SIEMENS

SIMATIC NET

PROFINET
I-device user programming interface

Programming Manual

10/2018

C79000-G8976-C378-05

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER

indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING

indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION

indicates that minor personal injury can result if proper precautions are not taken.

NOTICE

indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ WARNING

Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Division Process Industries and Drives
Postfach 48 48
90026 NÜRNBERG
GERMANY

C79000-G8976-C378-05
Ⓟ 11/2018 Subject to change

Preface

SIMATIC NET – I-device user programming interface

This manual will provide you with a sound basis for your user programs in the C/C++ programming language.

Are you a beginner? Then you can familiarize yourself systematically. Start with the overview of the I-device user programming interface. There you will find everything you need to know about the principles and range of functions of the interface.

Are you a professional programmer? Then you can get going straight away. The step sequences in the Quick Start and the comprehensive references will show you the shortest way to create your I-device user program.

Do you find examples useful? The supplied sample programs will provide you with a flexible basis with which you can put your own ideas into practice.

Purpose of the manual

Use this manual when you want to implement an I-device with the I-device user programming interface.

Overview of the documentation

We recommend that you familiarize yourself with the following documentation before you create an I-device user program:

Name of the document	Why you should read it
Manual PROFINET system description	This provides you with the basics of the topics involved in PROFINET IO: Network components, data exchange and communication, PROFINET IO, Component Based Automation, application example PROFINET IO and Component Based Automation.
Getting Started PROFINET IO	Documents of the class "Getting Started" guide you through the steps in commissioning through to a functioning application based on concrete examples.
Manual From PROFIBUS DP to PROFINET IO	Read this document if you already have a PROFIBUS system installed and want to convert to a PROFINET system.
Readme file • for the "SIMATIC NET PC Software" DVD • for the CDs "DK-16xx PN IO" and "DK HN-IE PN IO"	Here, you will find the latest information on the SIMATIC NET PC software products.
Installation manual for the "SIMATIC NET PC Software" DVD	You are guided step by step through the installation of the SIMATIC NET products on a PC (SOFTNET IE PN IO only).

Name of the document	Why you should read it
Manual Commissioning PC Stations	This provides you with the information you require for commissioning and configuring a PC as a PROFINET IO controller.
Manual IO-Base user programming interface	The reference manual when you create an IO-Base controller or IO-Base device user program. The reference manual when you create an IO-Base controller program. The IO Device interface contained in it is only relevant in CP 1616 or CP 1604 products up to and including version V2.6.
Manual Industrial Communication with PG/PC	This manual introduces you to industrial communication and explains the available communications protocols.
Manual SIMATIC NET - Twisted Pair and Fiber Optic Networks	Configure and set up your Industrial Ethernet networks with the aid of this document.

Required basic experience

We recommend that you have the following experience as a programmer of user programs:

- Experience of programming in C/C++
- Programming techniques:
  - Multithreading techniques
  - Callback routines
- Knowledge of the technical terminology in English
- Knowledge of the PROFINET IO system
- General experience in the area of automation engineering
- Basic knowledge of the configuration tool "STEP 7 Professional (TIA Portal)"

Validity of the manual

The manual describes the I-device user programming interface that is a component of several products:

- CP 1626
- DK-16xx PN IO as of V2.7

Certification

The products and systems listed in this document are manufactured and marketed using a quality management system complying with DIN ISO 9001 and certified by DQS. The DQS certificate is recognized in all IQNet countries (certificate register no. 2613).

Trademarks

The following and possibly other names not identified by the registered trademark sign ® are registered trademarks of Siemens AG:

SIMATIC NET, HARDNET, SOFTNET, CP 1612, CP 1613, CP 5612, CP 5613, CP 5614, CP 5622

Industry Online Support

In addition to the product documentation, the comprehensive online information platform of Siemens Industry Online Support at the following Internet address:
Link: (https://support.industry.siemens.com/cs/de/en/)

Apart from news, there you will also find:

- Project information: Manuals, FAQs, downloads, application examples etc.
- Contacts, Technical Forum
- The option submitting a support query:
  Link: (https://support.industry.siemens.com/My/ww/en/requests)
- Our service offer:

  Right across our products and systems, we provide numerous services that support you in every phase of the life of your machine or system - from planning and implementation to commissioning, through to maintenance and modernization.

You will find contact data on the Internet at the following address:
Link: (http://www.automation.siemens.com/aspa_app/?ci=yes&lang=en)

SITRAIN - Training for Industry

The training offer includes more than 300 courses on basic topics, extended knowledge and special knowledge as well as advanced training for individual sectors - available at more than 130 locations. Courses can also be organized individually and held locally at your location.

You will find detailed information on the training curriculum and how to contact our customer consultants at the following Internet address:

Link: (http://sitrain.automation.siemens.com/DE/sitrain/default.aspx?AppLang=en)

Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, systems, machines and networks.

In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.

Additionally, Siemens' guidance on appropriate security measures should be taken into account. For additional information on industrial security measures that may be implemented, please visit
(http://www.siemens.com/industrialsecurity)

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customers' exposure to cyber threats.

To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed under
(https://support.industry.siemens.com/cs/ww/en/ps/15247/pm)

SIMATIC NET glossary

Explanations of many of the specialist terms used in this documentation can be found in the SIMATIC NET glossary.

You will find the SIMATIC NET glossary on the Internet at the following address:

50305045 (http://support.automation.siemens.com/WW/view/en/50305045)

Recycling and disposal

The products are low in harmful substances, can be recycled and meet the requirements of the Directive 2012/19/EU for disposal of waste electrical and electronic equipment (WEEE).

Do not dispose of the products at public disposal sites.

For environmentally compliant recycling and disposal of your electronic waste, please contact a company certified for the disposal of electronic waste or your Siemens representative.

Note the different national regulations.

Table of contents

Overview of the real-time modes 1

1.1 Overview of the real-time modes

Below, you will find an overview of the real-time modes and a general description of the data traffic involved.

The I-device user programming interface supports the following modes:

- RT (Real Time)
- IRT (Isochronous Real Time) with the option of "high performance"

1.2 Real-time and isochronous real-time modes

Overview

The PROFINET standard distinguishes between the real-time (RT) and the isochronous real-time (IRT) modes.

RT mode

In real-time mode (RT), real-time frames (RT data) are transferred with high priority.

IRT mode

IRT with the option "high performance"

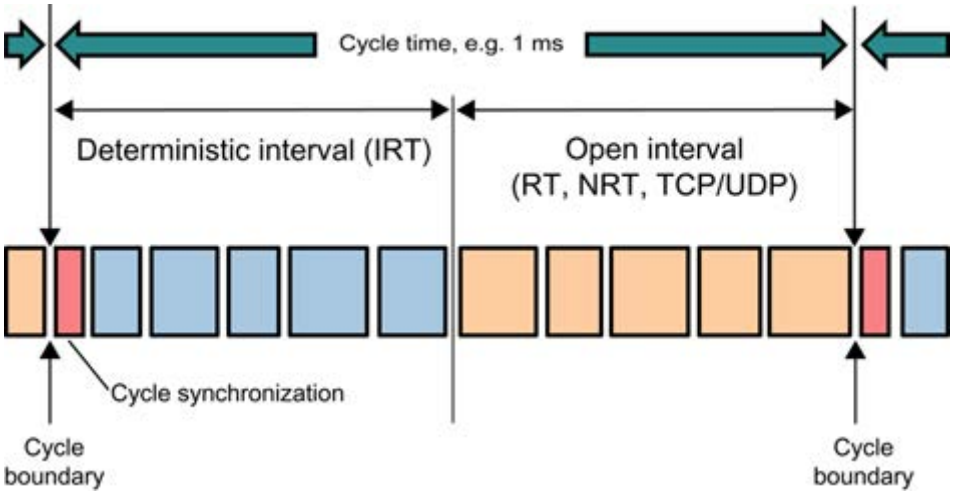
In IRT mode with high performance, real-time frames (IRT data) is transferred cyclically with high priority during a fixed period (deterministic interval). The end of this transfer interval is signaled to the I-device user program. This signal allows the I-device user program to be linked to the bus clock. During the open interval, RT, NRT (Non Real Time) and TCP/UDP frames are transferred.

In this mode, the I-device user programming interface provides other functions that need to be called in the initialization, operational and termination phases. The times and periods are specified in the configuration tool and transferred to the I-devices.

The topology needs to be configured in STEP 7 Professional (TIA Portal).

Note

In the following pages, the IRT mode with the "high Performance" option is called "IRT".



Quick start with I-device functionality 2

This section recommends a step-by-step procedure for creating a user program in the C/C++ programming language based on the I-device user programming interface.

In the first step, familiarize yourself with the basics of PROFINET IO. Gradually work through the steps and complete them by creating I-device user program.

2.1 Procedure

By following the steps below, you can create an I-device user program quickly and effectively:

Step	Description
1	Get to know the basics of PROFINET. You can do this by reading the "PROFINET System Description" manual.
2	Familiarize yourself with the essential properties of the I-device user programming interface for the I-device. You can do this by reading the section "Overview of the I-device user programming interface for I-devices (Page 15)" in this manual.
3	Familiarize yourself with the following files so that you know what support is already available for the following steps and how you can use this support. These are as follows: • Readme files with additional information and the latest modifications. • C header files of the I-device user programming interface: – "pniobase.h" contains functions and data structures used by the device and controller interfaces. – "pniousrd.h" contains the functions and data structures – "pnioerrx.h" contains all the error and return codes • The library "pniolib.lib" for linking to your I-device user program. • Sample programs and corresponding databases.
4	Work through the source text of the sample program "dev_rw_digital_io" in the "../examples/device/" subdirectory and check out the functions and data structures in the section "Overview of the I-device user programming interface for I-devices (Page 15)".
5	From your system configuration and your GSDML file find which data is to be sent and received (IO data, data records, alarms) and which I-devices are involved.
6	Complete the STEP 7 configuration and download it to the devices involved.
7	Modify the supplied sample program "dev_rw_digital_io" so that it can run on your system. In this way, you will get to know important techniques and no longer need to develop them yourself. Compile and link the modified sample program and test it on the system you have available.
8	Now create your own I-device user program.

2.2 Overview of the functionality of the products

Description

The table below provides you with an overview of the I-device functionality that can be used in the various SIMATIC NET products. All functions whose names end with “sync” are synchronous calls that do not require acknowledgment. All functions that end with “async” are asynchronous and are completed by calling an appropriate acknowledgment function or a callback.

Function groups and names	SIMATIC NET products	
	SOFTNET PN IO (RT)	CP 1626 (RT + IRT) with DK HN-IE PNIO and CP 1616 as of V2.7
Management functions		
PNIOD_init_open_sync()	–	x
PNIOD_open_sync()	–	x
PNIOD_close_sync()	–	x
PNIOD_start_sync()	–	x
PNIOD_stop_async()	–	x
PNIOD_CBF_SYNC_STOPPED	–	x
PNIOD_CBF_SYNC_CP_STOP_REQ	–	x
Interface for I-device configuration		
PNIOD_get_config_sync()	–	x
Interface for write IO data with the I-device		
PNIOD_trigger_data_write_sync()	–	x
Callback function PNIO_CBF_DATA_WRITE	–	x
Interface for read IO data with the I-device		
PNIOD_trigger_data_read_sync()	–	x
Callback function PNIO_CBF_DATA_READ	–	x
Interface for IRT mode with the I-device		
PNIO_CP_register_cbf()	–	x
Callback function PNIO_CP_CBE_APPL_START_IND	–	x
PNIO_CP_set_opdone()	–	x
Callback function PNIO_CP_CBE_APPL_FAULT_IND	–	x
Callback function PNIO_CP_CBE_BUSCYCLE_IND	–	x
Callback function PNIO_CP_CBE_PRM	–	x
Interface for read/write data record with the I-device		
Callback function PNIOD_CBF_ASYNC_REC_READ	–	x
PNIOD_rec_read_async_rsp	–	x
Callback function PNIOD_CBF_ASYNC_REC_WRITE	–	x
PNIOD_rec_write_async_rsp	–	x
Alarm and diagnostics interface for the I-device		
PNIOD_build_chan_prop_sync()	–	x

Function groups and names	SIMATIC NET products	
PNIOD_init_channel_add_sync()	–	x
PNIOD_diag_channel_add_sync()	–	x
PNIOD_init_generic_add_sync()	–	x
PNIOD_diag_generic_add_sync()	–	x
PNIOD_diag_remove_sync()	–	x
PNIOD_init_alarm_send_async()	–	x
PNIOD_alarm_send_async()	–	x
PNIOD_CBF_SYNC_ALARM_DONE	–	x
Interface for connection establishment and termination of the I-device		
Callback function PNIOD_CBF_ASYNC_CONNECT_IND	–	x
PNIOD_connect_async_rsp	–	x
Callback function PNIOD_CBF_ASYNC_OWNERSHIP_IND	–	x
PNIOD_ownership_async_rsp	–	x
Callback function PNIOD_CBF_ASYNC_PRM_END_IND	–	x
PNIOD_prm_end_async_rsp	–	x
Callback function PNIOD_CBF_ASYNC_IRT_INIT_INPUTS	–	x
PNIOD_irt_init_inputs_async_rsp	–	x
Callback function PNIOD_CBF_ASYNC_INDATA_IND	–	x
PNIOD_indata_async_rsp	–	x
Callback function PNIOD_CBF_SYNC_DISCONNECT_IND	–	x
Callback function PNIOD_CBF_SYNC_DATA_STATUS_IND	–	x

Overview of the I-device user programming interface for I-devices

3

This chapter explains the basic characteristics of the I-device user programming interface to prepare you for creating your own I-device user program.

