS_AP_PCK_HEADER_T tHead;
  PNS_IF_GET_XMAC_DIAGNOSIS_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_GET_XMAC_DIAGNOSIS_CNF_T;
```

© Hilscher 2011 – 2019

6.4.4 Process Alarm service

With this service, the application can request the stack to send a process alarm. Process alarms have high PROFINET priority in this implementation.

Note:
This service is obsolete from version V3.12.0.0. Please use the Send Alarm Service instead.

Note:
If for some reason the IO-Controller does not react to the alarm the application will NOT get a confirmation from the stack as the stack itself is waiting for a reaction of the IO-Controller.

Note:
PROFINET only allows one outstanding alarm per priority per time. However, the stack is implemented in the way that the user can request sending up to eight alarms simultaneously. Therefore, the application should not send more than one alarm request at a time to the protocol stack. After the protocol stacks confirms the process alarm, then the next Process Alarm Service can be activated. Take into account that the protocol stack has only limited capability of queuing process alarms. If the application sends more than one alarm request at a time, the stack has to buffer the packet. Therefore, the stack uses the DPM Mailbox packet buffers. Each pending Process Alarm Service will occupy one packet buffer, but these buffers are used for other application initiated services as well. Thus it is strongly recommended to implement process alarm queuing at application level.

Note:
Since GSDML file version V2.32, it is required to use the new keyword “MayIssueProcessAlarm” to indicate that a submodule might generate a process alarm. This keyword defaults to “false” for default Hilscher GSDML files. If this service is used, the GSDML file must be adapted and the keyword must be set to “true” for all affected submodules.

6.4.4.1 Process Alarm request

This packet causes the stack to send a Process Alarm to the IO-Controller.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	24 +n	Packet data length in bytes. n is the value of usLenAlarmData.
ulCmd	uint32_t	0x1F52	PNS_IF_SEND_PROCESS_ALARM_REQ
Data			
ulReserved	uint32_t	0	Reserved. Set to zero.
ulApi	uint32_t		The API the alarm belongs to.
ulSlot	uint32_t		The Slot the alarm belongs to.
ulSubslot	uint32_t		The Subslot the alarm belongs to.
hAlarmHandle	uint32_t		A user specific alarm handle. The application is free to choose any value.
usUserStructId	uint16_t		The application always has to provide a User Structure Identifier. The PROFINET specification allows an application not to provide a User Structure Identifier, which is not supported by the stack.
usAlarmDataLen	uint16_t	 1 … 172 173… 1024	The length of the alarm data Length supported by any IO-Controller. Length allowed by the specification but not supported by any IO-Controller. See parameter description below.
abAlarmData[1024]	uint8_t[]		The alarm data.

Table 132: PNS_IF_SEND_PROCESS_ALARM_REQ_T - Process Alarm request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_SEND_PROCESS_ALARM_REQ_DATA_Ttag
{
  uint32_t ulReserved ;
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t hAlarmHandle;
  uint16_t usUserStructId;
  uint16_t usAlarmDataLen;
  uint8_t abAlarmData[PNS_IF_MAX_ALARM_DATA_LEN]; /* ATTENTION:
Recommended not to use more than 172 byte of data for compatibility
reasons */
} __HIL_PACKED_POST PNS_IF_SEND_PROCESS_ALARM_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SEND_PROCESS_ALARM_REQ_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_SEND_PROCESS_ALARM_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_SEND_PROCESS_ALARM_REQ_T;
```

usAlarmDataLen

Note:
The PROFINET specification guarantees a total alarm length of 200 byte to be exchangeable between IO-Device and IO-Controller. IO-Controllers may support more than 200 byte but the IO-Device cannot rely on it. For this reason, we highly recommend to you **not** to generate any alarm that exceeds 200 bytes.

The 200 bytes include 28 bytes for submodule and protocol-related parameters. This means that the pure application data including the User Structure Identifier is limited to 174 byte in this case. Note that the User Structure Identifier is part of the 28 bytes and that it is already included in the `PNS_IF_SEND_PROCESS_ALARM_REQ_DATA_T` structure and therefore should not be used with alarm data length of more than 172 bytes.

6.4.4.2 Process Alarm confirmation

After the controller has confirmed the alarm, the protocol stack returns this packet to the application.

The reaction of the IO-Controller is reported to the application within the confirmation.

Packet description

Variable	**Type**	**Value / Range**	**Description**
ulLen	uint32_t	12	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F53	PNS_IF_SEND_PROCESS_ALARM_CNF
Data			
ulReserved	uint32_t		Reserved. Set to zero.
hAlarmHandle	uint32_t		The user specific alarm handle.
ulPnio	uint32_t		PROFINET error code consisting of ErrCode, ErrDecode, ErrCode1 and ErrCode2. For detailed explanation, see section PROFINET Status Code.

Table 133: PNS_IF_SEND_PROCESS_ALARM_CNF_T - Process Alarm confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_SEND_PROCESS_ALARM_CNF_DATA_Ttag
{
  uint32_t ulReserved ;
  uint32_t hAlarmHandle;
  /* Profinet error code, consists of ErrCode, ErrDecode, ErrCode1
and ErrCode2 */
  uint32_t ulPnio;
} __HIL_PACKED_POST PNS_IF_SEND_PROCESS_ALARM_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SEND_PROCESS_ALARM_CNF_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_SEND_PROCESS_ALARM_CNF_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_SEND_PROCESS_ALARM_CNF_T;
```

6.4.5 Diagnosis Alarm service

With this service, the application can request the stack to send a diagnosis alarm to the IO-Controller. The application has to use an “add diagnosis” service first (see sections *Add Channel Diagnosis service* [▸ page 221], *Add Extended Channel Diagnosis service* [▸ page 223], *Add Generic Diagnosis service* [▸ page 225]). The diagnosis handle that the stack returned back to the application using those services is needed here to send the alarm.

Diagnosis alarms have the low PROFINET priority in this implementation.

Note:
If for some reason the IO-Controller does not react to the alarm the application will NOT get a confirmation from the stack as the stack itself is waiting for a reaction of the IO-Controller.

Note:
PROFINET only allows one outstanding alarm per AR per priority at the same time. However, the stack is implemented in the way that diagnostic alarms will be handled with a state per alarm and therefore queue less. The application may issue one diagnostic alarm per diagnosis entry at the same time.

6.4.5.1 Diagnosis Alarm request

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	12	Packet data length in bytes.
ulCmd	uint32_t	0x1F4C	PNS_IF_SEND_DIAG_ALARM_REQ
Data			
ulReserved	uint32_t		Reserved. Set to zero.
hAlarmHandle	uint32_t		A user specific alarm handle. The application is free to choose any value.
hDiagHandle	uint32_t		The handle to the diagnosis record the diagnosis alarm shall be sent for.

Table 134: PNS_IF_SEND_DIAG_ALARM_REQ_T - Diagnosis Alarm request

© Hilscher 2011 – 2019

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_SEND_DIAG_ALARM_REQ_DATA_Ttag
{
  uint32_t ulReserved;
  uint32_t hAlarmHandle;
  uint32_t hDiagHandle;
} __HIL_PACKED_POST PNS_IF_SEND_DIAG_ALARM_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SEND_DIAG_ALARM_REQ_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_SEND_DIAG_ALARM_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_SEND_DIAG_ALARM_REQ_T;
```

6.4.5.2 Diagnosis Alarm confirmation

This packet is returned to the application by the stack. The reaction of the IO-Controller is reported to the application with this service.

If for some reason the IO-Controller does not respond to the Alarm the IO-Device application will **not** receive this confirmation packet.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	12	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F4D	PNS_IF_SEND_DIAG_ALARM_CNF
Data			
ulReserved	uint32_t		Reserved. Set to zero.
hAlarmHandle	uint32_t		The user specific alarm handle.
ulPnio	uint32_t		PROFINET error code, consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See section “PROFINET Status Code”.

Table 135: PNS_IF_DIAG_ALARM_CNF_T - Diagnosis Alarm confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_SEND_DIAG_ALARM_CNF_DATA_Ttag
{
  uint32_t ulReserved;
  uint32_t hAlarmHandle;
  /* Profinet error code, consists of ErrCode, ErrDecode, ErrCode1
and ErrCode2 */
  uint32_t ulPnio;
} __HIL_PACKED_POST PNS_IF_SEND_DIAG_ALARM_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SEND_DIAG_ALARM_CNF_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_SEND_DIAG_ALARM_CNF_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_SEND_DIAG_ALARM_CNF_T;
```

6.4.6 Return of Submodule Alarm service

The application has to use this service whenever it changes a provider IOPS of a submodule from BAD to GOOD. This service causes the stack to send a Return of Submodule alarm to the IO-Controller, which indicates that submodule provides valid data again.

Return of Submodule alarms use the low PROFINET priority in this implementation.

Note:
If for some reason the IO-Controller does not react to the alarm the application will NOT get a confirmation from the stack as the stack itself is waiting for a reaction of the IO-Controller

Note:
PROFINET only allows one outstanding alarm per priority per time. However, the stack is implemented in such a way, that the user can request sending up to 16 alarms alarm simultaneously. Then, the stack queues the outstanding requests and handles them in the order they have been reported.

6.4.6.1 Return of Submodule Alarm request

In order to cause a Return of Submodule alarm, send this packet to the stack.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	20	Packet data length in bytes
ulCmd	uint32_t	0x1F50	PNS_IF_RETURN_OF_SUB_REQ
Data			
ulReserved	uint32_t		Reserved. Set to zero.
ulApi	uint32_t		The API of the submodule.
ulSlot	uint32_t		The Slot of the Submodule.
ulSubslot	uint32_t		The Subslot of the submodule.
hAlarmHandle	uint32_t		A user specific alarm handle. The application is free to choose any value.

Table 136: PNS_IF_RETURN_OF_SUB_ALARM_REQ_T - Return of Submodule Alarm request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_RETURN_OF_SUB_ALARM_REQ_DATA_Ttag
{
  uint32_t ulReserved;
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t hAlarmHandle;
} __HIL_PACKED_POST PNS_IF_RETURN_OF_SUB_ALARM_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_RETURN_OF_SUB_ALARM_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_RETURN_OF_SUB_ALARM_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_RETURN_OF_SUB_ALARM_REQ_T;
```

6.4.6.2 Return of Submodule Alarm confirmation

The stack will return this packet.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	12	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F51	PNS_IF_RETURN_OF_SUB_CNF
Data			
ulReserved	uint32_t		Reserved. Set to zero.
hAlarmHandle	uint32_t		The user specific alarm handle.
ulPnio	uint32_t		PROFINET error code consisting of ErrCode, ErrDecode, ErrCode1 and ErrCode2. For detailed information, see section “PROFINET Status Code”..

Table 137: PNS_IF_RETURN_OF_SUB_ALARM_CNF_T - Return of Submodule Alarm confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_RETURN_OF_SUB_ALARM_CNF_DATA_Ttag
{
  uint32_t ulReserved;
  uint32_t hAlarmHandle;
  /* Profinet error code, consists of ErrCode, ErrDecode, ErrCode1
and ErrCode2 */
  uint32_t ulPnio;
} __HIL_PACKED_POST PNS_IF_RETURN_OF_SUB_ALARM_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_RETURN_OF_SUB_ALARM_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_RETURN_OF_SUB_ALARM_CNF_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_RETURN_OF_SUB_ALARM_CNF_T;
```

6.4.7 AR Abort Request service

With this service, the application requests the stack to abort an established AR.

Note:
If the device handle refers to an SR-AR Set, all SR-ARs of this set are aborted.

Note:
Be aware that in consequence of AR abort, a certification tool will report fails in testing. This service is intended for usage in special situations only and should not be used in regular environment: Use this service with care.

6.4.7.1 AR Abort Request

The application has to send this packet to force the stack to abort an established connection.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	4	Packet data length in bytes
ulCmd	uint32_t	0x1FD8	PNS_IF_ABORT_CONNECTION_REQ
Data			
hDeviceHandle	uint32_t		The device handle of the AR to abort.

Table 138: PNS_IF_ABORT_CONNECTION_REQ_T - AR Abort Request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
{
  uint32_t hDeviceHandle;
  /** the error status to report on bus. This field is ignored if
the packet is smaller
    * than 8 Bytes. Stack versions < 3.5.1.1 also ignore this field
*/
  uint32_t ulPnio;
} __HIL_PACKED_POST PNS_IF_ABORT_CONNECTION_REQ_DATA_T;

/* Request packet */
typedef __HIL_PACKED_PRE struct
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_ABORT_CONNECTION_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_ABORT_CONNECTION_REQ_T;
```

6.4.7.2 AR Abort Request confirmation

The stack will send this packet back to the application to confirm the abort of the AR.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1FD9	PNS_IF_ABORT_CONNECTION_CNF
Data			
hDeviceHandle	uint32_t		The device handle

Table 139: PNS_IF_ABORT_CONNECTION_CNF_T - AR Abort Request confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_DATA_Ttag
{
  /** The device handle */
  uint32_t hDeviceHandle;
} __HIL_PACKED_POST PNS_IF_HANDLE_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_HANDLE_PACKET_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_HANDLE_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_HANDLE_PACKET_T;

typedef PNS_IF_HANDLE_PACKET_T PNS_IF_ABORT_CONNECTION_CNF_T;
```

6.4.8 Plug Module service

With this service, the application can plug additional modules after the Set_Configuration Req packet (see section *Set Configuration request* [▸ page 54]) has been sent. It is also possible to (re)plug a module, which has been pulled by the application.

This service requires that all preceding plug/pull module/submodule services have been completed before a plug request packet is sent, i.e. that the respective confirmation packet has already been received.

Plugging a module does not lead to any action visible from the outside (e.g. no alarm is generated to an IO-Controller). Only plugging submodules is visible from the outside.

6.4.8.1 Plug Module request

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	18	Packet data length in bytes
ulCmd	uint32_t	0x1F04	PNS_IF_PLUG_MODULE_REQ
Data			
ulReserved	uint32_t		Reserved. Set to zero.
ulApi	uint32_t		The API of the module.
ulSlot	uint32_t		The Slot to plug the module to.
ulModuleId	uint32_t		The ModuleID of the module to be plugged.
usModState	uint16_t	0..1	Module state. Informative Only.

Table 140: PNS_IF_PLUG_MODULE_REQ_T - Plug Module request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_PLUG_MODULE_REQ_DATA_Ttag
{
  /** Obsolete field. set to zero */
  uint32_t ulReserved;
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulModuleId;
  uint16_t usModuleState; /* module state: 0 = correct module, 1 =
substitute module */
} __HIL_PACKED_POST PNS_IF_PLUG_MODULE_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_PLUG_MODULE_REQ_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_PLUG_MODULE_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_PLUG_MODULE_REQ_T;
```

6.4.8.2 Plug Module confirmation

The stack will return this packet to application.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	18	Packet data length in bytes
ulCmd	uint32_t	0x1F05	PNS_IF_PLUG_MODULE_CNF
Data			
ulReserved	uint32_t		Reserved. Set to zero.
ulApi	uint32_t		The API of the module.
ulSlot	uint32_t		The Slot to plug the module to.
ulModuleId	uint32_t		The ModuleID of the module to be plugged.
usModState	uint16_t	0..1	The Module state.

Table 141: PNS_IF_PLUG_MODULE_CNF_T - Plug Module confirmation

Packet structure reference

```
typedef PNS_IF_PLUG_MODULE_REQ_T PNS_IF_PLUG_MODULE_CNF_T;
```

6.4.9 Plug Submodule service

With this service, the application can plug additional submodules after the stack has been configured and/or while the device is communicating. With this service, it is also possible to (re)plug a submodule, which has been pulled by the application.

This service requires that all preceding plug/pull module/submodule services have been completed before a plug request packet is sent, i.e. that the respective confirmation packet has already been received.

If the plugged submodule was expected in a currently active AR, the controller will be notified about this by means of a plug submodule alarm. The controller will now commission the submodule by writing its parameters.

The following figure shows the Plug Submodule packet sequence.

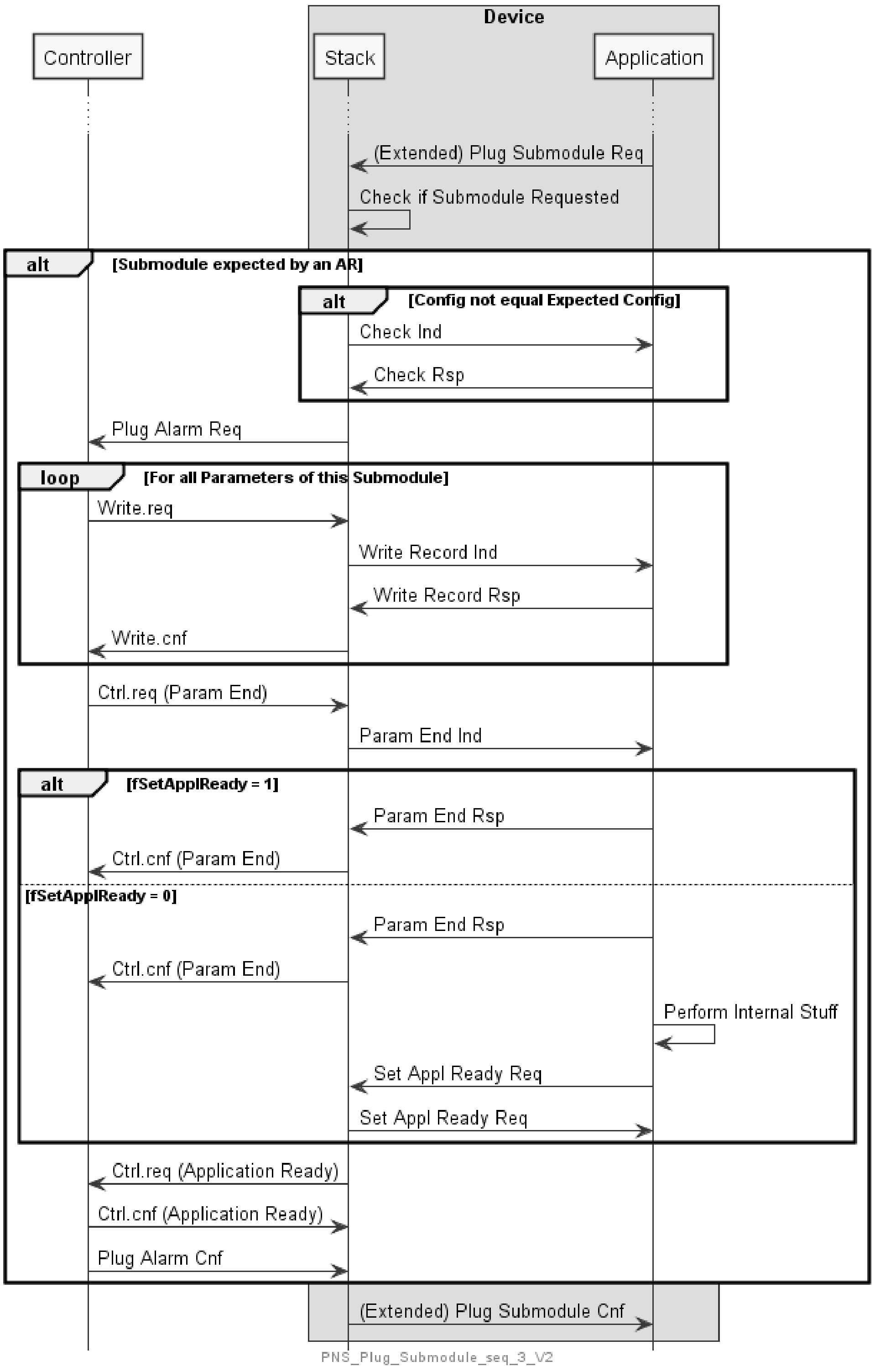


Figure 21: Plug Submodule: packet sequence

The stack also supports an extended plug submodule request allowing the application to plug modules and submodules using one single request, see *Extended Plug Submodule request* [▸ page 207].

6.4.9.1 Plug Submodule request

Packet description

Variable	**Type**	**Value / Range**	**Description**
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	50	Packet data length in bytes
ulCmd	uint32_t	0x1F08	PNS_IF_PLUG_SUBMODULE_REQ
Data			
ulReserved	uint32_t		Reserved. Set to zero.
ulApi	uint32_t		The API of the submodule.
ulSlot	uint32_t		The Slot to plug the submodule to.
ulSubslot	uint32_t		The Subslot to plug the submodule to.
ulSubmodId	uint32_t		The SubmoduleID of the submodule to be plugged.
ulProvDataLen	uint32_t		The provider data length, i.e. the length of the Input data of this submodule. This length describes the data sent by IO-Device and received by IO-Controller.
ulConsDataLen	uint32_t		The consumer data length, i.e. the length of the Output data of this submodule. This length describes the data sent by IO-Controller and received by IO-Device.
ulDPMOffsetCons	uint32_t		Offset in DPM to which consumer data for the submodule shall be copied. If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.
ulDPMOffsetProv	uint32_t		Offset in DPM from which provider data for the submodule shall be taken. If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.
usOffsetIOPSProvider	uint16_t		Offset of the IO provider state (provider) for this submodule. The offset is relative to the beginning of the IOPS block in DPM output area. **Note:** If the IOPS mode is set to “bitwise mode”, the application has to ensure that the offset addresses do not overlap. The stack will not check them for overlapping in this mode.
usOffsetIOPSConsumer	uint16_t		Offset of the IO provider state (consumer) for this submodule. The offset is relative to the beginning of the IOPS block in the DPM input area. **Note:** If the IOPS mode is set to “bitwise mode”, the application has to ensure that the offset addresses do not overlap. The stack will not check them for overlapping in this mode.
usOffsetIOCSProvider	uint16_t		Offset of the IO consumer state (provider) for this submodule. The offset is relative to the beginning of IOCS block in DPM output area. **Note:** If the IOCS mode is set to “bitwise mode”, the application has to ensure that the offset addresses do not overlap. The stack will not check them for overlapping in this mode.
usOffsetIOCSConsumer	uint16_t		Offset of the IO consumer state (consumer) for this submodule relative to the beginning of the IOCS block in DPM input area. **Note:** If the IOCS mode is set to “bitwise mode”, the application has to ensure that the offset addresses do not overlap. The stack will not check them for overlapping in this mode.
ulReserved	uint32_t	0	Reserved for future use. Set to zero.

Variable	Type	Value / Range	Description
usSubmodState	uint16_t	0..1	The submodule state: 0 = correct module, 1 = substitute module

Table 142: PNS_IF_PLUG_SUBMODULE_REQ_T - Plug Submodule request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_PLUG_SUBMODULE_REQ_DATA_Ttag
{
  /** Obsolete field. set to zero */
  uint32_t ulReserved1;
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t ulSubmodId;
  /* provider data is data sent by IO-Device and received by IO-
Controller */
  uint32_t ulProvDataLen;
  /* consumer data is data sent by IO-Controller and received by IO-
Device */
  uint32_t ulConsDataLen;
  /* offset in DPM where Output data (consumed by IO-Device from IO-
Controller) is copied to */
  uint32_t ulDPMOffsetCons;
  /* offset in DPM where Input data (provided by IO-Device to IO-
Controller) is taken from */
  uint32_t ulDPMOffsetProv;
  /* offset where to put IOPS provider state for this submodule
relative to beginning of IOPS block in dpm output area to */
  uint16_t usOffsetIOPSProvider;
  /* offset where to take IOPS consumer state of this submodule
relative to beginning of IOPS block in dpm input area from */
  uint16_t usOffsetIOPSConsumer;
  /* offset where to put IOCS provider state for this submodule
relative to beginning of IOCS block in dpm output area to */
  uint16_t usOffsetIOCSProvider;
  /* offset where to take IOCS consumer state of this submodule
relative to beginning of IOCS block in dpm input area from */
  uint16_t usOffsetIOCSConsumer;
  /* reserved for future use - maybe needed for DPM Area later */
  uint32_t ulReserved2;
  /* submodule state: 0 = correct submodule, 1 = substitute
submodule */
  uint16_t usSubmodState;
} __HIL_PACKED_POST PNS_IF_PLUG_SUBMODULE_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_PLUG_SUBMODULE_REQ_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_PLUG_SUBMODULE_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_PLUG_SUBMODULE_REQ_T;
```

6.4.9.2 Plug Submodule confirmation

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	50	Packet data length in bytes
ulCmd	uint32_t	0x1F09	PNS_IF_PLUG_SUBMODULE_CNF
Data			
ulReserved	uint32_t		Reserved. Set to zero.
ulApi	uint32_t		The API of the module.
ulSlot	uint32_t		The Slot to plug the submodule to.
ulSubslot	uint32_t		The Subslot to plug the submodule to.
ulSubmodId	uint32_t		The SubmoduleID of the submodule to be plugged.
ulProvDataLen	uint32_t		The provider data length, i.e. the length of the Input data of this submodule. This length describes the data sent by IO-Device and received by IO-Controller.
ulConsDataLen	uint32_t		The consumer data length, i.e. the length of the Output data of this submodule. This length describes the data sent by IO-Controller and received by IO-Device.
ulDPMOffsetCons	uint32_t		Offset in DPM where consumer data for the submodule shall be copied to. If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.
ulDPMOffsetProv	uint32_t		Offset in DPM where provider data for the submodule shall be taken from. If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.
usOffsetIOPSProvider	uint16_t		Offset of the IO provider state (provider) for this submodule. The offset is relative to the beginning of the IOPS block in DPM output area.
usOffsetIOPSConsumer	uint16_t		Offset of the IO provider state (consumer) for this submodule. The offset is relative to the beginning of the IOPS block in the DPM input area.
usOffsetIOCSProvider	uint16_t		Offset of the IO consumer state (provider) for this submodule. The offset is relative to the beginning of IOCS block in DPM output area.
usOffsetIOCSConsumer	uint16_t		Offset of the IO consumer state (consumer) for this submodule relative to the beginning of the IOCS block in DPM input area.
ulDPMOffsetIocsOut	uint32_t		Offset in DPM where IOCS is taken from.
ulReserved	uint32_t	0	Reserved for future use. Set to zero.
usSubmodState	uint16_t	0..1	The submodule state.

Table 143: PNS_IF_PLUG_SUBMODULE_CNF_T - Plug Submodule confirmation

Packet structure reference

```
typedef PNS_IF_PLUG_SUBMODULE_REQ_T PNS_IF_PLUG_SUBMODULE_CNF_T;
```

6.4.9.3 Extended Plug Submodule request

Receiving this packet the stack adds the described submodule and module to its internal configuration in order to use it for data exchange. The module will be added only if necessary. If the submodule has been requested by an IO-Controller for data exchange before, the stack will automatically notify the controller by issuing a plug submodule alarm.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	56	Packet data length in bytes
ulCmd	uint32_t	0x1F08	PNS_IF_PLUG_SUBMODULE_REQ (It’s the same command as standard plug submodule request)
Data			
ulReserved	uint32_t		Reserved. Set to zero.
ulApi	uint32_t		The API of the submodule.
ulSlot	uint32_t		The Slot to plug the submodule to.
ulSubslot	uint32_t		The Subslot to plug the submodule to.
ulSubmodId	uint32_t		The SubmoduleID of the submodule to be plugged.
ulProvDataLen	uint32_t		The provider data length, i.e. the length of the Input data of this submodule. This length describes the data sent by IO-Device and received by IO-Controller.
ulConsDataLen	uint32_t		The consumer data length, i.e. the length of the Output data of this submodule. This length describes the data sent by IO-Controller and received by IO-Device.
ulDPMOffsetCons	uint32_t		Offset in DPM where consumer data for the submodule shall be copied to. If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.
ulDPMOffsetProv	uint32_t		Offset in DPM where provider data for the submodule shall be taken from. If the length of data in this direction is 0 or if DPM is not used this value shall be set to 0xFFFFFFFF.
usOffsetIOPSProvider	uint16_t		Offset of the IO provider state (provider) for this submodule. The offset is relative to the beginning of the IOPS block in DPM output area. **Note:** If the IOPS mode is set to “bitwise mode”, the application has to ensure that the offset addresses do not overlap. The stack will not check them for overlapping in this mode.
usOffsetIOPSConsumer	uint16_t		Offset of the IO provider state (consumer) for this submodule. The offset is relative to the beginning of the IOPS block in the DPM input area. **Note:** If the IOPS mode is set to “bitwise mode”, the application has to ensure that the offset addresses do not overlap. The stack will not check them for overlapping in this mode.
usOffsetIOCSProvider	uint16_t		Offset of the IO consumer state (provider) for this submodule. The offset is relative to the beginning of IOCS block in DPM output area. **Note:** If the IOCS mode is set to “bitwise mode”, the application has to ensure that the offset addresses do not overlap. The stack will not check them for overlapping in this mode.

Variable	**Type**	**Value / Range**	**Description**
usOffsetIOCSConsumer	uint16_t		Offset of the IO consumer state (consumer) for this submodule relative to the beginning of the IOCS block in DPM input area. **Note:** If the IOCS mode is set to “bitwise mode”, the application has to ensure that the offset addresses do not overlap. The stack will not check them for overlapping in this mode.
ulReserved	uint32_t	0	Reserved for future use. Set to zero.
usSubmodState	uint16_t	0..1	The submodule state. See below
ulModuleId	uint32_t	1..0xFFFFFFFF	The module identifier.
usModuleState	uint16_t	0..1	The submodule state: 0 = correct module, 1 = substitute module

Table 144: PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_T – Extended Plug Submodule request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_DATA_Ttag
{
  /** Obsolete field. set to zero */
  uint32_t ulReserved1;
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t ulSubmodId;
  /* provider data is data sent by IO-Device and received by IO-
Controller */
  uint32_t ulProvDataLen;
  /* consumer data is data sent by IO-Controller and received by IO-
Device */
  uint32_t ulConsDataLen;
  /* offset in DPM where Output data (consumed by IO-Device from IO-
Controller) is copied to */
  uint32_t ulDPMOffsetCons;
  /* offset in DPM where Input data (provided by IO-Device to IO-
Controller) is taken from */
  uint32_t ulDPMOffsetProv;
  /* offset where to put IOPS provider state for this submodule
relative to beginning of IOPS block in dpm output area to */
  uint16_t usOffsetIOPSProvider;
  /* offset where to take IOPS provider state of this submodule
relative to beginning of IOPS block in dpm input area from */
  uint16_t usOffsetIOPSConsumer;
  /* offset where to put IOCS provider state for this submodule
relative to beginning of IOCS block in dpm output area to */
  uint16_t usOffsetIOCSProvider;
  /* offset where to take IOCS provider state of this submodule
relative to beginning of IOCS block in dpm input area from */
  uint16_t usOffsetIOCSConsumer;
  /* reserved for future use - maybe needed for DPM Area later */
  uint32_t ulReserved2;
  /* submodule state: 0 = correct submodule, 1 = substitute
submodule */
  uint16_t usSubmodState;
  /* Module identifier (new since V3.4 - Submodules can now be
plugged without prior plugging the modules)*/
  uint32_t ulModuleId;
  /* Module state (new since V3.4 - Submodules can now be plugged
without prior plugging the modules)*/
  uint16_t usModuleState;
} __HIL_PACKED_POST PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_T;
```

6.4.9.4 Extended Plug Submodule confirmation

This is the confirmation of the extended plug submodule request.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	56	Packet data length in bytes
ulCmd	uint32_t	0x1F09	PNS_IF_PLUG_SUBMODULE_CNF (It’s the same command as standard plug submodule confirmation)
Data			
ulReserved	uint32_t		Reserved. Set to zero.
ulApi	uint32_t		The API of the submodule.
ulSlot	uint32_t		The Slot the submodule was plugged into.
ulSubslot	uint32_t		The Subslot the submodule was plugged into
ulSubmodId	uint32_t		The SubmoduleID of the submodule.
ulProvDataLen	uint32_t		The provider data length, i.e. the length of the Input data of this submodule.
ulConsDataLen	uint32_t		The consumer data length, i.e. the length of the Output data of this submodule
ulDPMOffsetCons	uint32_t		Offset in DPM where consumer data for the submodule will be copied to.
ulDPMOffsetProv	uint32_t		Offset in DPM where provider data for the submodule will be taken from.
usOffsetIOPSProvider	uint16_t		Offset of the IO provider state (provider) for this submodule. The offset is relative to the beginning of the IOPS block in DPM output area.
usOffsetIOPSConsumer	uint16_t		Offset of the IO provider state (consumer) for this submodule. The offset is relative to the beginning of the IOPS block in the DPM input area.
usOffsetIOCSProvider	uint16_t		Offset of the IO consumer state (provider) for this submodule. The offset is relative to the beginning of IOCS block in DPM output area.
usOffsetIOCSConsumer	uint16_t		Offset of the IO consumer state (consumer) for this submodule relative to the beginning of the IOCS block in DPM input area.
ulReserved	uint32_t	0	Reserved for future use.
usSubmodState	uint16_t	0..1	The submodule state
ulModuleId	uint32_t	1..0xFFFFFFFF	The module identifier.
usModuleState	uint16_t	0..1	The module state

Table 145: PNS_IF_PLUG_SUBMODULE_EXTENDED_CNF_T – Extended Plug Submodule confirmation

Packet structure reference

```
typedef PNS_IF_PLUG_SUBMODULE_EXTENDED_REQ_T
PNS_IF_PLUG_SUBMODULE_EXTENDED_CNF_T;
```

6.4.10 Pull Module service

With this service, the application can pull modules. This removes the module and its submodules from the configuration. The stack will generate a Pull Module Alarm for each AR, which owned any submodule of this module.