Function calls and data access are described in detail in the Section "Description of the functions and data types for I-devices (Page 25)".

Note

The IO data directions are always described from the perspective of the higher-level IO controller. As a result, an I-device writes input data (data is sent to the IO controller) and reads output data (data sent by the IO controller).

This convention is used in the entire description of the I-device user programming interface for the I-device.

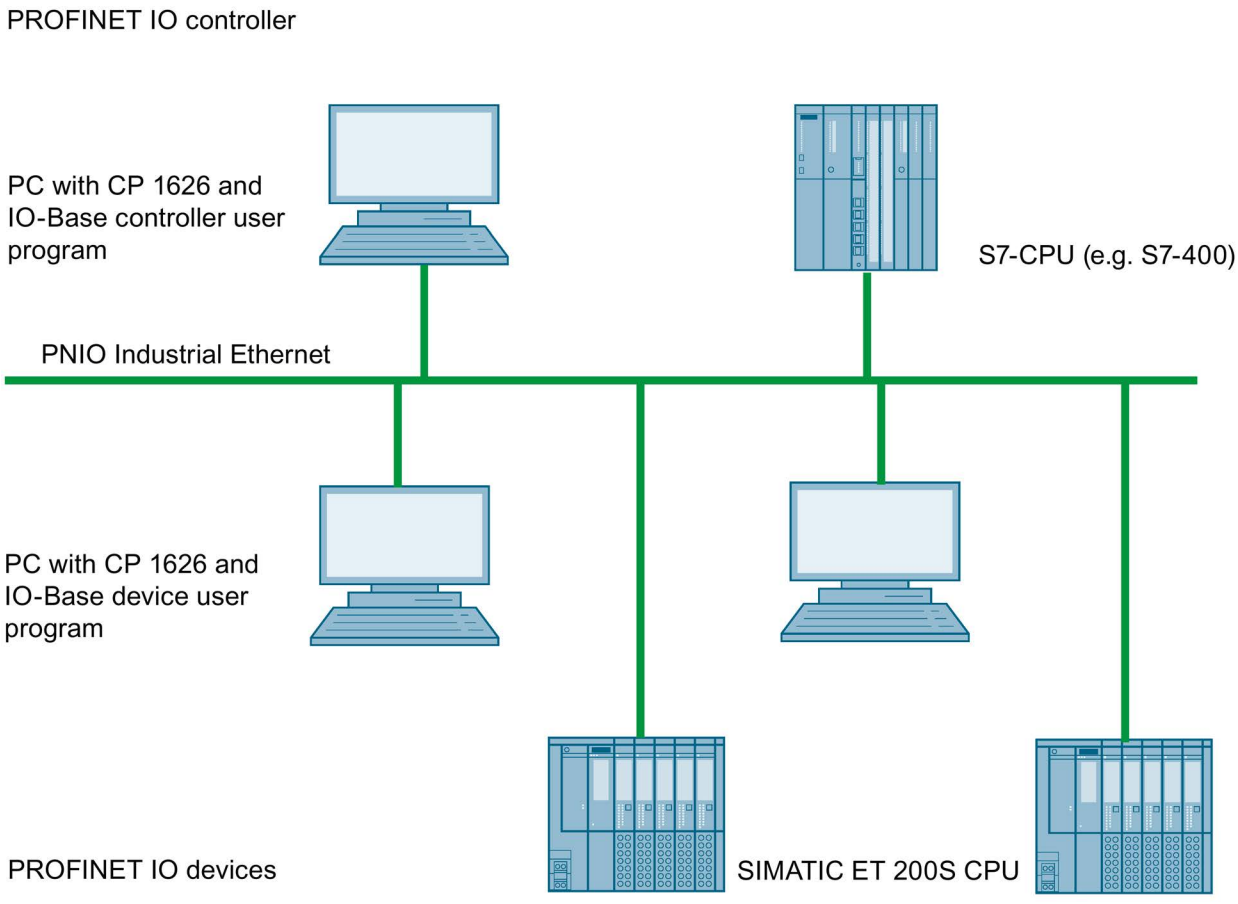
3.1 Typical use of the I-device user programming interface

Description

The following schematic illustrates a typical application: A PC communicates with a PROFINET IO controller over Industrial Ethernet.

The I-device user program runs on the PC. Data traffic is handled over a PROFINET IO communications processor with a SIMATIC S7 PROFINET CPU (PLC) or alternatively with a PC with a controller user program over Industrial Ethernet.

The supplied sample program relates to this configuration.



3.2 Software architecture on a PC with PROFINET IO

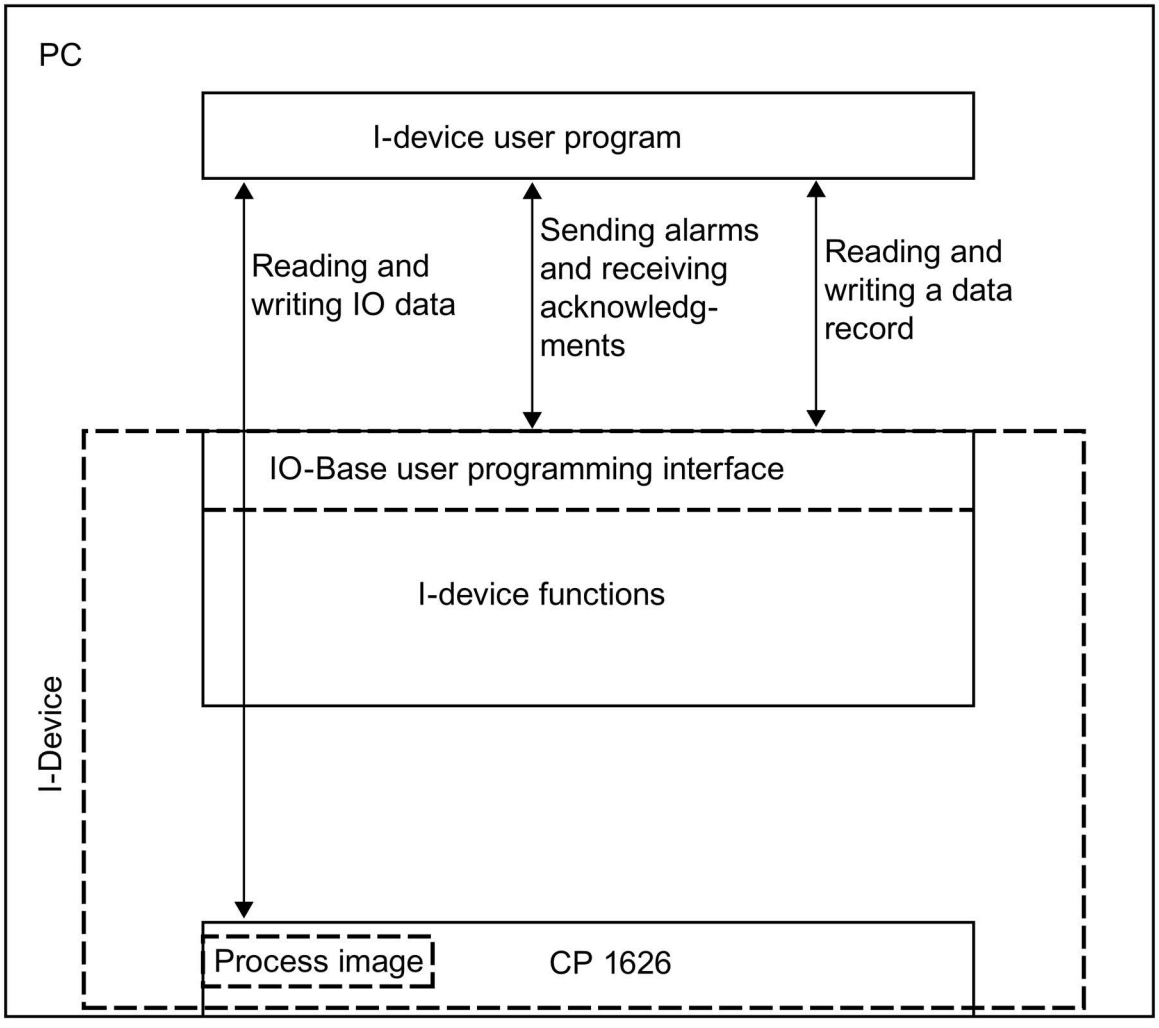
Description

With the I-device user programming interface, PROFINET IO provides all the functions that a C or C++ user program requires to communicate with PROFINET IO devices.

These are the following I-device functions:

- Read/write IO data
- Sending alarms and receiving alarm acknowledgments
- Read/write data record

The PROFINET IO software is connected to the process over a PROFINET IO communications processor. The communication with the IO controller is configured with STEP 7 Professional (TIA Portal).



Header files, libraries and sample programs

The following files are required for I-device support:

File type	File name	Purpose
Header file	pniobase.h	Contains global structures and definitions for IO controller and I-device.
Header file	pniousrd.h	Contains data types, constants, and function declarations. This header file is embedded in pniourd_host.h. The declarations contained from pniourd.h may only be used via pniousrd_host.h.
Header file	pnioerrx.h	Contains the error codes.
Library	libpniousr	For linking to your I-device user program.

3.3 Typical sequence of an I-device user program

Overview

The typical sequence of an I-device user program can be divided into 3 phases.

- Initialization phase
- Productive operation
- Completion phase

3.3.1 Initialization phase

Description

The initialization phase and connection establishment is divided into steps. A distinction must be made between function calls activated by the I-device user program and callback calls activated by the PNIO library.

Note

Unless otherwise specified, the only callback functions that may be used are the I-device functions specified in the following table in the "Purpose" column.

Initialization phase with RT

Step	Action	Purpose
1	PNIOD_init_open_sync()	Prepare parameters for the function PNIOD_open_sync
2	PNIOD_open_sync()	• Logging into the CP • Register callback functions
3	PNIOD_get_config_sync()	Read out the configuration of the I-device (e.g. module structure)
4	PNIOD_start_sync()	Activate I-device interface for connection establishment by the IO controller
5	Wait for PNIOD_CBF_ASYNC_CONNECT_IND	Wait for connection establishment by the IO controller
6	PNIOD_CBF_ASYNC_CONNECT_IND	I-device interface calls up this callback when an IO controller wants to establish the connection.
7	PNIOD_connect_async_rsp	Acknowledgment of the connection establishment request
8	Wait for PNIOD_CBF_ASYNC_OWNERSHIP_IND	-
9	PNIOD_CBF_ASYNC_OWNERSHIP_IND	The callback indicates that one or more submodules were assigned to an IO controller or this assignment was canceled. Assignment means that the IO controller can exchange data with this module.
10	PNIOD_ownership_async_rsp	Acknowledgment of the submodule assignment
11	Wait for PNIOD_CBF_ASYNC_PRM_END_IND	-
12	PNIOD_CBF_ASYNC_PRM_END_IND	The callback function indicates that the parameter assignment of one or more submodules by the IO controller is completed.
13	PNIOD_prm_end_async_rsp	Acknowledgment of the previous callback
14	Wait for PNIOD_CBF_ASYNC_INDATA_IND	-
15	PNIOD_CBF_ASYNC_INDATA_IND	The callback function indicates that cyclic data exchange has started.
16	PNIOD_indata_async_rsp	Acknowledgment of the previous callback

Initialization phase with IRT

Step	Action	Purpose
1	PNIOD_init_open_sync()	Prepare parameters for the function PNIOD_open_sync
2	PNIOD_open_sync()	• Logging into the CP • Register callback functions
3	PNIOD_get_config_sync()	Read out the configuration of the I-device (e.g. module structure)
4	PNIO_CP_register_cbf()	Register the callbacks "PNIO_CP_CBE_APPL_FAULT_IND" and "PNIO_CP_CBE_APPL_START_IND"
5	PNIOD_start_sync()	Activate I-device interface for connection establishment by the IO controller
6	Wait for PNIOD_CBF_ASYNC_CONNECT_IND	Wait for connection establishment by the IO controller
7	PNIOD_CBF_ASYNC_CONNECT_IND	I-device interface calls up this callback when an IO controller wants to establish the connection.
8	PNIOD_connect_async_rsp	Acknowledgment of the connection establishment request
9	Wait for PNIOD_CBF_ASYNC_OWNERSHIP_IND	-
10	PNIOD_CBF_ASYNC_OWNERSHIP_IND	The callback indicates that one or more submodules were assigned to an IO controller or this assignment was canceled. Assignment means that the IO controller can exchange data with this module.
11	PNIOD_ownership_async_rsp	Acknowledgment of the submodule assignment
12	Wait for PNIOD_CBF_ASYNC_PRM_END_IND	-
13	PNIOD_CBF_ASYNC_PRM_END_IND	The callback function indicates that the parameter assignment of one or more submodules by the IO controller is completed.
14	PNIOD_prm_end_async_rsp	Acknowledgment of the previous callback
15	PNIOD_CBF_ASYNC_IRT_INIT_INPUTS	Initiation of the exchange of IO data
16	PNIOD_irt_init_inputs_async_rsp	Acknowledgment of the previous callback
17	Wait for PNIOD_CBF_ASYNC_INDATA_IND	-
18	PNIOD_CBF_ASYNC_INDATA_IND	The callback function indicates that cyclic data exchange has started.
19	PNIOD_indata_async_rsp	Acknowledgment of the previous callback

3.3.2 Productive operation

Overview

Data is exchanged with the IO controller in productive operation. This is as follows:

- Read/write IO data
- Send alarms and receive their acknowledgments
- Read/write data record

The data traffic is explained in detail below.

Read RT IO data

Reading IO data (output data from the perspective of the IO controller) is done in two steps:

Step	Action	Purpose
1	PNIOD_trigger_data_read_sync() (DevHndl, pAddr, PNIO_ACCESS_RT_WITH_LOCK)	The user program signals the read request to the I-device interface.
2	PNIO_CBF_DATA_READ	The I-device interface calls this callback for the specified or all submodules with output data and, among other things, transfers the pointer to a data buffer with the output data received from the IO controller to the user program.

Write RT IO data

Writing RT IO data (input data from the perspective of the IO controller) is done in two steps:

Step	Action	Purpose
1	PNIOD_trigger_data_write_sync() (DevHndl, pAddr, PNIO_ACCESS_RT_WITH_LOCK)	The user program signals the write request to the I-device interface.
2	PNIO_CBF_DATA_WRITE	The I-device interface calls up this callback for the specified submodule or every submodule with input data. It also transfers, among other things, the pointer to a data buffer to which the input data to be sent to the IO controller will be copied.

Reading and writing IRT IO data

Reading and writing IRT IO data involves six steps:

Step	Action	Purpose
1	PNIO_CP_CBE_APP_START_IND	With this event, the I-device interface signals the I-device user program the start of the time window for processing isochronous data.
2	PNIOD_trigger_data_read_sync() (DevHndl, pAddr, PNIO_ACCESS_IRT_WITHOUT_LOCK)	The I-device user program signals the read request to the I-device interface. This causes the I-device interface to execute step 3. This call returns only after step 3 has been processed.
3	PNIO_CBF_DATA_READ	The I-device interface calls up this callback for the specified submodule or every submodule with output data. A pointer is transferred as a parameter in the callback that contains the output data received from the IO controller.
4	PNIOD_trigger_data_write_sync() (DevHndl, pAddr, PNIO_ACCESS_IRT_WITHOUT_LOCK)	The I-device user program signals the write request to the I-device interface. This causes the I-device interface to execute step 5. This call returns only after step 5 has been processed.
5	PNIO_CBF_DATA_WRITE	The I-device interface calls up this callback for the specified submodule or every submodule with input data. The parameters of the callback contain a pointer into which the input data to be sent to an IO controller needs to be copied by the I-device user program.
6	PNIO_CP_set_opdone()	The I-device user program uses this call to inform the I-device interface of the end of the isochronous data processing.

Responding to a read data record job

Responding to a read data record job consists of two steps:

Step	Action	Purpose
1	Wait until "PNIOD_CBF_ASYNC_REC_READ" is called.	As soon as the I-device interface receives a read data record job from the IO controller, it calls the "PNIO_CBF_REC_READ" callback registered by the I-device user program.
2	Call "PNIOD_rec_read_async_rsp"	Acknowledgment of the previous callback

Responding to a write data record job

Responding to a write data record job consists of two steps:

Step	Action	Purpose
1	Wait until "PNIOD_CBF_ASYNC_REC_WRITE" is called.	As soon as the I-device interface receives a write data record job from the IO controller, it calls the "PNIO_CBF_REC_WRITE" callback registered by the I-device user program.
2	Call "PNIOD_rec_write_async_rsp"	Acknowledgment of the previous callback

Sending alarms and receiving alarm acknowledgments

Alarms are processed in five steps as described below:

Step	Action	Purpose
1	Form channel properties with "PNIOD_build_chan_prop_sync()"	Generate channel properties (only with incoming diagnostics)
2	"PNIOD_diag_channel_add_sync()" or "PNIOD_diag_generic_add_sync()" or "PNIOD_diag_remove_sync()"	Add or remove diagnostics (only with diagnostics alarms)
3	PNIOD_init_alarm_send_async()	Preparation of the parameters for step 4
4	PNIOD_alarm_send_async()	Send alarm
5	PNIOD_CBF_SYNC_ALARM_DONE	Wait for acknowledgment of the alarm

Callback events

When the connection is being established, information is made available by the I-device interface using callbacks.

The following table lists the signaled information and the corresponding callback name.

Callback name	Signaled Information
PNIOD_CBF_ASYNC_CONNECT_IND	"Connect request" arrived from the controller
PNIOD_CBF_ASYNC_OWNERSHIP_IND	Change of ownership of submodules
PNIOD_CBF_ASYNC_PRM_END_IND	End of parameter assignment of submodules
PNIOD_CBF_ASYNC_IRT_INIT_INPUTS	Initialize controller inputs
PNIOD_CBF_ASYNC_INDATA_IND	Start of the cyclic data exchange
PNIOD_CBF_SYNC_DISCONNECT_IND	AR abort
PNIOD_CBF_SYNC_DATA_STATUS_IND	Change to the APDU status
PNIOD_CBF_SYNC_CP_STOP_REQ	Reconfiguration request:
PNIOD_CBF_SYNC_STOPPED	Message that the function "PNIOD_stop_async()" was executed, and the I-device was stopped successfully.
PNIOD_CBF_ASYNC_REC_READ	Read data record
PNIOD_CBF_ASYNC_REC_WRITE	Write data record

Note

These services involve passive functionality. All these callbacks are called by the PROFINET library as a reaction to PROFINET IO controller actions.

3.3.3 Completion phase

Description

The completion phase of the I-device covers three steps:

Step	Action	Purpose
1	PNIOD_stop_async()	The I-device is deactivated and can therefore no longer be reached by the IO controller. The I-device acknowledges the call for the function "PNIOD_stop_async()". If there is data exchange between the IO controller and the I-device or is currently being established, the I-device interface calls up the callback "PNIOD_CBF_SYNC_DISCONNECT_IND".
2	PNIOD_CBF_SYNC_STOPPED	Wait for "PNIOD_CBF_SYNC_STOPPED"
3	PNIOD_close_sync()	The I-device interface logs off from the CP.

Description of the functions and data types for I-devices

4

This section describes the individual functions of the I-device user programming interface for an I-device in detail and the data types used.

The section is primarily intended as a source of reference when you are creating your I-device user programs.

Note

Descriptions of functions, callbacks and data types mentioned in this document but not described can be found in the document "IO-Base User Programming Interface" on the product support pages (https://support.industry.siemens.com/cs/products?lc=en-WW).

4.1 PNIO_DEV_ADDR (IO device address type)

Description

The "PNIO_DEV_ADDR address" structure is used for addressing with all I-device interface functions described here.

Syntax

```
typedef struct {
 PNIO_ADDR_TYPE AddrType;
 PNIO_IO_TYPE IODataType;
   union {
 PNIO_UINT32 Addr;
      struct {
 PNIO_UINT32 reserved1[2];
 PNIO_UINT32 Slot;
 PNIO_UINT32 Subslot;
 PNIO_UINT32 reserved2;
 } Geo;
 } u;
} ATTR_PACKED PNIO_DEV_ADDR;
```

Parameter

Name	Description
AddrType	For the IO device must always be "PNIO_ADDR_GEO"!
IODataType	Information whether input or output type. Possible values: • PNIO_IO_IN for input data • PNIO_IO_OUT for output data Not relevant if the structure is used as input parameter.
Addr	Not relevant for IO device
addr.Geo.Slot	Slot number (geographical address) The value for the slot number is always 1.
addr.Geo.Subslot	Subslot number (geographical address) The value for the subslot can be found in "STEP 7 Professional (TIA Portal)" under "Operating mode > I-device communication > Transfer area > Subslot". Example: Transfer area 1: Subslot = 1000 Transfer area 2: Subslot = 1001, etc.

Note

The address of a submodule is the same as the address of a transfer area of the I-device in "STEP 7 Professional (TIA Portal)".

4.2 Management functions for an I-device

Overview

The I-device recognizes the following management functions:

- PNIOD_init_open_sync()
- PNIOD_open_sync()
- PNIOD_close_sync()
- PNIOD_start_sync()
- PNIOD_stop_async()
- PNIOD_CBF_SYNC_STOPPED
- PNIOD_CBF_SYNC_CP_STOP_REQ

These are described in detail in the following sections. Other management functions are available for IRT mode as described in the section "IRT mode (Page 80)".

4.2.1 Data structures

4.2.1.1 PNIOD_ANNOTATION

Description

The version identifier must the transferred with the function “PNIOD_open_sync()”. This provides information about the I-device, hardware version and software version of the user program. The software version consists of a letter from the alphabet followed by three numeric 16-bit values separated in each case by a period, for example V1.0.0.

Syntax

```
#define MAX_DEVICE_TYPE_LENGTH 25
#define MAX_ORDER_ID_LENGTH 20

typedef struct {
 PNIO_UINT8 DeviceType [MAX_DEVICE_TYPE_LENGTH + 1];
 PNIO_UINT8 OrderId [MAX_ORDER_ID_LENGTH + 1];
 PNIO_UINT16 HwRevision;
 PNIO_UINT8 SwRevisionPrefix;
 PNIO_UINT16 SwRevision1;
 PNIO_UINT16 SwRevision2;
 PNIO_UINT16 SwRevision3;
} ATTR_PACKED PNIOD_ANNOTATION;
```

Structure elements

Structure element	Description
DeviceType	Identifier for the I-device
OrderId	Order id for the I-device
HwRevision	Version number of the hardware
SwRevisionPrefix	Prefix for the software version of the user program
SwRevision1	First numeric value of the software version
SwRevision2	Second numeric value of the software version
SwRevision3	Third numeric value of the software version

4.2.1.2 PNIOD_CBF_FUNCTIONS

Description

The structure with the function pointers for the callbacks must be transferred when you use the "PNIOD_open_sync()" function.

All callbacks must be specified except for:

- PNIOD_CBF_ASYNC_IRT_INIT_INPUTS (only necessary for IRT, not required for RT)
- PNIOD_CBF_SYNC_CP_STOP_REQ
- PNIOD_CBF_SYNC_START_LED_FLASH
- PNIOD_CBF_SYNC_STOP_LED_FLASH

Syntax

```
#ifndef PNIOD_CBF_RESERVED
# define PNIOD_CBF_RESERVED
#endif /* PNIOD_CBF_RESERVED */
/* structure for callback function block */

typedef struct {
 PNIO_UINT32 size;
 PNIO_CBF_DATA_WRITE cbf_data_write;
 PNIO_CBF_DATA_READ cbf_data_read;
 PNIOD_CBF_ASYNC_REC_READ cbf_async_rec_read;
 PNIOD_CBF_ASYNC_REC_WRITE cbf_async_rec_write;
 PNIOD_CBF_SYNC_ALARM_DONE cbf_sync_alarm_done;
 PNIOD_CBF_ASYNC_CONNECT_IND cbf_async_connect_ind;
 PNIOD_CBF_ASYNC_OWNERSHIP_IND cbf_async_ownership_ind;
 PNIOD_CBF_ASYNC_INDATA_IND cbf_async_indata_ind;
 PNIOD_CBF_SYNC_DISCONNECT_IND cbf_sync_disconnect_ind;
 PNIOD_CBF_SYNC_DATA_STATUS_IND cbf_sync_data_status_ind;
 PNIOD_CBF_ASYNC_PRM_END_IND cbf_async_prm_end_ind;
 PNIOD_CBF_SYNC_STOPPED cbf_sync_device_stopped;
 PNIOD_CBF_ASYNC_IRT_INIT_INPUTS cbf_async_irt_init_inputs;
 PNIOD_CBF_SYNC_CP_STOP_REQ cbf_sync_cp_stop_req;
 PNIOD_CBF_SYNC_START_LED_FLASH cbf_sync_start_led_flash;
 PNIOD_CBF_SYNC_STOP_LED_FLASH cbf_sync_stop_led_flash;
```

```
 PNIOD_CBF_RESERVED
} ATTR_PACKED PNIOD_CBF_FUNCTIONS;
```

Structure elements

Structure element	Description
size	Size of the structure, corresponds to `sizeof (PNIOD_CBF_FUNCTIONS)`
PNIOD_CBF_RESERVED	Reserved for future versions

4.2.2 PNIOD_init_open_sync()

Description

When you call this function, you obtain a structure pre-filled with default values that needs to be adapted and used for the subsequent call of "PNIOD_open_sync()". Fill in all the values according to your requirements.

Syntax

```
PNIO_UINT32 ATTR_PACKED PNIOD_init_open_sync (
 PNIOD_OPEN_SYNC_PARAMS_TYPE **ppParams,
 PNIO_UINT16 CpIndex,
 PNIO_UINT32 *pDevHndl
);
```

Parameter

<table>
<tr><th>Name</th><th colspan="2">Description</th></tr>
<tr><td rowspan="6">ppParams</td><td colspan="2">Pointer to the address to be used for the created parameters structure.

<pre>typedef struct pniod_open_sync_params_tag {
    PNIO_UINT32   ExtPar;
    PNIOD_ANNOTATION   DevAnnotation;
    PNIOD_CBF_FUNCTIONS *pCbf;
    PNIO_UINT32   *pDevHndl;
} PNIOD_OPEN_SYNC_PARAMS_TYPE;</pre></td></tr>
<tr><th>Structure element</th><th>Description</th></tr>
<tr><td>ExtPar</td><td>"PNIO_CEP_MODE_CTRL" must always be transferred.
Default: PNIO_CEP_MODE_CTRL

Note on "PNIO_CEP_MODE_CTRL"
The I-device user program that sets this bit is given the right to change its own mode. This bit must always be set.