This service requires that all preceding plug/pull module/submodule services have been completed before a pull request packet is sent, i.e. that the respective confirmation packet has already been received.

The following figure shows the packet sequence.

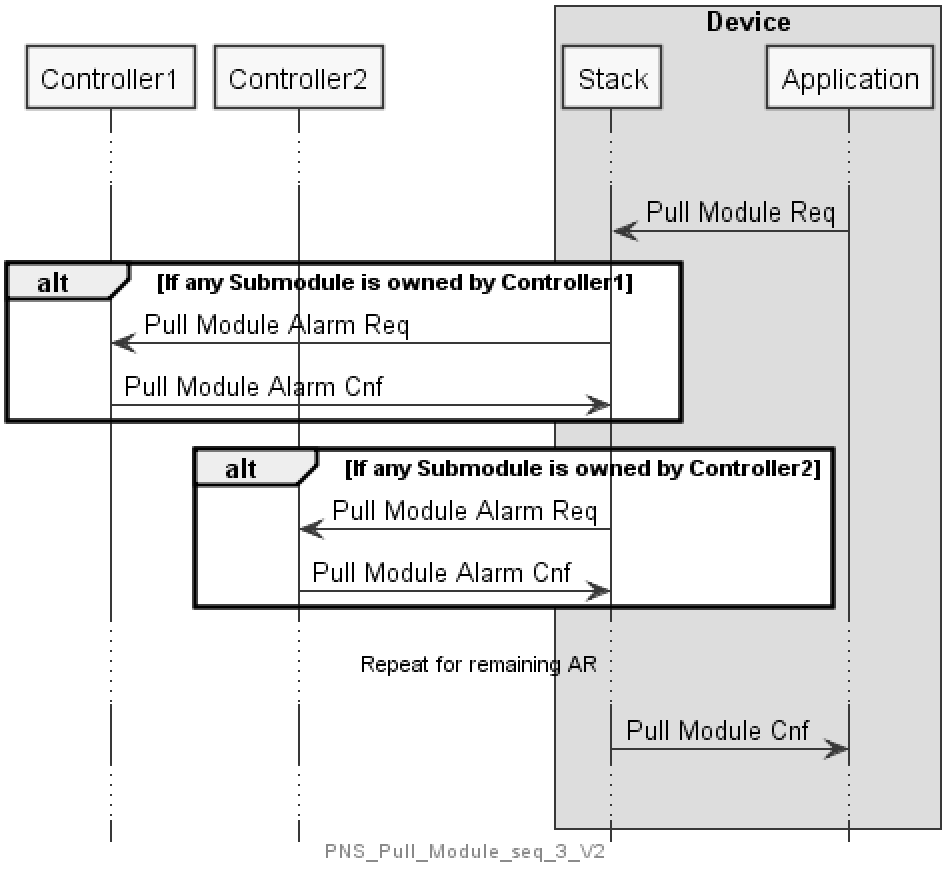


Figure 22: Pull Module: packet sequence

6.4.10.1 Pull Module request

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	12	Packet data length in bytes
ulCmd	uint32_t	0x1F06	PNS_IF_PULL_MODULE_REQ
Data			
ulReserved	uint32_t		Reserved. Set to Zero.
ulApi	uint32_t		The API of the module.
ulSlot	uint32_t		The Slot to pull the module from.

Table 146: PNS_IF_PULL_MODULE_REQ_T - Pull Module request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_PULL_MODULE_REQ_DATA_Ttag
{
  /** Obsolete field. set to zero */
  uint32_t ulReserved;
  uint32_t ulApi;
  uint32_t ulSlot;
} __HIL_PACKED_POST PNS_IF_PULL_MODULE_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_PULL_MODULE_REQ_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_PULL_MODULE_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_PULL_MODULE_REQ_T;
```

6.4.10.2 Pull Module confirmation

The stack will return this packet to the application.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	12	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F07	PNS_IF_PULL_MODULE_CNF
Data			
ulReserved	uint32_t		Reserved. Will be set to Zero.
ulApi	uint32_t		The API of the module.
ulSlot	uint32_t		The Slot to pull the module from.

Table 147: PNS_IF_PULL_MODULE_CNF_T – Pull Module confirmation

Packet structure reference

```
typedef PNS_IF_PULL_MODULE_REQ_T PNS_IF_PULL_MODULE_CNF_T;
```

6.4.11 Pull Submodule service

With this service, the application can pull submodules. This will remove the submodule from the stacks configuration. If the submodule is in use by any AR, a Pull Alarm will be generated automatically.

This service requires that all preceding plug/pull module/submodule services have been completed before a pull request packet is sent, i.e. that the respective confirmation packet has already been received.

The following figure shows the packet sequence.

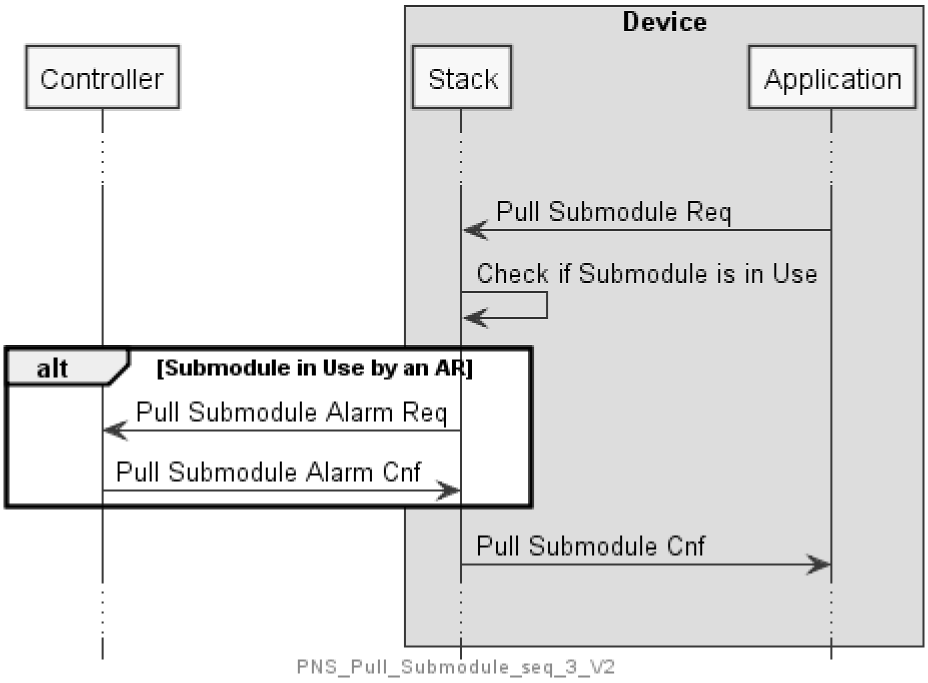


Figure 23: Pull Submodule: packet sequence

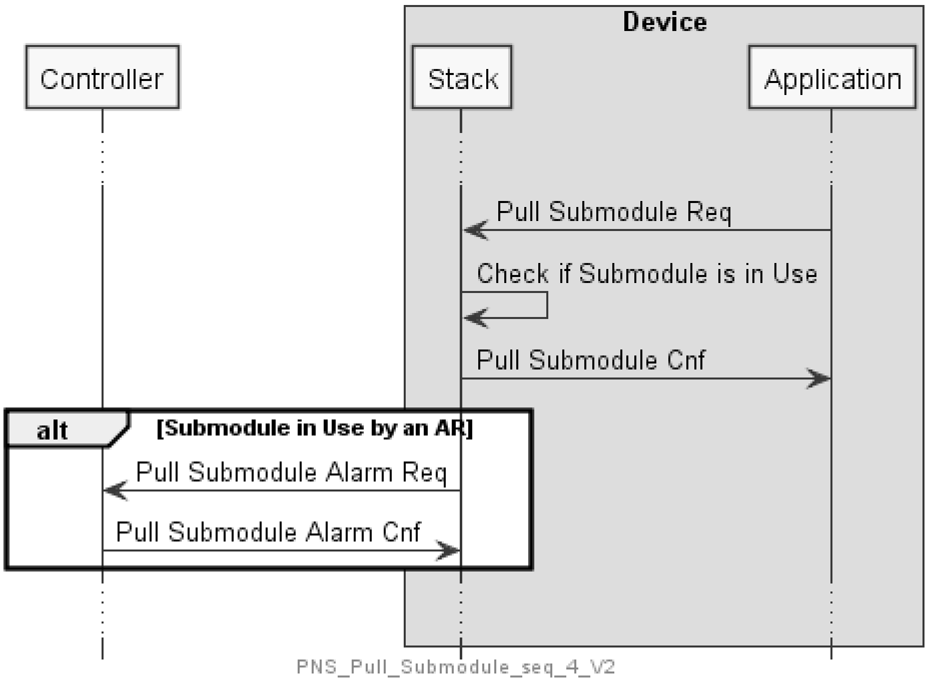


Figure 24: Pull Submodule: packet sequence

6.4.11.1 Pull Submodule request

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	16	Packet data length in bytes
ulCmd	uint32_t	0x1F0A	PNS_IF_PULL_SUBMODULE_REQ
Data			
ulReserved	uint32_t		Reserved. Set to zero.
ulApi	uint32_t		The API of the submodule.
ulSlot	uint32_t		The Slot to pull the submodule from.
ulSubslot	uint32_t		The subslot to pull the submodule from.

Table 148: PNS_IF_PULL_SUBMODULE_REQ_T - Pull Submodule request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_PULL_SUBMODULE_REQ_DATA_Ttag
{
  /** Obsolete field. set to zero */
  uint32_t ulReserved;
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
} __HIL_PACKED_POST PNS_IF_PULL_SUBMODULE_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_PULL_SUBMODULE_REQ_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_PULL_SUBMODULE_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_PULL_SUBMODULE_REQ_T;
```

6.4.11.2 Pull Submodule confirmation

The stack will return this packet to the application.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	16	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [page 292].
ulCmd	uint32_t	0x1F0B	PNS_IF_PULL_SUBMODULE_CNF
Data			
ulReserved	uint32_t		Reserved. Will be set to Zero.
ulApi	uint32_t		The API of the submodule.
ulSlot	uint32_t		The Slot to pull the submodule from.
ulSubslot	uint32_t		The subslot to pull the submodule from.

Table 149: PNS_IF_PULL_SUBMODULE_CNF_T - Pull Submodule confirmation

Packet structure reference

```
typedef PNS_IF_PULL_SUBMODULE_REQ_T PNS_IF_PULL_SUBMODULE_CNF_T;
```

6.4.12 Get Station Name service

With this service, the application can request the current *NameOfStation* from the stack.

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

6.4.12.1 Get Station Name request

In order to request the current *NameOfStation* from the stack, the application has to send the following request packet to the stack.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	0	Packet data length in bytes
ulCmd	uint32_t	0x1F8E	PNS_IF_GET_STATION_NAME_REQ

Table 150: PNS_IF_GET_STATION_NAME_REQ_T - Get Station Name request

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_GET_STATION_NAME_REQ_T;
```

6.4.12.2 Get Station Name confirmation

The protocol stack returns the following confirmation packet. It contains the current *NameOfStation* .

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	242	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F8F	PNS_IF_GET_STATION_NAME_CNF
Data			
usNameLen	uint16_t	0..240	Length of the current NameOfStation.
abNameOfStation[240]	uint8_t		The NameOfStation as ASCII byte-array. For details about allowed characters, see section *Name encoding* [▸ page 323].

Table 151: PNS_IF_GET_STATION_NAME_CNF_T - Get Station Name confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_GET_STATION_NAME_CNF_DATA_Ttag
{
  uint16_t usNameLen;
  uint8_t abNameOfStation[PNS_IF_MAX_NAME_OF_STATION];
} __HIL_PACKED_POST PNS_IF_GET_STATION_NAME_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_GET_STATION_NAME_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_GET_STATION_NAME_CNF_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_GET_STATION_NAME_CNF_T;
```

© Hilscher 2011 – 2019

6.4.13 Get IP Address service

With this service, the application can request the current IP-parameters (IP address, network mask, gateway address) from the stack.

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

6.4.13.1 Get IP Address request

In order to request the current IP-parameters from the stack, the application has to send the following request packet to the stack:

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	0	Packet data length in bytes
ulCmd	uint32_t	0x1FBC	PNS_IF_GET_IP_ADDR_REQ

Table 152: PNS_IF_GET_IP_ADDR_REQ_T - Get IP Address request

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_GET_IP_ADDR_REQ_T;
```

6.4.13.2 Get IP Address confirmation

The protocol stack returns the following confirmation packet. It contains the current IP parameters.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	12	Packet data length in bytes
ulSta	uint32_t	0	Status has to be okay for this service.
ulCmd	uint32_t	0x1FBD	PNS_IF_GET_IP_ADDR_CNF
Data			
ulIpAddr	uint32_t		The IP address.
ulNetMask	uint32_t		The network mask.
ulGateway	uint32_t		The gateway address.

Table 153: PNS_IF_GET_IP_ADDR_CNF_T - Get IP Address confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_GET_IP_ADDR_CNF_DATA_Ttag
{
  uint32_t ulIpAddr;
  uint32_t ulNetMask;
  uint32_t ulGateway;
} __HIL_PACKED_POST PNS_IF_GET_IP_ADDR_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_GET_IP_ADDR_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_GET_IP_ADDR_CNF_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_GET_IP_ADDR_CNF_T;
```

6.4.14 Add Channel Diagnosis service

With this service, the user application can add a diagnosis data record to a submodule.

Inside the confirmation packet, the stack sends a unique record handle to the application. The application has to store this handle, as it is needed to remove the diagnosis record later on.

It is also needed if the application wants to send the diagnosis alarm some time later than the record is added.

Note:
The stack does not automatically send a diagnosis alarm to the PROFINET IO-Controller. The application has to initiate this using the Diagnosis Alarm Service (see section *Diagnosis Alarm service* [▸ page 193]).

6.4.14.1 Add Channel Diagnosis request

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	22	Packet data length in bytes
ulCmd	uint32_t	0x1F46	PNS_IF_ADD_CHANNEL_DIAG_REQ
Data			
ulApi	uint32_t		The API of the submodule.
ulSlot	uint32_t		The Slot of the submodule.
ulSubslot	uint32_t		The Subslot of the submodule.
hDiagHandle	uint32_t	0	Not used inside this request. Will be used for confirmation by the stack. Set to zero.
usChannelNum	uint16_t		Channel Number for which the diagnosis data shall be added. Supported are Manufacturer specific Channel Numbers in the range of 0x0000-0x7FFF and the Channel number for the submodule itself 0x8000
usChannelProp	uint16_t		Channel properties. See table *Coding of Diagnosis* [▸ page 319]
usChannelErrType	uint16_t		Channel error type. See table *Coding of Diagnosis* [▸ page 319]

Table 154: PNS_IF_ADD_CHANNEL_DIAG_REQ_T - Add Channel Diagnosis request

© Hilscher 2011 – 2019

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_ADD_CHANNEL_DIAG_Ttag
{
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t hDiagHandle;
  uint16_t usChannelNum;
  uint16_t usChannelProp;
  uint16_t usChannelErrType;
} __HIL_PACKED_POST PNS_IF_ADD_CHANNEL_DIAG_T;

typedef __HIL_PACKED_PRE struct PNS_IF_ADD_CHANNEL_DIAG_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_ADD_CHANNEL_DIAG_T tData;
} __HIL_PACKED_POST PNS_IF_ADD_CHANNEL_DIAG_REQ_T;
```

6.4.14.2 Add Channel Diagnosis confirmation

With this packet, the stack informs the application about the success of adding diagnosis data.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	22	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F47	PNS_IF_ADD_CHANNEL_DIAG_CNF
Data			
ulApi	uint32_t		The API of the submodule.
ulSlot	uint32_t		The Slot of the submodule.
ulSubslot	uint32_t		The Subslot of the submodule.
hDiagHandle	uint32_t		A unique handle representing this diagnostic record inside the stack. The application shall store it as it is has to be used to be able to remove the record later on.
usChannelNum	uint16_t		Channel Number for which the diagnosis data shall be added. Supported are Manufacturer specific Channel Numbers in the range of 0x0000-0x7FFF and the Channel number for the submodule itself 0x8000
usChannelProp	uint16_t		Channel properties. See table *Coding of Diagnosis* [▸ page 319]
usChannelErrType	uint16_t		Channel error type. See table *Coding of Diagnosis* [▸ page 319]

Table 155: PNS_IF_ADD_CHANNEL_DIAG_CNF_T - Add Channel Diagnosis confirmation

Packet structure reference

```
typedef PNS_IF_ADD_CHANNEL_DIAG_REQ_T PNS_IF_ADD_CHANNEL_DIAG_CNF_T;
```

6.4.15 Add Extended Channel Diagnosis service

With this service, the user application can add an extended diagnosis data record to a submodule.

Inside the confirmation packet, the stack sends a unique record handle to the application. The application has to store this handle, as it is needed to remove the diagnosis record later on.

It is also needed if the application wants to send the diagnosis alarm some time later than the record is added.

Note:
The stack does not automatically send a diagnosis alarm to the PROFINET IO-Controller. The application has to initiate this using the Diagnosis Alarm Service (see section *Diagnosis Alarm service* [▹ page 193]).

6.4.15.1 Add Extended Channel Diagnosis request

The user application can add extended diagnosis data to a submodule using this packet.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	30	PNS_IF_ADD_EXT_DIAG_REQ_T - Packet data length in bytes
ulCmd	uint32_t	0x1F54	PNS_IF_ADD_EXTENDED_DIAG_REQ
Data			
ulApi	uint32_t		The API of the submodule.
ulSlot	uint32_t		The Slot of the submodule.
ulSubslot	uint32_t		The Subslot of the submodule.
hDiagHandle	uint32_t	0	Not used inside this request. Will be used for confirmation by the stack. Set to zero.
usChannelNum	uint16_t		Channel Number for which the diagnosis data shall be added. Supported are Manufacturer specific Channel Numbers in the range of 0x0000-0x7FFF and the Channel number for the submodule itself 0x8000
usChannelProp	uint16_t		Channel properties. See table *Coding of Diagnosis* [▹ page 319]
usChannelErrType	uint16_t		Channel error type. See table *Coding of Diagnosis* [▹ page 319]
usReserved	uint16_t		Reserved
ulExtChannelAddValue	uint32_t		Additional Value. Can be used to transfer additional information regarding the diagnosis. (E.g. Temperature) Can be displayed by engineering software using format strings defined in GSDML.
usExtChannelErrType	uint16_t	1 ... 0x7FFF	Extended channel error type. See table *Coding of Diagnosis* [▹ page 321]

Table 156: PNS_IF_ADD_EXTENDED_DIAG_REQ_T - Add Extended Channel Diagnosis request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_ADD_EXTENDED_DIAG_Ttag
{
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t hDiagHandle;
  uint16_t usChannelNum;
  uint16_t usChannelProp;
  uint16_t usChannelErrType;
  uint16_t usReserved;
  uint32_t ulExtChannelAddValue;
  uint16_t usExtChannelErrType;
} __HIL_PACKED_POST PNS_IF_ADD_EXTENDED_DIAG_T;

typedef __HIL_PACKED_PRE struct PNS_IF_ADD_EXTENDED_DIAG_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_ADD_EXTENDED_DIAG_T tData;
} __HIL_PACKED_POST PNS_IF_ADD_EXTENDED_DIAG_REQ_T;
```

6.4.15.2 Add Extended Channel Diagnosis confirmation

With this packet, the stack informs the application about the success of adding extended diagnosis data.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	30	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▹ page 292].
ulCmd	uint32_t	0x1F55	PNS_IF_ADD_EXTENDED_DIAG_CNF
Data			
ulApi	uint32_t		The API of the submodule.
ulSlot	uint32_t		The Slot of the submodule.
ulSubslot	uint32_t		The Subslot of the submodule.
hDiagHandle	uint32_t		A unique handle representing this diagnostic record inside the stack. Application shall store it as it is has to be used to be able to remove the record later on.
usChannelNum	uint16_t		Channel Number for which the diagnosis data shall be added. Supported are Manufacturer specific Channel Numbers in the range of 0x0000-0x7FFF and the Channel number for the submodule itself 0x8000.
usChannelProp	uint16_t		Channel properties. See table *Coding of Diagnosis* [▹ page 319]
usChannelErrType	uint16_t		Channel error type. See table *Coding of Diagnosis* [▹ page 319]
usReserved	uint16_t		Reserved
ulExtChannelAddValue	uint32_t	0	Currently not supported, set to zero.
usExtChannelErrType	uint16_t		Extended channel error type. See table *Coding of Diagnosis* [▹ page 321]

Table 157: PNS_IF_ADD_EXTENDED_DIAG_CNF_T - Add Extended Channel Diagnosis confirmation

Packet structure reference

```
typedef PNS_IF_ADD_EXTENDED_DIAG_REQ_T
PNS_IF_ADD_EXTENDED_DIAG_CNF_T;
```

6.4.16 Add Generic Diagnosis service

Note:
The stack does not automatically send a diagnosis alarm to the PROFINET IO-Controller. The application has to initiate this using the Diagnosis Alarm Service (see section *Diagnosis Alarm service* [▸ page 193]).

Note:
Usage of this service is not recommended. Generic diagnosis cannot be handled automatically by Engineering systems. The PI Diagnosis Guideline recommends not using this service.
It should be used in justified exceptions to this rule only.

Generic Diagnoses contain application specific amount of user data. Thus the memory must be allocated dynamically at runtime and can not be predicted. In turn the successful creation of a generic diagnosis entry depends on the following runtime properties:
- Available dynamic memory
- Memory Fragmentation due to runtime memory allocation
They should be used in justified exceptions to this rule only.
Generic Diagnoses contain application specific amount of user data. Thus the memory must be allocated dynamically at runtime and can not be predicted. In turn the successful creation of a generic diagnosis entry depends on the following runtime properties:

© Hilscher 2011 – 2019

6.4.16.1 Add Generic Channel Diagnosis request

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

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	26 + n	PNS_IF_ADD_GENERIC_DIAG_REQ - Packet data length in bytes + usDiagDataLen
ulCmd	uint32_t	0x1F58	PNS_IF_ADD_GENERIC_DIAG_REQ
Data			
ulApi	uint32_t		The API of the submodule.
ulSlot	uint32_t		The Slot of the submodule.
ulSubslot	uint32_t		The Subslot of the submodule.
hDiagHandle	uint32_t	0	Not used inside this request. Will be used for confirmation by the stack. Set to zero.
usChannelNum	uint16_t	0x8000	Channel Number for which the diagnosis data shall be added .Supported are Manufacturer specific Channel Numbers in the range of 0x0000-0x7FFF and the Channel number for the submodule itself 0x8000.
usChannelProp	uint16_t		Channel properties. See table *Coding of Diagnosis* [▸ page 319]
usUserStructId	uint16_t	0 - 0x7FFF	See table *Add Generic Channel Diagnosis request* [▸ page 227]
usReserved	uint16_t	0	Reserved
usDiagDataLen	uint16_t	 1 … 172 173 … 1024	The length of the diagnosis data Length supported by any IO-Controller. Length allowed by the specification but not supported by any IO-Controller. See parameter description below.
abDiagData[1024]	uint8_t		Diagnosis Data

Table 158: PNS_IF_ADD_GENERIC_DIAG_REQ_T - Add Generic Channel Diagnosis request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_ADD_GENERIC_DIAG_REQ_DATA_Ttag
{
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t hDiagHandle;
  uint16_t usChannelNum;
  uint16_t usChannelProp;
  uint16_t usUserStructId;
  uint16_t usReserved;
  uint16_t usDiagDataLen;
  uint8_t abDiagData[PNS_IF_MAX_ALARM_DATA_LEN]; /* ATTENTION:
Recommended not to use more than 172 byte of data for compatibility
reasons */
} __HIL_PACKED_POST PNS_IF_ADD_GENERIC_DIAG_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_ADD_GENERIC_DIAG_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_ADD_GENERIC_DIAG_REQ_T;
```

© Hilscher 2011 – 2019

Coding usUserStructId

Value (Hexadecimal)	Description
0 – 0x7FFF	Manufacturer Specific.

Table 159: Coding of usUserStructId

usDiagDataLen

Note:
The PROFINET specification guarantees a total diagnosis data length of 200 byte to be exchangeable between IO-Device and IO-Controller. IO-Controllers may support more than 200 byte but the IO-Device cannot rely on it. For this reason, we highly recommend to you **not** to generate any diagnosis data that exceeds 200 bytes.

The 200 bytes include 28 bytes for submodule and protocol-related parameters. This means that the pure application data including the User Structure Identifier is limited to 174 byte in this case. Note that the User Structure Identifier is part of the 28 bytes and that it is already included in the `PNS_IF_ADD_GENERIC_DIAG_REQ_DATA_T` structure and therefore should not be used with diagnosis data length of more than 172 bytes.

6.4.16.2 Add Generic Channel Diagnosis confirmation

With this packet the stack informs the application about the success of adding generic diagnosis data.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	22	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F59	PNS_IF_ADD_GENERIC_DIAG_CNF
Data			
ulApi	uint32_t		The API of the submodule.
ulSlot	uint32_t		The Slot of the submodule.
ulSubslot	uint32_t		The Subslot of the submodule.
hDiagHandle	uint32_t		A unique handle representing this diagnostic record inside the stack. Application shall store it as it is has to be used to be able to remove the record later on.
usChannelNum	uint16_t	0x8000	Channel number for which the diagnosis data has to be added. Supported are Manufacturer specific Channel Numbers in the range of 0x0000-0x7FFF and the Channel number for the submodule itself 0x8000.
usChannelProp	uint16_t		Channel properties. See table *Coding of Diagnosis* [▸ page 319]
usUserStructId	uint16_t		See table *Add Generic Channel Diagnosis request* [▸ page 227]

Table 160: PNS_IF_ADD_GENERIC_DIAG_CNF_T - Add Generic Channel Diagnosis confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_ADD_GENERIC_DIAG_CNF_DATA_Ttag
{
  uint32_t ulApi;
  uint32_t ulSlot;
  uint32_t ulSubslot;
  uint32_t hDiagHandle;
  uint16_t usChannelNum;
  uint16_t usChannelProp;
  uint16_t usUserStructId;
} __HIL_PACKED_POST PNS_IF_ADD_GENERIC_DIAG_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_ADD_GENERIC_DIAG_CNF_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_ADD_GENERIC_DIAG_CNF_T;
```

6.4.17 Remove Diagnosis service

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

6.4.17.1 Remove Diagnosis request

Using this packet the user application can remove diagnosis data from a submodule.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	4	Packet data length in bytes
ulCmd	uint32_t	0x1FE6	PNS_IF_REMOVE_DIAG_REQ
Data			
hDiagHandle	uint32_t		The unique diagnosis handle given to application by the stack while adding the diagnosis record.