Note on "PNIO_CEP_DISABLE_SHARE_DEVICE"
With the macro "PNIO_CEP_DISABLE_SHARE_DEVICE" the use of the communications processor as a shared I-device can be disabled. This functionality is only taken into account by the CP 1604 and CP 1616. The CP 1626 takes this setting from the configuration.</td></tr>
<tr><td>DevAnnotation</td><td>Pointer to structure for version identification of the module.
Default: All boxes 0</td></tr>
<tr><td>pCbf</td><td>Pointer to a table with callback functions
If the value "NULL" is entered for a callback function, the corresponding service is not supported. Refer to the definition of the "PNIO_CFB_FUNCTIONS" structure in the header file "pniousrd.h" for the callbacks that need to be registered.
Default: All boxes NULL</td></tr>
<tr><td>*pDevHndl</td><td>Pointer to handle assigned to the I-device. This must be included in all further function calls.</td></tr>
<tr><td>CpIndex</td><td colspan="2">Module index
Used for unique identification of the communications module; must be 1.</td></tr>
<tr><td>pDevHndl</td><td colspan="2">Pointer to handle assigned to the I-device. This must be used in all further function calls. This pointer is also entered in the structure "ppParams”.</td></tr>
</table>

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_INTERNAL
- PNIO_ERR_SEQUENCE
- PNIO_ERR_OS_RES
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_NO_FW_COMMUNICATION

4.2.3 PNIOD_open_sync()

Description

Using this function, the I-device user program registers an IO device with the I-device interface.

In addition to this, the callback functions used are registered using a structure with function pointers.

Note

The structure to be transferred here is generated by the "PNIOD_init_open_sync()" function and is released after calling the "PNIOD_open_sync()" function successfully. It can then no longer be used.

Syntax

```
typedef struct pniod_open_sync_params_tag{
 PNIO_UINT32 ExtPar;
 PNIOD_ANNOTATION DevAnnotation;
 PNIOD_CBF_FUNCTIONS *pCbf;
 PNIOD_OPEN_SYNC_RESERVED
} ATTR PACKED PNIOD OPEN SYNC PARAMS TYPE;
```

Parameter

Name	Description	
ExtPar	The parameter consists of a bit-by-bit combination of the following values:	
	PNIO_CEP_MODE_CTRL	Must always be set.
	PNIO_CEP_PE_MODE_ENABLE	Must be set if you want to shut down and start up the host PC using PROFIenergy jobs from the PNIO controller to save energy.
	PNIO_CEP_DISABLE_SHARE_DEVICE	Must be set if only one AR is to be supported (no Shared Device).
DevAnnotation	Pointer to structure for version identification of the module.	
pCbf	Pointer to structure with function pointers of the callback functions If the value "NULL" is entered for a callback function, the I-device does not support the service.. Refer to the definition of the "PNIO_CFB_FUNCTIONS" structure in the header file "pniousrd.h" for the callbacks that need to be registered.	

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_OS_RES
- PNIO_ERR_PRM_EXT_PAR
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_POINTER

4.2.4 PNIOD_close_sync()

Description

With this function, the I-device user program logs off an I-device from the I-device interface that was previously logged on with "PNIOD_open_sync()".

Note

The "PNIOD_close_sync( )" function may only be called when all jobs have been acknowledged by the I-device user program with asynchronous callbacks. If the IO device has already been started by calling "PNIOD_start_sync()", "PNIOD_stop_async()" must first be called before the "PNIOD_close_sync()" call can be executed.

If this is not adhered to, "PNIO_ERR_SEQUENCE" is returned and the "PNIO_device_close( )" job will need to be executed again.

Note

All previously registered callback functions are deregistered. On completion of the PNIOD_close_sync( ) function, no further callback functions will be called for the deregistered I-device.

Note

After successfully deregistering, the handle assigned to the I-device is no longer and may no longer be used by the I-device user program.

Syntax

```
PNIO_UINT32 PNIOD_close_sync(
 PNIO_UINT32 DevHndl
);
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_OS_RES

4.2.5 PNIOD_start_sync()

Description

With the call of "PNIOD_start_sync()" by the user program, the IO device is activated. An IO controller can only communicate with the I-device after this function has been called.

Syntax

```
PNIO_UINT32 PNIOD_start_sync(
 PNIO_UINT32 DevHndl
);
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_OS_RES

4.2.6 PNIOD_stop_async()

Description

With the call of "PNIOD_stop_async()" by the user program, the IO device is deactivated. This means that the IO device instance can no longer be addressed by the IO controller and existing application relations to IO controllers are terminated. This function must be called before the function "PNIOD_close_sync()" if the function "PNIOD_start_sync()" had been called successfully previously.

Note

The function PNIOD_stop_async() is only completed when the following callback has been called:

```
void (*PNIOD_CBF_SYNC_STOPPED)

       (PNIOD_CBF_SYNC_STOPPED_PARAMS_TYPE *pParams)
```

Syntax

```
PNIO_UINT32 PNIOD_stop_async(
 PNIO_UINT32 DevHndl
);
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_OS_RES

4.2.7 PNIOD_CBF_SYNC_STOPPED

Description

The callback "PNIOD_CBF_SYNC_STOPPED" is called by the I-device interface as a result of calling the function "PNIOD_stop_async()" to inform the I-device user program that the deactivation of the IO device triggered by calling "PNIOD_stop_async()" is complete.

Syntax

```
typedef void (*PNIOD_CBF_SYNC_STOPPED)
(PNIOD_CBF_SYNC_STOPPED_PARAMS_TYPE *pParams);
```

```
typedef struct pniod_cbf_sync_stopped_params_tag {

 PNIO_UINT32 DevHndl;

 PNIO_UINT32 ErrorCode;

 PNIOD_CBF_SYNC_STOPPED_RESERVED

} ATTR_PACKED PNIOD_CBF_SYNC_STOPPED_PARAMS_TYPE;
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
ErrorCode	Error message on the I-device stopping

Return values

No return values.

4.2.8 PNIOD_CBF_SYNC_CP_STOP_REQ

Description

The callback "PNIOD_CBF_SYNC_CP_STOP_REQ" notifies you that a firmware update, reconfiguration or a reset was requested. Call "PNIOD_close_sync()" immediately. Afterwards, it is possible to call the functions "PNIOD_init_open_sync()" and "PNIOD_open_sync()" again, but is only successful when the reconfiguration is completed.

Syntax

```
typedef void (*PNIOD_CBF_SYNC_CP_STOP_REQ)
(PNIOD_CBF_SYNC_CP_STOP_REQ_PARAMS_TYPE *pParams);
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"

Return values

No return values.

4.3 Interface for I-device configuration

Overview

The I-device recognizes the following functions for configuration:

- PNIOD_get_config_sync()
- PNIOD_get_non_routing_bits()

These is described in detail in the following sections.

Note

AR stands for Application Relation and represents a relationship between an IO controller and an IO device that comprises a number of communications relations.

4.3.1 PNIOD_get_config_sync()

Description

The "PNIOD_get_config_sync()" function allows you to read out the submodule configuration of the I-device. It lists submodules configured as both CD and TM transfer areas. The CD transfer areas (controller device transfer areas) can be read and/or written by the I-device application. The transfer areas of the "TM" type (IO routing areas) are used to transfer IO data between the control and process level; see operating instructions "CP 1604/CP 1616", section "How does the IO router work?".

Note

The data of the I-device configuration is available after a successful "PNIOD_open_sync()".

Syntax

```
PNIOD_GET_CONFIG_SYNC_PARAMS_TYPE PNIO_CODE_ATTR

PNIOD_get_config_sync(

 PNIO_UINT32 DevHndl);
```

Parameters

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"

Return values

A pointer is returned to the following structure if successful, or "NULL" if an error occurs. The structure was allocated by the IO-Base library and cannot be released by the I-device application. The pointer is valid until the "PNIOD_close()" function is called.

```
typedef struct pniod_get_config_sync_params_tag {

 PNIO_UINT16 VendorId;

 PNIO_UINT16 DeviceId;

 PNIO_UINT32 MaxARsIOC;

 PNIO_UINT16 MinDeviceInterval;

 PNIO_UINT16 NrOfSubmods;

 PNIO_UINT32 LowestTransferSubmod;

 PNIOD_GET_CONFIG_SYNC_ELEMENT_TYPE* pSubmodList;

 PNIOD_GET_CONFIG_SYNC_RESERVED
```

```
} ATTR_PACKED PNIOD_GET_CONFIG_SYNC_PARAMS_TYPE;
```

Structure element	Description
VendorID	Manufacturer ID for the device, must be applied for from the PNO. (For all seems products: 0x002a)
DeviceID	Device ID must be applied for from the PNO and be unique within the PNIO products of a manufacturer..
MaxARsIOC	Maximum number of supported "single ARs". These include ARs of the type "IOC_AR_SINGLE" (controller AR with RT) and "IOC_AR_SINGLE_RTC3" (controller AR with IRT high performance).
MinDeviceInterval	MinDeviceInterval value from the GSDML file. AR establishment attempts by IO controllers are automatically rejected if they do not correspond to the condition SendClock * ReductionRatio >= MinDeviceInterval.
NrOfSubmods	Number of sub modules involved; in other words the elements in "pSubmodList"
LowestTransferSubmod	The lowest slot number configured for a transfer submodule.

pSubModList	Submodules involved.
	
	"PNIOD_GET_CONFIG_SYNC_ELEMENT_TYPE" has the following structure:
	
	`typedef struct pniod_get_config_sync_element_tag {`
	`  PNIO_UINT32 Api;`
	`  PNIO_DEV_ADDR Addr;`
	`  PNIO_UINT32 ModIdent;`
	`  PNIO_UINT32 SubIdent;`
	`  PNIO_UINT32 InDatLength;`
	`  PNIO_UINT32 OutDatLength;`
	`  PNIO_UINT32 Properties;`
	`  PNIO_UINT32 Reserved[4];`
	`} ATTR_PACKED PNIOD_GET_CONFIG_SYNC_ELEMENT_TYPE;`

Name	Description
Api	"Application process identifier" of the submodule involved
Addr	Slot or subslot number of the module
ModIdent	"module ident number" - unique type ID of the module
SubIdent	"submodule ident number" - unique type ID of the submodule
InDatLength	Length of the output data
Out-DatLength	Length of the input data
Properties	If bit 0 = 1: The module is an IO routing area If bit 0 = 0: The module is a CD transfer area
Reserved	not relevant

4.4 Interface for read and write data record

Overview

The following callback functions are available for an I-device for read data record and write data record:

- PNIOD_CBF_ASYNC_REC_READ
- PNIOD_rec_read_async_rsp
- PNIOD_CBF_ASYNC_REC_WRITE
- PNIOD_rec_write_async_rsp

These are described in detail in the following sections.

4.4.1 PNIOD_CBF_ASYNC_REC_READ

Description

By calling this callback, the user is signaled that a read data record job has arrived.

Syntax

```
typedef void (*PNIOD_CBF_ASYNC_REC_READ)
 (PNIOD_CBF_ASYNC_REC_READ_PARAMS_TYPE *pParams);

typedef struct pniod_cbf_async_rec_read_params_tag {
 PNIO_UINT32 DevHndl;
 PNIO_UINT32 Api;
 PNIO_UINT16 SessionKey;
 PNIO_UINT32 SequenceNum;
 PNIO_DEV_ADDR Addr;
 PNIO_UINT32 RecordIndex;
 PNIO_UINT32 BufLen;
 PNIO_UINT8 *pBuffer;
 PNIO_ERR_STAT PnioStatus;
 PNIOD_CBF_ASYNC_REC_READ_RESERVED
} ATTR PACKED PNIOD CBF ASYNC REC READ PARAMS TYPE;
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
Api	"Application process identifier" for the submodule
SessionKey	"Session key" for unique identification of the relevant AR, 0 with "Implicit Read"
SequenceNum	Not relevant
Addr	Address of the submodule
RecordIndex	Index of the data record to be read.
BufLen	Length of the read buffer made available, referenced by "pBuffer".
pBuffer	Pointer to the data buffer into which the user program should copy the data record content. The format of this data is specified by the PNIO standard. You will find examples in the supplied sample programs. The buffer remains valid until the "PNIOD_rec_read_async_rsp()" function has been called.
PnioStatus	PNIO status (consisting of ErrCode, ErrDecode,ErrCode1 and ErrCode2)

Note

The structure pointer "pParam" transferred with the callback function remains valid until the "PNIOD_rec_read_async_rsp()" function has been called.

Return values

No return values.

4.4.2 PNIOD_rec_read_async_rsp

Description

This function is used to acknowledge the callback "PNIOD_CBF_ASYNC_REC_READ".

The read data record job only counts as completed when this function is called As the transfer parameter the pointer "pParams" from the callback "PNIOD_CBF_ASYNC_REC_READ" must be used.

Syntax

```
PNIO_UINT32 PNIOD_rec_read_async_rsp (PNIOD_CBF_ASYNC_REC_READ_PARAMS_TYPE *pParams);
```

Parameter

The parameters result from those of the callback "PNIOD_CBF_ASYNC_REC_READ". Only the following values need adjusting :

Name	Description
BufLen	Number of bytes that were copied to the buffer. This value must be less than or equal to the buffer length contained in the "BufLen" parameter in the "PNIOD_CBF_ASYNC_REC_READ" callback. If the I-device user program wants to reject the read job, the "BufLen" parameter needs to be set to 0.
PnioStatus	PNIO status (consisting of ErrCode, ErrDecode, ErrCode1 and ErrCode2), see PNIO standard.
pBuffer	The content of the data record to be read must be entered in this buffer as long as the length of the buffer (BufLen) is adequate.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_INTERNAL
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN
- PNIO_ERR_OS_RES

4.4.3 PNIOD_CBF_ASYNC_REC_WRITE

Description

The "PNIOD_CBF_ASYNC_REC_WRITE" callback shows you that a write data record job has arrived for the IO device.