Table 161: PNS_IF_REMOVE_DIAG_REQ_T - Remove Diagnosis request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_REMOVE_DIAG_REQ_DATA_Ttag
{
  uint32_t hDiagHandle;
} __HIL_PACKED_POST PNS_IF_REMOVE_DIAG_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_REMOVE_DIAG_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_REMOVE_DIAG_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_REMOVE_DIAG_REQ_T;
```

6.4.17.2 Remove Diagnosis confirmation

With this packet, the stack informs the application about the success of removing diagnosis data.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [page 292].
ulCmd	uint32_t	0x1FE7	PNS_IF_REMOVE_DIAG_CNF
Data			
hDiagHandle	uint32_t		The unique diagnosis handle given to application by the stack while adding the diagnosis record.

Table 162: PNS_IF_REMOVE_DIAG_CNF_T - Remove Diagnosis confirmation

Packet structure reference

```
typedef PNS_IF_REMOVE_DIAG_REQ_T PNS_IF_REMOVE_DIAG_CNF_T;
```

6.4.18 Get Submodule Configuration service

The application can read the information about all previously configured submodules.

Note:
The confirmation packet has a limited size, depending on the maximum length of the dual-port memory channel. The maximum count of returned submodule configuration data is limited by MAX_SUBMODULE_CNT (see Packet Structure Reference). If more submodules are configured, error code TLR_E_FAIL will be returned in `ulSta`.

If no submodule is configured, error code TLR_E_PNS_IF_NO_MODULE_CONFIGURED will be returned in `ulSta`.

6.4.18.1 Get Submodule Configuration request

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	0	Packet data length in bytes
ulCmd	uint32_t	0x1F22	PNS_IF_GET_CONFIGURED_SUBM_REQ

Table 163: PNS_IF_GET_CONFIGURED_SUBM_REQ_T - Get Submodule configuration

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_GET_CONFIGURED_SUBM_REQ_T;
```

6.4.18.2 Get Submodule Configuration confirmation

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	4 + ulSubmCnt * sizeof(PNS_IF_CONFIGURED_SUBM_STRUCT_T)	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F23	PNS_IF_GET_CONFIGURED_SUBM_CNF
Data			
ulSubmCnt	uint32_t	1 … MAX_SUBMODULE_CNT	Count of configured submodules
PNS_IF_CONFIGURED_SUBM_STRUCT_T atSubm[MAX_SUBMODULE_CNT]	Structure		Array of PNS_IF_CONFIGURED_SUBM_STRUCT_T

Table 164: PNS_IF_GET_CONFIGURED_SUBM_CNF_T - Get Submodule configuration

© Hilscher 2011 – 2019

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
{
  uint32_t ulApi;
  uint16_t usSlot;
  uint16_t usSubslot;
  uint32_t ulModuleId;
  uint32_t ulSubmoduleId;
} __HIL_PACKED_POST PNS_IF_CONFIGURED_SUBM_STRUCT_T;

#define MAX_SUBMODULE_CNT 95 /* max. count of submodules */

typedef __HIL_PACKED_PRE struct
{
  uint32_t ulSubmCnt;
  PNS_IF_CONFIGURED_SUBM_STRUCT_T atSubm[MAX_SUBMODULE_CNT];
} __HIL_PACKED_POST PNS_IF_GET_CONFIGURED_SUBM_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_GET_CONFIGURED_SUBM_CNF_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_GET_CONFIGURED_SUBM_CNF_T;
```

The structure PNS_IF_CONFIGURED_SUBM_STRUCT_T has following members:

Variable	Type
ulApi	uint32_t
usSlot	uint16_t
usSubslot	uint16_t
ulModuleId	uint32_t
ulSubmoduleId	uint32_t

Table 165: Elements of PNS_IF_CONFIGURED_SUBM_STRUCT_T

6.4.19 Set Submodule State service

With this service, the application has the possibility to change the submodule state. This is useful in e.g. gateway applications. Using this service the user application is able to influence the occurrence of a submodule in the *ModuleDiffBlock*.

Note:
This service is only usable for the user application if the user application handles IOxS, as well. If IOxS is not handled by the application, the stack will not allow the usage of this service.

A possible use case for this service is the following scenario:

A gateway application requires some parameter records to configure the underlying network. The content of the records is expected by the underlying network to work properly. Missing or invalid parameters disallow using some (or all) of the nodes of the underlying network.

To be able to receive the parameter records the application is required to adapt the local PROFINET submodule configuration during Connection establishment by evaluating *Check Indication* [▸ page 107] and using the *Extended Plug Submodule request* [▸ page 207]. After the parameters have been received via *Write Record indication* [▸ page 130] and the application received the *Parameter End indication* [▸ page 114], it configures and parameterizes the underlying network. If an error occurs in this phase, it can be indicated to the IO-Controller by setting the submodule state of the erroneous submodules to “ApplicationReady pending”. In addition, the application might generate a specific diagnosis to indicate the problem on an additional level. The IOPS of the affected submodules needs to be set to “bad”. Afterwards, the application has to send ApplicationReady using the *Application Ready request* [▸ page 117].

Note:
The submodule state is bound to the submodule and is only changed by the user application. Thus if the state is set to “ApplicationReady pending” and the AR is terminated and reestablished afterwards, the submodule will be reported with “ApplicationReady pending” by the protocol stack.

Setting the submodule state back to “good” differs in handling depending if the data exchange is active or not.

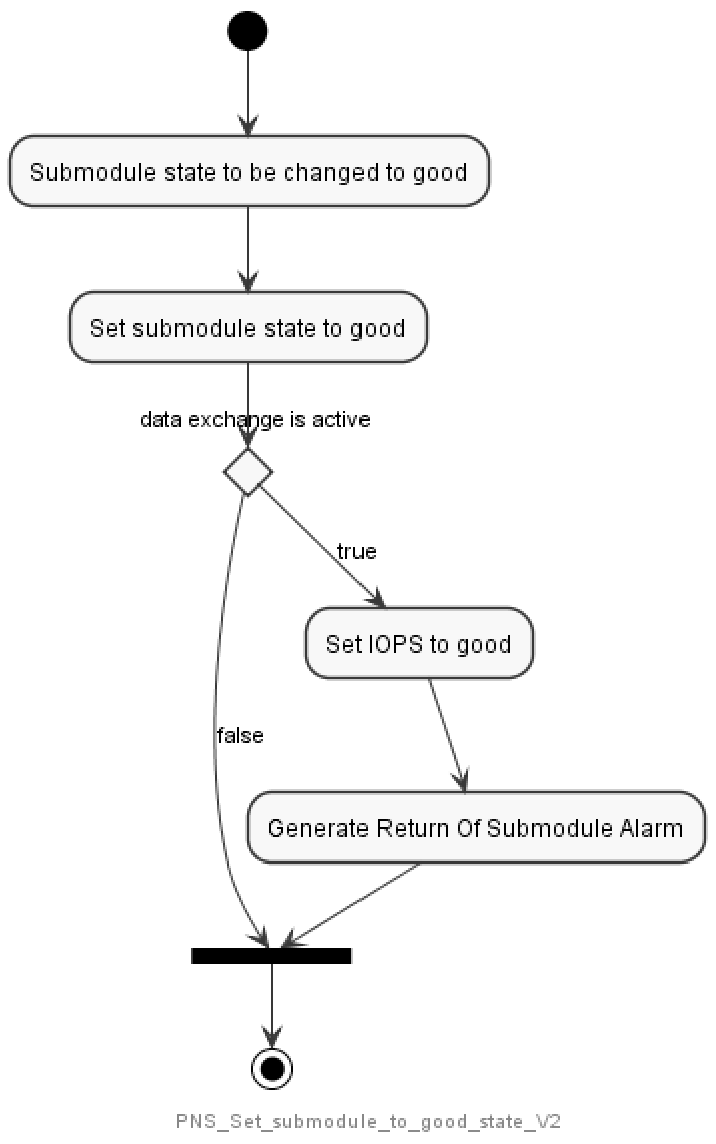

Submodule state to be changed to good
Set submodule state to good
data exchange is active
true
false
Set IOPS to good
Generate Return Of Submodule Alarm
PNS_Set_submodule_to_good_state_V2

Figure 25: SetSubModuleState from bad to good

6.4.19.1 Set Submodule State request

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

Only these states are supported: “ApplicationReady pending”, “Submodule ordinated locked” and “okay”.

It is possible to set the state of multiple submodules at the same time to the state “ApplicationReady pending”. However, it is not possible to set multiple submodules at the same time to the state “okay”. Setting the submodule state “okay” needs to be done in a single request for each submodule whose state is now “okay”.

Note:
The request packet has a limited size, depending on maximal DPM packet length. The maximal count of configured submodules is limited to 98. If more submodules are configured, the error code ERR_HIL_FAIL in ulSta will be returned.

Note:
A submodule in the state “ApplicationReady pending” shall have its IOPS set to “bad”. In consequence if the submodule state changes back to “good” the IOPS needs to be changed to “good” by the application. This requires sending a ReturnOfSubmodule Alarm (see section *Return of Submodule Alarm service* [▸ page 195]) which must be done by the application as well.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	16 + n	Packet data length in bytes
ulCmd	uint32_t	0x1F92	PNS_IF_SET_SUBM_STATE_REQ
Data			
ulSubmCnt	uint32_t	1…129	The amount of submodules contained in the packet.
ulApi	uint32_t		The API of the first submodule.
usSlot	uint16_t		The slot of the first submodule.
usSubslot	uint16_t		The subslot of the first submodule.
usSubmState	uint16_t		The new submodule state (see below)
usModuleState	uint16_t		Reserved for future use. Set to 0.

Table 166: PNS_IF_SET_SUBM_STATE_REQ–T - Set Submodule State request

Valu e	Name	
0	PNS_IF_SET_SUBM_STATE_SUBM_OKAY	The submodule state is okay, it is ready for valid data exchange.
1	PNS_IF_SET_SUBM_STATE_SUBM_SUPERO RD_LOCKED	The submodule is locked by application
2	PNS_IF_SET_SUBM_STATE_SUBM_APPL_R EADY_PENDING	The submodule is not ready for valid data exchange

Table 167: Values of usSubmState

Packet structure reference

```
/* the submodule is locked by application */
/* not yet supported by the stack, for future use */
#define PNS_IF_SET_SUBM_STATE_SUBM_SUPERORD_LOCKED (1)
/* the submodule is not yet ready for data exchange but not locked
*/
#define PNS_IF_SET_SUBM_STATE_SUBM_APPL_READY_PENDING (2)
/* the submodule is no longer locked */
#define PNS_IF_SET_SUBM_STATE_SUBM_OKAY (0)

typedef __HIL_PACKED_PRE struct PNS_IF_SET_SUBM_STATE_SUBMBLOCK_Ttag
{
  /* the API the submodule belongs to */
  uint32_t ulApi;
  /* the slot the submodule resides */
  uint16_t usSlot;
  /* the subslot the submodule resides */
  uint16_t usSubslot;
  /* the submodule state (see above) */
  uint16_t usSubmState;
  /* the module state, reserved for future use! */
  uint16_t usModuleState;
} __HIL_PACKED_POST PNS_IF_SET_SUBM_STATE_SUBMBLOCK_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SET_SUBM_STATE_DATA_REQ_Ttag
{
  /* amount of submodules contained in this packet */
  uint32_t ulSubmCnt;
  /* the first of ulSubmCnt submodule datasets */
  PNS_IF_SET_SUBM_STATE_SUBMBLOCK_T atSubm[98];
} __HIL_PACKED_POST PNS_IF_SET_SUBM_STATE_DATA_REQ_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SET_SUBM_STATE_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_SET_SUBM_STATE_DATA_REQ_T tData;
} __HIL_PACKED_POST PNS_IF_SET_SUBM_STATE_REQ_T;
```

6.4.19.2 Set Submodule State confirmation

The stack will return this packet to the application.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F93	PNS_IF_SET_SUBM_STATE_CNF

Table 168: PNS_IF_SET_SUBM_STATE_CNF–T - Set Submodule State confirmation

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_SET_SUBM_STATE_CNF_T;
```

6.4.20 Get Parameter service

The application can use this service to retrieve runtime parameters from the protocol stack.

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

Note:
The protocol stack ignores the submodule reference parameters for global parameters.

6.4.20.1 Get Parameter request

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	2 or 12	Packet data length in bytes: 2 for generic parameters, 12 for submodule specific parameters
ulCmd	uint32_t	0x1F64	PNS_IF_GET_PARAM_REQ
Data			
usPrmType	uint16_t	1 ... 7	Parameters to be retrieved, see table below.
usPadding	uint16_t	0	Set `usPadding` to zero for future compatibility.
ulApi	uint32_t		API of the submodule
usSlot	uint16_t		Slot of the submodule
usSubslot	uint16_t		Subslot of the submodule

Table 169: PNS_IF_GET_PARAM_REQ_T - Get Parameter request

© Hilscher 2011 – 2019

Packet structure reference

```
enum PNS_IF_PARAM_Etag
{
  PNS_IF_PARAM_MRP = 1,
  PNS_IF_PARAM_SUBMODULE_CYCLE = 2,
  PNS_IF_PARAM_ETHERNET = 3,
  PNS_IF_PARAM_DIAGNOSIS = 4, /* only useable for port- and
interface submodules */
  PNS_IF_PARAM_IM0_DATA = 5,
  PNS_IF_PARAM_IM5_DATA = 6,
  PNS_IF_PARAM_PORT_STATISTIC = 7, /* only useable for port-
submodules */
};

typedef enum PNS_IF_PARAM_Etag PNS_IF_PARAM_E;

typedef __HIL_PACKED_PRE struct PNS_IF_PARAM_COMMON_Ttag
{
  /** mandatory */
  uint16_t usPrmType;
  /** following parameters are optional */
  uint16_t usPadding;
  /** api of the associated submodule */
  uint32_t ulApi;
  /** slot of the associated submodule */
  uint16_t usSlot;
  /** subslot of the associated submodule */
  uint16_t usSubslot;
} __HIL_PACKED_POST PNS_IF_PARAM_COMMON_T;

typedef __HIL_PACKED_PRE struct PNS_IF_GET_PARAM_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_PARAM_COMMON_T tData;
} __HIL_PACKED_POST PNS_IF_GET_PARAM_REQ_T;
```

The following values are valid for `usPrmType`:

Symbolic name	Value	Description
PNS_IF_PARAM_MRP	1	Media Redundancy Parameters (Global Parameter)
PNS_IF_PARAM_SUBMODULE_CYCLE	2	Submodule Process Data Cycle (Output)
PNS_IF_PARAM_ETHERNET	3	Port submodule-related Ethernet parameters
PNS_IF_PARAM_DIAGNOSIS	4	Delivers all PDEV diagnosis entries. The application can use this service to read diagnosis entries generated by the protocol stack. Thus, all diagnoses generated by the protocol stack for submodules whose subslot is in the range 0x8000 to 0x8002 will be returned. In other words: Diagnoses generated by the application for these submodules will not be returned.
PNS_IF_PARAM_IM0_DATA	5	The values the protocol stack will deliver for I&M0 if handling of I&M by protocol stack is enabled.
PNS_IF_PARAM_IM5_DATA	6	The values the protocol stack will deliver form I&M5 if handling of I&M by protocol stack is enabled.
PNS_IF_PARAM_PORT_STATISTIC	7	Lists the current switch statistic counters including a validity mask of the given port submodule.

Table 170: PNS_IF_PARAM_E - Valid parameter options

6.4.20.2 Get Parameter confirmation

The stack will return this packet in answer to the Get Parameter Request. In case of success, this packet contains the requested information.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	> 2	Packet data length in bytes. Depends on the requested information.
ulSta	uint32_t		See section *Error codes and status codes* [page 292].
ulCmd	uint32_t	0x1F65	PNS_IF_GET_PARAM_CNF
Data			
union PNS_IF_PARAM_T			
tCommon	PNS_IF_PARAM_COMMON_T		Common structure
tMrp	PNS_IF_PARAM_MRP_T		MRP parameter
tSubmoduleCycle	PNS_IF_PARAM_SUBMODULE_CYCLE_T		Submodule cycle parameter
tEthernetPrm	PNS_IF_PARAM_ETHERNET_T		Port-submodule Ethernet parameter
tDiagnosisData	PNS_IF_DIAGNOSIS_T		PDev pending diagnosis
tIM0Data	PNS_IF_IM0_DATA_T		I&M0 data implemented in the protocol stack.
tIM5Data	PNS_IF_IM5_DATA_T		I&M5 data implemented in protocol stack.
tPortStatistic	PNS_IF_PARAM_PORT_STATISTIC_DATA_T		Port-submodule: switch statistic counters

Table 171: PNS_IF_GET_PARAM_CNF_T - Get Parameter confirmation

Packet structure reference

```
/** MRP was disabled by protocol stack itself */
#define PNS_IF_MRP_STATE_DISABLED (0)
/** MRP was enabled by protocol with default values.
* That happens if no explicit MRP parameterization took place */
#define PNS_IF_MRP_STATE_ENABLED_DEFAULT (1)
/** MRP was enabled explicitly by parameterization
* The controller enabled MRP explicitly by means of MRP parameter
record */
#define PNS_IF_MRP_STATE_ENABLED_PRM (2)
/** MRP was disabled explicitly by parameterization
* The controller disabled MRP explicitly by means of MRP parameter
record */
#define PNS_IF_MRP_STATE_DISABLED_PRM (3)

#define PNS_IF_MRP_ROLE_NONE (0)
#define PNS_IF_MRP_ROLE_MRP_CLIENT (1)
#define PNS_IF_MRP_ROLE_MRP_MANAGER (2)

typedef __HIL_PACKED_PRE struct PNS_IF_PARAM_MRP_Ttag
{
  uint16_t usPrmType;
  uint8_t bState;
  uint8_t bRole;
  HIL_UUID_T tUUID;
  uint8_t szDomainName[PNS_IF_MAX_NAME_OF_STATION];
} __HIL_PACKED_POST PNS_IF_PARAM_MRP_T;


typedef __HIL_PACKED_PRE struct PNS_IF_PARAM_SUBMODULE_CYCLE_Ttag
{
  uint16_t usPrmType;

  uint16_t usPadding;
  /** api of the associated submodule */
  uint32_t ulApi;
  /** slot of the associated submodule */
  uint16_t usSlot;
  /** subslot of the associated submodule */
  uint16_t usSubslot;
  /** update interval of this submodules process data in nanoseconds
*/
  uint32_t ulUpdateInterval;
  /** send base clock for the associated iocr */
  uint16_t usSendClock;
  /** send clock reduction for the associated iocr */
  uint16_t usReductionRatio;
  /** missing frames until timeout of the associated iocr */
  uint16_t usDataHoldFactor;
} __HIL_PACKED_POST PNS_IF_PARAM_SUBMODULE_CYCLE_T;

typedef __HIL_PACKED_PRE struct PNS_IF_PARAM_ETHERNET_Ttag
{
  uint16_t usPrmType;
  /* real Mutype */
  uint16_t usMauType;
  /* measured power budget in dB */
  uint32_t ulPowerBudget;
  /* measured cable delay in ns */
  uint32_t ulCableDelay;
} __HIL_PACKED_POST PNS_IF_PARAM_ETHERNET_T;

typedef __HIL_PACKED_PRE struct PNS_IF_DIAGNOSIS_ENTRY_Ttag
{
  /* subslot the diagnosis alarm belongs to */
  uint16_t usSubslot;
  /* Channel properties */
  uint16_t usChannelProp;
  /* Channel error type */
  uint16_t usChannelErrType;
  /* Channel extended error type */
  uint16_t usExtChannelErrType;
} __HIL_PACKED_POST PNS_IF_DIAGNOSIS_ENTRY_T;

#define MAX_DIAGNOSIS_ENTRIES_CNT 32

typedef __HIL_PACKED_PRE struct PNS_IF_DIAGNOSIS_Ttag
{
  uint16_t usPrmType;
  /* The amount of diagnosis entries contained in the confirmation
*/
  uint16_t usDiagnosisCnt;
  /** Diagnosis data, shall contain usDiagnosisCnt entries */
  PNS_IF_DIAGNOSIS_ENTRY_T atDiagnosis[MAX_DIAGNOSIS_ENTRIES_CNT];
} __HIL_PACKED_POST PNS_IF_DIAGNOSIS_T;

typedef __HIL_PACKED_PRE struct PNS_IF_PARAM_IM0_DATA_Ttag
{
  uint16_t usPrmType;
  uint16_t usPadding;
  PNS_IF_IM0_DATA_T tIM0Data;
} __HIL_PACKED_POST PNS_IF_PARAM_IM0_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_PARAM_IM5_DATA_Ttag
{
  uint16_t usPrmType;
  uint16_t usPadding;
  PNS_IF_IM5_DATA_T tIM5Data;
} __HIL_PACKED_POST PNS_IF_PARAM_IM5_DATA_T;

#define PNS_IF_VALID_MASK_PORT_STATISTIC_INOCTETS (0x0001)
#define PNS_IF_VALID_MASK_PORT_STATISTIC_OUTOCTETS (0x0002)
#define PNS_IF_VALID_MASK_PORT_STATISTIC_INDISCARDS (0x0004)
#define PNS_IF_VALID_MASK_PORT_STATISTIC_OUTDISCARDS (0x0008)
```

```
#define PNS_IF_VALID_MASK_PORT_STATISTIC_INERRORS (0x0010)
#define PNS_IF_VALID_MASK_PORT_STATISTIC_OUTERRORS (0x0020)

typedef __HIL_PACKED_PRE struct
PNS_IF_PARAM_PORT_STATISTIC_DATA_Ttag
{
  uint16_t usPrmType;
  uint16_t usPadding;
  uint32_t ulValidMask; /* indicator which of the fields in this
structure contain valid data */
  uint32_t ulInOctets; /* number of octets received by the port */
  uint32_t ulOutOctets; /* number of octets sent by the port */
  uint32_t ulInDiscards; /* number of frames discarded or dropped
due to low resources on reception */
  uint32_t ulOutDiscards; /* number of frames discarded before
sending */
  uint32_t ulInErrors; /* number of frames received with detected
error */
  uint32_t ulOutErrors; /* number of frames sent with detected error
*/
} __HIL_PACKED_POST PNS_IF_PARAM_PORT_STATISTIC_DATA_T;

typedef union PNS_IF_PARAM_Ttag PNS_IF_PARAM_T;

union PNS_IF_PARAM_Ttag
{
  PNS_IF_PARAM_COMMON_T tCommon;
  PNS_IF_PARAM_MRP_T tMrp;
  PNS_IF_PARAM_SUBMODULE_CYCLE_T tSubmoduleCycle;
  PNS_IF_PARAM_ETHERNET_T tEthernetPrm;
  PNS_IF_DIAGNOSIS_T tDiagnosisData;
  PNS_IF_PARAM_IM0_DATA_T tIM0Prm;
  PNS_IF_PARAM_IM5_DATA_T tIM5Prm;
  PNS_IF_PARAM_PORT_STATISTIC_DATA_T tPortStatistic;
};

typedef __HIL_PACKED_PRE struct PNS_IF_GET_PARAM_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_PARAM_T tData;
} __HIL_PACKED_POST PNS_IF_GET_PARAM_CNF_T;
```

Media Redundancy Parameters (usPrmType = 1)

Coding of `tMrp`

Variable	Type	Value / Range	Description
usPrmType	uint16_t	1	Mrp parameter option value
bState	uint8_t	0-3	Mrp state, see table *PNS_IF_PARAM_MRP_STATE_E - Valid values for MRP state* [▷ page 243]
bRole	uint8_t	0-1	Mrp Role of the Device, see *PNS_IF_PARAM_MRP_ROLE_E - Valid values for MRP Role* [▷ page 243]
tUUID	HIL_UUID_T		Mrp UUID
szDomainName	uint8_t[240]		Mrp Domain Name. String Length is Packet Length minus 20. For details about allowed characters, see section *Name encoding* [▷ page 323].

Table 172: PNS_IF_PARAM_MRP_T - MRP parameter

The following values are valid for `bState`

Symbolic Name	Numerical Value	Description
MRP_STATE_DISABLED	0	MRP was disabled by the protocol stack
MRP_STATE_ENABLED_DEFAULT	1	MRP was enabled by the protocol stack using default MRP parameters. (No parameter was written by controller / Factory Reset)
MRP_STATE_ENABLED_PRM	2	MRP was enabled on behalf of Controller. (The corresponding parameter was set with MRP enabled)
MRP_STATE_DISABLED_PRM	3	MRP was disabled on behalf of Controller. (The corresponding parameter was et with MRP disabled)

Table 173: PNS_IF_PARAM_MRP_STATE_E - Valid values for MRP state

The following values are valid for `bRole`

Symbolic Name	Numerical Value	Description
MRP_ROLE_NONE	0	MRP is disabled
MRP_ROLE_MRP_CLIENT	1	MRP is configured for Client Mode.

Table 174: PNS_IF_PARAM_MRP_ROLE_E - Valid values for MRP Role

Submodule Process Data Cycle (usPrmType = 2)

Coding of `tSubmoduleCycle`

Variable	Type	Value / Range	Description
usPrmType	uint16_t	2	Submodule cycle option value
usPadding	uint16_t		Ignore for future compatibility
ulApi	uint32_t		API of the submodule
usSlot	uint16_t		Slot of the submodule
usSubslot	uint16_t		Subslot of the submodule
ulUpdateInterval	uint32_t		Output process data update interval in nanoseconds
usSendClock	uint16_t		PROFINET Send Clock
usReductionRatio	uint16_t		PROFINET Reduction Ratio
usDataHoldFactor	uint16_t		Output process data watchdog factor: The output process data timeout is usDataHoldFactor x ulUpdateInterval.

Table 175: PNS_IF_PARAM_SUBMODULE_CYCLE_T - Submodule cycle parameter

Ethernet Parameters (usPrmType = 3)

Coding of `tEthernetPrm`

Variable	Type	Value / Range	Description
usPrmType	uint16_t	3	Ethernet parameter option value
usMauType	uint16_t		Real MAU type
ulPowerBudget	uint32_t		Measured power budget in dB
ulCableDelay	uint32_t		Measured cable delay in ns

Table 176: PNS_IF_PARAM_ETHERNET_T - Submodule cycle parameter

Diagnosis (usPrmType = 4)

Coding of `tDiagnosisData`

Variable	Type	Value / Range	Description
usPrmType	uint16_t	4	Diagnosis option value
usDiagnosisCnt	uint16_t	0..32	Number of diagnosis entries contained in the confirmation
atDiagnosis[32]	PNS_IF_DIAGNOSIS_ENTRY_T		Diagnosis data, see *PNS_IF_DIAGNOSIS_ENTRY_T - Diagnosis entry* [▸ page 244].

Table 177: PNS_IF_DIAGNOSIS_T - Diagnosis data

Coding of `atDiagnosis`

Variable	Type	Value / Range	Description
usSubslot	uint16_t		Subslot the diagnosis alarm belongs to
usChannelProp	uint16_t		Diagnosis channel properties. See table *Coding of Diagnosis* [▸ page 319].
usChannelErrType	uint16_t		Diagnosis error type, see table *Coding of Diagnosis* [▸ page 319]
usExtChannelErrType	uint16_t		Diagnosis extended error type, see *Coding of Diagnosis* [▸ page 321]

Table 178: PNS_IF_DIAGNOSIS_ENTRY_T - Diagnosis entry

I&M0 Parameters (usPrmType = 5)

Coding of `tIM0Prm`

Variable	Type	Value / Range	Description
usPrmType	uint16_t	5	Submodule cycle option value
usPadding	uint16_t		Ignore for future compatibility
tIM0Data	PNS_IF_IM0_DATA_T		See *Read I&M response* [▸ page 164].

Table 179: PNS_IF_PARAM_IM0_DATA_T - Stack I&M0 record content

I&M5 Parameters (usPrmType = 6)

Coding of `tIM5Prm`

Variable	Type	Value / Range	Description
usPrmType	uint16_t	6	Submodule cycle option value
usPadding	uint16_t		Ignore for future compatibility
tIM5Data	PNS_IF_IM5_DATA_T		See *Read I&M response* [▸ page 164].

Table 180: PNS_IF_PARAM_IM5_DATA_T – Stack I&M5 record content

Switch statistic counters (usPrmType = 7)

Coding of `tPortStatistic`

Variable	Type	Value / Range	Description
usPrmType	uint16_t	7	PNS_IF_PARAM_PORT_STATISTIC
usPadding	uint16_t	0	-
ulValidMask	uint32_t	0..0x3F	Bit field indicating which elements of this structure contain valid data.
ulInOctets	uint32_t		Number of octets received by this port.
ulOutOctets	uint32_t		Number of octets sent by this port.
ulInDiscards	uint32_t		Number of frames that the switch • did not forward to higher sublayer due to resource constraints • discarded after receipt due to port state
ulOutDiscards	uint32_t		Number of valid frames that were discarded.
ulInErrors	uint32_t		Number of received erroneous frames.
ulOutErrors	uint32_t		Number of frames that could not be transmitted due to an error.

Table 181: PNS_IF_PARAM_PORT_STATISTIC_DATA_T – Port statistics data

6.4.21 Add PE Entity service

The Add PE Entity service is part of the PE ASE implementation of the PROFINET Device stack. The application has to use this service to add a PE entity to the PE ASE. The implementation supports one PE entity per submodule.

The protocol stack will validate the parameters of the request packet and add the entity if the addressed submodule

- exists, and
- is plugged, and
- has no PE entity yet.

The entities' operational mode will be initialized to the value "Ready to operate" (0xFF). The protocol stack will internally generate the PROFINET alarms associated with this operation.

Note:
By default, all PE entity services are deactivated. In order to activate the PE entity services, use the *Set OEM Parameters request* [page 72] with ulParamType set to 10.

6.4.21.1 Add PE Entity request

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	8	Packet data length in bytes
ulCmd	uint32_t	0x1F94	PNS_IF_ADD_PE_ENTITY_REQ
Data			
ulApi	uint32_t		The API of the submodule associated with the the PE entity.
usSlot	uint16_t		The slot of the submodule associated with the PE entity.
usSubslot	uint16_t		The subslot of the submodule associated with the PE entity.

Table 182: PNS_IF_ADD_PE_ENTITY_REQ–T – Add PE Entity request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_ADD_PE_ENTITY_REQ_DATA_Ttag
{
  uint32_t ulAPI;
  uint16_t usSlot;
  uint16_t usSubslot;
} __HIL_PACKED_POST PNS_IF_ADD_PE_ENTITY_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_ADD_PE_ENTITY_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_ADD_PE_ENTITY_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_ADD_PE_ENTITY_REQ_T;
```

© Hilscher 2011 – 2019

6.4.21.2 Add PE Entity confirmation

The protocol stack returns this packet to the application in order to confirm adding a PE entity.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F95	PNS_IF_ADD_PE_ENTITY_CNF

Table 183: PNS_IF_ADD_PE_ENTITY_CNF–T – Add PE Entity confirmation

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_ADD_PE_ENTITY_CNF_T;
```

© Hilscher 2011 – 2019

6.4.22 Remove PE Entity service

The Remove PE Entity service is part of the PE ASE implementation of the PROFINET Device Stack. The application has to use this service to remove a PE entity from the PE ASE.

The protocol stack will validate the parameters of the packet and removes the PE entity if the addressed submodule

- is plugged, and
- independent whether the submodule has an PE entity or not.

Note:
By default, all PE entity services are deactivated. In order to activate the PE entity services, use the *Set OEM Parameters request* [page 72] with ulParamType set to 10.

This service exists for symmetric reasons only and thus this service is not intended to be used as there is no real use case for it.

6.4.22.1 Remove PE Entity request

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	8	Packet data length in bytes
ulCmd	uint32_t	0x1F96	PNS_IF_REMOVE_PE_ENTITY_REQ
Data			
ulApi	uint32_t		The API of the submodule associated with the PE entity.
usSlot	uint16_t		The Slot of the submodule associated with the PE entity.
usSubslot	uint16_t		The subslot of the submodule associated with the PE entity.