Syntax

```
typedef void (PNIOD_CBF_ASYNC_REC_WRITE)(
PNIOD_CBF_ASYNC_REC_WRITE_PARAMS_TYPE *pParams
);
typedef struct pniod_cbf_async_rec_write_params_tag {
    PNIO_UINT32          DevHndl;
    PNIO_UINT32          Api;
    PNIO_UINT16          SessionKey;
    PNIO_UINT32          SequenceNum;
    PNIO_DEV_ADDR        Addr;
    PNIO_UINT32          RecordIndex;
    PNIO_UINT32          BufLen;
    PNIO_UINT8         *pBuffer;
    PNIO_ERR_STAT        PnioStatus;
    PNIO_BOOL            IsDeviceAccess;
} PNIOD_CBF_ASYNC_REC_WRITE_PARAMS_TYPE;
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
Api	"Application process identifier" for the submodule
SessionKey	"Session key" for unique identification of the relevant AR
SequenceNum	Not relevant
Addr	Address of the submodule
RecordIndex	Index of the data record to be written.
BufLen	Length of the data record to be written
pBuffer	Pointer to the data buffer from which the user program should copy the data record content. The buffer remains valid until the "PNIOD_rec_write_async_rsp()" function has been called.
PnioStatus	PNIO status (consisting of ErrCode, ErrDecode,ErrCode1 and ErrCode2), see PNIO standard.
IsDeviceAccess	This flag shows whether or not a write job originates from a DeviceAccess AR

Note

The structure pointer "pParam" transferred with the callback function remains valid until the "PNIOD_rec_write_async_rsp()" function has been called.

Return values

No return values.

4.4.4 PNIOD_rec_write_async_rsp

Description

This function is used to acknowledge the callback "PNIOD_CBF_ASYNC_REC_WRITE".

The read data record job only counts as completed when this function is called As the transfer parameter the pointer "pParams" from the callback "PNIOD_CBF_ASYNC_REC_WRITE" must be used.

Syntax

```
PNIO_UINT32 PNIOD_rec_write_async_rsp(PNIOD_CBF_ASYNC_REC_WRITE_PARAMS_TYPE
*pParams);
```

Parameter

In the response, the values of the callback must be adopted. Only the following values need filling in:

Name	Description
BufLen	Number of bytes that were copied from the buffer. The number of bytes actually written must always be returned even if this is less than the data record length.
PnioStatus	PNIO status (consisting of ErrCode, ErrDecode, ErrCode1 and ErrCode2), see PNIO standard.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_INTERNAL
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN
- PNIO_ERR_OS_RES

4.5 Establishment and termination of ARs

Overview

The following functions for establishing and terminating the ARs are available for an I-device:

- PNIOD_CBF_ASYNC_CONNECT_IND
- PNIOD_connect_async_rsp()
- PNIOD_CBF_ASYNC_OWNERSHIP_IND
- PNIOD_ownership_async_rsp()
- PNIOD_CBF_ASYNC_PRM_END_IND
- PNIOD_prm_end_async_rsp()
- PNIOD_CBF_ASYNC_IRT_INIT_INPUTS
- PNIOD_irt_init_inputs_async_rsp()
- PNIOD_CBF_ASYNC_INDATA_IND
- PNIOD_indata_async_rsp()
- PNIOD_CBF_SYNC_DISCONNECT_IND
- PNIOD_CBF_SYNC_DATA_STATUS_IND

These are described in detail in the following sections.

Note

AR stands for Application Relation and represents a relationship between an IO controller and an IO device that comprises a number of communications relations.

4.5.1 PNIOD_CBF_ASYNC_CONNECT_IND

Description

The "PNIOD_CBF_ASYNC_CONNECT_IND" callback indicates that a connect request was received and therefore an AR establishment sequence initiated.

Syntax

```
void (PNIOD_CBF_ASYNC_CONNECT_IND)(
PNIOD_CBF_ASYNC_CONNECT_IND_PARAMS_TYPE *pParam
);
typedef struct pniod_cbf_async_connect_ind_params_tag {
    PNIO_UINT32      DevHndl;
    PNIO_UINT32      HostIp;
    PNIO_UINT16      ArType;
    PNIO_UUID_TYPE   ArUUID;
    PNIO_UINT32      ArProperties;
    PNIO_UUID_TYPE   CmiObjUUID;
    PNIO_UINT8     *pCmiStationName;
    PNIO_UINT16      SessionKey;
} PNIOD_CBF_ASYNC_CONNECT_IND_PARAMS_TYPE;
```

Parameter

Note

The structure pointer "pParam" transferred with the callback function remains valid until the "PNIOD_connect_async_rsp()" function has been called.

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
HostIp	IP address of the IO controller

ArType	Type of the AR to be established. In the current version, the following AR types are supported:
	`typedef enum{`
	` PNIO_AR_TYPE_SINGLE = 0x00000001,`
	` PNIO_AR_TYPE_SINGLE_RTC3 = 0x00000010`
	`}PNIO_AR_TYPE_ENUM;`
	
	The AR type "PNIO_AR_TYPE_SINGLE" is an AR of the RT mode, "PNIO_AR_TYPE_SINGLE_RTC3" on the other hand, is an AR of the IRT mode.

ArUUID	For internal purposes, see PNIO standard.
ArProperties	For internal purposes, see PNIO standard.
CmiObjUUID	For internal purposes, see PNIO standard.
pCmiStationName	For internal purposes, see PNIO standard.
SessionKey	"Session key" for unique identification of the relevant AR. This parameter is required during the further communication between the user program and I-device programming interface to establish an assignment to a specific AR.

Return values

No return values.

4.5.2 PNIOD_connect_async_rsp

Description

This function is used to acknowledge the callback "PNIOD_CBF_ASYNC_CONNECT_IND".

The function "PNIOD_connect_async_rsp" may not be called up within the "PNIOD_CBF_ASYNC_CONNECT_IND" callback, but only after leaving the "PNIOD_CBF_ASYNC_CONNECT_IND" callback. As the transfer parameter the pointer "pParam" from the callback "PNIOD_CBF_ASYNC_CONNECT_IND" must be used.

Syntax

```
PNIO_UINT32 PNIOD_connect_async_rsp (PNIOD_CBF_ASYNC_CONNECT_IND_PARAMS_TYPE
*pParams);
```

Parameter

All the values from the "PNIOD_CBF_ASYNC_CONNECT_IND" callback are adopted unchanged.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_INTERNAL
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN
- PNIO_ERR_OS_RES

4.5.3 PNIOD_CBF_ASYNC_OWNERSHIP_IND

Description

The "PNIOD_CBF_ASYNC_OWNERSHIP_IND" callback indicates that one or more submodules were assigned to an IO controller or this assignment was canceled. Assignment means that the IO controller can exchange data with this submodule via an AR.

Syntax

```
void (*PNIOD_CBF_ASYNC_OWNERSHIP_IND)
 (PNIOD_CBF_ASYNC_OWNERSHIP_IND_PARAMS_TYPE
*pParams);


typedef struct pniod_cbf_async_ownership_ind_params_tag {
 PNIO_UINT32 DevHndl;
 PNIO_UINT16 SessionKey;
 PNIO_AR_CONTEXT ArContext;
 PNIO_UINT16 NrOfSubmods;
 PNIOD_CBF_ASYNC_OWNERSHIP_ELEMENT_TYPE *pSubmodList;
 PNIOD_CBF_ASYNC_OWNERSHIP_IND_RESERVED
} ATTR_PACKED PNIOD_CBF_ASYNC_OWNERSHIP_IND_PARAMS_TYPE;
```

Parameter

Note

The structure pointer "pParam" transferred with the callback function remains valid until the "PNIOD_ownership_async_rsp()" function has been called.

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
SessionKey	"Session key" for unique identification of the AR

ARContext	Event that led to a change of ownership.
	The following values are possible:
	`typedef enum {`
	`  PNIO_AR_CONTEXT_CONN = 1`
	`  PNIO_AR_CONTEXT_DISC = 2`
	`} PNIO_AR_CONTEXT;`

NrOfSubmods	Number of submodules involved; in other words the elements in "pSubmodList"
pSubModList	Submodules involved

Name	Description
OwnerId	When "PNIO_AR_CONTEXT_CONN": !=0: From the perspective of the "pnioLib", the submodule can be included in the AR with the specified "session key". When "PNIO_AR_CONTEXT_CONN": ==0: From the perspective of the "pnioLib", the submodule cannot be included in the AR with the specified "session key" (e.g. because it is already assigned to another AR). When "PNIO_AR_CONTEXT_DISC": Always !=0, meaning: The submodule will be removed from the AR with the specified "session key".
IsWrong	When "PNIO_FALSE": Module and submodule identification number of the relevant submodule match in the desired and actual configuration. When "PNIO_TRUE": Module and submodule identification number of the relevant submodule do not match in the desired and actual configuration. The user program must decide whether the submodule from the I-device configuration is compatible with the expected submodule from the configuration of the higher-level IO controller.

Return values

No return values.

4.5.4 PNIOD_ownership_async_rsp

Description

This function serves to acknowledge the "PNIOD_CBF_ASYNC_OWNERSHIP_IND" callback.

The function "PNIOD_ownership_async_rsp" may not be called within the "PNIOD_CBF_ASYNC_OWNERSHIP_IND" callback, but only after leaving the "PNIOD_CBF_ASYNC_OWNERSHIP_IND" callback. As the transfer parameter the pointer "pParam" from the callback” PNIOD_CBF_ASYNC_OWNERSHIP_IND” must be used.

Syntax

```
PNIO_UINT32 PNIOD_ownership_async_rsp (PNIOD_CBF_ASYNC_OWNERSHIP_IND_PARAMS_TYPE
*pParams);


typedef struct pniod_cbf_async_ownership_ind_params_tag {
 PNIO_UINT32 DevHndl;
 PNIO_UINT16 SessionKey;
 PNIO_AR_CONTEXT ArContext;
 PNIO_UINT16 NrOfSubmods;
```

```
 PNIOD_CBF_ASYNC_OWNERSHIP_ELEMENT_TYPE *pSubmodList;

 PNIOD_CBF_ASYNC_OWNERSHIP_IND_RESERVED

} ATTR_PACKED PNIOD_CBF_ASYNC_OWNERSHIP_IND_PARAMS_TYPE;
```

Parameter

In the submodule list (pSubModList) the following values of the individual modules need to be adapted: "OwnerId" and "IsWrong"

Name	Description	

pSubmodList	`typedef struct pniod_cbf_async_ownership_element_tag {`
	`PNIO_UINT32 Api;`
	`PNIO_DEV_ADDR Addr;`
	`PNIO_UINT32 ModIdent;`
	`PNIO_UINT32 SubIdent;`
	`PNIO_UINT32 InDatLength;`
	`PNIO_UINT32 OutDatLength;`
	`PNIO_UINT16 OwnerId;`
	`PNIO_BOOL IsWrong;`
	`PNIOD_CBF_ASYNC_OWNERSHIP_IND_ELEMENT_RESERVED`
	`} ATTR_PACKED PNIOD_CBF_ASYNC_OWNERSHIP_ELEMENT_TYPE;`	

	Structure element	**Description**
	Api	"Application Process Identifier", to which the relevant submodule is assigned.
	Addr	Address of the submodule
	ModIdent	Module identification number of the associated module.
	SubIdent	Submodule identification number of the submodule to be plugged in
	InDatLength	Length of the input data
	OutDatLength	Length of the output data
	OwnerId	When PNIO_AR_CONTEXT_CONN:!=0 (unchanged compared with "PNIOD_CBF_ASYNC_OWNERSHIP_IND"): From the perspective of the user, the submodule can be included in the AR with the specified "session key" (only permitted when the submodule can be included in the AR from the perspective of "pnioLib”). When PNIO_AR_CONTEXT_CONN:==0: The submodule cannot be accepted in the AR from the perspective of the user (e.g. not compatible, not ready for parameter assignment). When PNIO_AR_CONTEXT_DISC: unchanged compared with "PNIOD_CBF_ASYNC_OWNERSHIP_IND"):
	IsWrong	The value “IsWrong” transferred to the user program in "PNIOD_CBF_ASYNC_OWNERSHIP_IND" must be retained. Exception: If “IsWrong” was set to “PNIO_TRUE” by “pnioLib” and the module and submodule numbers of the submodule from "PNIOD_get_config_sync()" and "PNIOD_CBF_ASYNC_OWNERSHIP_IND” are not identical but fully compatible, “IsWrong” must be set to “PNIO_FALSE”.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_INTERNAL
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_INVALIDARG
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN
- PNIO_ERR_OS_RES

4.5.5 PNIOD_CBF_ASYNC_PRM_END_IND

Description

The "PNIOD_CBF_ASYNC_PRM_END_IND" callback shows that the parameter assignment phase is completed for all submodules of this AR.

Note

If the I-device is used in RT mode, before the I-device application calls the acknowledgment all the input data must initially be written using the function "PNIOD_trigger_data_write_sync".

Syntax