Table 184: PNS_IF_REMOVE_PE_ENTITY_REQ–T – Remove PE Entity request

Packet structure reference

```
typedef PNS_IF_ADD_PE_ENTITY_REQ_DATA_T
PNS_IF_REMOVE_PE_ENTITY_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_REMOVE_PE_ENTITY_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_REMOVE_PE_ENTITY_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_REMOVE_PE_ENTITY_REQ_T;
```

6.4.22.2 Remove PE Entity confirmation

The protocol stack returns this packet to the application in order to confirm removing a PE entity.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F97	PNS_IF_REMOVE_PE_ENTITY_CNF

Table 185: PNS_IF_REMOVE_PE_ENTITY_CNF–T – Remove PE Entity confirmation

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_REMOVE_PE_ENTITY_CNF_T;
```

6.4.23 Update PE Entity service

The Update PE Entity service is part of the PE ASE implementation of the PROFINET stack. The application has to use this service to update the operational mode of a PE entity within the PE ASE. The protocol stack will internally generate the PROFINET alarms associated with this operation.

The protocol stack will validate the parameters of the packet and updates the operational mode if the addressed submodule

- is plugged, and
- uses an PE entity.

Note:
By default, all PE entity services are deactivated. In order to activate the PE entity services, use the *Set OEM Parameters request* [▸ page 72] with ulParamType set to 10.

6.4.23.1 Update PE Entity request

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	9	Packet data length in bytes
ulCmd	uint32_t	0x1F98	PNS_IF_UPDATE_PE_ENTITY_REQ
Data			
ulApi	uint32_t		The API of the submodule associated with the PE entity.
usSlot	uint16_t		The slot of the submodule associated with the PE entity.
usSubslot	uint16_t		The subslot of the submodule associated with the PE entity.
bOperationalMode	uint8_t	0 … 0xFF	The current operational mode of the PE entity as defined by PROFIenergy specification.

Table 186: PNS_IF_UPDATE_PE_ENTITY_REQ–T – Update PE Entity request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct
PNS_IF_UPDATE_PE_ENTITY_REQ_DATA_Ttag
{
  uint32_t ulAPI;
  uint16_t usSlot;
  uint16_t usSubslot;
  uint8_t bOperationalMode;
} __HIL_PACKED_POST PNS_IF_UPDATE_PE_ENTITY_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_UPDATE_PE_ENTITY_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  PNS_IF_UPDATE_PE_ENTITY_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_UPDATE_PE_ENTITY_REQ_T;
```

6.4.23.2 Update PE Entity confirmation

The protocol stack returns this packet to the application in order to confirm updating a PE entity. The confirmation is be returned independently of associated PROFINET alarm processing, e.g. before the required alarms have been issued.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F99	PNS_IF_UPDATE_PE_ENTITY_CNF

Table 187: PNS_IF_UPDATE_PE_ENTITY_CNF–T – Update PE Entity confirmation

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T PNS_IF_UPDATE_PE_ENTITY_CNF_T;
```

6.4.24 Send Alarm service

The application can use this service to send a user specific alarm.

Note:
This service replaces the *Process Alarm service* [▸ page 189] and can be used to indicate Process alarm or other user specific alarm types.

6.4.24.1 Send Alarm request

This packet causes the stack to send a user specific Alarm to the IO-Controller.

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	LFW: 0x00000020, LOM: PNS_IF	LFW: AP task, LOM: PNS_IF
ulLen	uint32_t	22 + n	Packet data length in bytes. n is the value of usLenAlarmData.
ulCmd	uint32_t	0x1F5C	PNS_IF_SEND_ALARM_REQ
Data			
ulApi	uint32_t		The API the alarm belongs to.
ulSlot	uint32_t		The Slot the alarm belongs to.
ulSubslot	uint32_t		The Subslot the alarm belongs to.
hAlarmHandle	uint32_t		A user specific alarm handle. The application is free to choose any value.
usAlarmType	uint16_t		Alarm type to generate see below
usUserStructId	uint16_t	0x0000 .. 0x7FFF, 0X8320	The application always has to provide a User Structure Identifier. The PROFINET specification allows an application not to provide a User Structure Identifier, which is not supported by the stack.
usAlarmDataLen	uint16_t	 1 … 172 173 … 1024	The length of the alarm data Length supported by any IO-Controller. Length allowed by the specification but not supported by any IO-Controller. See parameter description below.
tAlarmData	PNS_IF_ALARM_DATA_T		The alarm data.

Table 188: PNS_IF_SEND_ALARM_REQ_T Send Alarm request

© Hilscher 2011 – 2019

Packet structure reference

```
typedef enum {
  PNS_IF_ALARM_TYPE_PROCESS = 0x0002,
  PNS_IF_ALARM_TYPE_STATUS = 0x0005,
  PNS_IF_ALARM_TYPE_UPDATE = 0x0006,
  PNS_IF_ALARM_TYPE_ISO_PROBLEM = 0x0010,
  PNS_IF_ALARM_TYPE_UPLOAD_RETRIEVAL = 0x001E,
  PNS_IF_ALARM_TYPE_MANUF_SPEC_START = 0x0020,
  PNS_IF_ALARM_TYPE_MANUF_SPEC_END = 0x007F,
} PNS_IF_ALARM_TYPE_E;

typedef enum {
  PNS_IF_ALARM_USER_STRUCTURE_ID_MANUF_SPEC_STAR = 0x0000,
  PNS_IF_ALARM_USER_STRUCTURE_ID_MANUF_SPEC_END = 0x7FFF,
} PNS_IF_ALARM_USER_STRUCTURE_ID_E;

/** Structure definition of Record data objects.
 * These data are commissioning parameters which shall be
permanently stored at the IO Controller or IO Supervisor.
 * Record data objects which are part of the GSDML file and conveyed
from the engineering system shall not be part of Upload and
retrieval records */
typedef __HIL_PACKED_PRE struct
PNS_IF_UPLOAD_RETRIEVAL_RECORD_DATA_Ttag
{
  uint32_t ulRecordIndex;
  uint32_t ulRecordLength;
}__HIL_PACKED_POST PNS_IF_UPLOAD_RETRIEVAL_RECORD_DATA_T;

/* Upload & Retrieval sub-Alarm type */
typedef enum PNS_IF_ALARM_UPLOADT_RETRIEVAL_SUBTYPE_Etag
{
  PNS_IF_ALARM_UPLOAD_SELECTED_RECORDS = 0x0001,
  PNS_IF_ALARM_RETRIEVAL_SELECTED_RECORDS = 0x0002,
  PNS_IF_ALARM_RETRIEVAL_ALL_RECORDS = 0x0003,
} PNS_IF_ALARM_UPLOADT_RETRIEVAL_SUBTYPE_E;

/** Structure definition of Upload & Retrieval alarm item data .*/
typedef __HIL_PACKED_PRE struct PNS_IF_UPLOAD_RETRIEVAL_DATA_Ttag
{
  /** Alarm subtype see \ref
PNS_IF_ALARM_UPLOADT_RETRIEVAL_SUBTYPE_Etag*/
  uint8_t bAlarmSubtype;
  /** This part of Upload & Retrieval alarm data has a dynamic
structure.
   * It shall contain device specific records data.
   * The Length of this structure will be derived from
usAlarmDataLen */
  PNS_IF_UPLOAD_RETRIEVAL_RECORD_DATA_T tRecords[];
}__HIL_PACKED_POST PNS_IF_UPLOAD_RETRIEVAL_DATA_T;

typedef __HIL_PACKED_PRE struct
PNS_IF_PROCESS_ALARM_CHANNEL_DATA_Ttag
{
  /* Channel number issuing the process alarm */
  uint16_t usChannelNum;
  /* Channel properties */
  uint16_t usChannelProp;
  /* Reason for process alarm */
  uint16_t usReason;
  /* Extended Reason for process alarm */
  uint16_t usExtReason;
  /* Additional Value */
  uint8_t abReasonAddValue[128]; /* hint: no additional reason is
coded with length 4 and value 00:00:00:00 */
} __HIL_PACKED_POST PNS_IF_PROCESS_ALARM_CHANNEL_DATA_T;

/* alarm data */
```

```
typedef union PNS_IF_ALARM_DATA_Ttag
{
  uint8_t abManufacturerData[PNS_IF_MAX_ALARM_DATA_LEN]; /*
ATTENTION: Recommended not to use more than 172 byte of data for
compatibility reasons */
  PNS_IF_UPLOAD_RETRIEVAL_DATA_T tUploadRetrieval;
  PNS_IF_PROCESS_ALARM_CHANNEL_DATA_T tProcessAlarmChannelData; /*
only used for USI 0x8320 */
}PNS_IF_ALARM_DATA_T;

/* Request packet */
typedef __HIL_PACKED_PRE struct PNS_IF_SEND_ALARM_REQ_DATA_Ttag
{
  /* API of the submodule that issues the alarm */
  uint32_t ulApi;
  /* Slot of the submodule that issues the alarm */
  uint32_t ulSlot;
  /* Subslot of the submodule that issues the alarm */
  uint32_t ulSubslot;
  /* AlarmHandle freely chosen by application for identification
purpose */
  uint32_t hAlarmHandle;
  /* AlarmType to generate, see PNS_IF_USER_ALARMTYPES_E */
  uint16_t usAlarmType;
  /* User structure identifier of the alarm will only be used in
case
   * of manufacturer specific alarm type */
  uint16_t usUserStructId;
  /* length of the alarm data (thus length of content in
abAlarmData) */
  uint16_t usAlarmDataLen;
  /* alarm data */
  PNS_IF_ALARM_DATA_T tAlarmData;
} __HIL_PACKED_POST PNS_IF_SEND_ALARM_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SEND_ALARM_REQ_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_SEND_ALARM_REQ_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_SEND_ALARM_REQ_T;
```

Valid values for `usAlarmType`:

Symbolic Name	**Numerical Value**	**Description**
PNS_IF_ALARM_TYPE_PROCESS	0x0002	Process alarm notification
PNS_IF_ALARM_TYPE_STATUS	0x0005	status change notification
PNS_IF_ALARM_TYPE_UPDATE	0x0006	Change of parameter notification
PNS_IF_ALARM_TYPE_ISO_PROBLEM	0x0010	Isochronous Mode Problem Notification
PNS_IF_ALARM_TYPE_UPLOAD_RETRIEVAL	0x001E	Alarm notification for upload and storage of device specific record data objects
PNS_IF_ALARM_TYPE_MANUF_SPEC	0x0020 - 0x007F	manufacturer specific alarm types
Reserved	other	Reserved for future usage

Table 189: Valid values for Alarm Type

Valid values for `usUserStructId`:

Symbolic Name	Numerical Value	Description
PNS_IF_ALARM_USER_STRUCTURE_ID_MANUF_SPEC	0x0000 – 0x7FFF	Manufacturer-specific structure
-	0x8320	Process Alarm with channel coding. Only if usAlarmType = PNS_IF_ALARM_TYPE_PROCESS.
other	Reserved	usUserStructId will in case of upload and retrieval alarm type automatically assigned by the stack.

Table 190: Valid values for User Structure Identifier

usAlarmDataLen

Note:
The PROFINET specification guarantees a total alarm length of 200 byte to be exchangeable between IO-Device and IO-Controller. IO-Controllers may support more than 200 byte but the IO-Device cannot rely on it. For this reason, we highly recommend to you **not** to generate any alarm that exceeds 200 bytes.

The 200 bytes include 28 bytes for submodule and protocol-related parameters. This means that the pure application data including the User Structure Identifier is limited to 174 byte in this case. Note that the User Structure Identifier is part of the 28 bytes and that it is already included in the `PNS_IF_SEND_PROCESS_ALARM_REQ_DATA_T` structure and therefore should not be used with alarm data length of more than 172 bytes.

tAlarmData

Variable	Type	Value / Range	Description
abManufacturerData	uint8_t[1024]	1 - 3	Manufacturer-specific alarm data
tUploadRetrieval	PNS_IF_UPLOAD_RETRIEVAL_DATA_T		Upload and retrieval alarm data. Must be used in conjunction with usAlarmType = PNS_IF_ALARM_TYPE_UPLOAD_RETRIEVAL
tProcessAlarmChannelData	PNS_IF_PROCESS_ALARM_CHANNEL_DATA_T		Process Alarm with channel coding. Must be used in conjunction with usAlarmType = PNS_IF_ALARM_TYPE_PROCESS and usUserStructId = 0x8320

Table 191: Alarm data definition

abManufacturerData

This parameter can be used to report optional manufacturer data. It shall only be used in conjunction with following alarm types:

- Process alarm
- Status alarm
- Update alarm
- Isochronous mode problem alarm
- Manufacturer specific alarm

tUploadRetrieval

This parameter shall only be used if the alarm type is set to “Upload and Retrieval alarm”.

With this parameter, the application indicates the existence of device specific record objects that shall be stored outside of IO Device or retrieval request of already stored record objects.

Variable	Type	Value / Range	Description
bAlarmSubtype	uint8_t	1 - 3	Alarm subtype of upload and retrieval notification
tUploadRetrievalData	PNS_IF_UPLOAD_RETRIEVAL_RECORD_DATA_T[1-127]		The number of entries in the list is determined by the packet length.

Table 192: Upload and Retrieval structure definition

Valid values are for `bAlarmSubtype`:

Symbolic Name	Numerical Value	Description
PNS_IF_ALARM_UPLOAD_SLECTED_RECORDS	1	Shall be used to indicate the existence of record data object, which shall be stored outside of IO Device.
PNS_IF_ALARM_RETRIEVAL_SLECTED_RECORDS	2	Shall be used to send a retrieval request for specifc record objects
PNS_IF_ALARM_RETRIEVAL_ALL_RECORDS	3	Shall be used to send a retrieval request for all stored record objects

Table 193: Valid value for Upload and Retrieval Alarm Subtype

Variable	Type	Value / Range	Description
ulRecordIndex	uint32_t	0x00000000 - 0x0000FFFF	Index of record data object that shall be uploaded or retrieval
ulRecordLength	uint32_t	0x00000001-0xFFFFFFFF	Length of record data object that shall be uploaded or retrieval

Table 194: Upload and Retrieval Record Data Description

Process Alarm with channel coding

To send a Process Alarm with channel coding, the application has to use usAlarmType = PNS_IF_ALARM_TYPE_PROCESS and usUserStructId = 0x8320. This coding allows the OEM to describe the alarm content in the GSDML file.

Variable	Type	Value / Range	Description
usChannelNum	uint16_t	See section *Add Channel Diagnosis service* [▹ page 221].	Channel number issuing the process alarm.
usChannelProp	uint16_t	See section *Add Channel Diagnosis service* [▹ page 221].	Channel properties
usReason	uint16_t	For values, see table below.	Reason for process alarm.
usExtReason	uint16_t	For values, see table below.	Extended Reason for process alarm.
abReasonAddValue[128]	uint8_t	For length and values, see table below.	Reason additional values

Table 195: Channel coding structure definition

Value for usReason:

Value	Description	Text
0x0000	Reserved	Unknown reason
0x0001	Internal gate opening (Gate start)	Counter or way measurement starts Internal gate opening (Gate start)
0x0002	Internal gate closing (Gate stop)	Counter or way measurement stops Internal gate closing (Gate stop)
0x0003	Overflow (High counting limit violated)	Counter or way measurement register Overflow (High counting limit violated)
0x0004	Underflow (Low counting limit violated)	Counter or way measurement register Underflow (Low counting limit violated)
0x0005	Compare event for DQ0 has occured	Compare of counter or way measurement register with a parameterized value “DQ0” (or “first digital output”) shows hit (Compare event for “first digital output” has occurred)
0x0006	Compare event for DQ1 has occured	Compare of counter or way measurement register with a parameterized value “DQ1” (or “second digital output”) shows hit (Compare event for “second digital output” has occurred)
0x0007	Zero pass	Counter or way measurement signed register hits or passes zero (Zero pass)
0x0008	New capture value available	New capture value available
0x0009	Synchronization of the counter by an external signal	Counter or way measurement register value is set by an external signal (Synchronization of the counter by an external signal)
0x000A	Direction reversal	Counter or way measurement register changes counting direction (Direction reversal)
0x000B – 0x000F	Reserved	Unknown reason
0x0010	Rising edge	The digital channel detects a rising edge (Rising edge)
0x0012	Falling edge	The digital channel detects a falling edge (Falling edge)
0x0013	Upper user limit 1 exceeded	The parameterized compare value 1 for the channel is exceeded (Upper user limit 1 exceeded)
0x0014	Lower user limit 1 undershot	The parameterized compare value 1 for the channel is undershot (Lower user limit 1 undershot)
0x0015	Upper user limit 2 exceeded	The parameterized compare value 2 for the channel is exceeded (Upper user limit 2 exceeded)
0x0016	Lower user limit 2 undershot	The parameterized compare value 2 for the channel is undershot (Lower user limit 2 undershot)
0x0017	Upper user limit 3 exceeded	The parameterized compare value 3 for the channel is exceeded (Upper user limit 3 exceeded)

Value	Description	Text
0x0018	Lower user limit 3 undershot	The parameterized compare value 3 for the channel is undershot (Lower user limit 3 undershot)
0x0019	Upper user limit 4 exceeded	The parameterized compare value 4 for the channel is exceeded Upper user limit 4 exceeded
0x001A	Lower user limit 4 undershot	The parameterized compare value 4 for the channel is undershot (Lower user limit 4 undershot)
0x001B – 0x00FF	Reserved	Unknown reason
0x0100 – 0x7FFF	Manufacturer specific	Manufacturer specific reason
0x8000 – 0x8FFF	Reserved	Reserved
0x9000 – 0x9FFF	Reserved for profiles	Profile specific reason
0xA000 – 0xFFFF	Reserved	Reserved

Table 196: Values for usReason

Value for usExtReason:

Value	Description	Text
0x0000	Unspecified	Unspecified
0x0001 – 0x7FFF	Manufacturer specific	Manufacturer specific reason detail
0x8000 – 0x8FFF	Reserved	Reserved
0x9000 – 0x9FFF	Reserved for profiles	Profile specific reason detail
0xA000 – 0xFFFF	Reserved	Reserved

Table 197: Values for usExtReason

Length and value for abReasonAddValue:

Length	Value	Description
0x04	0x00, 0x00, 0x00, 0x00	No additional information Independent from usReason and usExtReason.
	other	Definition depending on usReason and usExtReason
0x05 – 0x80	0x00 .. 0xFF (for each byte)	Definition depending on usReason and usExtReason
other	-	Reserved

Table 198: Length and values for abReasonAddValue

6.4.24.2 Send Alarm confirmation

This packet is returned to the application by the stack after the controller confirmed the alarm. The reaction of the IO-Controller is reported to the application within the confirmation.

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	12	Packet data length in bytes
ulSta	uint32_t		See section *Error codes and status codes* [▸ page 292].
ulCmd	uint32_t	0x1F5E	PNS_IF_SEND_ALARM_CNF
Data			
ulReserved	uint32_t		Reserved
hAlarmHandle	uint32_t		The user specific alarm handle.
ulPnio	uint32_t		PROFINET error code consists of ErrCode, ErrDecode, ErrCode1 and ErrCode2. See section *PROFINET status codes* [▸ page 311].

Table 199: PNS_IF_SEND_ALARM_CNF_T - Send Alarm confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct PNS_IF_SEND_ALARM_CNF_DATA_Ttag
{
  uint32_t ulReserved ;
  uint32_t hAlarmHandle;
  /* Profinet error code, consists of ErrCode, ErrDecode, ErrCode1
and ErrCode2 */
  uint32_t ulPnio;
} __HIL_PACKED_POST PNS_IF_SEND_ALARM_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct PNS_IF_SEND_ALARM_CNF_Ttag
{
  /** packet header */
  HIL_PACKET_HEADER_T tHead;
  /** packet data */
  PNS_IF_SEND_ALARM_CNF_DATA_T tData;
} __HIL_PACKED_POST PNS_IF_SEND_ALARM_CNF_T;
```

7 Linkable Object Module (LOM)

7.1 Usage of Linkable Object Module

Note:
This section only applies if the stack is used as Linkable Object Module. If the stack is used as Loadable Firmware this section can be ignored. For more details of the configuration, see the example provided on NXLOM CD. In case of any lack of clarity within this document please refer to the example provided on NXLOM CD.

If the stack is used as Linkable Object Module, the user has to create its own configuration file (which among others contains task start-up parameters and hardware resource declarations). The PROFINET stack itself is started by invoking the function `PNS_StackInit()` from the user application.

Note:
Since PROFINET Stack Version 3.5.0.0 the PROFINET Stack is started by calling the function `PNS_StackInit()`. This function automatically initializes required resources and launches the affected tasks.

7.1.1 Config.c

The `config.c` file contains among others the hardware resource declarations and the static task list.

7.1.1.1 Hardware Resources

Besides the standard rcX resources and the user application resources, the following hardware resources should be declared. These are used by the PROFINET Device stack.

netX 100 or netX 500

Resource	Peripheral Type	Peripheral Subtype	Instance
Ethernet Physical Interface	RX_PERIPHERAL_TYPE_PHY		0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XMACRPU	0, 1 and 3*
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XMACTPU	0, 1 and 3*
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XPEC	0, 1 and 3
Fifo units	RX_PERIPHERAL_TYPE_FIFOCHANNEL		0, 1 and 3
Ethernet Driver	RX_PERIPHERAL_TYPE_EDD		0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msync0	0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msync1	1
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msync3	3
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com0	0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com1	1
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com3	3
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_systime_ns	0

Table 200: Hardware resources used by the PROFINET Device stack for netX 100/500

See also section *Disable XMAC3* [▸ page 263].

netX 50

Resource	Peripheral Type	Peripheral Subtype	Instance
Ethernet Physical Interface	RX_PERIPHERAL_TYPE_PHY		0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XMACRPU	0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XMACTPU	0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XPEC	0 and 1
Fifo units	RX_PERIPHERAL_TYPE_FIFOCHANNEL		0 and 1
Ethernet Driver	RX_PERIPHERAL_TYPE_EDD		0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msync0	0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msync1	1
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com0	0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com1	1
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_systime_ns	0

Table 201: Hardware resources used by the PROFINET Device stack for netX 50

netX 51

Resource	Peripheral Type	Peripheral Subtype	Instance
Ethernet Physical Interface	RX_PERIPHERAL_TYPE_PHY		0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XMACRPU	0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_XMACTPU	0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_RPEC	0 and 1
xC Units	RX_PERIPHERAL_TYPE_XC	RX_XC_TYPE_TPEC	0 and 1
Fifo units	RX_PERIPHERAL_TYPE_FIFOCHANNEL		0 and 1
Ethernet Driver	RX_PERIPHERAL_TYPE_EDD		0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msync0	0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_msync1	1
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com0	0
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NETX_VIC_IRQ_STAT_com1	1
Interrupt	RX_PERIPHERAL_TYPE_INTERRUPT	SRT_NX51_vic_irq_status_systime_ns	0

Table 202: Hardware resources used by the PROFINET Device stack for netX 51

7.1.1.2 Disable XMAC3

netX100 or netX500

Using netX100 or netX500 it is possible to exclude the XMACRPU3 and XMACTPU3 units from the PROFINET Device stack hardware list (*Hardware resources used by the PROFINET Device stack for netX 100/500* [▸ page 261]), but XPEC3 unit is still in use. XMACRPU3 and XMACTPU3 can be used for another purpose.

Without the XMACRPU3 and XMACTPU3 units the PROFINET stack has the following restrictions:

- disabled generation of the external SYNC signals on Sync0 and Sync1 pins that is described in reference [],
- it is not possible to build synchronous application described in reference [] because of disabling of synchronization interface,
- certification for "Conformance Class C" is not possible because there is no external SYNC signal to check the synchronization (no sync pin available), but IRT mode is still available.

To exclude the XMACRPU3 and XMACTPU3 units from usage of the PROFINET Device stack do not define their names in the EDD-parameter list RX_EDD_PARAMETERS_T, just initialize to `NULL`:

```
STATIC RX_EDD_PARAMETERS_T g_atEddParam[]=
{
...
{ RX_EDD_PARAM_XPEC_NAME, "PNS_XPEC", 0 }, /* XPEC0 */
{ RX_EDD_PARAM_XMAC_RPU_NAME, "PNS_XMACRPU", 0 }, /* XMAC0_RPU */
{ RX_EDD_PARAM_XMAC_TPU_NAME, "PNS_XMACTPU", 0 }, /* XMAC0_TPU */

{ RX_EDD_PARAM_XPEC_NAME, "PNS_XPEC", 1 }, /* XPEC1 */
{ RX_EDD_PARAM_XMAC1_RPU_NAME, "PNS_XMACRPU", 1 }, /* XMAC1_TPU */
{ RX_EDD_PARAM_XMAC1_TPU_NAME, "PNS_XMACTPU", 1 }, /* XMAC1_RPU */

{ RX_EDD_PARAM_XPEC3_NAME, "PNS_XPEC", 3 }, /* XPEC3 */
{ RX_EDD_PARAM_XMAC3_RPU_NAME, NULL, 3 }, /* don't use XMAC3_RPU for
EDD */
{ RX_EDD_PARAM_XMAC3_TPU_NAME, NULL, 3 }, /* don't use XMAC3_TPU for
EDD */
...
};

STATIC RX_EDD_SET_T g_atEdd[] = {
{{PNS_EDD_IDENTIFY_NAME,RX_PERIPHERAL_TYPE_EDD,0}, /* Ethernet
Device Driver */
0, /* use Edd0 */
"PROFINET Switch-Cut-Through", /* NIC name */
RX_EDD_MODE_DEFAULT,
FALSE,
g_atEddParam,
&trEddHalPND
} };
```

7.1.1.3 Systime Unit

The PROFINET Device stack uses the SysTime unit (namely the SysTime-compare interrupt `SRT_NETX_VIC_IRQ_STAT_systime_ns`) for internal RT scheduling. The stack reconfigures the SysTime unit differently compared to default rcX, therefore the rcX function delivers following information:

```
SYSTIME_TIMESTAMP_T timestamp;
Drv_SysTimeGetTime(&timestamp);

Timestamp.ulTimeNs - lower 4 bytes of the system time in step of 10
ns
Timestamp. ulTimeS - higher 4 bytes of the system time in step of 10
ns

So the “timestamp‟ is a 64 bit value of system time in step of 10 ns
```

7.1.1.4 Static Task List

Since PROFINET Stack version 3.5, the static task list should contain the Timer Task, The TCP/IP Task and the application tasks:

```
/* Stack-sizes for the static tasks */
#define TSK_STACK_SIZE_TLR_TIMER 512 /* Memory assignement for the
TimerTask */
#define TSK_STACK_SIZE_TCP_TASK 1024 /* Stack Size in multiples of
UINTs */
#define TSK_STACK_SIZE_PCK_DEMO 2048 /* Stack Size in multiples of
UINTs */

/* Stack for the "TLR TIMER" task /
STATIC UINT auTskStack_Tlr_Timer[TSK_STACK_SIZE_TLR_TIMER];

/* Stack for the "TCP_UDP" task */
STATIC UINT auTskStack_Tcp_Task[TSK_STACK_SIZE_TCP_TASK];

/* Stack for the "Packet API Demo" task */
STATIC UINT auTskStack_Pck_API_Demo[TSK_STACK_SIZE_PCK_DEMO];

RX_STATIC_TASK_T g_atStaticTasks[] = {
{"TlrTimer", /* Identification */
TSK_PRIO_12, TSK_TOK_12, /* Priority Token ID */
0, /* Instance 0 */
&auTskStack_Tlr_Timer[0], /* Pointer to Stack */
TSK_STACK_SIZE_TLR_TIMER, /* Size of Task Stack */
0,
RX_TASK_AUTO_START, /* Start task automatically */
(VOID*)TaskEnter_TlrTimer, /* Task main() Function */
(VOID*)TaskExit_TlrTimer, /* Task leave callback */
(UINT32)&g_tTlrTimerPrm, /* Startup Parameter */
{0,0,0,0,0,0,0,0} /* Reserved */
},
{"TCP_UDP", /* Identification */
TSK_PRIO_30, TSK_TOK_30, /* Priority and Token ID */
0, /* Instance 0 */
&auTskStack_Tcp_Task[0], /* Pointer to Stack */
TSK_STACK_SIZE_TCP_TASK, /* Size of Task Stack */
0,
RX_TASK_AUTO_START, /* Start task automatically */
(void FAR*)TaskEnter_TcpipTcpTask, /* Task main() Function */
NULL, /* Task leave callback */
(UINT32)&g_tTcpIpPrm, /* Startup Parameter */
{0,0,0,0,0,0,0,0} /* Reserved */
},
/* User AP-Task for acyclical and low-priority services */
{"APP_LOW", /* Identification */
TSK_PRIO_40, TSK_TOK_40, /* Priority and Token ID */
0, /* Instance to 0 */
& auTskStack_Pck_API_Demo[0], /* Pointer to Stack */
TSK_STACK_SIZE_PCK_DEMO, /* Size of Task Stack */
0,
RX_TASK_AUTO_START, /* Start task automatically */
(void FAR*)TaskEnter_UserAp, /* Task main() Function */
NULL, /* Task leave callback */
0, /* Startup Parameter */
{0,0,0,0,0,0,0,0} /* Reserved */
},
```

The startup parameters of the Timer Task and TCP/IP Task shall be set as follows (this depends on the used version as well):

```
STATIC CONST TLR_TIMER_STARTUPPARAMETER_T g_tTlrTimerPrm =
{
TLR_TASK_TIMER, /* ulTaskIdentifier */
1, /* ulParamVersion */
160, /* application timer resources */
2, /* interrupt timer resources */
2 /* retry packet resources */
};

STATIC CONST TCPIP_TCP_TASK_STARTUPPARAMETER_T g_tTcpIpPrm =
{
TLR_TASK_TCPUDP, /* ulTaskIdentifier */
TCPIP_STARTUPPARAMETER_VERSION_6, /* ulParamVersion */
TCPIP_SRT_QUE_ELEM_CNT_AP_DEFAULT, /* ulQueElemCntAp */
TCPIP_SRT_POOL_ELEM_CNT_DEFAULT, /* ulPoolElemCnt */
TCPIP_SRT_FLAG_FAST_START, /* ulStartFlags */
TCPIP_SRT_TCP_CYCLE_EVENT_DEFAULT, /* ulTcpCycleEvent */
TCPIP_SRT_QUE_FREE_ELEM_CNT_DEFAULT, /* ulQueFreeElemCnt */
32, /* ulSocketMaxCnt */
TCPIP_SRT_ARP_CACHE_SIZE_DEFAULT, /* ulArpCacheSize */
PNS_EDD_IDENTIFY_NAME, /* pszEddName */
TCPIP_SRT_EDD_QUE_POOL_ELEM_CNT_DEFAULT, /* ulEddQuePoolElemCnt */
TCPIP_SRT_EDD_OUT_BUF_MAX_CNT_DEFAULT, /* ulEddOutBufMaxCnt */
NULL, /* Pointer to EthIntf Config */
60, /* ARP cache timeout in seconds */
NULL,
.ulNetLoadMaxFramesPerTick =
TCPIP_SRT_NETLOAD_MAXFRAMESPERTICK_DEFAULT,
.ulNetLoadMaxPendingARP = TCPIP_SRT_NETLOAD_MAXPENDING_ARP_DEFAULT,
.ulNetLoadMaxPendingMCastARP =
TCPIP_SRT_NETLOAD_MAXPENDING_MCASTARP_DEFAULT,
.ulNetLoadMaxPendingIP = TCPIP_SRT_NETLOAD_MAXPENDING_IP_DEFAULT,
.ulNetLoadMaxPendingMCastIP =
TCPIP_SRT_NETLOAD_MAXPENDING_MCASTIP_DEFAULT,
};
```

7.1.2 PNS_StackInit()

In order to start the PROFINET Device stack, the user application shall invoke the function PNS_StackInit(). The function will setup the resources needed by the stack and launch all required tasks. The function has to be provided with parameters describing the hardware resources:

```
/* PROFINET Device stack - configuration parameters */
PROFINET_IODEVICE_STARTUPPARAMETER_T tPrm;

/* PROFINET Device stack - configured task resources */
PROFINET_IODEVICE_TASK_RESOURCES_T* ptRsc;

eRslt= PNS_StackInit(&tPNSParam, &ptPNSRsc);
```

The configuration structure PROFINET_IODEVICE_STARTUPPARAMETER_T of the PROFINET Device stack has following fields:

ulParamVersion

This value needs to be adapted to the stack version used, different versions of parameters will not be accepted by the StackInit function.

ulInstance

Used as a handle variable for the interrupts described above and EDD.

ulFlags

Defines the following options:

Option	Value	Description
PROFINET_IODEVICE_STARTUP_FLAG_USE_IRT	0x0001	Obsolete, set to 0
PROFINET_IODEVICE_STARTUP_FLAG_SYNC_APPL_WITH_FIQ	0x0002	Obsolete, set to 0
PROFINET_IODEVICE_STARTUP_FLAG_SYNC_APPL_WITH_IRQ	0x0004	Obsolete, set to 0
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_HW	0x0008	Obsolete, use values 0x0100 or/and 0x0200 instead
PROFINET_IODEVICE_STARTUP_FLAG_DISALLOW_IO _SUPERVOR_AR	0x0010	Shall be set if IO-Supervisor AR shall not be supported by the stack
PROFINET_IODEVICE_STARTUP_FLAG_USE_LINKLOCAL_IP	0x0020	Set this flag to configure link local IP address (generated from MAC address) instead of zero IP address
---	0x0040	Reserved
---	0x0080	Reserved
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0	0x0100	Shall be set, if hardware has fiber optic on port1
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1	0x0200	Shall be set, if hardware has fiber optic on port2
PROFINET_IODEVICE_STARTUP_FLAG_IM5_SUPPORTED	0x0400	Shall be set to support I&M5 (used for Hilscher's internal purposes)
PROFINET_IODEVICE_STARTUP_FLAG_FODMI_TASK_DISABLED	0x0800	Shall be set to disable the fiber optic service task (FODMI)
PROFINET_IODEVICE_STARTUP_FLAG_DISALLOW_SRAR	0x1000	Shall be set if system redundancy AR shall not be supported by the stack
PROFINET_IODEVICE_STARTUP_FLAG_DISALLOW_IRT	0x2000	Shall be set if RT Class3 (IRT) AR shall not be supported by the stack

Table 203: ulFlags options

ulMinDeviceInterval

This is a GSDML-file parameter representing the minimum interval time for sending cyclic IO data.

ulMaxAr

This parameter represents the maximal count of controllers can be connected to the device. See section *Multiple ARs* [▸ page 32]for details.

pszEddName

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

pszPhy0Name, pszPhy1Name

Obsolete parameters, not used any more.

pszIrqRTSchedName

Specify the name of the RT Scheduler Interrupt here. The interrupt with the corresponding name shall be defined in the configuration file, it is the SysTime-compare interrupt `SRT_NETX_VIC_IRQ_STAT_systime_ns`.

pfnFodmiTaskInitFn

Pointer to the startup function of the fiber optic service task. Shall be initialized to "FODMITask_Init" if flags `PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0` or/and `PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1` is/are set.

pvFodmiTaskInitPrm

Pointer to the initialization parameters for fiber optic service task. Shall be initialized with address of the startup parameters for FODMI-Task `FODMI_TASK_INIT_PRM_T` if flags `PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0` or/and `PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT`1 is/are set.

tTaskCmdev, tTaskRtc, tTaskRta, tTaskLldp, tTaskMibDB, tTaskPnsif, tTaskRpc, tTaskSnmp, tTaskFodmi

Configurations of the priorities for corresponding tasks

patPNSModules

Reserved, set to NULL.

usMauTypePort0, usMauTypePort1

MAU type of Fiber Optic device, shall be used if hardware has fiber optic (flags `PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0` or/and `PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1` set). If set to "0" the default MAU type "100BasePXFD" is used.

structure tSNMPdatabase

Used to add custom MIB variables to the SNMP-database. It is a rarely used feature for customer applications inasmuch as the PROFINET stack takes care about all required SNMP-variables. For more information see reference [].

.pvCustomGroups

Pointer to the array of custom MIB groups. It is the same parameter as MIB_DATABASE_STARTUPPARAMETER_V3_T::ptCustomGroups for SNMP-database task. Set to "NULL" if an application does not support custom SNMP variables

.ulCustomGroupsCount

Number of groups in the array of custom MIB groups. It is the same parameter as MIB_DATABASE_STARTUPPARAMETER_V3_T::ulCustomGroupsCount for SNMP-database task. Set to "0" if an application does not support custom SNMP variables

ulActivePROFINETPorts

Specify the number of Ethernet Ports here. Typically, this value is 2 for all netX based products. Definition of ulActivePROFINETPorts = 1 means a single port PROFINET device: logical an Ethernet port 1 is the active PROFINET port; information about port 2 will be not delivered via common services like SNMP and RPC (it is still usable for communication, physical is not deactivated).

Common implementation of a PROFINET IO-device with 2 Ethernet shall set this parameter to 2.

ulActiveLLDPPorts

This parameter defines amount of active LLDP Ethernet ports. Parameterization ulActiveLLDPPorts = 1 forces an activity of LLDP service only on the physical Ethernet port 1.

The combination of parameter
"ulActivePROFINETPorts = 1" and "ulActiveLLDPPorts = 2"
is allowed.

usOEMVendorID

This field is relevant only in case I&M5 is enabled. In this case the value of this field will be used inside DCP Identify Response frames as content for the OEMVendorBlock. Set to 0 if I&M5 is not used.

usOEMDeviceID

This field is relevant only in case I&M5 is enabled. In this case the value of this field will be used inside DCP Identify Response frames as content for the OEMVendorBlock. Set to 0 if I&M5 is not used.

Configuration Example (not valid for a fiber optic device):

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

/* common parameters */
tPnsParam.ulParamVersion =
PROFINET_IODEVICE_STARTUPPARAMETER_VERSION_V6
tPNSParam.ulInstance = 0;
tPNSParam.ulFlags = 0;
tPNSParam.ulMinDeviceInterval = 32; /* 1ms */
tPNSParam.ulMaxAr = 2;
tPNSParam.pszEddName = "ETHERNET";
tPNSParam.pszPhy0Name = NULL;
tPNSParam.pszPhy1Name = NULL;
tPNSParam.pszIrqRTSchedName = "PNS_RT_IRQ";
tPNSParam.pfnFodmiTaskInitFn = NULL;
tPNSParam. pvFodmiTaskInitPrm = NULL;

/* priorities of all tasks, started by PROFINET */
tPNSParam.tTaskCmdev = {NULL, "PNIO_CMDEV", TSK_PRIO_18},
tPNSParam.tTaskRtc = {NULL, "PNIO_RTC", TSK_PRIO_4},
tPNSParam.tTaskRta = {NULL, "PNIO_RTA", TSK_PRIO_14},
tPNSParam.tTaskLldp = {NULL, "LLDP-Task", TSK_PRIO_13},
tPNSParam.tTaskMibDB = {NULL, "Mib-Database", TSK_PRIO_31},
tPNSParam.tTaskPnsif = {NULL, "PNS_IF", TSK_PRIO_21},
tPNSParam.tTaskRpc = {NULL, "RPC", TSK_PRIO_16},
tPNSParam.tTaskSnmp = {NULL, "SNMP-Server", TSK_PRIO_32},
tPNSParam.tTaskFodmi = {NULL, "FODMI", TSK_PRIO_51},

/* fixed modules */
tPNSParam.patPNSModules = NULL;

/* MAU types are applied for fiber optic only */
tPNSParam.usMauTypePort0 = 0;
tPNSParam.usMauTypePort1 = 0;

/* no custom SNMP variables */
tPNSParam.tSNMPdatabase.pvCustomGroups = NULL;
tPNSParam.tSNMPdatabase.ulCustomGroupsCount = 0;

/* 2-Ethernet port has a device */
tPNSParam.ulActivePROFINETPorts = 2;

/* LLDP service is active on both Ethernet ports */
tPNSParam.ulActiveLLDPPorts = 2;

/* now start the stack */
if (TLR_S_OK == PNS_StackInit(&tPNSParam, &ptPNSRsc))
{
hQuePnsIf = ptPNSRsc->hQuePnsif;
}
```

7.1.3 Task Priorities

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

Note:
Any change in the priority order of the tasks may lead to problems which will be difficult to detect. It is recommended that each application task shall have a lower priority than the PNS_IF-Task.

Task Name	Priority	Description
RX_TIMER	*TSK_PRIO_DEF_RX_TIMER*	*rcX Timer interrupt task priority**
PND_HAL_RT	TSK_PRIO_3	PND Switch RT task
PNIO_RTC	TSK_PRIO_4	RTC task
RX_TIMER	*TSK_PRIO_5**	*rcX Timer interrupt task priority**
PND_HAL_NWC	TSK_PRIO_7	PND Switch NWC task
APP_HI	TSK_PRIO_8	Application task, **handles cyclical data** (see *Requirements to the Application* [▸ page 287])
PND_HAL_NRT	TSK_PRIO_11	PND Switch NRT task
TlrTimer	TSK_PRIO_12	TLR Timer Task
LLDP-Task	TSK_PRIO_13	LLDP task
PNIO_RTA	TSK_PRIO_14	RTA task
RPC	TSK_PRIO_16	RPC task
PNIO_CMDEV	TSK_PRIO_18	CMDEV task
PNS_IF	TSK_PRIO_21	PROFINET Stack Interface task
TCP_UDP	TSK_PRIO_30	TCP/IP task
Mib-Database	TSK_PRIO_31	SNMP MIB task
SNMP-Server	TSK_PRIO_32	SNMP-Task
APP_LOW	TSK_PRIO_40	Application task, **handles acyclical services** (see *Requirements to the Application* [▸ page 287])
FODMI	TSK_PRIO_51	Fiber Optic Diagnostic Media Interface

Table 204: Overview about the recommended task priorities

* priority of the RX_TIMER should be set to TSK_PRIO_5 only for Isochronous Applications with fast cycles 250µs), see also section *Isochronous Application* [▸ page 42].

7.1.4 Fiber optic device

Using Linkable Object Module, it is possible to build a device that uses a fiber optic medium.

7.1.4.1 Fiber optic configuration

As explained above the PROFINET stack starts, if a user application invokes the function PNS_StackInit() with configuration parameters. They include a pointer to a configuration structure FODMI_TASK_INIT_PRM_T for the fiber optic service task FODMI. This task operates the link LEDs and link activity LEDs for fiber optic ports and reads diagnostic data from the fiber optic transceivers via I2C bus.

Configuration structure FODMI_TASK_INIT_PRM_T has following fields:

ulCyclicDiagnosis_ms

This is a period for sending diagnosis information, evaluated in milliseconds (use 1000ms).

pszLEDPort1Link

The name of the LED for link (up/down) signaling of the port 1.

pszLEDPort1Act

The name of the LED for link activity signaling of the port 1.

pszLEDPort2Link

The name of the LED for link (up/down) signaling of the port 2.

pszLEDPort2Act

The name of the LED for link activity signaling of the port 2.

ulCyclicLEDActState_ms

This is a period for checking the activity of the fiber optic ports, evaluated in milliseconds (use 250ms).

bSDA1PinIdx (used only for netX50)

MMIO pin number on the netX for I2C serial data line (SDA) to connect the fiber optic transceiver for the port 1.

bSDA2PinIdx (used only for netX50)

MMIO pin number on the netX for I2C serial data line (SDA) to connect the fiber optic transceiver for the port 2.

- **bSCLPinIdx** (used only for netX50)

MMIO pin number on the netX for I2C serial clock line (SCL) commonly used for fiber optic transceiver s for both ports.

ulPintype (used only for netX500 and netX100)

Pin to switch between fiber optic transceivers on port1 and port 2.

Type of the *switch-pin*. Allowed values:

```
TAG_FIBER_OPTIC_IF_DMI_PINTYPE_NONE = 0 /* not used */
TAG_FIBER_OPTIC_IF_DMI_PINTYPE_GPIO = 1 /* GPIO */
TAG_FIBER_OPTIC_IF_DMI_PINTYPE_PIO = 2 /* PIO or HIFPIO */
```

(HIFPIO used if uPin greater than 32)

uPin (used only for netX500 and netX100)

Number of the *switch-pin*.

Allowed values:

0 – 15 if ulPintype=TAG_FIBER_OPTIC_IF_DMI_PINTYPE_GPIO

0 – 84 if ulPintype=TAG_FIBER_OPTIC_IF_DMI_PINTYPE_PIO

fPinInvert (used only for netX500 and netX100)

Set to 1 to invert the signal of the *switch-pin*. Possible values 0 or 1.

pszPnsIfQueName *(not used)*

Set to NULL.

pszPioGpioHifPioPinIdentifyName (used only for netX500 and netX100)

Name of GPIO, PIO or HIFPIO used for *switch-pin* to identify (depends on ulPintype)

Notes

- Because of differences in the netX chips the fields bSDA1PinIdx, bSDA2PinIdx, bSCLPinIdx are evaluated only for netX50 but the fields ulPintype, uPin, fPinInvert – for netX500 and netX100.
- **The netX50** has only one I2C channel and a flexible MMIO matrix that allows to connect the fiber optic transceivers (I2C bus) to any MMIO pin. The MMIO matrix is used to switch the I2C bus (exactly pin I2C_SDA) between two fiber optic transceivers (the clock pin I2C_SCL is common for both transceivers). The FODMI task does configure of MMIO martix and switch between SDA pins on its own according to the fields bSDA1PinIdx, bSDA2PinIdx and bSCLPinIdx.
- Be sure to select the fiber optic type for the PHY conection in the “IO Config Register”.
- **The netX500 and netX100** have only one I2C channel and fixed pins for I2C bus. Therefore the hardware design (schematic) has to implement a switch of the I2C bus between two fiber optic transceivers. The *switch-pin* will control this switch, it can be configured on any CPIO, PIO or HIFPIO using fields ulPintype and uPin.
- Be sure to select the fiber optic type for the PHY conection in the “IO Config Register”.

- **The netX51** has two I2C channels and a flexible MMIO matrix that allows to connect the fiber optic transceivers to any MMIO pin. For each fiber optic transceiver is used own I2C channel.
- The FODMI task does not configure MMIO matrix, please configure MMIO matrix for all used I2C channels (ports) before starting the stack.
- Select position A or B for fiber optic PHY0 or PHY1 in the “IO Config Register”, see also the pinning table of the netX51.

Please refer to the “Design-In Guide” of used netX chip for layout and wiring and to the configuration examples shown below

For a fiber optic device all parameters for the FODMI task should be set, flags – which port is for fiber optic and MAU types for these ports.

The following examples show the fiber optic specific configuration for different netX chips and different number of fiber optic ports.

Example 1

chip type – **netX50**, fiber optic ports – port 1 and port 2, MAU types – default for both ports, I2C bus connected to the MMIO matrix, MMIO pin number 21 used for I2C-clock wire, MMIO pin number 2 for I2C-SDA wire of the port 1 and MMIO pin number 3 for I2C-SDA wire of the port 2

```
int main(void)
{
...
...
/* before the operating system starts */
volatile unsigned long* pulAccessKey =
(volatile unsigned long*) NETX_IO_CFG_ACCESS_KEY;
volatile unsigned long* pulIOConfig = (volatile unsigned
long*)NETX_IO_CFG;

unsigned long val = *pulIOConfig;

/* allow internal connection of the i2c module to MMIO matrix */
Val |= MSK_NETX_IO_CFG_sel_i2c_mmio;

/* Select the fiber optic type for the PHY0/1 conection */
val |= ( MSK_NETX_IO_CFG_sel_fo0 | MSK_NETX_IO_CFG_sel_fo1 );

/* unlock the netX access-key mechanism and set the values */
*pulAccessKey = *pulAccessKey;
*pulIOConfig = val;

...
...
}

FODMI_TASK_INIT_PRM_T g_tFodmiInitPrm =
{
.ulCyclicDiagnosis_ms = 1000,
.pszLEDPort1Link = "POF_PORT0LINK",
.pszLEDPort1Act = "POF_PORT0ACT",
.pszLEDPort2Link = "POF_PORT1LINK",
.pszLEDPort2Act = "POF_PORT1ACT",
.ulCyclicLEDActState_ms = 250,
.bSDA1PinIdx = 2, /* MMIO2 is SDA for port 1 */
.bSDA2PinIdx = 3, /* MMIO3 is SDA for port 2 */
.bSCLPinIdx = 21, /* MMIO21 is SCL for clock */
.ulPintype = TAG_FIBER_OPTIC_IF_DMI_PINTYPE_NONE,
.uPin = 0, /* not used */
.fPinInvert = 0, /* not used */
.pszPnsIfQueName = NULL,
.pszPioGpioHifPioPinIdentifyName = NULL, /* not used */
};
...
...
PROFINET_IODEVICE_STARTUPPARAMETER_T tPNSParam /* PNS Stack
Parameters */

tPNSParam.ulFlags =
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0|
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1;
...
tPNSParam.pfnFodmiTaskInitFn = (FN_FODMI_TASK_INIT) FODMITask_Init;
tPNSParam. pvFodmiTaskInitPrm= &g_tFodmiInitPrm;
...
tPNSParam.usMauTypePort0 = 0; /* default MAU type: polymer optical
fiber */
tPNSParam.usMauTypePort1 = 0; /* default MAU type: polymer optical
fiber */
...
/* now start the stack */
if (TLR_S_OK == PNS_StackInit(&tPNSParam, &ptPNSRsc))
{
hQuePnsIf = ptPNSRsc->hQuePnsif;
}
```

All used LEDs should be configured in config.c (RX_LED_SET_T). The FODMI task uses their names to identify them in runtime. It applies to the pszPioGpioHifPioPinIdentifyName also.

Example 2

chip type – **netX100**, fiber optic ports – port 1 and port 2, MAU types – “100BaseFXFD” for both ports. GPIO number 7 used to switch between fiber optic transceivers

```
int main(void)
{
...
...
/* before the operating system starts */
volatile unsigned long* pulAccessKey =
(volatile unsigned long*) NETX_IO_CFG_ACCESS_KEY;
volatile unsigned long* pulIOConfig = (volatile unsigned
long*)NETX_IO_CFG;

unsigned long val = *pulIOConfig;

/* Select the fiber optic type for the PHY0/1 conection */
val |= ( MSK_NETX_IO_CFG_sel_fo0 | MSK_NETX_IO_CFG_sel_fo1 );

/* unlock the netX access-key mechanism and set the values */
*pulAccessKey = *pulAccessKey;
*pulIOConfig = val;

...
...
}

FODMI_TASK_INIT_PRM_T g_tFodmiInitPrm =
{
.ulCyclicDiagnosis_ms = 1000,
.pszLEDPort1Link = "POF_PORT0LINK",
.pszLEDPort1Act = "POF_PORT0ACT",
.pszLEDPort2Link = "POF_PORT1LINK",
.pszLEDPort2Act = "POF_PORT1ACT",
.ulCyclicLEDActState_ms = 250,
.bSDA1PinIdx = 0, /* not used */
.bSDA2PinIdx = 0, /* not used */
.bSCLPinIdx = 0, /* not used */
.ulPintype = TAG_FIBER_OPTIC_IF_DMI_PINTYPE_GPIO,
.uPin = 7, /* GPIO 7 */
.fPinInvert = 0, /* not invert */
.pszPnsIfQueName = NULL,
.pszPioGpioHifPioPinIdentifyName = "GPIOFODMI",
};
...
...
PROFINET_IODEVICE_STARTUPPARAMETER_T tPNSParam; /* PNS Stack
Parameters */

tPNSParam.ulFlags =
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0|
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1;
...
tPNSParam.pfnFodmiTaskInitFn = (FN_FODMI_TASK_INIT) FODMITask_Init;
tPNSParam. pvFodmiTaskInitPrm= &g_tFodmiInitPrm;
...
PNSParam.usMauTypePort0 = 0x12; /* MAU type: glas optical fiber */
tPNSParam.usMauTypePort1 = 0x12; /* MAU type: glas optical fiber */
.........
```

Example 3

chip type – **netX50**, fiber optic port – port 1, MAU type – “100BaseFXFD” for port 1, I2C bus connected to the MMIO matrix, MMIO pin number 21 used for I2C-clock wire and MMIO pin number 2 for I2C-SDA wire of the port 1. The second port is connected with twisted pair

```
int main(void)
{
...
...
/* before the operating system starts */
volatile unsigned long* pulAccessKey =
(volatile unsigned long*) NETX_IO_CFG_ACCESS_KEY;
volatile unsigned long* pulIOConfig = (volatile unsigned
long*)NETX_IO_CFG;

unsigned long val = *pulIOConfig;

/* allow internal connection of the i2c module to MMIO matrix */
Val |= MSK_NETX_IO_CFG_sel_i2c_mmio;

/* Select the fiber optic type for the PHY0 conection */
val |= ( MSK_NETX_IO_CFG_sel_fo0);

/* unlock the netX access-key mechanism and set the values */
*pulAccessKey = *pulAccessKey;
*pulIOConfig = val;

...
...
}

FODMI_TASK_INIT_PRM_T g_tFodmiInitPrm =
{
.ulCyclicDiagnosis_ms = 1000,
.pszLEDPort1Link = "POF_PORT0LINK",
.pszLEDPort1Act = "POF_PORT0ACT",
.pszLEDPort2Link = "", /* not used */
.pszLEDPort2Act = "", /* not used */
.ulCyclicLEDActState_ms = 250,
.bSDA1PinIdx = 2, /* MMIO2 is SDA for port 1 */
.bSDA2PinIdx = 0, /* not used */
.bSCLPinIdx = 21, /* MMIO21 is SCL for clock */
.ulPintype = TAG_FIBER_OPTIC_IF_DMI_PINTYPE_NONE,
.uPin = 0, /* not used */
.fPinInvert = 0, /* not used */
.pszPnsIfQueName = NULL,
.pszPioGpioHifPioPinIdentifyName = NULL, /* not used */};
...
...
PROFINET_IODEVICE_STARTUPPARAMETER_T tPNSParam; /* PNS Stack
Parameters */

tPNSParam.ulFlags =
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0;
...
tPNSParam.pfnFodmiTaskInitFn = (FN_FODMI_TASK_INIT) FODMITask_Init;
tPNSParam. pvFodmiTaskInitPrm= &g_tFodmiInitPrm;
...
tPNSParam.usMauTypePort0 = 0x12; /* MAU type: glas optical fiber */
tPNSParam.usMauTypePort1 = 0; /* not used */
...
```

Example 4

chip type – **netX51**, fiber optic ports – port 1 and port 2, MAU types – “100BaseFXFD” for both ports. I2C buses are connected to the MMIO matrix. FODMI task does use the I2C0 bus to service the port 1 using MMIO pin number 2 for I2C0_SCL wire and MMIO pin number 3 for I2C0_SDA wire, and does use the I2C1 bus to service the port 2 using MMIO pin number 9 for I2C1_SCL wire and MMIO pin number 8 for I2C1_SDA

```
int main(void)
{
unsigned long val;
volatile unsigned long* pulAccessKey =
(volatile unsigned long*) Adr_NX51_asic_ctrl_access_key;
volatile unsigned long* pulIOConfig =
(volatile unsigned long*) Adr_NX51_io_config;
volatile unsigned long* pulMMIOConfig =
(volatile unsigned long*) NX51_NETX_MMIO_CTRL_AREA;
...
...
/* before the operating system starts */
...
...
/* configure MMIO matrix for both I2C channels used by FODMI task (2
fiber
optic ports) */

/* unlock the netX access-key mechanism and set the values */
*pulAccessKey = *pulAccessKey;
/* set MMIO pin number 2 for I2C0_SCL wire */
pulMMIOConfig[2] = MMIO_CONFIG_I2C0_SCL_MMIO;

*pulAccessKey = *pulAccessKey;
pulMMIOConfig[3] = MMIO_CONFIG_I2C0_SDA_MMIO;

*pulAccessKey = *pulAccessKey;
pulMMIOConfig[9] = MMIO_CONFIG_I2C1_SCL;

*pulAccessKey = *pulAccessKey;
pulMMIOConfig[8] = MMIO_CONFIG_I2C1_SDA;


/* prevent activation of fiber optics on both positions A and B */
val = *pulIOConfig;
val &= ~(MSK_NX51_io_config_sel_fo0_a | MSK_NX51_io_config_sel_fo0_b
|
MSK_NX51_io_config_sel_fo1_a | MSK_NX51_io_config_sel_fo1_b);

/* Select position A for conection of the fiber optic PHY0 and PHY1
*/
val |= (MSK_NX51_io_config_sel_fo0_a |
MSK_NX51_io_config_sel_fo1_a);


/* allow internal connection of the i2c module to MMIO matrix */
Val |= MSK_NETX_IO_CFG_sel_i2c_mmio;

/* unlock the netX access-key mechanism and set the values */
*pulAccessKey = *pulAccessKey;
*pulIOConfig = val;
...
...
}

FODMI_TASK_INIT_PRM_T g_tFodmiInitPrm =
{
.ulCyclicDiagnosis_ms = 1000,
.pszLEDPort1Link = "POF_PORT0LINK",
.pszLEDPort1Act = "POF_PORT0ACT",
.pszLEDPort2Link = "POF_PORT1LINK",
.pszLEDPort2Act = "POF_PORT1ACT",
.ulCyclicLEDActState_ms = 250,
.bSDA1PinIdx = 0, /* not used */
.bSDA2PinIdx = 0, /* not used */
.bSCLPinIdx = 0, /* not used */
.ulPintype = 0, /* not used */
.uPin = 0, /* not used */
.fPinInvert = 0, /* not used */
.pszPnsIfQueName = NULL,
.pszPioGpioHifPioPinIdentifyName = NULL, /* not used */
};
...
...
PROFINET_IODEVICE_STARTUPPARAMETER_T tPNSParam /* PNS Stack
Parameters */

tPNSParam.ulFlags =
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT0|
PROFINET_IODEVICE_STARTUP_FLAG_FIBER_OPTIC_PORT1;
...
tPNSParam.pfnFodmiTaskInitFn = (FN_FODMI_TASK_INIT) FODMITask_Init;
tPNSParam. pvFodmiTaskInitPrm= &g_tFodmiInitPrm;
...
tPNSParam.usMauTypePort0 = 0x12; /* MAU type: glas optical fiber */
tPNSParam.usMauTypePort1 = 0x12; /* MAU type: glas optical fiber */
...
/* now start the stack */
if (TLR_S_OK == PNS_StackInit(&tPNSParam, &ptPNSRsc))
{
hQuePnsIf = ptPNSRsc->hQuePnsif;
}
```

7.1.4.2 Medium Attachment Unit for Fiber Optic

The fiber optic device can differentiate between following types of the Medium Attachment Units (MAU):

MAU Type	Value	Description
100BaseFXFD	0x12	Used transceiver for glas optical fiber (GOF)
100BasePXFD	0x36	Used transceiver for polymer optical fiber (POF)

Table 205: MAU types for fiber optic ports

© Hilscher 2011 – 2019

7.1.5 PROFINET Netload Requirements

The PROFINET Certification requires at least to pass the **Netload Class I** tests since V2.31 (mandatory). If the device does not pass these tests the Conformance Test will not be passed and the device does not get a certificate. The application and firmware must be designed as follows in order to be able to pass the tests:

- The application shall be split into at least two tasks. One task (e.g. "APP_HI") is entirely dedicated to the handling of the process data while the acyclic services are handled by the other task(s) (e.g. "APP_LOW"). The process data task must be configured with a priority between the "PND_HAL_NWC" task and "RTA" task. Therefore this task must be designed carefully in order to avoid unintentional consumption of CPU power. If an application can process the cyclical-data really fast (e.g. one or two checks, apply to outputs and fetch inputs) the “APP_HI” can be omitted and whole processing performed in the "pfnEventHandler" on "PNS_IF_IO_EVENT_CONSUMER (PROVIDER)_UPDATE_DONE" (see section *Set IO Image service* [▸ page 87] for details). The acyclic application task shall have a lower priority than the protocol stack (see *Task Priorities* [▸ page 271] for recommended task priorities).

For netX51 a special memory setup is required to pass the Netload Test Class III:

- In order to improve execution speed the netX51 has internal SRAM memory used by the protocol stack and application. The linker script shall be designed that important parts of the protocol stack and the process data handling of the application is relocated into the SRAM areas. The startup code must be adapted as well to perform the necessary relocations from SDRAM to SRAM at startup. An example startup script and linker script can be found in the corresponding LOM example.
- To improve execution speed of separate tasks it is recommended to allocate the stacks of "TCP_UDP" task, PROFINET tasks (by priority: "PND_HAL_RT", "PNIO_RTC", "PND_HAL_NWC", "PND_HAL_NRT", "PNIO_RTA", "PNIO_CMDEV") and Application's tasks also in the internal SRAM memory. E.g. the following part of linker script shows how to allocate the stacks of "TCP_UDP" and "Packet API Demo" tasks in the internal memory (see section *Static Task List* [▸ page 264] for definition of the task stacks):

```
MEMORY
{
.........
/* itcm & dtcm */
ITCM(rx): ORIGIN = 0x00000080, LENGTH = 128K + 128K + 64K - 0x80 /*
INTRAM0/1/2 */
DTCM(rw): ORIGIN = 0x04050000, LENGTH = 64K + 64K + 32K + 32K /*
INTRAM3/4/5/6 */
.........
}

.itcm :
{
.........
}>ITCM AT>SDRAM

.dtcm :
{
```

```
.........
/* stack of the "TLR TIMER" task*/
*Config_netX51.o(.bss. auTskStack_Tlr_Timer)

/* stack of the "Packet API Demo" task*/
*Config_netX51.o(.bss.auTskStack_Pck_API_Demo)

/* stack of the "TCP_UDP" task*/
*Config_netX51.o(.bss.auTskStack_Tcp_Task)

.../* or stack of ALL static tasks declared in the
"Config_netX51.c" /
*Config_netX51.o(.bss*)

.........

/* For example the stacks of some PROFINET tasks: */

/*stacks of the "HAL_RTC" task */
*Hal_EddPROFINETDevice_common.o(.bss.aulHal_RTC_TaskStack*)
/*stacks of the "HAL_NWC" task */
*Hal_EddPROFINETDevice_common.o(.bss.aulHal_NWC_TaskStack*)
/*stacks of the "HAL_NRT" task */
*Hal_EddPROFINETDevice_common.o(.bss.aulHal_NRT_TaskStack*)

/*stack of the "PNIO_RTC" task */
*PNIORTC_Init.o(.bss.g_RTC_Task_Stack*)

/*stack of the "PNIO_RTA" task */
*PNIORTA_Init.o(.bss.s_auiRTATask_TaskStack*)

/*stack of the "PNIO_CMDEV" task */
*PNIOCMDEV_init.o(.bss.s_auiCMDEVTask_TaskStack*)

.........
}>DTCM AT>SDRAM
```

8 PROFINET Certification

Before any PROFINET device appears on the market it has to pass PROFINET certification tests to prove the conformance according to one of the defined PROFINET conformance classes (A, B or C).

To achieve conformance class A (without SNMP) or B (with SNMP) the PROFINET Real Time (RT) protocol behavior and the behavior on network load have to be tested on the device.

If a PROFINET device shall conform to conformance class C, it has to fulfill some additional requirements including PROFINET Isochronous Real Time (IRT). This chapter explains common requirements to a PROFINET device for all types of the certification tests.

For detailed information about PROFINET Certification please also refer to reference [3]

8.1 RT tests (Conformance class A, B and C)

8.1.1 Description

The common PROFINET RT certification tests have to be passed by all IO Device implementations. These test cases cover the following functionality

- basic state machine behavior
- different parts of protocol (coding)
- reaction of device on erroneous configurations and situations
- acyclic services
- cyclic data and data-status (IOXS)

8.1.2 General requirements for RT tests

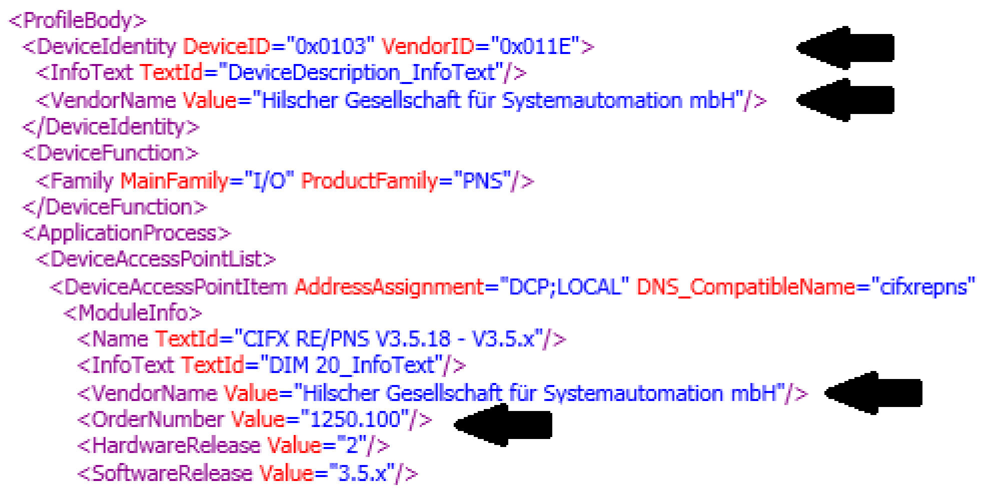
In order to execute all test cases the examiner in the certification laboratory should have a possibility to trigger alarms (diagnosis or process alarms) on a device. For example additional push button causes an alarm, “magic” sequence of push buttons, deliberate shorting of outputs and so on. This is required if the device generates alarm / diagnosis at runtime. If no alarm and no diagnosis are generated at runtime, these tests can be skipped.

It is recommended (but not required) to have a possibility to change input values on a device. For example set some inputs of an IO-Device to “HI-level”, change measurement values if it is a sensor and so on.

A small informal paper describing how to trigger alarms and change IO Data needs to be given to the certification laboratory together with the device.

8.1.3 Common checks before certification (GSDML)

The device has to be consistent with its own GSDML file. The “DeviceID” and “VendorID” in the GSDML shall be the same that was used in *Set Configuration service* [▸ page 53] for fields “PNS_IF_DEVICE_PARAMETER_T::ulVendorId” and “PNS_IF_DEVICE_PARAMETER_T:: ulDeviceId”, or used in data base (if the stack configured with data base), or used in tag list (if according tag was enabled in firmware). The vendor name shall also be correct:



```
<ProfileBody>
  <DeviceIdentity DeviceID="0x0103" VendorID="0x011E">
    <InfoText TextId="DeviceDescription_InfoText"/>
    <VendorName Value="Hilscher Gesellschaft für Systemautomation mbH"/>
  </DeviceIdentity>
  <DeviceFunction>
    <Family MainFamily="I/O" ProductFamily="PNS"/>
  </DeviceFunction>
  <ApplicationProcess>
    <DeviceAccessPointList>
      <DeviceAccessPointItem AddressAssignment="DCP;LOCAL" DNS_CompatibleName="cifxrepns"
        <ModuleInfo>
          <Name TextId="CIFX RE/PNS V3.5.18 - V3.5.x"/>
          <InfoText TextId="DIM 20_InfoText"/>
          <VendorName Value="Hilscher Gesellschaft für Systemautomation mbH"/>
          <OrderNumber Value="1250.100"/>
          <HardwareRelease Value="2"/>
          <SoftwareRelease Value="3.5.x"/>
```

Figure 26: Vendor and device identification in the GSDML file

In addition it is required that the OrderNumber, HardwareRelease and SoftwareRelease from the GSDML file match the configuration of the PROFINET IO-Device. The values are contained in PNS_IF_DEVICE_PARAMETER_T.

If a new product is created based on Hilscher GmbH reference GSDML files, it is highly recommended to remove all non-matching DAPs from the new GSDML file. Otherwise it may lead to confusion why a new product already has several DAPs inside the GSDML file.

8.1.4 Basic application behavior

If the application takes care of storing device NameOfStation and IP parameters it has to store these parameters according to the following rules:

IP parameters (see section *Save IP Address service* [▸ page 140]):

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

Device NameOfStation (see section *Save Station Name service* [▸ page 137]):

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

On *Reset Factory Settings indication* [▸ page 149] the application shall delete the permanently stored Station Name, set the permanently stored IP parameters to 0.0.0.0 and use them after the next PowerUp cycle (see section *Reset Factory Settings service* [▸ page 146]).

8.2 IRT tests (Conformance class C only)

8.2.1 Description

The common PROFINET IRT certification tests have to be passed by all IO Device implementations. These test cases cover the following functionality

- basic IRT-related state machine behavior
- synchronization related tests
- different parts of protocol (coding)
- reaction of the device on erroneous configurations and situations
- cyclic data and data status (IOXS)

8.2.2 General requirements for IRT tests

There are no special additional requirements besides those from the RT tests.

8.2.3 Hardware requirements for IRT tests

To evaluate the synchronization capability of the device it is required to offer an external synchronization pin to the examiner in the certification laboratory. The synchronization jitter of this pin needs to be verified. This pin shall be present for the device used during certification test. If this pin is not available at the devices used in the field, this is accepted.

8.2.4 Software requirements for IRT tests

Regarding application handling nothing special needs to be taken in account. So for certification no additional software requirements exist compared to the ones defined for RT tests.

8.2.5 GSDML requirements for IRT tests

The GSDML file needs to contain the required IRT related keywords. Please refer to Hilscher reference GSDML files for details. The following listing will just briefly name the required keywords, there may be more keywords required by GSDML checker tool.

- InterfaceSubmoduleItem
  - SupportedRT_Classes="RT_CLASS_1;RT_CLASS_3"
- RT_Class3Properties
  - ForwardingMode="Relative"
  - MaxBridgeDelay="5500"
  - MaxNumberIR_FrameData="256"
  - StartupMode="Advanced;Legacy"
- SynchronisationMode
  - MaxLocalJitter="50"
  - SupportedRole="SyncSlave"
  - SupportedSyncProtocols="PTCP"
  - T_PLL_MAX="1000"
- RT_Class3TimingProperties
  - ReductionRatio="1 2 4 8 16"
  - SendClock="8 16 32 64 128"
- PortSubmoduleItem
  - MaxPortRxDelay="340"
  - MaxPortTxDelay="92"

8.3 Network load tests

8.3.1 Description

Starting with certification for PROFINET specification V2.3 (expressed in GSDML file of the IO-Device by using PNIO_Version=”V2.31”) it is mandatory to execute special network load tests during certification.

These tests verify the correct behavior of the PROFINET IO device during different network loads.

The following classes of network load are defined:

- network load class I
- network load class II
- network load class III

The device has to pass at least the tests for “network load class I” to fulfill the certification requirements.

8.3.2 Requirements to the Application

The main point in the application is the optimization. The following recommendations will help to achive at least network load class I:

- Separate cyclic and acyclic parts of application;
- Arrange the tasks priorities according to the recommendations in table *Overview about the recommended task priorities* [▸ page 271];
- Allocate all important parts of the protocol stack and the process data handling of the application in to internal SRAM memory (netX51);
- Allocate the stacks of tasks also in internal SRAM memory (netX51).

For more information to the optimization see chapter *PROFINET Netload Requirements* [▸ page 280] and refer to the LOM example.

8.4 How to handle I&M Data

In order to fulfill PROFINET conformance needs, any PROFINET IO device should be able to handle the I&M0, I&M1, I&M2, I&M3 and I&M0 Filter data.

8.4.1 Overview

Identification & Maintenance (I&M) is an integral part of each PROFINET Device implementation. It provides standarized information about a device and its parts. The I&M Information is accessible through PROFINET Record Objects and is always bound to a submodule belonging to the item to be described. An item means here the PROFINET Device itself or a part of this device e.g. a plugable module for modular devices. Submodules can provide own I&M objects or share the I&M objects of other submodules. The I&M objects can be grouped into three kinds of information as described as follows.

I&M0

I&M0 is a read only information which describes the associated item. The following fields are defined:

Field	Description	Usage Hints
Vendor ID	The PNO vendor ID of the associated item. E.g. the vendor of the device or the vendor of a pluggable module/submodule.	This is the vendor ID of the manufacturer of the item and not the vendor ID of the PROFINET protocol vendor. Do not use the Hilscher vendor ID.
Order ID	The order ID of the associated item.	Order ID as defined by the manufacturer of the item. Must be equal to any order ID markings on the item itself.
Serial Number	The serial number of the associated item.	Must be an unique serial number associated with the item. Must be equal to any serial number markings on the item itself.
Hardware Revision	The revision of the hardware of the item.	Must be equal to any hardware revision markings on the item itself.
Software Revision	The software revision of the item.	This is the software version of the whole item including the PROFINET Protocol implementation and the Application. This version is managed by the manufacturer of the item. It must be changed whenever a part of the software within the item (including the PROFINET Protocol implementation if it is part of the item) changes. This is not the version number of the PROFINET Protocol implementation. Do not use the Hilscher Version of the PROFINET Protocol implementation.
Revision Counter	Counts the changes of I&M1 to I&M4 objects	-
Profile ID	The Profile of the item if applicable.-	-
Profile Specific Type		
Supported I&M objects	Bitmask defining which I&M objects are supported by this item.	-

Table 206: Fields of I&M0

I&M1 to I&M4

The I&M1 to I&M4 objects provide a non-volatile storage for PROFINET engineering related information. This information is typically generated by the engineering software and stored within the objects at engineering time. The information must be stored by the device in non-volatile memory. The objects must be stored physically within the associated item. This means in particular, if a plugable module is removed from one backplane and plugged into another backplane, it must deliver the same I&M1 to I&M4 information as stored before.

I&M5

Finally, the I&M5 record provides information about the PROFINET protocol implementation itself. It is quite similar to I&M0 but describes the PROFINET protocol implementation instead. Thus it is typically handled by the PROFINET Protocol implementation itself.

I&M0 filter

Addtionally to these submodule specific I&M objects, each PROFINET device must support the global I&M Filter Data object, the so called "I&M0FilterData". This is a read-only object which is used to determine which submodules are associated with dedicated I&M objects.

For certification it is required that at exactly one is marked as “device representative” in I&M0 Filter Data object.

8.4.2 Structure and access paths of I&M objects

The following figure shows the structural organization of I&M Records within a device and the access paths:

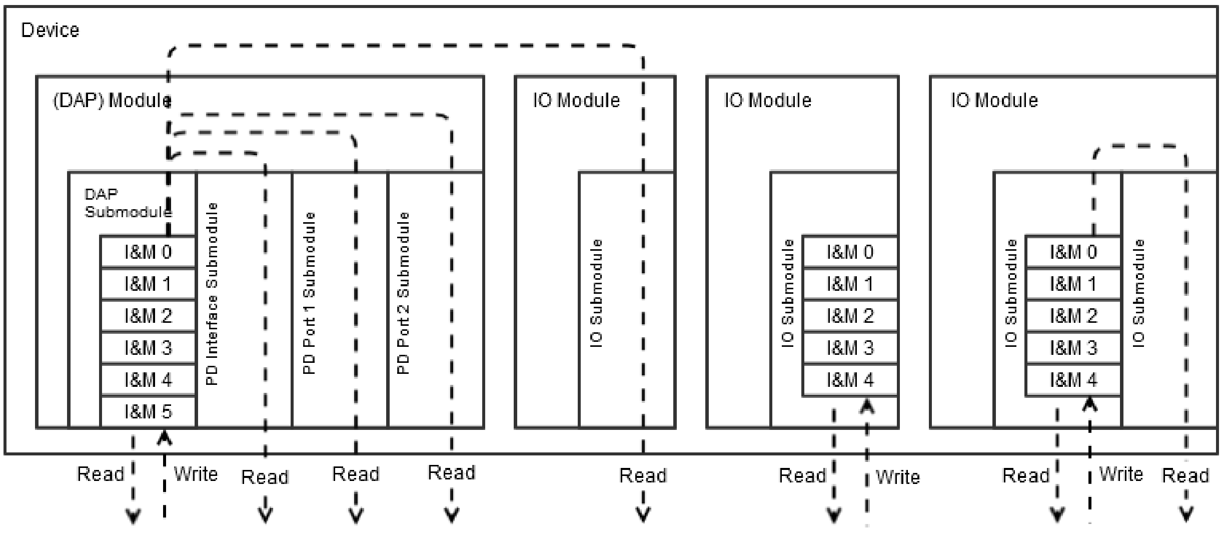


Figure 27: Structural organization of I&M records within a Device and the access paths

Structural organization of I&M Records within a Device and the Access Paths

According to I&M a submodule can be characterized as follows:

- **Module Representative:** The submodule is a representative for the module. This means that the submodule provides the I&M information of the superordinate module.
- **Device Representative:** The submodule is a representative for the device. In this case the submodule provides the I&M information of the superordinate device. Exactly one submodule of the device must have this property. Typical the DAP submodule is chosen for this purpose.
- **Default:** The submodule only represents itself or does not have any I&M information at all. In the latter case only I&M Read Access is allowed and the Read will deliver the I&M information of the Module Representative or the Device Representative.

When reading the I&M objects of a submodule the following order is used to deliver the requested data:

1. If the submodule provides an own I&M0 object, the I&M read access will deliver the submodule's I&M objects
2. If the submodule has no own I&M0 object and the superordinate module has a module representative, the module representative's I&M objects will be delivered.
3. If the submodule has no own I&M0 object and the superordinate module has no representativ the device representatives's I&M objects will be delivered.

In contrast to this, writing I&M1 to I&M4 objects is only possible on the submodules associated with the I&M objects itself.

Finally, the I&M0 Filter Data object contains the information which submodules are associated with their own I&M data, which submodules represent their superordinate module and which submodule represents the whole device. This object is global and a read-only object and it is not associated with any submodule.

8.4.3 Usage of I&M with Hilscher PROFINET protocol

The PROFINET Device protocol implementation provided by Hilscher supports two modes of operation with I&M. For simple applications the PROFINET protocol implementation provides I&M0 to I&M5 objects within the DAP submodule. In this case, the I&M is fully handled by the PROFINET Protocol implementation without any interaction with the application. If a more complex I&M structure is required, the PROFINET Protocol implementation can forward I&M accesses to the Application. In this case all I&M objects must be handled by the application. The desired mode of operation is configured using the Set Configuration Service.

If the PROFINET protocol implementation is configured to handle the I&M internally, the following parameters will be used within the I&M0 object:

Field	Source of value
Vendor ID	Vendor ID in Set Configuration Service or Configuration Database. Can be overwritten by tag list if configuration database is used.
Order ID	Order ID in Set Configuration Service or configuration database.
Serial Number	Defaults to the serial number from Security Memory / Flash device label. This is the serial number of the Hilscher communication module if applicable. It shall be changed to the serial number of the manufacturers device using the Set OEM Parameters service.
Hardware Revision	Hardware revision in Set Configuration Service. If a Configuration Database is used the hardware revision from Security Memory / Flash device label is used. This is the hardware revision of the Hilscher communication module if applicable.
Software Revision	Software revision in Set Configuration Service. If a Configuration Database is used the PROFINET Protocol implementations software revision will be used.
Revision Counter	Internally stored an incremented on each change of I&M1 to I&M4.
Profile ID	Default is '0x00' (Manufacturer specific). Can be changed to a desired value using Set OEM Parameters service.
Profile Specific Type	Default is '0x05' (Generic Device). Can be changed to a desired value using Set OEM Parameters service.
Supported I&M objects	Defaults to I&M0 to I&M5. Can be changed to a desired value using Set OEM Parameters service.

Table 207: Parameters of the I&M0 object

If the application requested to handle the I&M objects, all I&M accesses will be forwarded to the application using the I&M Read and I&M Write service. For details on the data structures of the I&M services refer to the protocol API. The application must store the data of I&M write services permanently, so that the I&M1 to I&M4 objects and the I&M0 revision counter can be recovered after a power cycle. The I&M0 Filter data structure must be filled by the application with a list of the submodules which are associated with dedicated I&M data. Exactly one of these submodules must be characterized as device representative. Characterization as a module representative is optional and only sensible if the module has more than one submodule. The protocol stack will then internally build up to three lists out of the information provided by the application.

9 Error codes and status codes

9.1 Common status codes

Hexadecimal value	Definition and description
0x00000000	SUCCESS_HIL_OK Operation succeeded.
0xC0000001	ERR_HIL_FAIL Common error, detailed error information optionally present in the data area of packet.
0xC0000002	ERR_HIL_UNEXPECTED Unexpected failure.
0xC0000003	ERR_HIL_OUTOFMEMORY Ran out of memory.
0xC0000004	ERR_HIL_UNKNOWN_COMMAND Unknown Command in Packet received.
0xC0000005	ERR_HIL_UNKNOWN_DESTINATION Unknown Destination in Packet received.
0xC0000006	ERR_HIL_UNKNOWN_DESTINATION_ID Unknown Destination Id in Packet received.
0xC0000007	ERR_HIL_INVALID_PACKET_LEN Packet length is invalid.
0xC0000008	ERR_HIL_INVALID_EXTENSION Invalid Extension in Packet received.
0xC0000009	ERR_HIL_INVALID_PARAMETER Invalid Parameter in Packet found.
0xC000000A	ERR_HIL_INVALID_ALIGNMENT Invalid alignment.
0xC000000C	ERR_HIL_WATCHDOG_TIMEOUT Watchdog error occurred.
0xC000000D	ERR_HIL_INVALID_LIST_TYPE List type is invalid.
0xC000000E	ERR_HIL_UNKNOWN_HANDLE Handle is unknown.
0xC000000F	ERR_HIL_PACKET_OUT_OF_SEQ A packet index has been not in the expected sequence.
0xC0000010	ERR_HIL_PACKET_OUT_OF_MEMORY The amount of fragmented data contained the packet sequence has been too large.
0xC0000011	ERR_HIL_QUE_PACKETDONE The packet done function has failed.
0xC0000012	ERR_HIL_QUE_SENDPACKET The sending of a packet has failed.
0xC0000013	ERR_HIL_POOL_PACKET_GET The request of a packet from packet pool has failed.
0xC0000014	ERR_HIL_POOL_PACKET_RELEASE The release of a packet-to-packet pool has failed.
0xC0000015	ERR_HIL_POOL_GET_LOAD The get packet pool load function has failed.
0xC0000016	ERR_HIL_QUE_GET_LOAD The get queue load function has failed.

Hexadecimal value	Definition and description
0xC0000017	ERR_HIL_QUE_WAITFORPACKET The waiting for a packet from queue has failed.
0xC0000018	ERR_HIL_QUE_POSTPACKET The posting of a packet has failed.
0xC0000019	ERR_HIL_QUE_PEEKPACKET Peeking a packet from queue has failed.
0xC000001A	ERR_HIL_REQUEST_RUNNING Request is already running.
0xC000001B	ERR_HIL_CREATE_TIMER Creating a timer failed.
0xC000001C	ERR_HIL_BUFFER_TOO_SHORT Supplied buffer too short for the data.
0xC000001D	ERR_HIL_NAME_ALREADY_EXIST Supplied name already exists.
0xC000001E	ERR_HIL_PACKET_FRAGMENTATION_TIMEOUT The packet fragmentation has timed out.
0xC0000100	ERR_HIL_INIT_FAULT General initialization fault.
0xC0000101	ERR_HIL_DATABASE_ACCESS_FAILED Database access failure.
0xC0000102	ERR_HIL_CIR_MASTER_PARAMETER_FAILED Master parameter cannot activated at state operate.
0xC0000103	ERR_HIL_CIR_SLAVE_PARAMTER_FAILED Slave parameter cannot activated at state operate.
0xC0000119	ERR_HIL_NOT_CONFIGURED Configuration not available
0xC0000120	ERR_HIL_CONFIGURATION_FAULT General configuration fault.
0xC0000121	ERR_HIL_INCONSISTENT_DATA_SET Inconsistent configuration data.
0xC0000122	ERR_HIL_DATA_SET_MISMATCH Configuration data set mismatch.
0xC0000123	ERR_HIL_INSUFFICIENT_LICENSE Insufficient license.
0xC0000124	ERR_HIL_PARAMETER_ERROR Parameter error.
0xC0000125	ERR_HIL_INVALID_NETWORK_ADDRESS Network address invalid.
0xC0000126	ERR_HIL_NO_SECURITY_MEMORY Security memory chip missing or broken.
0xC0000127	ERR_HIL_NO_MAC_ADDRESS_AVAILABLE No MAC address available.
0xC0000140	ERR_HIL_NETWORK_FAULT General communication fault.
0xC0000141	ERR_HIL_CONNECTION_CLOSED Connection closed.
0xC0000142	ERR_HIL_CONNECTION_TIMEOUT Connection timeout.

Hexadecimal value	Definition and description
0xC0000143	ERR_HIL_LONELY_NETWORK Lonely network.
0xC0000144	ERR_HIL_DUPLICATE_NODE Duplicate network address.
0xC0000145	ERR_HIL_CABLE_DISCONNECT Cable disconnected.
0xC0000180	ERR_HIL_BUS_OFF Bus Off flag is set.
0xC0000181	ERR_HIL_CONFIG_LOCK Changing configuration is not allowed.
0xC0000182	ERR_HIL_APPLICATION_NOT_READY Application is not at ready state.
0xC0000183	ERR_HIL_RESET_IN_PROCESS Application is performing a reset.
0xC0000200	ERR_HIL_WATCHDOG_TIME_INVALID Watchdog time is out of range.
0xC0000201	ERR_HIL_APPLICATION_ALREADY_REGISTERED Application is already registered.
0xC0000202	ERR_HIL_NO_APPLICATION_REGISTERED No application registered.
0xC0000203	ERR_HIL_INVALID_COMPONENT_ID Invalid component identifier.
0xC0000204	ERR_HIL_INVALID_DATA_LENGTH Invalid data length.
0xC0000205	ERR_HIL_DATA_ALREADY_SET The data was already set.
0xC0000206	ERR_HIL_NO_LOGBOOK_AVAILABLE Logbook not available.
0xC0001000	ERR_HIL_INVALID_HANDLE No description available - ERR_HIL_INVALID_HANDLE.
0xC0001001	ERR_HIL_UNKNOWN_DEVICE No description available - ERR_HIL_UNKNOWN_DEVICE.
0xC0001002	ERR_HIL_RESOURCE_IN_USE No description available - ERR_HIL_RESOURCE_IN_USE.
0xC0001003	ERR_HIL_NO_MORE_RESOURCES No description available - ERR_HIL_NO_MORE_RESOURCES.
0xC0001004	ERR_HIL_DRV_OPEN_FAILED No description available - ERR_HIL_DRV_OPEN_FAILED.
0xC0001005	ERR_HIL_DRV_INITIALIZATION_FAILED No description available - ERR_HIL_DRV_INITIALIZATION_FAILED.
0xC0001006	ERR_HIL_DRV_NOT_INITIALIZED No description available - ERR_HIL_DRV_NOT_INITIALIZED.
0xC0001007	ERR_HIL_DRV_ALREADY_INITIALIZED No description available - ERR_HIL_DRV_ALREADY_INITIALIZED.
0xC0001008	ERR_HIL_CRC No description available - ERR_HIL_CRC.
0xC0001010	ERR_HIL_DRV_INVALID_RESOURCE No description available - ERR_HIL_DRV_INVALID_RESOURCE.

Hexadecimal value	Definition and description
0xC0001011	ERR_HIL_DRV_INVALID_MEM_RESOURCE No description available - ERR_HIL_DRV_INVALID_MEM_RESOURCE.
0xC0001012	ERR_HIL_DRV_INVALID_MEM_SIZE No description available - ERR_HIL_DRV_INVALID_MEM_SIZE.
0xC0001013	ERR_HIL_DRV_INVALID_PHYS_MEM_BASE No description available - ERR_HIL_DRV_INVALID_PHYS_MEM_BASE.
0xC0001014	ERR_HIL_DRV_INVALID_PHYS_MEM_SIZE No description available - ERR_HIL_DRV_INVALID_PHYS_MEM_SIZE.
0xC0001015	ERR_HIL_DRV_UNDEFINED_HANDLER No description available - ERR_HIL_DRV_UNDEFINED_HANDLER.
0xC0001020	ERR_HIL_DRV_ILLEGAL_VECTOR_ID No description available - ERR_HIL_DRV_ILLEGAL_VECTOR_ID.
0xC0001021	ERR_HIL_DRV_ILLEGAL_IRQ_MASK No description available - ERR_HIL_DRV_ILLEGAL_IRQ_MASK.
0xC0001022	ERR_HIL_DRV_ILLEGAL_SUBIRQ_MASK No description available - ERR_HIL_DRV_ILLEGAL_SUBIRQ_MASK.
0xC0001100	ERR_HIL_DPM_CHANNEL_UNKNOWN No description available - ERR_HIL_DPM_CHANNEL_UNKNOWN.
0xC0001101	ERR_HIL_DPM_CHANNEL_INVALID No description available - ERR_HIL_DPM_CHANNEL_INVALID.
0xC0001102	ERR_HIL_DPM_CHANNEL_NOT_INITIALIZED No description available - ERR_HIL_DPM_CHANNEL_NOT_INITIALIZED.
0xC0001103	ERR_HIL_DPM_CHANNEL_ALREADY_INITIALIZED No description available - ERR_HIL_DPM_CHANNEL_ALREADY_INITIALIZED.
0xC0001120	ERR_HIL_DPM_CHANNEL_LAYOUT_UNKNOWN No description available - ERR_HIL_DPM_CHANNEL_LAYOUT_UNKNOWN.
0xC0001121	ERR_HIL_DPM_CHANNEL_SIZE_INVALID No description available - ERR_HIL_DPM_CHANNEL_SIZE_INVALID.
0xC0001122	ERR_HIL_DPM_CHANNEL_SIZE_EXCEEDED No description available - ERR_HIL_DPM_CHANNEL_SIZE_EXCEEDED.
0xC0001123	ERR_HIL_DPM_CHANNEL_TOO_MANY_BLOCKS No description available - ERR_HIL_DPM_CHANNEL_TOO_MANY_BLOCKS.
0xC0001130	ERR_HIL_DPM_BLOCK_UNKNOWN No description available - ERR_HIL_DPM_BLOCK_UNKNOWN.
0xC0001131	ERR_HIL_DPM_BLOCK_SIZE_EXCEEDED No description available - ERR_HIL_DPM_BLOCK_SIZE_EXCEEDED.
0xC0001132	ERR_HIL_DPM_BLOCK_CREATION_FAILED No description available - ERR_HIL_DPM_BLOCK_CREATION_FAILED.
0xC0001133	ERR_HIL_DPM_BLOCK_OFFSET_INVALID No description available - ERR_HIL_DPM_BLOCK_OFFSET_INVALID.
0xC0001140	ERR_HIL_DPM_CHANNEL_HOST_MBX_FULL No description available - ERR_HIL_DPM_CHANNEL_HOST_MBX_FULL.
0xC0001141	ERR_HIL_DPM_CHANNEL_SEGMENT_LIMIT No description available - ERR_HIL_DPM_CHANNEL_SEGMENT_LIMIT.
0xC0001142	ERR_HIL_DPM_CHANNEL_SEGMENT_UNUSED No description available - ERR_HIL_DPM_CHANNEL_SEGMENT_UNUSED.
0xC0001143	ERR_HIL_NAME_INVALID No description available - ERR_HIL_NAME_INVALID.

Hexadecimal value	Definition and description
0xC0001144	ERR_HIL_UNEXPECTED_BLOCK_SIZE No description available - ERR_HIL_UNEXPECTED_BLOCK_SIZE.
0xC0001145	ERR_HIL_COMPONENT_BUSY The component is busy and cannot handle the requested service.
0xC0001150	ERR_HIL_INVALID_HEADER Invalid (file) header. E.g. wrong CRC/MD5/Cookie.
0xC0001151	ERR_HIL_INCOMPATIBLE Firmware does not match device.
0xC0001152	ERR_HIL_NOT_AVAILABLE Update file or destination (XIP-Area) not found.
0xC0001153	ERR_HIL_READ Failed to read from file/area.
0xC0001154	ERR_HIL_WRITE Failed to write from file/area.
0xC0001155	ERR_HIL_IDENTICAL Update firmware and installed firmware are identical.
0xC0001156	ERR_HIL_INSTALLATION Error during installation of firmware.
0xC0001157	ERR_HIL_VERIFICATION Error during verification of firmware.
0xC0001158	ERR_HIL_INVALIDATION Error during invalidation of firmware files.
0xC0001160	ERR_HIL_FORMAT Volume is not formatted.
0xC0001161	ERR_HIL_VOLUME (De-)Initialization of volume failed.
0xC0001162	ERR_HIL_VOLUME_DRV (De-)Initialization of volume driver failed.
0xC0001163	ERR_HIL_VOLUME_INVALID The volume is invalid.
0xC0001164	ERR_HIL_VOLUME_EXCEEDED Number of supported volumes exceeded.
0xC0001165	ERR_HIL_VOLUME_MOUNT The volume is mounted (in use).
0xC0001166	ERR_HIL_ERASE Failed to erase file/directory/flash.
0xC0001167	ERR_HIL_OPEN Failed to open file/directory.
0xC0001168	ERR_HIL_CLOSE Failed to close file/directory.
0xC0001169	ERR_HIL_CREATE Failed to create file/directory.
0xC0001170	ERR_HIL_MODIFY Failed to modify file/directory.
0x0000F005	SUCCESS_HIL_FRAGMENTED Fragment accepted.
0xC000F006	ERR_HIL_RESET_REQUIRED Reset required.

Hexadecimal value	Definition and description
0xC000F007	ERR_HIL_EVALUATION_TIME_EXPIRED Evaluation time expired. Reset required.
0xC000DEAD	ERR_HIL_FIRMWARE_CRASHED The firmware has crashed and the exception handler is running.

Table 208: Common status codes

9.2 PROFINET IO-Device Interface Task

Hexadecimal value	Definition and description
0xC0300001	ERR_PNS_IF_COMMAND_INVALID Invalid command.
0xC0300002	ERR_PNS_IF_OS_INIT_FAILED Initialization of PNS Operating system adaption failed.
0xC0300003	ERR_PNS_IF_SET_INIT_IP_FAILED Initialization of PNS IP address failed.
0xC0300004	ERR_PNS_IF_PNIO_SETUP_FAILED PROFINET IO-Device Setup failed.
0xC0300005	ERR_PNS_IF_DEVICE_INFO_ALREADY_SET Device information set already.
0xC0300006	ERR_PNS_IF_SET_DEVICE_INFO_FAILED Setting of device information failed.
0xC0300007	ERR_PNS_IF_NO_DEVICE_SETUP PROFINET IO-Device stack is not initialized. Send PNS_IF_SET_DEVICEINFO_REQ before PNS_IF_OPEN_DEVICE_REQ.
0xC0300008	ERR_PNS_IF_DEVICE_OPEN_FAILED Opening a device instance failed.
0xC0300009	ERR_PNS_IF_NO_DEVICE_INSTANCE No device instance open.
0xC030000A	ERR_PNS_IF_PLUG_MODULE_FAILED Plugging a module failed.
0xC030000B	ERR_PNS_IF_PLUG_SUBMODULE_FAILED Plugging a submodule failed.
0xC030000C	ERR_PNS_IF_DEVICE_START_FAILED Start of PROFINET IO-Device failed.
0xC030000D	ERR_PNS_IF_EDD_ENABLE_FAILED Start of network communication failed.
0xC030000E	ERR_PNS_IF_ALLOC_MNGMNT_BUFFER_FAILED Allocation of a device instance management buffer failed.
0xC030000F	ERR_PNS_IF_DEVICE_HANDLE_NULL Given device handle is NULL.
0xC0300010	ERR_PNS_IF_SET_APPL_READY_FAILED Command PNS_IF_SET_APPL_READY_REQ failed.
0xC0300011	ERR_PNS_IF_SET_DEVSTATE_FAILED Command PNS_IF_SET_DEVSTATE_REQ failed.
0xC0300012	ERR_PNS_IF_PULL_SUBMODULE_FAILED Pulling the submodule failed.
0xC0300013	ERR_PNS_IF_PULL_MODULE_FAILED Pulling the module failed.

Hexadecimal value	Definition and description
0xC0300014	ERR_PNS_IF_WRONG_DEST_ID Destination ID in command invalid.
0xC0300015	ERR_PNS_IF_DEVICE_HANDLE_INVALID Device Handle in command invalid.
0xC0300016	ERR_PNS_IF_CALLBACK_TIMEOUT PNS stack callback timeout.
0xC0300017	ERR_PNS_IF_PACKET_POOL_EMPTY PNS_IF packet pool empty.
0xC0300018	ERR_PNS_IF_ADD_API_FAILED Command PNS_IF_ADD_API_REQ failed.
0xC0300019	ERR_PNS_IF_SET_SUB_STATE_FAILED Setting submodule state failed.
0xC030001A	ERR_PNS_IF_NO_NW_DBM_ERROR No network configuration DBM-file.
0xC030001B	ERR_PNS_IF_NW_SETUP_TABLE_ERROR Error during reading the "SETUP" table of the network configuration DBM-file .
0xC030001C	ERR_PNS_IF_CFG_SETUP_TABLE_ERROR Error during reading the "SETUP" table of the config.xxx DBM-file .
0xC030001D	ERR_PNS_IF_NO_CFG_DBM_ERROR No config.xxx DBM-file.
0xC030001E	ERR_PNS_IF_DBM_DATASET_ERROR Error getting dataset pointer.
0xC030001F	ERR_PNS_IF_SETUPEX_TABLE_ERROR Error getting dataset pointer(SETUP_EX table).
0xC0300020	ERR_PNS_IF_AP_TABLE_ERROR Error getting either dataset pointer or number of datasets(AP table).
0xC0300021	ERR_PNS_IF_MODULES_TABLE_ERROR Error getting either dataset pointer or number of datasets(MODULE table).
0xC0300022	ERR_PNS_IF_SUBMODULES_TABLE_ERROR Error getting either dataset pointer or number of datasets(SUBMODULE table).
0xC0300023	ERR_PNS_IF_PNIO_SETUP_ERROR Error setting up PNIO configuration(PNIO_setup()).
0xC0300024	ERR_PNS_IF_MODULES_GET_REC Error getting record of "MODULES" linked table.
0xC0300025	ERR_PNS_IF_SUBMODULES_GET_REC Error getting record of "SUBMODULES" linked table.
0xC0300026	ERR_PNS_IF_PNIOD_MODULE_ID_TABLE_ERROR Error accessing "PNIOD_MODULE_ID" table or table record error.
0xC0300027	ERR_PNS_IF_SIGNALS_TABLE_ERROR Error accessing "SIGNALS" table or table record error.
0xC0300028	ERR_PNS_IF_MODULES_IO_TABLE_ERROR Error accessing "MODULES_IO" table or table record error.
0xC0300029	ERR_PNS_IF_CHANNEL_SETTING_TABLE_ERROR Error accessing "CHANNEL_SETTING" table or table record error.
0xC030002A	ERR_PNS_IF_WRITE_DBM Error writing DBM-file.
0xC030002B	ERR_PNS_IF_DPM_CONFIG No basic DPM configuration.

Hexadecimal value	Definition and description
0xC030002C	ERR_PNS_IF_WATCHDOG Application did not trigger the watchdog.
0xC030002D	ERR_PNS_IF_SIGNALS_SUBMODULES Data length in "SIGNALS" table does not correspond to that in "SUBMODULES" table.
0xC030002E	ERR_PNS_IF_READ_DPM_SUBAREA Failed to read DPM subarea.