```
void (PNIOD_CBF_ASYNC_PRM_END_IND)(
PNIOD_CBF_ASYNC_PRM_END_IND_PARAMS_TYPE *pParam
);
typedef struct pniod_cbf_async_prm_end_ind_params_tag {
    PNIO_UINT32      DevHndl;
    PNIO_UINT16      SessionKey;
    PNIO_AR_CONTEXT  ArContext;
    PNIO_UINT16      NrOfSubmods;
    PNIOD_CBF_ASYNC_PRM_END_ELEMENT_TYPE *pSubmodList;
} PNIOD_CBF_ASYNC_PRM_END_IND_PARAMS_TYPE;
```

Parameter

Note

The structure pointer "pParam" transferred with the callback function remains valid until the "PNIOD_prm_end_async_rsp()" function has been called.

Name	Description	
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"	
SessionKey	"Session key" for unique identification of the relevant AR.	

ArContext	Event that led to the parameter assignment.
	The following values are possible:
	`typedef enum{`
	`  PNIO_AR_CONTEXT_CONN = 1,`
	`  PNIO_AR_CONTEXT_DISC = 2,`
	`}PNIO_AR_CONTEXT;`	

	PNIO_AR_CONTEXT_CONN = 1	AR establishment by the IO controller
	PNIO_AR_CONTEXT_DISC = 2	AR termination by the IO controller
NrOfSubmods	Number of submodules involved; in other words the elements in "pSubmodList"	

pSubmodList	Pointer to a list of submodules to have parameters assigned with the following structure:
	`typedef struct pniod_cbf_async_prm_end_element_tag {`
	`  PNIO_UINT32 Api;`
	`  PNIO_DEV_ADDR Addr;`
	`  PNIO_BOOL ApplReadyPending;`
	`  PNIO_UINT32 Reserved[5];`
	`} ATTR_PACKED PNIOD_CBF_ASYNC_PRM_END_ELEMENT_TYPE;`	

	Api	"Application process identifier" for the submodule
	Addr	Address of the submodule involved
	ApplReadyPending	Internally recognized parameter assignment error (e.g. configuration error)

Return values

No return values.

4.5.6 PNIOD_prm_end_async_rsp

Description

This function serves to acknowledge the "PNIOD_CBF_ASYNC_PRM_END_IND" callback.

The function "PNIOD_prm_end_async_rsp" may not be called within the "PNIOD_CBF_ASYNC_PRM_END_IND" callback, but only after leaving the "PNIOD_CBF_ASYNC_PRM_END_IND" callback. As the transfer parameter the pointer "pParam" from the callback "PNIOD_CBF_ASYNC_PRM_END_IND" must be used.

Syntax

```
PNIO_UINT32 PNIOD_prm_end_async_rsp (PNIOD_CBF_ASYNC_PRM_END_IND_PARAMS_TYPE
*pParams);
```

Parameter

In the submodule list (pSubModList) the following value of the individual modules needs to be adapted: "ApplReadyPending"

Set "ApplReadyPending" to "PNIO_FALSE", if the parameter assignment of this submodule was successful and the submodule is ready for cyclic data exchange. In all other cases, the value must be set to "PNIO_TRUE".

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_INTERNAL
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN
- PNIO_ERR_OS_RES

4.5.7 PNIOD_CBF_ASYNC_IRT_INIT_INPUTS

Description

The "PNIOD_CBF_ASYNC_IRT_INIT_INPUTS" callback shows that the IRT synchronization with the IO controller was successful. Before the acknowledgment by the I-device application can take place all the input data must initially be written using the function "PNIOD_trigger_data_write_sync".

Syntax

```
void (PNIOD_CBF_ASYNC_IRT_INIT_INPUTS)(
PNIOD_CBF_ASYNC_IRT_INIT_INPUTS_PARAMS_TYPE *pParam
);
typedef struct pniod_cbf_async_irt_init_inputs_params_tag {
    PNIO_UINT32      DevHndl;
    PNIO_UINT16      SessionKey;
} PNIOD CBF ASYNC IRT INIT INPUTS PARAMS TYPE;
```

Parameter

Note

The structure pointer "pParam" transferred with the callback function remains valid until the "PNIOD_irt_init_inputs_async_rsp()" function has been called.

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
SessionKey	"Session key" for unique identification of the relevant AR

Return values

No return values.

4.5.8 PNIOD_irt_init_inputs_async_rsp

Description

This function serves to acknowledge the "PNIOD_CBF_ASYNC_IRT_INIT_INPUTS" callback.

The function "PNIOD_irt_init_inputs_async_rsp" may not be called within the "PNIOD_CBF_ASYNC_IRT_INIT_INPUTS" callback, but only after leaving the "PNIOD_CBF_ASYNC_IRT_INIT_INPUTS" callback. When calling this function, it must be certain that the input data and their consumer and provider status were written at least once. As the transfer parameter the pointer "pParam" from the callback "PNIOD_CBF_ASYNC_IRT_INIT_INPUTS" must be used.

Syntax

```
PNIO_UINT32 PNIOD_irt_init_inputs_async_rsp
(PNIOD_CBF_ASYNC_IRT_INIT_INPUTS_PARAMS_TYPE *pParams);
```

Parameter

The values from the callback do not need to be changed.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_INTERNAL
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN
- PNIO_ERR_OS_RES

4.5.9 PNIOD_CBF_ASYNC_INDATA_IND

Description

The "PNIOD_CBF_ASYNC_INDATA_IND" callback indicates that cyclic data exchange has started.

Syntax

```
void (PNIOD_CBF_ASYNC_INDATA_IND)(
PNIOD_CBF_ASYNC_INDATA_IND_PARAMS_TYPE *pParam
);
typedef struct pniod_cbf_async_indata_ind_params_tag {
    PNIO_UINT32      DevHndl;
    PNIO_UINT16      SessionKey;
    PNIO_UINT16      NrOfSubmods;
    PNIOD_CBF_ASYNC_INDATA_ELEMENT_TYPE *pSubmodList;
} PNIOD_CBF_ASYNC_INDATA_IND_PARAMS_TYPE;
```

Parameter

Note

The structure pointer "pParam" transferred with the callback function remains valid until the "PNIOD_indata_async_rsp()" function has been called.

<table>
<tr><th>Name</th><th colspan="2">Description</th></tr>
<tr><td>DevHndl</td><td colspan="2">"DevHndl" from "PNIOD_init_open_sync()"</td></tr>
<tr><td>SessionKey</td><td colspan="2">"Session key" for unique identification of the relevant AR</td></tr>
<tr><td>NrOfSubmods</td><td colspan="2">Number of submodules involved; in other words the elements in "pSubmodList"</td></tr>
<tr><td rowspan="4">pSubmodList</td><td colspan="2">Pointer to a list of the submodules involved in the data exchange.

<pre>typedef struct pniod_cbf_async_indata_element_tag {
 PNIO_UINT32 Api;
 PNIO_DEV_ADDR Addr;
 PNIOD_CBF_ASYNC_INDATA_IND_ELEMENT_RESERVED
} ATTR_PACKED PNIOD_CBF_ASYNC_INDATA_ELEMENT_TYPE;</pre></td></tr>
<tr><th>Structure element</th><th>Description</th></tr>
<tr><td>Api</td><td>"Application Process Identifier", to which the relevant submodule is assigned.</td></tr>
<tr><td>Addr</td><td>Address of the submodule involved.</td></tr>
</table>

Return values

No return values.

4.5.10 PNIOD_indata_async_rsp

Description

This function serves to acknowledge the "PNIOD_CBF_ASYNC_INDATA_IND" callback.

The function "PNIOD_indata_async_rsp" may not be called within the "PNIOD_CBF_ASYNC_INDATA_IND" callback, but only after leaving the "PNIOD_CBF_ASYNC_INDATA_IND" callback. As the transfer parameter the pointer "pParam" from the callback "PNIOD_CBF_ASYNC_INDATA_IND" must be used.

Syntax

```
PNIO_UINT32 PNIOD_indata_async_rsp (PNIOD_CBF_ASYNC_INDATA_IND_PARAMS_TYPE *pParams);
```

Parameter

The values from the callback may not be changed.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_INTERNAL
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_INVALIDARG
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN
- PNIO_ERR_OS_RES

4.5.11 PNIOD_CBF_SYNC_DISCONNECT_IND

Description

The "PNIOD_CBF_SYNC_DISCONNECT_IND" callback indicates that the relevant AR was aborted.

Syntax

```
void (PNIOD_CBF_SYNC_DISCONNECT_IND)(
PNIOD_CBF_SYNC_DISCONNECT_IND_PARAMS_TYPE *pParam
);
typedef struct pniod_cbf_sync_disconnect_ind_params_tag {
    PNIO_UINT32      DevHndl;
    PNIO_UINT16      SessionKey;
    PNIO_AR_REASON   ReasonCode;
    PNIO_BOOL        WasInData;
} PNIOD_CBF_SYNC_DISCONNECT_IND_PARAMS_TYPE;
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
SessionKey	"Session key" for unique identification of the relevant AR. Since this AR no longer exists, the specified "session key." Becomes invalid when "PNIOD_CBF_SYNC_DISCONNECT_IND" is called.
ReasonCode	The reason why the AR was aborted. Possible reasons: see PNIO_AR_REASON in the header file pniousrd.h in the appendix
WasInData	Value "TRUE" if the aborted AR was already in cyclic data exchange, otherwise "FALSE".

Return values

No return values.

4.5.12 PNIOD_CBF_SYNC_DATA_STATUS_IND

Description

The "PNIOD_CBF_SYNC_DATA_STATUS_IND" callback indicates that the APDU status of a higher-level IO controller has changed.

Syntax

```
void (PNIOD_CBF_SYNC_DISCONNECT_IND)(
PNIOD_CBF_SYNC_DATA_STATUS_IND_PARAMS_TYPE *pParam
);
typedef struct pniod_cbf_sync_data_status_ind_params_tag {
    PNIO_UINT32          DevHndl;
    PNIO_UINT16          SessionKey;
    PNIO_UINT8           CurrentStatus;
    PNIO_UINT8           PreviousStatus;
} PNIOD_CBF_SYNC_DATA_STATUS_IND_PARAMS_TYPE;
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
SessionKey	"Session key" for unique identification of the relevant AR.
CurrentStatus	Current (new) value of the APDU status. Possible values: see PNIO_DATA_STATUS_XXX defines in pniousrd.h.
PreviousStatus	Previous (old) value of the APDU status. Possible values: see PNIO_DATA_STATUS_XXX defines in pniousrd.h.

Return values

No return values.

4.6 Alarm and diagnostics interface

Overview

The following functions for the alarms and diagnostics are available for an I-device:

- PNIOD_build_chan_prop_sync()
- PNIOD_init_channel_add_sync()
- PNIOD_diag_channel_add_sync()
- PNIOD_init_generic_add_sync()
- PNIOD_diag_generic_add_sync()
- PNIOD_diag_remove_sync()
- PNIOD_init_alarm_send_async()
- PNIOD_alarm_send_async()
- PNIOD_CBF_SYNC_ALARM_DONE()

These are described in detail in the following sections.

4.6.1 PNIOD_build_chan_prop_sync()

Description

The "PNIOD_build_chan_prop_sync()" function generates the channel properties required to add diagnostics and for sending a diagnostics alarm (for the parameter "pData").

Syntax

```
PNIO_UINT32 PNIOD_build_chan_prop_sync(
 PNIO_UINT32 DevHndl,
 PNIO_BOOL MaintRequired,
 PNIO_BOOL MaintDemanded,
 PNIO_UINT16 Type,
 PNIO_UINT16 Spec,
 PNIO_UINT16 Dir
);
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
MaintRequired	Maintenance Required (must be set to the value "TRUE" to generate "maintenance required" diagnostics information)
MaintDemanded	"Maintenance Demanded" (must be set to the value TRUE to generate "maintenance demanded" diagnostics information)
Type	Packing size for the "Channel Diagnosis Property" Possible values are as follows: • PNIO_DIAG_CHANPROP_TYPE_SUBMOD • PNIO_DIAG_CHANPROP_TYPE_1BIT • PNIO_DIAG_CHANPROP_TYPE_2BIT • PNIO_DIAG_CHANPROP_TYPE_4BIT • PNIO_DIAG_CHANPROP_TYPE_BYTE • PNIO_DIAG_CHANPROP_TYPE_WORD • PNIO_DIAG_CHANPROP_TYPE_DWORD • PNIO_DIAG_CHANPROP_TYPE_LWORD When the diagnostics information to be generated relates to an entire submodule (channel number equals 0x8000), the type is always 0. If on the other hand, the diagnostics information relates to a single channel, its length needs to be specified here.
Spec	Meaning of the "Channel Diagnosis Property". Possible values are as follows: • PNIO_DIAG_CHAN_SPEC_ERROR_APPEARS: (error appears) • PNIO_DIAG_CHAN_SPEC_ERROR_DISAPP_FREE: (error cleared, no further errors) • PNIO_DIAG_CHAN_SPEC_ERROR_DISAPP_REMAIN: (error cleared , further errors exist)
Dir	Data direction of the module for which the diagnostics data will be created. Possible values are as follows: • PNIO_DIAG_CHAN_DIRECTION_MANUFACTURE (vendor-specific) • PNIO_DIAG_CHAN_DIRECTION_INPUT: (input) • PNIO_DIAG_CHAN_DIRECTION_OUTPUT: (output) • PNIO_DIAG_CHAN_DIRECTION_BIDIRECT: (input/output)

Return values

Packed bit structure for channel properties.

4.6.2 PNIOD_init_diag_channel_add_sync()

Description

Generates a default structure for "PNIOD_diag_channel_add_sync()".