0xC030002F	ERR_PNS_IF_MOD_0_SUB_1 Error configuring Module 0 Submodule 1.
0xC0300030	ERR_PNS_IF_SIGNALS_LENGTH Length of I/O signals is bigger than the size of DPM subarea.
0xC0300031	ERR_PNS_IF_SUB_TRANSFER_DIRECTION A submodule cannot have input and outputs at the same time.
0xC0300032	ERR_PNS_IF_FORMAT_PNVOLUME Error while formatting PNVOLUME.
0xC0300033	ERR_PNS_IF_MOUNT_PNVOLUME Error while mounting PNVOLUME.
0xC0300034	ERR_PNS_IF_INIT_REMOTE Error during initialization of the remote resources of the stack.
0xC0300035	ERR_PNS_IF_WARMSTART_CONFIG_REDUNDANT Warmstart parameters are redundant. The stack was configured with DBM or packets.
0xC0300036	ERR_PNS_IF_WARMSTART_PARAMETER Incorrect warmstart parameter(s).
0xC0300037	ERR_PNS_IF_SET_APPL_STATE_READY PNIO_set_appl_state_ready() returns error.
0xC0300038	ERR_PNS_IF_SET_DEV_STATE PNIO_set_dev_state() returns error.
0xC0300039	ERR_PNS_IF_PROCESS_ALARM_SEND PNIO_process_alarm_send() returns error.
0xC030003a	ERR_PNS_IF_RET_OF_SUB_ALARM_SEND PNIO_ret_of_sub_alarm_send() returns error.
0xC030003b	ERR_PNS_IF_DIAG_ALARM_SEND PNIO_diag_alarm_send() returns error.
0xC030003c	ERR_PNS_IF_DIAG_GENERIC_ADD PNIO_diag_generic_add() returns error.
0xC030003d	ERR_PNS_IF_DIAG_GENERIC_REMOVE PNIO_diag_generic_remove() returns error.
0xC030003e	ERR_PNS_IF_DIAG_CHANNEL_ADD PNIO_diag_channel_add() returns error.
0xC030003f	ERR_PNS_IF_DIAG_CHANNEL_REMOVE PNIO_diag_channel_remove() returns error.
0xC0300040	ERR_PNS_IF_EXT_DIAG_CHANNEL_ADD PNIO_ext_diag_channel_add() returns error.
0xC0300041	ERR_PNS_IF_EXT_DIAG_CHANNEL_REMOVE PNIO_ext_diag_channel_remove() returns error.
0xC0300042	ERR_PNS_IF_STATION_NAME_LEN Parameter station name length is incorrect.
0xC0300043	ERR_PNS_IF_STATION_NAME Parameter station name is incorrect.

Hexadecimal value	Definition and description
0xC0300044	ERR_PNS_IF_STATION_TYPE_LEN Parameter station type length is incorrect.
0xC0300045	ERR_PNS_IF_DEVICE_TYPE Parameter device type is incorrect.
0xC0300046	ERR_PNS_IF_ORDER_ID Parameter order id is incorrect.
0xC0300047	ERR_PNS_IF_INPUT_STATUS Parameter input data status bytes length is incorrect.
0xC0300048	ERR_PNS_IF_OUTPUT_STATUS Parameter output data status bytes length is incorrect.
0xC0300049	ERR_PNS_IF_WATCHDOG_PARAMETER Parameter watchdog timing is incorrect(must be >= 10).
0xC030004a	ERR_PNS_IF_OUT_UPDATE Parameter output data update timing is incorrect.
0xC030004b	ERR_PNS_IF_IN_UPDATE Parameter input data update timing is incorrect.
0xC030004c	ERR_PNS_IF_IN_SIZE Parameter input memory area size is incorrect.
0xC030004d	ERR_PNS_IF_OUT_SIZE Parameter output memory area size is incorrect.
0xC030004e	ERR_PNS_IF_GLOBAL_RESOURCES Unable to allocate memory for global access to local resources.
0xC030004f	ERR_PNS_IF_DYNAMIC_CFG_PCK Unable to allocate memory for dynamic configuration packet.
0xC0300050	ERR_PNS_IF_DEVICE_STOP Unable to stop device.
0xC0300051	ERR_PNS_IF_DEVICE_ID Parameter device id is incorrect.
0xC0300052	ERR_PNS_IF_VENDOR_ID Parameter vendor id is incorrect.
0xC0300053	ERR_PNS_IF_SYS_START Parameter system start is incorrect.
0xC0300054	ERR_PNS_IF_DYN_CFG_IO_LENGTH The length of IO data expected by the controller exceeds the limit specified in warmstart parameters.
0xC0300055	ERR_PNS_IF_DYN_CFG_MOD_NUM The count of the IO modules expected by the controller exceeds the supported by the stack count.
0xC0300056	ERR_PNS_IF_ACCESS_LOCAL_RSC No global access to local resources.
0xC0300057	ERR_PNS_IF_PULL_PLUG Plugging and pulling modules during creation of communication is not allowed.
0xC0300058	ERR_PNS_IF_AR_NUM Maximum number of ARs is 1.
0xC0300059	ERR_PNS_IF_API_NUM Only API = 0 is supported.
0xC030005a	ERR_PNS_IF_ALREADY_OPEN Device is already opened.

Hexadecimal value	Definition and description
0xC030005b	ERR_PNS_IF_API_ADDED Application is already added.
0xC030005c	ERR_PNS_IF_CONFIG_MODE Configuration modes should not be mixed(DBM-files,application,warmstart message).
0xC030005d	ERR_PNS_IF_UNK_LED_MODE Unknown LED mode.
0xC030005e	ERR_PNS_IF_PHYSICAL_LINK Physical link rate is less than 100 Mbit.
0xC030005f	ERR_PNS_IF_MAX_SLOT_SUBSLOT Number of slots or subslots too big.
0xC0300060	ERR_PNS_IF_AR_REASON_MEM AR error. Out of memory.
0xC0300061	ERR_PNS_IF_AR_REASON_FRAME AR error. Add provider or consumer failed.
0xC0300062	ERR_PNS_IF_AR_REASON_MISS AR error. Consumer missing.
0xC0300063	ERR_PNS_IF_AR_REASON_TIMER AR error. CMI timeout.
0xC0300064	ERR_PNS_IF_AR_REASON_ALARM AR error. Alarm open failed.
0xC0300065	ERR_PNS_IF_AR_REASON_ALSND AR error. Alarm send confirmation failed.
0xC0300066	ERR_PNS_IF_AR_REASON_ALACK AR error. Alarm acknowledge send confirmation failed.
0xC0300067	ERR_PNS_IF_AR_REASON_ALLEN AR error. Alarm data too long.
0xC0300068	ERR_PNS_IF_AR_REASON_ASRT AR error. Alarm indication error.
0xC0300069	ERR_PNS_IF_AR_REASON_RPC AR error. RPC client call confirmation failed.
0xC030006A	ERR_PNS_IF_AR_REASON_ABORT AR error. Abort request.
0xC030006B	ERR_PNS_IF_AR_REASON_RERUN AR error. Re-Run.
0xC030006C	ERR_PNS_IF_AR_REASON_REL AR error. Release indication received.
0xC030006D	ERR_PNS_IF_AR_REASON_PAS AR error. Device deactivated.
0xC030006E	ERR_PNS_IF_AR_REASON_RMV AR error. Device/ar removed.
0xC030006F	ERR_PNS_IF_AR_REASON_PROT AR error. Protocol violation.
0xC0300070	ERR_PNS_IF_AR_REASON_NARE AR error. NARE error.
0xC0300071	ERR_PNS_IF_AR_REASON_BIND AR error. RPC-Bind error.
0xC0300072	ERR_PNS_IF_AR_REASON_CONNECT AR error. RPC-Connect error.

Hexadecimal value	Definition and description
0xC0300073	ERR_PNS_IF_AR_REASON_READ AR error. RPC-Read error.
0xC0300074	ERR_PNS_IF_AR_REASON_WRITE AR error. RPC-Write error.
0xC0300075	ERR_PNS_IF_AR_REASON_CONTROL AR error. RPC-Control error.
0xC0300076	ERR_PNS_IF_AR_REASON_UNKNOWN AR error. Unknown.
0xC0300077	ERR_PNS_IF_INIT_WATCHDOG Watchdog initialization failed.
0xC0300078	ERR_PNS_IF_NO_PHYSICAL_LINK The Device is not connected to a network.
0xC0300079	ERR_PNS_IF_DPM_CYCLIC_IO_RW Failed to copy from DPM or to DPM the cyclic IO data.
0xC030007A	ERR_PNS_IF_SUBMODULE Submodule number is wrong.
0xC030007B	ERR_PNS_IF_MODULE Module number is wrong.
0xC030007C	ERR_PNS_IF_NO_AR The AR was closed or the AR handle is not valid.
0xC030007D	ERR_PNS_IF_WRITE_REC_RES_TIMEOUT Timeout while waiting for response to write_record_indication.
0xC030007E	ERR_PNS_IF_UNREGISTERED_SENDER The sender of the request in not registered with request PNS_IF_REGISTER_AP_REQ.
0xC030007F	ERR_PNS_IF_RECORD_HANDLE_INVALID Unknown record handle.
0xC0300080	ERR_PNS_IF_REGISTER_AP Another instance is registered at the moment.
0xC0300081	ERR_PNS_IF_UNREGISTER_AP One instance cannot unregister another one.
0xC0300082	ERR_PNS_IF_CONFIG_DIFFER The Must-configuration differs from the Is-configuration.
0xC0300083	ERR_PNS_IF_NO_COMMUNICATION No communication processing.
0xC0300084	ERR_PNS_IF_BAD_PARAMETER At least one parameter in a packet was wrong or/and did not meet the requirements.
0xC0300085	ERR_PNS_IF_AREA_OVERFLOW Input or Output data requires more space than available.
0xC0300086	ERR_PNS_IF_WRM_PCK_SAVE Saving Warmstart Configuration for later use was not successful.
0xC0300087	ERR_PNS_IF_AR_REASON_PULLPLUG AR error. Pull and Plug are forbidden after check.rsp and before in-data.ind.
0xC0300088	ERR_PNS_IF_AR_REASON_AP_RMV AR error. AP has been removed.
0xC0300089	ERR_PNS_IF_AR_REASON_LNK_DWN AR error. Link "down".
0xC030008A	ERR_PNS_IF_AR_REASON_MMAC AR error. Could not register multicast-MAC.

Hexadecimal value	Definition and description
0xC030008B	ERR_PNS_IF_AR_REASON_SYNC AR error. Not synchronized (Cannot start companion-AR).
0xC030008C	ERR_PNS_IF_AR_REASON_TOPO AR error. Wrong topology(Cannot start companion-AR).
0xC030008D	ERR_PNS_IF_AR_REASON_DCP_NAME AR error. DCP. Station Name changed.
0xC030008E	ERR_PNS_IF_AR_REASON_DCP_RESET AR error. DCP. Reset to factory-settings.
0xC030008F	ERR_PNS_IF_AR_REASON_PRM AR error. Cannot start companion-AR because a 0x8ipp submodule in the first AR /has appl-ready-pending/ is locked/ is wrong or pulled/ .
0xC0300090	ERR_PNS_IF_PACKET_MNGMNT Packet management error.
0xC0300091	ERR_PNS_IF_WRONG_API_NUM Invalid parameter API.
0xC0300092	ERR_PNS_IF_WRONG_MODULE_ID Invalid parameter ModuleIdentifier (a module with different ModuleIdentifier is already plugged).
0xC0300093	ERR_PNS_IF_WRONG_MODULE_NUM Obsolete, no longer used.
0xC0300094	ERR_PNS_IF_UNS_AREA Obsolete, no longer used.
0xC0300095	ERR_PNS_IF_WRONG_SUB_ID Obsolete, no longer used.
0xC0300096	ERR_PNS_IF_WRONG_SUBMODULE_NUM Obsolete, no longer used.
0xC0300097	ERR_PNS_IF_DEVICE_STOP_FAILED Obsolete, no longer used.
0xC0300098	ERR_PNS_IF_EDD_DISABLE_FAILED Obsolete, no longer used.
0xC0300099	ERR_PNS_IF_WRITE_IN Obsolete, no longer used.
0xC030009A	ERR_PNS_IF_READ_OUT Obsolete, no longer used.
0xC030009B	ERR_PNS_IF_PNIO_STATUS Obsolete, no longer used.
0xC030009C	ERR_PNS_IF_WRONG_MODULE_ADDRESS Obsolete, no longer used.
0xC030009D	ERR_PNS_IF_UNK_DEVICE_STATE Obsolete, no longer used.
0xC030009E	ERR_PNS_IF_ALARM_DATA_LEN Invalid alarm data length.
0xC030009F	ERR_PNS_IF_UNK_SUBMODULE_STATE Obsolete, no longer used.
0xC03000A0	ERR_PNS_IF_BAD_DIAG_HANDLE Invalid parameter Diagnosis handle.
0xC03000A1	ERR_PNS_IF_UNS_STRUCT_ID Obsolete, no longer used.

Hexadecimal value	Definition and description
0xC03000A2	ERR_PNS_IF_UNK_ALARM_STATE Obsolete, no longer used.
0xC03000A3	ERR_PNS_IF_DIAG_DATA_LEN Obsolete, no longer used.
0xC03000A4	ERR_PNS_IF_BAD_CHANNEL_ERR_TYPE Obsolete, no longer used.
0xC03000A5	ERR_PNS_IF_BAD_CHANNEL_PROP Obsolete, no longer used.
0xC03000A6	ERR_PNS_IF_BAD_CHANNEL_NUM Obsolete, no longer used.
0xC03000A7	ERR_PNS_IF_RCX_RESTART Obsolete, no longer used.
0xC03000A8	ERR_PNS_IF_CFG_MNGMNT Obsolete, no longer used.
0xC03000A9	ERR_PNS_IF_UNK_INTERN_REQ Obsolete, no longer used.
0xC03000AA	ERR_PNS_IF_CFG_STORE Obsolete, no longer used.
0xC03000AB	ERR_PNS_IF_CFG_DELETE_FAILED An internal error occurred while deleting the configuration.
0xC03000AC	ERR_PNS_IF_READ_CFG Obsolete, no longer used.
0xC03000AD	ERR_PNS_IF_ACCESS_SYS_VOLUME Obsolete, no longer used.
0xC03000AE	ERR_PNS_IF_ACCESS_BCKUP_VOLUME Obsolete, no longer used.
0xC03000AF	ERR_PNS_IF_CFG_BAD_LEN Obsolete, no longer used.
0xC03000B0	ERR_PNS_IF_WRM_CFG_MNGMNT Obsolete, no longer used.
0xC03000B1	ERR_PNS_IF_RESET_FACTORY_IND No registered application. Reset_to_factory_settings Indication failed.
0xC03000B2	ERR_PNS_IF_MODULE_ALREADY_PLUGGED A module was already plugged to the slot.
0xC03000B3	ERR_PNS_IF_OSINIT Failed to init the OS adaptation layer.
0xC03000B4	ERR_PNS_IF_OSSOCKINIT Failed to init the TCPIP adaptation layer.
0xC03000B5	ERR_PNS_IF_INVALID_NETMASK Invalid subnetwork mask.
0xC03000B6	ERR_PNS_IF_INVALID_IP_ADDR Invalid IP address.
0xC03000B7	ERR_PNS_IF_STA_STARTUP_PARAMETER Erroneous Task start-up parameters.
0xC03000B8	ERR_PNS_IF_INIT_LOCAL Failed to initialize the Task local resources.
0xC03000B9	ERR_PNS_IF_APP_CONFIG_INCOMPLETE The configuration per packets is incomplete.

Hexadecimal value	Definition and description
0xC03000BA	ERR_PNS_IF_INIT_EDD EDD Initialization failed.
0xC03000BB	ERR_PNS_IF_DPM_NOT_ENABLED DPM is not enabled.
0xC03000BC	ERR_PNS_IF_READ_LINK_STATUS Reading Link Status failed.
0xC03000BD	ERR_PNS_IF_INVALID_GATEWAY Invalid gateway address (not reachable with configured netmask).
0xC0300100	ERR_PNS_IF_PACKET_SEND_FAILED Error while sending a packet to another task.
0xC0300101	ERR_PNS_IF_RESOURCE_OUT_OF_MEMORY Insufficient memory to handle the request.
0xC0300102	ERR_PNS_IF_NO_APPLICATION_REGISTERED No application to send the indication to is registered.
0xC0300103	ERR_PNS_IF_INVALID_SOURCE_ID The host-application returned a packet with invalid (changed) SourceID.
0xC0300104	ERR_PNS_IF_PACKET_BUFFER_FULL The buffer used to store packets exchanged between host-application and stack is full.
0xC0300105	ERR_PNS_IF_PULL_NO_MODULE Pulling the (sub)module failed because no module is plugged into the slot specified.
0xC0300106	ERR_PNS_IF_PULL_NO_SUBMODULE Pulling the submodule failed because no submodule is plugged into the subslot specified.
0xC0300107	ERR_PNS_IF_PACKET_BUFFER_RESTORE_ERROR The packet buffer storing packets exchanged between host-application and stack returned an invalid packet.
0xC0300108	ERR_PNS_IF_DIAG_NO_MODULE Diagnosis data not accepted because no module is plugged into the slot specified.
0xC0300109	ERR_PNS_IF_DIAG_NO_SUBMODULE Diagnosis data not accepted because no submodule is plugged into the subslot specified.
0xC030010A	ERR_PNS_IF_CYCLIC_EXCHANGE_ACTIVE The services requested are not available while cyclic communication is running.
0xC030010B	ERR_PNS_IF_FATAL_ERROR_CLB_ALREADY_REGISTERED This fatal error callback function could not be registered because there is already a function registered.
0xC030010C	ERR_PNS_IF_ERROR_STACK_WARMSTART_CONFIGURATION The stack did not accept the warmstart parameters.
0xC030010D	ERR_PNS_IF_ERROR_STACK_MODULE_CONFIGURATION The stack did not accept the module configuration packet.
0xC030010E	ERR_PNS_IF_CHECK_IND_FOR_UNEXPECTED_MODULE The stack sent a Check Indication for an unexpected module. This module was not part of the CR Info Indication.
0xC030010F	ERR_PNS_IF_CHECK_IND_FOR_UNEXPECTED_SUBMODULE The stack sent a Check Indication for an unexpected submodule. This submodule was not part of the CR Info Indication.
0xC0300110	ERR_PNS_DIAG_BUFFER_FULL No more diagnosis records can be added to the stack because the maximum amount is already reached.
0xC0300111	ERR_PNS_IF_CHECK_IND_FOR_UNEXPECTED_API The stack sent a Check Indication for an unexpected API. This API was not part of the CR Info Indication.

Hexadecimal value	Definition and description
0xC0300112	ERR_PNS_IF_DPM_ACCESS_WITH_INVALID_OFFSET The DPM shall be accessed with an invalid data offset.
0xC0300113	ERR_PNS_IF_DUPLICATE_INPUT_CR_INFO The stack indicated to CR Info Indications with type input.
0xC0300114	ERR_PNS_IF_DUPLICATE_OUTPUT_CR_INFO The stack indicated to CR Info Indications with type output.
0xC0300115	ERR_PNS_IF_FAULTY_CR_INFO_IND_RECEIVED The stack indicated a faulty CR Info Indications.
0xC0300116	ERR_PNS_IF_CONFIG_RELOAD_RUNNING The request cannot be executed because configuration reload or ChannelInit is running.
0xC0300117	ERR_PNS_IF_NO_MAC_ADDRESS_SET There is no valid chassis MAC address set Without MAC address the stack will not work.
0xC0300118	ERR_PNS_IF_SET_PORT_MAC_NOT_POSSIBLE The Port MAC addresses have to be set before sending Set-Configuration Request to the stack.
0xC030011A	ERR_PNS_IF_INVALID_MODULE_CONFIGURATION Evaluating the module configuration failed.
0xC030011B	ERR_PNS_IF_CONF_IO_LEN_TO_BIG The sum of IO-data length exceeds the maximum allowed value.
0xC030011C	ERR_PNS_IF_NO_MODULE_CONFIGURED The module configuration does not contain at least one module.
0xC030011D	ERR_PNS_IF_INVALID_SW_REV_PREFIX The value of bSwRevisionPrefix is invalid.
0xC030011E	ERR_PNS_IF_RESERVED_VALUE_NOT_ZERO The value of usReserved it not zero.
0xC030011F	ERR_PNS_IF_IDENTIFY_CMDEV_QUEUE_FAILED Identifying the stack message queue CMDEV failed.
0xC0300120	ERR_PNS_IF_CREATE_SYNC_QUEUE_FAILED Creating the sync message queue failed.
0xC0300121	ERR_PNS_IF_CREATE_ALARM_LOW_QUEUE_FAILED Creating the low alarm message queue failed.
0xC0300122	ERR_PNS_IF_CREATE_ALARM_HIGH_QUEUE_FAILED Creating the high alarm message queue failed.
0xC0300123	ERR_PNS_IF_CFG_PACKET_TO_SMALL While evaluating SetConfiguration packet the packet length was found smaller than amount of configured modules needs.
0xC0300124	ERR_PNS_IF_FATAL_ERROR_OCCURRED A fatal error occurred prior to this request. Therefore this request cannot be fulfilled.
0xC0300125	ERR_PNS_IF_SUBMODULE_NOT_IN_CYCLIC_EXCHANGE The request cannot be executed because the submodule is not in cyclic data exchange.
0xC0300126	ERR_PNS_IF_SERVICE_NOT_AVAILABLE_THROUGH_DPM This service is not available through DPM.
0xC0300127	ERR_PNS_IF_INVALID_PARAMETER_VERSION The version of parameters is invalid (most likely too old).
0xC0300128	ERR_PNS_IF_DATABASE_USAGE_IS_FORBIDDEN The usage of database is forbidden by startup parameters of task.
0xC0300129	ERR_PNS_IF_RECORD_LENGTH_TOO_BIG The amount of record data is too big.

Hexadecimal value	Definition and description
0xC030012A	ERR_PNS_IF_IDENTIFY_LLDP_QUEUE_FAILED Identifying the stack message queue LLDP failed.
0xC030012B	ERR_PNS_IF_INVALID_TOTAL_PACKET_LENGTH SetConfiguration Requests total packet length is invalid.
0xC030012C	ERR_PNS_IF_APPLICATION_TIMEOUT The application needed too much time to respond to an indication.
0xC030012D	ERR_PNS_IF_PACKET_BUFFER_INVALID_PACKET The packet buffer storing packets exchanged between host-application and stack returned a faulty packet.
0xC030012E	ERR_PNS_IF_NO_IO_IMAGE_CONFIGURATION_AVAILABLE The request cannot be handled until a valid IO Image configuration is available.
0xC030012F	ERR_PNS_IF_IO_IMAGE_ALREADY_CONFIGURED A valid IO Image configuration is already available.
0xC0300130	ERR_PNS_IF_INVALID_PDEV_SUBSLOT A submodule may only be plugged into a PDEV-subslot which does not exceed the number of supported interfaces and port numbers.
0xC0300131	ERR_PNS_IF_NO_DAP_PRESENT The module configuration does not contain a Device Access Point DAP-submodule in slot 0 subslot 1.
0xC0300132	ERR_PNS_IF_PLUG_SUBMOD_OUTPUT_SIZE_EXCEEDED Output size of the submodule exceeded. Configured value of ulCompleteOutputSize is smaller than the Output size of all plugged input modules. Upgrade ulCompleteOutputSize.
0xC0300133	ERR_PNS_IF_PLUG_SUBMOD_INPUT_SIZE_EXCEEDED Input size of the submodule exceeded. Configured value of ulCompleteInputSize is smaller than the Input size of all plugged input modules. Upgrade ulCompleteInputSize.
0xC0300134	ERR_PNS_IF_PLUG_SUBMOD_NO_MODULE_ATTACHED_TO_ADD_TO No module attached to add the submodule to.
0xC0300135	ERR_PNS_IF_PLUG_SUBMOD_ALREADY_PLUGGED_THIS_SUBMOD Submodule already plugged.
0xC0300136	ERR_PNS_IF_SETIOXS_INVALID_PROV_IMAGE Invalid IOXS provider image.
0xC0300137	ERR_PNS_IF_SETIOXS_INVALID_CONS_IMAGE Invalid IOXS consumer image.
0xC0300138	ERR_PNS_IF_INVALID_IOPS_MODE Invalid IOPS mode.
0xC0300139	ERR_PNS_IF_INVALID_IOCS_MODE Invalid IOCS mode.
0xC030013A	ERR_PNS_IF_INVALID_API Invalid API.
0xC030013B	ERR_PNS_IF_INVALID_SLOT Invalid slot.
0xC030013C	ERR_PNS_IF_INVALID_SUBSLOT Invalid subslot.
0xC030013D	ERR_PNS_IF_INVALID_CHANNEL_NUMBER Invalid channel number.
0xC030013E	ERR_PNS_IF_INVALID_CHANNEL_PROPERTIES Invalid channel properties.
0xC030013F	ERR_PNS_IF_CHANNEL_ERRORTYPE_NOT_ALLOWED Invalid channel error type not allowed.

Hexadecimal value	Definition and description
0xC0300140	ERR_PNS_IF_EXT_CHANNEL_ERRORTYPE_NOT_ALLOWED Invalid channel EXT error type not allowed.
0xC0300141	ERR_PNS_IF_INVALID_USER_STRUCT_IDENTIFIER Invalid user struct identifier.
0xC0300142	ERR_PNS_IF_INVALID_SUBMODULE Invalid submodule.
0xC0300143	ERR_PNS_IF_INVALID_IM_TYPE Invalid IM type.
0xC0300144	ERR_PNS_IF_IDENTIFY_FODMI_QUEUE_FAILED Failed to identify the FODMI Queue.
0xC0300145	ERR_PNS_IF_DPM_MAILBOX_OVERFLOW The DPM Receive Mailbox Queue run out of space. Most likely the host did not fetch the packets.
0xC0300146	ERR_PNS_IF_APPL_IM_ACCESS_DENIED The application denied read/write access to I&M record object.
0xC0300147	ERR_PNS_IF_APPL_IM_INVALID_INDEX The application does not implement the requested I&M record object.
0xC0300148	ERR_PNS_IF_TAGLIST_INVALID_SUBMODULE_NUMBER Invalid number of max supported submodules.
0xC0300149	ERR_PNS_IF_TAGLIST_INVALID_ADDITIONAL_AR_NUMBER Invalid number of max supported additional IO ARs.
0xC030014A	ERR_PNS_IF_TAGLIST_INVALID_IMPLICIT_AR_NUMBER Invalid number of max supported implicit IO ARs.
0xC030014B	ERR_PNS_IF_TAGLIST_INVALID_DAAR_NUMBER Invalid number of max supported Device Access ARs.
0xC030014C	ERR_PNS_IF_TAGLIST_INVALID_MIN_RPC_BUFFER_SIZE Invalid RPC buffer size.
0xC030014D	ERR_PNS_IF_TAGLIST_INVALID_DIAGNOSIS_ENTRIES_NUM Invalid number of max supported diagnosis entries .
0xC030014E	ERR_PNS_IF_TAGLIST_INVALID_ARSET_NUM Invalid number of max supported AR sets.
0xC030014F	ERR_PNS_IF_PE_ENTITY_EXISTS A PE Entity was already added to this submodule.
0xC0300150	ERR_PNS_IF_NO_PE_ENTITY The submodule does not have a PE Entity.
0xC0300A00	ERR_PNS_IF_CM_AR_REASON_NONE None. Unused.
0xC0300A03	ERR_PNS_IF_CM_AR_REASON_MEM AR Out of memory.
0xC0300A04	ERR_PNS_IF_CM_AR_REASON_FRAME AR add provider or consumer failed.
0xC0300A05	ERR_PNS_IF_CM_AR_REASON_MISS AR consumer DHT/WDT expired.
0xC0300A06	ERR_PNS_IF_CM_AR_REASON_TIMER AR cmi timeout.
0xC0300A07	ERR_PNS_IF_CM_AR_REASON_ALARM AR alarm-open failed.
0xC0300A08	ERR_PNS_IF_CM_AR_REASON_ALSND AR alarm-send.cnf(-).

Hexadecimal value	Definition and description
0xC0300A09	ERR_PNS_IF_CM_AR_REASON_ALACK AR alarm-ack-send.cnf(-).
0xC0300A0A	ERR_PNS_IF_CM_AR_REASON_ALLEN AR alarm data too long.
0xC0300A0B	ERR_PNS_IF_CM_AR_REASON_ASRT AR alarm.ind(err).
0xC0300A0C	ERR_PNS_IF_CM_AR_REASON_RPC AR rpc-client call.cnf(-).
0xC0300A0D	ERR_PNS_IF_CM_AR_REASON_ABORT AR abort.req.
0xC0300A0E	ERR_PNS_IF_CM_AR_REASON_RERUN AR re-run aborts existing AR.
0xC0300A0F	ERR_PNS_IF_CM_AR_REASON_REL AR release.ind received.
0xC0300A10	ERR_PNS_IF_CM_AR_REASON_PAS AR device deactivated.
0xC0300A11	ERR_PNS_IF_CM_AR_REASON_RMV AR removed.
0xC0300A12	ERR_PNS_IF_CM_AR_REASON_PROT AR protocol violation.
0xC0300A13	ERR_PNS_IF_CM_AR_REASON_NARE AR name resolution error.
0xC0300A14	ERR_PNS_IF_CM_AR_REASON_BIND AR RPC-Bind error.
0xC0300A15	ERR_PNS_IF_CM_AR_REASON_CONNECT AR RPC-Connect error.
0xC0300A16	ERR_PNS_IF_CM_AR_REASON_READ AR RPC-Read error.
0xC0300A17	ERR_PNS_IF_CM_AR_REASON_WRITE AR RPC-Write error.
0xC0300A18	ERR_PNS_IF_CM_AR_REASON_CONTROL AR RPC-Control error.
0xC0300A19	ERR_PNS_IF_CM_AR_REASON_PULLPLUG AR forbidden pull or plug after check.rsp and before in-data.ind.
0xC0300A1A	ERR_PNS_IF_CM_AR_REASON_AP_RMV AR AP removed.
0xC0300A1B	ERR_PNS_IF_CM_AR_REASON_LNK_DWN AR link down.
0xC0300A1C	ERR_PNS_IF_CM_AR_REASON_MMAC AR could not register multicast-mac address.
0xC0300A1D	ERR_PNS_IF_CM_AR_REASON_SYNC Not synchronized (cannot start companion-ar).
0xC0300A1E	ERR_PNS_IF_CM_AR_REASON_TOPO Wrong topology (cannot start companion-ar).
0xC0300A1F	ERR_PNS_IF_CM_AR_REASON_DCP_NAME DCP, station-name changed.
0xC0300A20	ERR_PNS_IF_CM_AR_REASON_DCP_RESET DCP, reset to factory-settings.

Hexadecimal value	Definition and description
0xC0300A21	ERR_PNS_IF_CM_AR_REASON_PRM 0x8ipp submodule in the first AR has either an appl-ready-pending (erroneous parameterization) or is locked (no parameterization) or is wrong or pulled (no parameterization).
0xC0300A22	ERR_PNS_IF_CM_AR_REASON_IRDATA No irdata record yet.
0xC0300A23	ERR_PNS_IF_CM_AR_REASON_PDEV Ownership of PDEV.
0xC0300AFF	ERR_PNS_IF_CM_AR_REASON_MAX Max. Unused.

Table 209: PROFINET IO-Device interface task

9.3 Socket API

Hexadecimal value	Definition and description
0xC0C90001	ERR_SOCK_UNSUPPORTED_SOCKET Unsupported socket domain, type and protocol combination.
0xC0C90002	ERR_SOCK_INVALID_SOCKET_HANDLE Invalid socket handle.
0xC0C90003	ERR_SOCK_SOCKET_CLOSED Socket was closed.
0xC0C90004	ERR_SOCK_INVALID_OP The command is invalid for the particular socket.
0xC0C90005	ERR_SOCK_INVALID_ADDRESS_FAMILY An invalid address family was used for this socket
0xC0C90006	ERR_SOCK_IN_USE The specified address is already in use.
0xC0C90007	ERR_SOCK_HUP The remote side closed the connection
0xC0C90008	ERR_SOCK_WOULDBLOCK The operation would block
0xC0C90009	ERR_SOCK_ROUTE No IP route to destination address.
0xC0C9000A	ERR_SOCK_IS_CONNECTED IP socket already connected.
0xC0C9000B	ERR_SOCK_CONNECTION_ABORTED TCP connection aborted.
0xC0C9000C	ERR_SOCK_CONNECTION_RESET TCP connection reset.
0xC0C9000D	ERR_SOCK_CONNECTION_CLOSED TCP connection closed.
0xC0C9000E	ERR_SOCK_NOT_CONNECTED IP Socket not connected.
0xC0C9000F	ERR_SOCK_NETWORK_INTERFACE Low-level network interface error.

Table 210: Socket API error codes

9.4 PROFINET status codes

The PROFINET status code is used by the PROFINET protocol to indicate success or failure. This chapter shall give a short introduction to this topic, as the status helps to solve problems and is used in some services as well (The according field is often named `ulPnio`). These services include:

- *Read Record service* [▸ page 125]
- *Write Record service* [▸ page 129]
- *AR Abort Indication service* [▸ page 134]
- *AR Abort Request service* [▸ page 198]

The status code is an unsigned 32-bit integer value which can be structured into four fields as shown in the following figure.

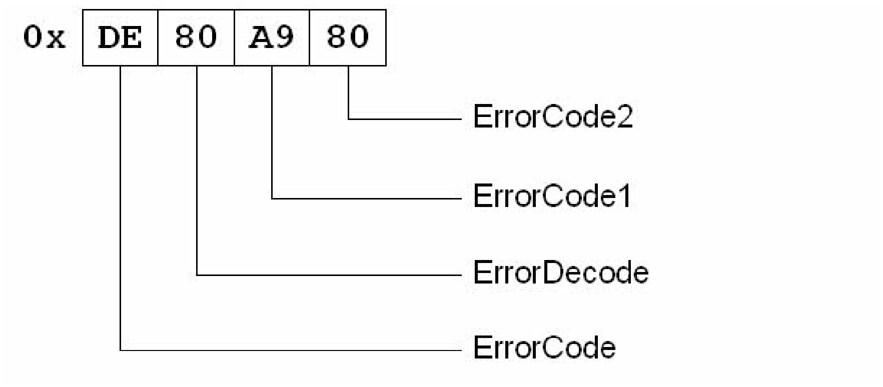


Figure 28: Structure of the PROFINET status code

The fields define a hierarchy on the error codes. The ordering is as follows: ErrorCode, ErrorDecode, ErrorCode1 and ErrorCode2. The meaning of lower order fields depends on the values of the higher order fields. The special value 0x00000000 means no error or success.

9.4.1 The ErrorCode field

This field defines the domain, in which the error occured. The following table shows the valid values:

Value	Meaning/Use	Description
0x20 - 0x3F	Manufacturer specific: for Log Book	To be used when the application generates a log book entry. Meaning of the values is manufacturer specific (Defined by manufacturer of the device.)
0x81	PNIO: for all errors not covered elsewhere	Used for all errors not covered by the other domains
0xCF	RTA error: used in RTA error PDUs	Used in alarm error telegrams (Low Level)
0xDA	AlarmAck: used in RTA data PDUs	Used in alarm data telegrams (High Level)
0xDB	IODConnectRes: RPC Connect Response	Used to indicate errors occurred when handling a Connect request.
0xDC	IODReleaseRes: RPC Release Response	Used to indicate errors occurred when handling a Release request.
0xDD	IODControlRes: RPC Control Response	Used to indicate errors occurred when handling a Control request.
0xDE	IODReadRes: RPC Read Response	Used to indicate errors occurred when handling a Read request.
0xDF	IODWriteRes: RPC Write Response	Used to indicate error occurred when handling a Write response

Table 211: Coding of PNIO status ErrorCode (Excluding reserved values)

9.4.2 The ErrorDecode field

This field defines the context, under which the error occurred.

Value	Meaning/Use	Description
0x80	PNIORW: Application errors of the services Read and Write	Used in the context of Read and Write services handled by either the PROFINET Device stack or by the Application
0x81	PNIO: Other Services	Used in context with any other services.
0x82	Manufacturer Specific: LogBook Entries	Used in context with LogBook entries generated by the stack or by the application.

Table 212: Coding of PNIO status ErrorDecode (Excluding reserved values)

9.4.3 The ErrorCode1 and ErrorCode2 fields

The ErrorCode1 and ErrorCode2 fields finally specify the error. The meaning of the both fields depends on the value of the ErrorDecode Field.

9.4.3.1 ErrorCode1 and ErrorCode2 for ErrorDecode = PNIORW

If the field ErrorDecode is set to the value PNIORW, the field ErrorCode2 shall be encoded user specific and the ErrorCode1 field is split into an ErrorClass Nibble and an ErrorDecode nibble as shown in the following table.

Note:
As the ErrorCode2 field can be freely chosen in this case, the application may use it to provide more detailed information about the error (e.g. why writing the record into the module was not possible, which value of the parameter was wrong…)

ErrorClass (Bit 7 – 4)	Meaning	ErrorDecode (Bit 3- 0)	Meaning
0xA	Application	0x0	Read Error
		0x1	Write Error
		0x2	Module Failure
		0x7	Busy
		0x8	Version Conflict
		0x9	Feature Not Supported
		0xA-0xF	User Specific
0xB	Access	0x0	Invalid Index
		0x1	Write Length Error
		0x2	Invalid Slot/Subslot
		0x3	Type Conflict
		0x4	Invalid Area/Api
		0x5	State Conflict
		0x6	Access Denied
		0x7	Invalid Range
		0x8	Invalid Parameter
		0x9	Invalid Type
		0xA	Backup
		0xB-0xF	User specific
0xC	Resource	0x0	Read Constrain Conflict
		0x1	Write Constrain Conflict
		0x2	Resource Busy