Syntax

```
PNIO_UINT32 PNIOD_init_diag_channel_add_sync(
 PNIOD_DIAG_CHANNEL_ADD_SYNC_PARAMS_TYPE **ppParams,
 PNIO_UINT32 DevHndl
);
```

Parameter

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>ppParams</td><td>
<pre>typedef struct pniod_diag_channel_add_sync_params_tag {
    PNIO_UINT32   DevHndl;
    PNIO_UINT32   Api;
    PNIO_DEV_ADDR   Addr;
    PNIO_UINT32   DiagTag;
    PNIO_UINT16   ChanDiagType;
    PNIO_UINT16   ChanNum;
    PNIO_UINT16   ChanProp;
    PNIO_UINT16   ChanErrType;
    PNIO_UINT16   ExtChanErrType;
    PNIO_UINT32   ExtChanAddValue;
    PNIO_UINT32   QuaChanQualifier;
} PNIOD_DIAG_CHANNEL_ADD_SYNC_PARAMS_TYPE;</pre>
<table>
<tr><th>Structure element</th><th>Description</th></tr>
<tr><td>DevHndl</td><td>DevHndl from "PNIOD_init_open_sync()"
Default: 1</td></tr>
<tr><td>Api</td><td>"Application Process Identifier", to which the relevant submodule is assigned.
Default: 0</td></tr>
<tr><td>Addr</td><td>Address of the submodule for which diagnostics will be added
Default: PNIO_ADDR_GEO; all further boxes 0</td></tr>
<tr><td>DiagTag</td><td>Unique reference to the diagnostics data record to be entered.
Default: 0</td></tr>
<tr><td>ChanDiagType</td><td>Subtype of the channel diagnostics.
The following can be selected:
<pre>#define PNIOD_CHAN_DIAG_TYPE_CHANNEL 0x00
#define PNIOD_CHAN_DIAG_TYPE_EXT_CHANNEL 0x01
#define PNIOD_CHAN_DIAG_TYPE_QUA_CHANNEL 0x02
#define PNIOD_CHAN_DIAG_TYPE_NONE 0x04</pre>
Default: PNIOD_CHAN_DIAG_TYPE_NONE</td></tr>
</table>
</td></tr>
</table>

Name	Description	
	ChanNum	Channel number Range of values 0 ... 0x8000 Default: 0x8000
	ChanProp	Channel properties from "PNIOD_build_chan_prop_sync()" Default: 0xFFFF
	ChanErrType	Channel error type For possible values, see the header file pniousrd.h in the appendix. Default: 0 (unknown error)
	ExtChanErrType	Additional information on the channel error For possible values, see the PNIO standard. Default: 0xFFFF
	ExtChanAddValue	Additional information on the channel error can be obtained with this. For possible values, see the PNIO standard. Default: 0xFFFFFFFF
	QuaChanQualifier	Further additional information on the channel error can be obtained with this. For possible values, see the PNIO standard. Default: 0
DevHndl	DevHndl from "PNIOD_init_open_sync()"	

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_INTERNAL
- PNIO_ERR_SEQUENCE
- PNIO_ERR_PRM_BUF
- PNIO_ERR_OS_RES

4.6.3 PNIOD_diag_channel_add_sync()

Description

The "PNIOD_diag_channel_add_sync()" function is used to load a channel diagnostics data record in a subslot.

Note

The structure generated by the "PNIOD_diag_channel_add_sync_init" function is released after a successful call and can no longer be used.

Note

If the submodule involved is part of an application relation, a diagnostics alarm must be sent following "PNIOD_diag_channel_add_sync()" to inform the IO controller about the diagnostics.

Syntax

```
PNIO_UINT32 PNIOD_diag_channel_add_sync(
PNIOD_DIAG_CHANNEL_ADD_SYNC_PARAMS_TYPE **ppParam
);
typedef struct pniod_diag_channel_add_sync_params_tag {
    PNIO_UINT32      DevHndl;
    PNIO_UINT32      Api;
    PNIO_DEV_ADDR    Addr;
    PNIO_UINT32      DiagTag;
    PNIO_UINT16      ChanDiagType;
    PNIO_UINT16      ChanNum;
    PNIO_UINT16      ChanProp;
    PNIO_UINT16      ChanErrType;
    PNIO_UINT16      ExtChanErrType;
    PNIO_UINT32      ExtChanAddValue;
    PNIO_UINT32      QuaChanQualifier;
} PNIOD_DIAG_CHANNEL_ADD_SYNC_PARAMS_TYPE;
```

Parameter

Name	Description
DevHndl	DevHndl from "PNIOD_init_open_sync()"
Api	"Application process identifier" for the submodule
Addr	Address of the submodule
DiagTag	Unique reference to the diagnostics data record to be entered.
ChanDiagType	Subtype of the channel diagnostics The following can be selected: • PNIOD_CHAN_DIAG_TYPE_CHANNEL 0x00 • PNIOD_CHAN_DIAG_TYPE_EXT_CHANNEL 0x01 • PNIOD_CHAN_DIAG_TYPE_QUA_CHANNEL 0x02 • PNIOD_CHAN_DIAG_TYPE_NONE 0x04
ChanNum	Channel number Range of values 0 ... 0x8000
ChanProp	Channel properties from "PNIOD_build_chan_prop_sync()"
ChanErrType	Channel error type For possible values see header file "pniousrd.h"
ExtChanErrType	Additional information on the channel error For possible values, see the PNIO standard.
ExtChanAddValue	Additional information on the channel error can be obtained with this. For possible values, see the PNIO standard.
QuaChanQualifier	Further additional information on the channel error can be obtained with this. For possible values, see the PNIO standard.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_POINTER
- PNIO_ERR_OS_RES

4.6.4 PNIOD_init_diag_generic_add_sync()

Description

Generates a default structure for "PNIOD_diag_generic_add_sync()".

Syntax

```
PNIO_UINT32 PNIO_CODE_ATTR PNIOD_init_diag_generic_add_sync(
 PNIOD_DIAG_GENERIC_ADD_SYNC_PARAMS_TYPE **ppParams,
 PNIO_UINT32 DevHndl
);
```

Parameter

Name	Description	

ppParams	`typedef struct pniod_diag_generic_add_sync_params_tag {`
	`  PNIO_UINT32 DevHndl;`
	`  PNIO_UINT32 Api;`
	`  PNIO_DEV_ADDR Addr;`
	`  PNIO_UINT32 DiagTag;`
	`  PNIO_UINT16 ChannelNum;`
	`  PNIO_UINT16 ChanProp;`
	`  PNIO_UINT16 UserStructIdent;`
	`  PNIO_UINT16 InfoDataLen;`
	`  PNIO_UINT8 *pInfoData;`
	`} PNIOD_DIAG_GENERIC_ADD_SYNC_PARAMS_TYPE;`	

	Structure element	**Description**
	DevHndl	DevHndl from "PNIOD_init_open_sync()" Default: 1
	Api	"Application Process Identifier", to which the relevant submodule is assigned. Default: 0
	Addr	Address of the submodule for which diagnostics will be added Default: PNIO_ADDR_GEO; all further boxes 0
	DiagTag	Unique reference to the diagnostics data record to be entered. Default: 0
	ChannelNum	Channel number Range of values 0 ... 0x8000 Default: 0x8000
	ChanProp	Channel properties from "PNIOD_build_chan_prop_sync()"
	UserStructIdent	"User structure identifier", provides information about the structure of the diagnostics data. Default: 0xFFFF

Name	Description	
	InfoDataLen	Length of the diagnostics data in bytes, max. 1404 bytes Default: 0
	pInfoData	Pointer to the diagnostics data in the network format Default: 0
DevHndl	DevHndl from "PNIOD_init_open_sync()"	

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_POINTER
- PNIO_ERR_OS_RES

4.6.5 PNIOD_diag_generic_add_sync()

Description

The "PNIOD_diag_generic_add_sync()" function is used to load a vendor-specific diagnostics data record in a subslot.

Note

After successfully calling the function, the structure is enabled and can no longer be used.

Note

If the submodule involved is part of an application relation, a diagnostics alarm must be sent following "PNIOD_diag_generic_add_sync()" to inform the IO controller about the diagnostics.

Syntax

```
PNIOD_DIAG_GENERIC_ADD_SYNC_PARAMS_TYPE* PNIOD_diag_generic_add_sync(
 PNIO_UINT32 DevHndl
);
typedef struct pniod_diag_generic_add_sync_params_tag {
    PNIO_UINT32          DevHndl;
    PNIO_UINT32          Api;
    PNIO_DEV_ADDR        Addr;
    PNIO_UINT32          DiagTag;
    PNIO_UINT16          ChannelNum;
    PNIO_UINT16          ChanProp;
    PNIO_UINT16          UserStructIdent;
    PNIO_UINT16          InfoDataLen;
    PNIO_UINT8          *pInfoData;
} PNIOD_DIAG_GENERIC_ADD_SYNC_PARAMS_TYPE;
```

Parameter

Name	Description
DecHndl	DevHndl from "PNIOD_init_open_sync()"
Api	"Application process identifier" for the submodule
Addr	Address of the submodule
DiagTag	Unique reference to the diagnostics data record to be entered.
ChannelNum	Channel number Range of values 0 ... 0x8000
ChanProp	Channel properties from "PNIOD_build_chan_prop_sync()" Default: 0xFFFF
UserStructI-dent	"User structure identifier", provides information about the structure of the diagnostics data
InfoDataLen	Length of the diagnostics data in bytes, maximum 1404 bytes
pInfoData	Pointer to the diagnostics data in the network format

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_POINTER
- PNIO_ERR_PRM_BUF
- PNIO_ERR_OS_RES

4.6.6 PNIOD_diag_remove_sync()

Description

The "PNIOD_diag_remove_sync()" function is used to remove a diagnostics data record from a subslot. The type of diagnostics data record does not matter (channel or vendor-specific diagnostics).

Note

If the submodule involved is part of an application relation, a diagnostics alarm must be sent following "PNIOD_diag_remove_sync()" to inform the IO controller about the removal of the diagnostics.

Syntax

```
PNIO_UINT32 PNIOD_diag_remove_sync(
 PNIO_UINT32 DevHndl,
 PNIO_UINT32 Api,
 PNIO_DEV_ADDR *pAddr,
 PNIO_UINT32 DiagTag
);
```

Parameter

Name	Description
DevHndl	DevHndl from "PNIOD_init_open_sync()"
Api	"Application process identifier" for the submodule
pAddr	Pointer to the address of the submodule for which diagnostics will be removed
DiagTag	Unique reference to the diagnostics data record to be entered.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_BUF
- PNIO_ERR_OS_RES

4.6.7 PNIOD_init_alarm_send_async()

Description

Generates a default structure for "PNIOD_alarm_send_async()".

Syntax

```
PNIO_UINT32 PNIO_CODE_ATTR PNIOD_init_alarm_send_async(
 PNIOD_ALARM_SEND_ASYNC_PARAMS_TYPE **ppParams,
 PNIO_UINT32 DevHndl
);
```

Parameter

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>ppParams</td><td>
<pre>typedef struct pniod_alarm_send_async_params_tag {
    PNIO_UINT32   DevHndl;
    PNIO_UINT32   Api;
    PNIO_UINT16   SessionKey;
    PNIO_UINT16   AlarmType;
    PNIO_DEV_ADDR   Addr;
    PNIO_UINT8   *pData;
    PNIO_UINT32   DataLen;
    PNIO_UINT16   UserStructIdent;
    PNIO_UINT32   UserHndl;
} PNIOD_ALARM_SEND_ASYNC_PARAMS_TYPE;</pre>
<table>
<tr><th>Structure element</th><th>Description</th></tr>
<tr><td>DevHndl</td><td>DevHndl from "PNIOD_init_open_sync()"
Default: 1</td></tr>
<tr><td>Api</td><td>"Application Process Identifier", to which the relevant submodule is assigned.
Default: 0</td></tr>
<tr><td>SessionKey</td><td>"Session key" for unique identification of the relevant AR
Default: 0</td></tr>
</table>
</td></tr>
</table>

Name	Description	
	AlarmType	Type of the alarm to be sent Default: 0 (PNIOD_ALARM_TYPE_NONE) For possible values, see the pniousrd.h **Note on alarm type "PNIOD_ALARM_TYPE_RET_OF_SUB":** Parameters of the function "PNIOD_alarm_send_async()" AlarmType = PNIOD_ALARM_TYPE_RET_OF_SUB pData = NULL DataLen = 0 UserStructIdent = 0 The other parameters must be allocated according to the example program "dev_handle_alerts".
	Addr	Address of the submodule for which diagnostics will be added Default: PNIO_ADDR_GEO; all further boxes 0
	pData	Pointer to a data buffer that contains the alarm data in the network format See sample programs and the PNIO standard Default: NULL
	DataLen	Length of the alarm data in bytes Default: 0
	UserStructIdent	"User structure identifier" Default: 0xFFFF
	UserHndl	Handle from the area 0...0x3FFFFFFF to be freely assigned by the user program for unique identification of the alarm by "PNIOD_CBF_SYNC_ALARM_DONE()" on arrival of the acknowledgement
DevHndl	DevHndl from "PNIOD_init_open_sync()"	

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_INTERNAL
- PNIO_ERR_SEQUENCE

Error code for NULL value as parameter: PNIO_ERR_NO_RESOURCE

4.6.8 PNIOD_alarm_send_async()

Description

The "PNIOD_alarm_send_async()" function is used to send an alarm. Depending on the content of “AlarmType”, diagnostics, process, return of submodule or upload and storage alarms can be sent. The arrival of the alarm acknowledgement is signaled by calling the "PNIOD_CBF_SYNC_ALARM_DONE" callback.

Note

The structure generated by "PNIOD_alarm_send_async_init()" is released again after a successful function call. This is done by the PNIO library and can no longer be used afterwards.