		0x3	Resource Unavailable
		0x8-0xF	User specific
0xD-0xF	User Specific	0x0-0xF	User specific

Table 213: Coding of ErrorCode1 for ErrorDecode = PNIORW (Excluding reserved values)

9.4.3.2 ErrorCode1 and ErrorCode2 for ErrorDecode = PNIO

The meaning of ErrorCode1 and ErrorCode2 for ErrorDecode = PNIO is shown in the following table. This kind of PROFINET status codes should only be used in conjunction with the *AR Abort Request Service* [▸ page 198] (see end of table).

Coding of ErrorCode1 and ErrorCode 2 for ErrorDecode = PNIO:

ErrCode1	Description	ErrCode2	Description / Use
0x01	Connect Parameter Error. Faulty ARBlockReq	0x00-0x0D	Error in one of the Block Parameters
0x02	Connect Parameter Error. Faulty IOCRBlockReq	0x00-0x1C	Error in one of the Block Parameters
0x03	Connect Parameter Error. Faulty ExpectedSubmoduleBlockReq	0x00-0x10	Error in one of the Block Parameters
0x04	Connect Parameter Error. Faulty AlarmCRBlockReq	0x00-0x0F	Error in one of the Block Parameters
0x05	Connect Parameter Error. Faulty PrmServerBlockReq	0x00-0x08	Error in one of the Block Parameters
0x06	Connect Parameter Error. Faulty MCRBlockReq	0x00-0x08	Error in one of the Block Parameters
0x07	Connect Parameter Error. Faulty ARRPCBlockReq	0x00-0x04	Error in one of the Block Parameters
0x08	Read Write Record Parameter Error. Faulty Record	0x00-0x0C	Error in one of the Block Parameters
0x09	Connect Parameter Error Faulty IRInfoBlock	0x00-0x05	Error in one of the Block Parameters
0x0A	Connect Parameter Error Faulty SRInfoBlock	0x00-0x05	Error in one of the Block Parameters
0x0B	Connect Parameter Error Faulty ARFSUBlock	0x00-0x05	Error in one of the Block Parameters
0x14	IODControl Parameter Error. Faulty ControlBlockConnect	0x00-0x09	Error in one of the Block Parameters
0x15	IODControl Parameter Error. Faulty ControlBlockPlug	0x00-0x09	Error in one of the Block Parameters
0x16	IOXControl Parameter Error. Faulty ControlBlock after a connection establishment	0x00-0x07	Error in one of the Block Parameters
0x17	IOXControl Parameter Error. Faulty ControlBlock a plug alarm	0x00-0x07	Error in one of the Block Parameters
0x18	IODControl Parameter Error Faulty ControlBlockPrmBegin	0x00-0x09	Error in one of the Block Parameters
0x19	IODControl Parameter Error Faulty SubmoduleListBlock	0x00-0x07	Error in one of the Block Parameters
0x28	Release Parameter Error. Faulty ReleaseBlock	0x00-0x07	Error in one of the Block Parameters
0x32	Response Parameter Error. Faulty ARBlockRes	0x00-0x08	Error in one of the Block Parameters
0x33	Response Parameter Error. Faulty IOCRBlockRes	0x00-0x06	Error in one of the Block Parameters
0x34	Response Parameter Error. Faulty AlarmCRBlockRes	0x00-0x06	Error in one of the Block Parameters
0x35	Response Parameter Error. Faulty ModuleDiffBlock	0x00-0x0D	Error in one of the Block Parameters
0x36	Response Parameter Error. Faulty ARRPCBlockRes	0x00-0x04	Error in one of the Block Parameters
0x37	Response Parameter Error Faulty ARServerBlockRes	0x00-0x05	Error in one of the Block Parameters

ErrCode1	Description	ErrCode2	Description / Use
0x3C	AlarmAck Error Codes	0x00	Alarm Type Not Supported
		0x01	Wrong Submodule State
0x3D	CMDEV	0x00	State conflict
		0x01	Resource
		0x02-0xFF	State machine specific
0x3E	CMCTL	0x00	State conflict
		0x01	Timeout
		0x02	No data send
0x3F	CTLDINA	0x00	No DCP active
		0x01	DNS Unknown_RealStationName
		0x02	DCP No_RealStationName
		0x03	DCP Multiple_RealStationName
		0x04	DCP No_StationName
		0x05	No_IP_Addr
		0x06	DCP_Set_Error
0x40	CMRPC	0x00	ArgsLength invalid
		0x01	Unknown Blocks
		0x02	IOCR Missing
		0x03	Wrong AlarmCRBlock count
		0x04	Out of AR Resources
		0x05	AR UUID Unknown
		0x06	State conflict
		0x07	Out of Provider, Consumer or Alarm Resources
		0x08	Out of memory
		0x09	PDev already owned
		0x0A	ARset State conflict during connection establishment
		0x0B	ARset Parameter conflict during connection establishment
0x41	ALPMI	0x00	Invalid state
		0x01	Wrong ACK-PDU
0x42	ALPMR	0x00	Invalid state
		0x01	Wrong Notification PDU
0x43	LMPM	0x00–0xFF	Ethernet Switch Errors
0x44	MAC	0x00-0xFF	
0x45	RPC	0x01	CLRPC_ERR_REJECTED: EPM or Server rejected the call.
		0x02	CLRPC_ERR_FAULTED: Server had fault while executing the call
		0x03	CLRPC_ERR_TIMEOUT: EPM or Server did not respond
		0x04	CLRPC_ERR_IN_ARGS; Broadcast or maybe “ndr_data” too large
		0x05	CLRPC_ERR_OUT_ARGS: Server sent back more than “alloc_len”
		0x06	CLRPC_ERR_DECODE: Result of EPM Lookup could not be decoded
		0x07	CLRPC_ERR_PNIO_OUT_ARGS: Out-args not “PN IO signature”, too short or inconsistent
		0x08	CLRPC_ERR_PNIO_APP_TIMEOUT: RPC call was terminated after RPC application timeout

ErrCode1	Description	ErrCode2	Description / Use
0x46	APMR	0x00	Invalid state
		0x01	LMPM signaled an error
0x47	APMS	0x00	Invalid state
		0x01	LMPM signaled an error
		0x02	Timeout
0x48	CPM	0x00	Invalid state
0x49	PPM	0x00	Invalid state
0x4A	DCPUCS	0x00	Invalid state
		0x01	LMPM signaled an error
		0x02	Timeout
0x4B	DCPUCR	0x00	Invalid state
		0x01	LMPM signaled an error
0x4C	DCPMCS	0x00	Invalid state
		0x01	LMPM signaled an error
0x4D	DCPMCR	0x00	Invalid state
		0x01	LMPM signaled an error
0x4E	FSPM	0x00-0xFF	FAL Service Protocol Machine error
0x64	CTLSM	0x00	Invalid state
		0x01	CTLSM signaled error
0x65	CTLRDI	0x00	Invalid state
		0x01	CTLRDI signaled an error
0x66	CTLRDR	0x00	Invalid state
		0x01	CTLRDR signaled an error
0x67	CTLWRI	0x00	Invalid state
		0x01	CTLWRI signaled an error
0x68	CTLWRR	0x00	Invalid State
		0x01	CTLWRR signaled an error
0x69	CTLIO	0x00	Invalid state
		0x01	CTLIO signaled an error
0x6A	CTLSU	0x00	Invalid state
		0x01	AR add provider or consumer failed
		0x02	AR alarm-open failed
		0x03	AR alarm-ack-send
		0x04	AR alarm-send
		0x05	AR alarm-ind
0x6B	CTLRPC	0x00	Invalid state
		0x01	CTLRPC signaled an error
0x6C	CTLPBE	0x00	Invalid state
		0x01	CTLPBE signaled an error
0xC8	CMSM	0x00	Invalid state
		0x01	CMSM signaled an error
0xCA	CMRDR	0x00	Invalid state
		0x01	CMRDR signaled an error
0xCC	CMWRR	0x00	Invalid state
		0x01	AR is not in state Primary. (Write not allowed)
		0x02	CMWRR signaled an error
0xCD	CMIO	0x00	Invalid state
		0x01	CMIO signaled an error

ErrCode1	Description	ErrCode2	Description / Use
0xCE	CMSU	0x00	Invalid state
		0x01	AR add provider or consumer failed
		0x02	AR alarm-open failed
		0x03	AR alarm-send
		0x04	AR alam-ack-send
		0x05	AR alarm-ind
0xD0	CMINA	0x00	Invalid state
		0x01	CMINA signaled an error
0xD1	CMPBE	0x00	Invalid state
		0x01	CMPBE signaled an error
0xD2	CMDMC	0x00	Invalid state
		0x01	CMDMC signaled an error
0xFD	Used by RTA for protocol error (RTA_ERR_CLS_PROTOCOL)	0x00	Reserved
		0x01	error within the coordination of sequence numbers (RTA_ERR_CODE_SEQ)
		0x02	instance closed (RTA_ERR_ABORT)
		0x03	AR out of memory (RTA_ERR_ABORT)
		0x04	AR add provider or consumer failed (RTA_ERR_ABORT)
		0x05	AR consumer DHT / WDT expired (RTA_ERR_ABORT)
		0x06	AR cmi timeout (RTA_ERR_ABORT)
		0x07	AR alarm-open failed (RTA_ERR_ABORT)
		0x08	AR alarm-send.cnf(-) (RTA_ERR_ABORT)
		0x09	AR alarm-ack- send.cnf(-) (RTA_ERR_ABORT)
		0x0A	AR alarm data too long (RTA_ERR_ABORT)
		0x0B	AR alarm.ind(err) (RTA_ERR_ABORT)
		0x0C	AR rpc-client call.cnf(-) (RTA_ERR_ABORT)
		0x0D	AR abort.req (RTA_ERR_ABORT)
		0x0E	AR re-run aborts existing (RTA_ERR_ABORT)
		0x0F	AR release.ind received (RTA_ERR_ABORT)

ErrCode1	Description	ErrCode2	Description / Use
0xFD (continued)	Used by RTA for protocol error (RTA_ERR_CLS_PROTOCOL)	0x10	AR device deactivated (RTA_ERR_ABORT)
		0x11	AR removed (RTA_ERR_ABORT)
		0x12	AR protocol violation (RTA_ERR_ABORT)
		0x13	AR name resolution error (RTA_ERR_ABORT)
		0x14	AR RPC-Bind error (RTA_ERR_ABORT)
		0x15	AR RPC-Connect error (RTA_ERR_ABORT)
		0x16	AR RPC-Read error (RTA_ERR_ABORT)
		0x17	AR RPC-Write error (RTA_ERR_ABORT)
		0x18	AR RPC-Control error (RTA_ERR_ABORT)
		0x19	AR forbidden pull or plug after check.rsp and before in- data.ind (RTA_ERR_ABORT)
		0x1A	AR AP removed (RTA_ERR_ABORT)
		0x1B	AR link down (RTA_ERR_ABORT)
		0x1C	AR could not register multicast-mac address (RTA_ERR_ABORT)
		0x1D	not synchronized (cannot start companion-ar) (RTA_ERR_ABORT)
		0x1E	wrong topology (cannot start companion-ar) (RTA_ERR_ABORT)
		0x1F	dcp, station-name changed (RTA_ERR_ABORT)
0xFD (continued)	Used by RTA for protocol error (RTA_ERR_CLS_PROTOCOL)	0x20	dcp, reset to factory-settings (RTA_ERR_ABORT)
		0x21	cannot start companion-AR because a 0x8ipp submodule in the first AR... (RTA_ERR_ABORT)
		0x22	no irdata record yet (RTA_ERR_ABORT)
		0x23	PDEV (RTA_ERROR_ABORT)
		0x24	PDEV, no port offers required speed / duplexity (RTA_ERR_ABORT)
		0x25	IP-suite [of the IOC] changed by means of DCP set(IPParameter) or local engineering (RTA_ERR_ABORT)
		0x26	IOCARSR RDHT expired (RTA_ERROR_ABORT)
		0xC9	AR removed by reason of watchdog timeout in the Application task (RTA_ERROR_ABORT)
		0xCA	AR removed by reason of pool underflow in the Application task (RTA_ERROR_ABORT)
		0xCB	AR removed by reason of unsuccessful packet sending (Queue) inside OS in the Application task (RTA_ERROR_ABORT)
		0xCC	AR removed by reason of unsuccessful memory allocation in the Application task (RTA_ERROR_ABORT)
0xFF	User specific	0x00-0xFE	User specific
		0xFF	Recommended for “User abort” without further detail

Table 214: Coding of ErrorCode1 for ErrorDecode = PNIO (Excluding reserved values)

9.4.3.3 ErrorCode1 and ErrorCode2 for ErrorDecode is manufacturer specific

If ErrorDecode is set to manufacturer specific, the values of the fields ErrorCode1 and ErrorCode2 could be freely chosen.

10 Coding of Diagnosis

The following tables represent diagnosis specific object and structure defined in PROFINET specification (IEC 61158-6-10).

Coding of Channel Properties

Bit No	Description
D0 - D7	Data type of this channel (see table *Coding of Diagnosis* [▸ page 319])
D8	Accumulative. It should be set if the diagnosis is accumulated from several channels
D9 – D10	Maintenance (see table *Coding of Diagnosis* [▸ page 319])
D11 – D12	Specifier. It will be handled by the Stack and shall not be set by application.
D13 - D15	Direction (see table *Coding of Diagnosis* [▸ page 319])

Table 215: Coding of the field Channel Properties

Coding of Channel Properties.Data type

Value (Hexadecimal)	Description
0x00	Should be used if ChannelNumber is 0x8000 or If none of the below defined types are appropriate
0x01	1 Bit
0x02	2 Bits
0x03	4 Bits
0x04	8 Bits
0x05	16 Bits
0x06	32 Bits
0x07	64 Bits
0x07 – 0xFF	Reserved

Table 216: Coding of the field data type in field Channel Properties

Coding of Channel Properties.Maintenance

Value (Hexadecimal)	Description
0x00	Diagnosis
0x01	Maintenance Required
0x02	Maintenance Demanded
0x03	Qualified Diagnosis

Table 217: Coding of the field Maintenance in field Channel Properties

Coding of Channel Properties.Direction

Value (Hexadecimal)	Description
0x00	Manufacturer specific
0x01	Input
0x02	Output
0x03	Input / Output
0x04 – 0xFF	Reserved

Table 218: Coding of the field Direction in field Channel Properties

Coding of Channel error type

Value (Hexadecimal)	Description
0x0000	Reserved
0x0001	Short circuit
0x0002	Undervoltage

0x0003	Overvoltage
0x0004	Overload
0x0005	Overtemperature
0x0006	Line break
0x0007	Upper limit value exceeded
0x0008	Lower limit value exceeded
0x0009	Error
0x000A	Simulation active
0x000B – 0x000E	Reserved
0x000F	Manufacturer specific, recommended for “parameterization missing”
0x0010	Manufacturer specific, recommended for “parameterization fault”
0x0011	Manufacturer specific, recommended for “power supply fault”
0x0012	Manufacturer specific, recommended for “fuse blown / open”
0x0013	Manufacturer specific, recommended for “communication fault”
0x0014	Manufacturer specific, recommended for “ground fault”
0x0015	Manufacturer specific, recommended for “reference point lost”
0x0016	Manufacturer specific, recommended for “process event lost / sampling error”
0x0017	Manufacturer specific, recommended for “threshold warning”
0x0018	Manufacturer specific, recommended for “output disabled”
0x0019	Manufacturer specific, recommended for “safety event”
0x001A	Manufacturer specific, recommended for “external fault”
0x001B - 0x001E	Manufacturer specific
0x001F	Manufacturer specific, recommended for “temporary fault”
0x0020 - 0x00FF	Reserved for common profiles (assigned by PROFIBUS International)
0x0100 - 0x7FFF	Manufacturer specific
0x8000	Data transmission impossible
0x8001	Remote mismatch
0x8002	Media redundancy mismatch
0x8003	Sync mismatch
0x8004	Isochronous mode mismatch
0x8005	Multicast CR mismatch
0x8006	Reserved
0x8007	Fiber optic mismatch
0x8008	Network component function mismatch
0x8009	Time mismatch
0x800A	Dynamic frame packing function mismatch
0x800B	Media redundancy with planned duplication mismatch
0x800C	System redundancy mismatch
0x800D	Multiple interface mismatch
0x800E	Nested diagnosis indication
0x800F - 0x8FFF	Reserved
0x9000 - 0x9FFF	Reserved for profiles
0xA000 - 0xFFFF	Reserved

Table 219: Coding of Channel error type.

Note: values 0x8000 – 0x800E will be triggered by the Stack internally and shall not be set by application

Coding of Extended channel error type

The value of this field depends on the field Channel error type. The values shall be set according to following tables:

Value (hexadecimal)	Description	Usage
0x0000	Reserved	
0x0001 - 0x7FFF	Manufacturer specific	Alarm / Diagnosis
0x8000	Accumulative info	Alarm / Diagnosis
0x8001 - 0x8FFF	Reserved	
0x9000 - 0x9FFF	Profile specific error codes	Alarm / Diagnosis
0xA000 - 0xFFFF	Reserved	

Table 220: Coding of Extended channel error type for Channel Error Type 1 - 0x7FFF

In conjunction with profile specific error types the extended channel error type will be set according to following tabels:

Coding usExtChannelErrType in conjunction with Data transmission impossible error type

Value	Description
0x8000	Link State Mismatch – Link down
0x8001	MAUType Mismatch
0x8003	Line delay mismatch

Table 221: Coding usExtChannelErrType in conjunction with Data transmission impossible error type

Coding of Extended error Type in conjunction with Remote Mismatch error type

Value	Description
0x8000	Peer Chassis ID mismatch
0x8001	Peer Port ID mismatch
0x8002	RT Class 3 mismatch
0x8003	Peer MAUType mismatch
0x8004	Peer MRP-Domain ID mismatch
0x8005	No peer detected
0x8007	Peer CableDelay mismatch
0x8008	Peer PTCP mismatch
Other	Reserved

Table 222: Coding of Extended error Type in conjunction with Remote Mismatch error type

Coding of Extended error Type in conjunction with Sync Mismatch error type

Value	Description
0x8000	No Sync Message Received
0x8001	Jitter out of Boundary”
Other	Reserved

Table 223: Coding of Extended error Type in conjunction with Sync Mismatch error type

Coding of Extended error Type in conjunction with Fiber optic mismatch error type

Value	Description
0x8000	Power budget mismatch
Other	Reserved

Table 224: Coding of Extended error Type in conjunction with Fiber optic mismatch error type

11 Appendix

11.1 Name encoding

The name is an OctetString with 1 to 240 octets. A name can contain one or more labels separated by a dot [.].

The definition of IETF RFC 5890 and the following syntax applies:

- 1 or more labels, separated by [.]
- Total length is 1 to 240
- Label length is 1 to 63
- Labels consist of [a-z0-9-]
- Labels do not start with [-]
- Labels do not end with [-]
- Labels do not use multiple concatenated [-] except for IETF RFC 5890
- The first label does not have the form “port-xyz” or “port-xyz-abcde” with a, b, c, d, e, x, y, z = 0..9, to avoid wrong similarity with the field AliasNameValue
- Station names do not have the form a.b.c.d with a, b, c, d = 0...999

11.2 Legal notes

Copyright

© Hilscher Gesellschaft für Systemautomation mbH

All rights reserved.

The images, photographs and texts in the accompanying materials (in the form of a user's manual, operator's manual, Statement of Work document and all other document types, support texts, documentation, etc.) are protected by German and international copyright and by international trade and protective provisions. Without the prior written consent, you do not have permission to duplicate them either in full or in part using technical or mechanical methods (print, photocopy or any other method), to edit them using electronic systems or to transfer them. You are not permitted to make changes to copyright notices, markings, trademarks or ownership declarations. Illustrations are provided without taking the patent situation into account. Any company names and product designations provided in this document may be brands or trademarks by the corresponding owner and may be protected under trademark, brand or patent law. Any form of further use shall require the express consent from the relevant owner of the rights.

Important notes

Utmost care was/is given in the preparation of the documentation at hand consisting of a user's manual, operating manual and any other document type and accompanying texts. However, errors cannot be ruled out. Therefore, we cannot assume any guarantee or legal responsibility for erroneous information or liability of any kind. You are hereby made aware that descriptions found in the user's manual, the accompanying texts and the documentation neither represent a guarantee nor any indication on proper use as stipulated in the agreement or a promised attribute. It cannot be ruled out that the user's manual, the accompanying texts and the documentation do not completely match the described attributes, standards or any other data for the delivered product. A warranty or guarantee with respect to the correctness or accuracy of the information is not assumed.

We reserve the right to modify our products and the specifications for such as well as the corresponding documentation in the form of a user's manual, operating manual and/or any other document types and accompanying texts at any time and without notice without being required to notify of said modification. Changes shall be taken into account in future manuals and do not represent an obligation of any kind, in particular there shall be no right to have delivered documents revised. The manual delivered with the product shall apply.

Under no circumstances shall Hilscher Gesellschaft für Systemautomation mbH be liable for direct, indirect, ancillary or subsequent damage, or for any loss of income, which may arise after use of the information contained herein.

Liability disclaimer

The hardware and/or software was created and tested by Hilscher Gesellschaft für Systemautomation mbH with utmost care and is made available as is. No warranty can be assumed for the performance or flawlessness of the hardware and/or software under all application conditions and scenarios and the work results achieved by the user when using the hardware and/or software. Liability for any damage that may have occurred as a result of using the hardware and/or software or the corresponding documents shall be limited to an event involving willful intent or a grossly negligent violation of a fundamental contractual obligation. However, the right to assert damages due to a violation of a fundamental contractual obligation shall be limited to contract-typical foreseeable damage.

It is hereby expressly agreed upon in particular that any use or utilization of the hardware and/or software in connection with

- Flight control systems in aviation and aerospace;
- Nuclear fusion processes in nuclear power plants;
- Medical devices used for life support and
- Vehicle control systems used in passenger transport

shall be excluded. Use of the hardware and/or software in any of the following areas is strictly prohibited:

- For military purposes or in weaponry;
- For designing, engineering, maintaining or operating nuclear systems;
- In flight safety systems, aviation and flight telecommunications systems;
- In life-support systems;
- In systems in which any malfunction in the hardware and/or software may result in physical injuries or fatalities.

You are hereby made aware that the hardware and/or software was not created for use in hazardous environments, which require fail-safe control mechanisms. Use of the hardware and/or software in this kind of environment shall be at your own risk; any liability for damage or loss due to impermissible use shall be excluded.

Warranty

Hilscher Gesellschaft für Systemautomation mbH hereby guarantees that the software shall run without errors in accordance with the requirements listed in the specifications and that there were no defects on the date of acceptance. The warranty period shall be 12 months commencing as of the date of acceptance or purchase (with express declaration or implied, by customer's conclusive behavior, e.g. putting into operation permanently).

The warranty obligation for equipment (hardware) we produce is 36 months, calculated as of the date of delivery ex works. The aforementioned provisions shall not apply if longer warranty periods are mandatory by law pursuant to Section 438 (1.2) BGB, Section 479 (1) BGB and Section 634a (1) BGB [Bürgerliches Gesetzbuch; German Civil Code] If, despite of all due care taken, the delivered product should have a defect, which already existed at the time of the transfer of risk, it shall be at our discretion to either repair the product or to deliver a replacement product, subject to timely notification of defect.

The warranty obligation shall not apply if the notification of defect is not asserted promptly, if the purchaser or third party has tampered with the products, if the defect is the result of natural wear, was caused by unfavorable operating conditions or is due to violations against our operating regulations or against rules of good electrical engineering practice, or if our request to return the defective object is not promptly complied with.

Costs of support, maintenance, customization and product care

Please be advised that any subsequent improvement shall only be free of charge if a defect is found. Any form of technical support, maintenance and customization is not a warranty service, but instead shall be charged extra.

Additional guarantees

Although the hardware and software was developed and tested in-depth with greatest care, Hilscher Gesellschaft für Systemautomation mbH shall not assume any guarantee for the suitability thereof for any purpose that was not confirmed in writing. No guarantee can be granted whereby the hardware and software satisfies your requirements, or the use of the hardware and/or software is uninterruptable or the hardware and/or software is fault-free.

It cannot be guaranteed that patents and/or ownership privileges have not been infringed upon or violated or that the products are free from third-party influence. No additional guarantees or promises shall be made as to whether the product is market current, free from deficiency in title, or can be integrated or is usable for specific purposes, unless such guarantees or promises are required under existing law and cannot be restricted.

Confidentiality

The customer hereby expressly acknowledges that this document contains trade secrets, information protected by copyright and other patent and ownership privileges as well as any related rights of Hilscher Gesellschaft für Systemautomation mbH. The customer agrees to treat as confidential all of the information made available to customer by Hilscher Gesellschaft für Systemautomation mbH and rights, which were disclosed by Hilscher Gesellschaft für Systemautomation mbH and that were made accessible as well as the terms and conditions of this agreement itself.

The parties hereby agree to one another that the information that each party receives from the other party respectively is and shall remain the intellectual property of said other party, unless provided for otherwise in a contractual agreement.

The customer must not allow any third party to become knowledgeable of this expertise and shall only provide knowledge thereof to authorized users as appropriate and necessary. Companies associated with the customer shall not be deemed third parties. The customer must obligate authorized users to confidentiality. The customer should only use the confidential information in connection with the performances specified in this agreement.

The customer must not use this confidential information to his own advantage or for his own purposes or rather to the advantage or for the purpose of a third party, nor must it be used for commercial purposes and this confidential information must only be used to the extent provided for in this agreement or otherwise to the extent as expressly authorized by the disclosing party in written form. The customer has the right, subject to the obligation to confidentiality, to disclose the terms and conditions of this agreement directly to his legal and financial consultants as would be required for the customer's normal business operation.

Export provisions

The delivered product (including technical data) is subject to the legal export and/or import laws as well as any associated regulations of various countries, especially such laws applicable in Germany and in the United States. The products / hardware / software must not be exported into such countries for which export is prohibited under US American export control laws and its supplementary provisions. You hereby agree to strictly follow the regulations and to yourself be responsible for observing them. You are hereby made aware that you may be required to obtain governmental approval to export, reexport or import the product.

Terms and conditions

Please read the notes about additional legal aspects on our netIOT web site under http://www.netiot.com/netiot/netiot-edge/terms-and-conditions/.

11.3 Third party software licenses

SNMP

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

/**

Copyright 1988, 1989 by Carnegie Mellon University

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its

documentation for any purpose and without fee is hereby granted,

provided that the above copyright notice appear in all copies and that

both that copyright notice and this permission notice appear in

supporting documentation, and that the name of CMU not be

used in advertising or publicity pertaining to distribution of the

software without specific, written prior permission.

CMU DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING

ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL

CMU BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR

ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,

WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,

ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS

SOFTWARE.

**/

List of figures

List of tables

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

Support

Phone: +49 (0) 6190 9907-99
E-mail: de.support@hilscher.com

SUBSIDIARIES

China

Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-mail: info@hilscher.cn

Support

Phone: +86 (0) 21-6355-5161
E-mail: cn.support@hilscher.com

France

Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-mail: info@hilscher.fr

Support

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

India

Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
E-mail: info@hilscher.in

Italy

Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-mail: info@hilscher.it

Support

Phone: +39 02 25007068
E-mail: it.support@hilscher.com

Japan

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

Support

Phone: +81 (0) 3-5362-0521
E-mail: jp.support@hilscher.com

Korea

Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-mail: info@hilscher.kr

Switzerland

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

Support

Phone: +49 (0) 6190 9907-99
E-mail: ch.support@hilscher.com

USA

Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-mail: info@hilscher.us

Support

Phone: +1 630-505-5301
E-mail: us.support@hilscher.com

© Hilscher 2011 – 2019