Note

The relevant submodule must be in an application in relation to an IO controller so that an alarm can be sent.

Note

When sending diagnostics alarms, a corresponding diagnostics entry must be generated or removed (with an outgoing alarm) so that the diagnostics can be read out using PNIO data records.

Note

Only one alarm may ever be sent at one time; in other words, the user program must first wait for "PNIOD_CBF_SYNC_ALARM_DONE" after each "PNIOD_alarm_send_async()" call before a further alarm can be sent.

Syntax

```
PNIO_UINT32 PNIOD_alarm_send_async(
PNIOD_ALARM_SEND_ASYNC_PARAMS_TYPE **ppParam
);
typedef struct pniod_alarm_send_async_params_tag {
    PNIO_UINT32          DevHndl;
    PNIO_UINT32          Api;
    PNIO_UINT16          SessionKey;
    PNIO_UINT16          AlarmType;
    PNIO_DEV_ADDR        Addr;
    PNIO_UINT8         *pData;
    PNIO_UINT32          DataLen;
    PNIO_UINT16          UserStructIdent;
    PNIO_UINT32          UserHndl;
} PNIOD_ALARM_SEND_ASYNC_PARAMS_TYPE;
```

Parameter

Name	Description
DevHndl	DevHndl from "PNIOD_init_open_sync()", see PNIO standard
Api	"Application process identifier" for the submodule
SessionKey	"Session key" for unique identification of the relevant AR
AlarmType	Type of the alarm to be sent
Addr	Address of the submodule
pData	Pointer to a data buffer that contains the alarm data in the network format See sample programs and the PNIO standard
DataLen	Length of the alarm data in bytes
UserStructIdent	"User structure identifier"
UserHndl	Handle from the area 0...0x3FFFFFFF to be freely assigned by the user program for unique identification of the alarm by "PNIOD_CBF_SYNC_ALARM_DONE()" on arrival of the acknowledgement

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_BUF
- PNIO_ERR_OS_RES
- PNIO_ERR_PRM_POINTER

4.6.9 PNIOD_CBF_SYNC_ALARM_DONE

Description

The "PNIOD_CBF_SYNC_ALARM_DONE" callback indicates that the IO controller has acknowledged a previously sent alarm.

Syntax

```
typedef void (PNIOD_CBF_SYNC_ALARM_DONE)(
PNIOD_CBF_SYNC_ALARM_DONE_PARAMS_TYPE *pParam
);
typedef struct pniod_cbf_sync_alarm_done_params_tag {
    PNIO_UINT32          DevHndl;
    PNIO_UINT32          UserHndl;
    PNIO_UINT32          Status;
    PNIO_ERR_STAT        PnioState;
} PNIOD_CBF_SYNC_ALARM_DONE_PARAMS_TYPE;
```

Parameter

Name	Description
DevHndl	DevHndl from "PNIOD_init_open_sync()"
UserHndl	UserHndl assigned by the user program in the function "PNIOD_init_alarm_send_async()".
Status	If successful, "PNIO_OK" is returned. If an error occurs see header file "pnioerrx.h"
Pni-oState	PNIO error code transferred with the alarm acknowlwdgment.

Return values

No return values.

4.7 Interface for reading and writing IO data

Overview

The following functions are available for an I-device for read and write IO data:

- PNIOD_trigger_data_read_sync()
- PNIO_CBF_DATA_READ()
- PNIOD_trigger_data_write_sync()
- PNIO_CBF_DATA_WRITE()

These are described in detail in the following sections.

Note

Other functions are available for IRT mode as described in the section "IRT mode (Page 80)".

4.7.1 PNIOD_trigger_data_read_sync()

Description

The "PNIOD_trigger_data_read_sync()" function triggers the I-device application to read IO data once for the specified submodule. This function applies both to RT and IRT mode.

Note

If you receive the return value "PNIO_ERR_ASYNCBUF_TOGGLE_TIMEOUT", this means that your driver porting did not react to the interrupt of the communications processor in good time. This is however necessary so that you receive current RT process data.

Syntax

```
PNIO_UINT32 PNIOD_trigger_data_read_sync(
 PNIO_UINT32 DevHndl,
 PNIO_DEV_ADDR *pAddr,
 PNIO_ACCESS_ENUM AccessType
);
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
pAddr	Pointer to the address of the submodule for which "PNIO_CBF_DATA_READ" will be called If the parameter "pAddr" has the value "NULL", the data of all submodules will be read. This means that for every submodule "PNIO_CBF_DATA_READ" is then called.
AccessType	Specifies the type of access to the IO data. Possible values are: PNIO_ACCESS_RT_WITH_LOCK, if the submodule is part of an RT AR. PNIO_ACCESS_IRT_WITHOUT_LOCK, if the submodule is part of an IRT AR.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_BUF
- PNIO_ERR_WRONG_HND
- PNIO_ERR_ASYNCBUF_TOGGLE_TIMEOUT
- PNIO_WARN_IRT_INCONSISTENT
- PNIO_WARN_NO_SUBMODULES
- PNIO_ERR_SESSION

4.7.2 PNIO_CBF_DATA_READ()

Description

The "PNIO_CBF_DATA_READ" callback prompts you to read the IO data of a submodule stored in the data buffer. In addition to this, the local status (IOCS) must be transferred with the return value.

Syntax

```
PNIO_IOXS (PNIO_CBF_DATA_READ)(
 PNIO_UINT32 DevHndl,
 PNIO_DEV_ADDR *pAddr,
 PNIO_UINT32 BufLen,
 PNIO_UINT8 *pBuffer,
 PNIO_IOXS Iops
);
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
pAddr	Pointer to the address of the submodule for which the IO data will be read
BufLen	Length of the data buffer to be read
pBuffer	Data buffer from which the data will be read.
Iops	Provider status Describes the controller status of the submodule.

Return values

Possible return values:

- PNIO_S_GOOD: The submodule can output the data.
- PNIO_S_BAD: The submodule is not functioning.

Note

When this function is called for the first time after calling "PNIOD_prm_end_async_rsp" and the application is ready to provide the cyclic data, the return value must be set to "PNIO_S_GOOD". If this is not possible because the application is not ready yet, the return value is set to "PNIO_S_GOOD" at a later time. A 'Return-Of-Submodule-Alarm' must be sent to the IO controller with the help of the "PNIOD_alarm_send_async" in this case.

4.7.3 PNIOD_trigger_data_write_sync()

Description

The "PNIOD_trigger_data_write_sync()" function triggers the writing of IO data for the specified submodule(s) once. Following this, the "PNIO_DEVICE_CBF_DATA_WRITE" callback is called for all submodules involved with which the user program can then write the corresponding IO data to the process image. This function applies both to RT and IRT mode.

Note

If you receive the return value "PNIO_ERR_ASYNCBUF_TOGGLE_TIMEOUT", this means that your driver porting did not react to the interrupt of the communications processor in good time. This is however necessary so that you receive current RT process data.

Syntax

```
PNIO_UINT32 PNIO_CODE_ATTR PNIOD_trigger_data_write_sync(
 PNIO_UINT32 DevHndl,
 PNIO_DEV_ADDR *pAddr,
 PNIO_ACCESS_ENUM AccessType
);
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
pAddr	Pointer to the address of the submodule for which "PNIO_CBF_DATA_WRITE()" will be called If the parameter "pAddr" has the value "NULL", the data of all submodules will be written. This means that for every submodule "PNIO_CBF_DATA_WRITE" is then called.
AccessType	Specifies the type of access to the IO data. Possible values are: PNIO_ACCESS_RT_WITH_LOCK, if the submodule is part of an RT AR. PNIO_ACCESS_IRT_WITHOUT_LOCK, if the submodule is part of an IRT AR.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_ASYNCBUF_TOGGLE_TIMEOUT
- PNIO_ERR_PRM_ACCESS_TYPE
- PNIO_WARN_NO_SUBMODULES

4.7.4 PNIO_CBF_DATA_WRITE( )

Description

The "PNIO_CBF_DATA_WRITE" callback prompts you to write the IO data for the relevant submodule to the specified data buffer. In addition to this, the local status (IOPS) must be transferred with the return value.

Syntax

```
PNIO_IOXS (PNIO_CBF_DATA_WRITE)(
 PNIO_UINT32 DevHndl,
 PNIO_DEV_ADDR *pAddr,
 PNIO_UINT32 BufLen,
 PNIO_UINT8 *pBuffer,
 PNIO_IOXS Iocs
);
```

Parameter

Name	Description
DevHndl	"DevHndl" from "PNIOD_init_open_sync()"
pAddr	Pointer to the address of the submodule for which the IO data will be written.
BufLen	Length of the data buffer to be written
pBuffer	Data buffer to which the data will be written
Iocs	Consumer status Describes the controller status of the submodule

Return values

Possible return values:

- PNIO_S_GOOD: The submodule transfers correct data.
- PNIO_S_BAD: The submodule is not functioning.

Note

When this function is called for the first time after calling "PNIOD_prm_end_async_rsp" and the application is ready to provide the cyclic data, the return value must be set to "PNIO_S_GOOD". If this is not possible because the application is not ready yet, the return value is set to "PNIO_S_GOOD" at a later time. A 'Return-Of-Submodule-Alarm' must be sent to the IO controller with the help of the "PNIOD_alarm_send_async" in this case.

4.7.5 IRT mode

Overview

This interface consists of the following functions and callback events:

- PNIO_CP_register_cbf()
- PNIO_CP_CBE_APP_START_IND
- PNIO_CP_set_opdone()
- PNIO_CP_CBE_APP_FAULT_IND
- PNIO_CP_CBE_BUSCYCLE_IND

Note

The functions "PNIOD_trigger_data_read_sync()" and "PNIOD_trigger_data_write_sync()" may only be called between the "PNIO_CP_CBE_APP_START_IND" callback and the acknowledgment with the "PNIO_CP_set_opdone()" function.

4.7.5.1 PNIO_CP_register_cbf()

Description

This registers a callback.

Note

Callbacks can only be registered by the IO device user program prior to "PNIOD_start_sync()".

Note

The "PNIO_CP_CBE_APPL_FAULT_IND" callback must be registered before "PNIO_CP_CBE_APPL_START_IND" callback.

Syntax

```
PNIO_UINT32 PNIO_CP_register_cbf(
 PNIO_UINT32 CpHandle,
 PNIO_CP_CBE_TYPE CbeType,
 PNIO_CP_CBF Cbf
);
```

Parameter

Name	Description
CpHandle	Handle from "PNIOD_init_open_sync()"
CbeType	Callback event type for which the "Cbf" callback will be registered
Cbf	Address of the callback that will be started after the arrival of the "CbeType" callback. **Note** The function pointer must not be NULL. **Note** For a callback from which a callback is already registered, a callback can only be registered again after"PNIOD_close_sync()".

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_ALLREADY_DONE
- PNIO_ERR_INTERNAL
- PNIO_ERR_MODE_VALUE
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_NEWCYCLE_SEQUENCE_REG
- PNIO_ERR_INVALID_CONFIG

4.7.5.2 PNIO_CP_CBE_APPL_START_IND

Description

The "PNIO_CP_CBE_APP_START_IND" callback signals the end of the IRT data transfer phase and the transfer of the IRT input data from the process image to the host memory by DMA to its I-device user program.

From this point onwards, all IRT input data is in the host memory. This allows your user program to access consistent IRT data.

Note

Since before completion of the initialization phase ("PNIOD_CBF_ASYNC_INDATA_IND" / "PNIOD_indata_async_rsp") the output data is not yet valid, this may only be read afterwards.

The following syntax shows the specific parameters for this event as part of the "union" from the PNIO_CP_CBE_PRM structure; see the section "PNIO_CP_CBE_PRM (Page 86)".

Syntax

```
typedef struct {
 PNIO_UINT32 CpHandle;
 PNIO_CYCLE_INFO CycleInfo;
} ATTR_PACKED PNIO_CP_CBE_PRM_APP_START_IND;
```

Parameter

Name	Description
CpHandle	Handle from "PNIOD_open_sync()"
CycleInfo	Information on the current cycle

Note

Within the "PNIO_CP_CBE_APP_START_IND" callback, you must not call any functions that put the real-time capability of your I-device user program at risk. This means functions that take longer to process such as file operations or screen displays. Refer to the description of your operating system to find out which functions might be involved in this situation.

As a general recommendation, we advise you to restrict yourself to use of the functions for accessing the IRT data of the I-device user interface.

Return values

No return values.

4.7.5.3 PNIO_CP_set_opdone()

Description

By calling this function, the I-device user program signals the I-device interface that it has completed isochronous processing of the IRT IO data. The I-device interface than initiates transmission of the IRT output data from the host memory to the process image by DMA.

Syntax

```
PNIO_UINT32 PNIO_CP_set_opdone(
 PNIO_UINT32 CpHandle;
 PNIO_CYCLE_INFO *CycleInfo;
);
```

Parameter

Name	Description
CpHandle	Handle from "PNIOD_init_open_sync()"
CycleInfo	Information on the current cycle in which this call was executed. If the pointer was "0", nothing is returned.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_WRONG_HND

4.7.5.4 PNIO_CP_CBE_APPL_FAULT_IND

Description

The "PNIO_CP_CBE_APPL_FAULT_IND" callback signals the violation of the isochronous mode to your I-device user program. This means that your user program called "PNIO_CP_set_opdone()" too late after receiving the "PNIO_CP_CBE_APPL_START_IND" callback.

The following syntax shows the specific parameters for this event as part of the "union" from the PNIO_CP_CBE_PRM structure; see the section "PNIO_CP_CBE_PRM (Page 86)".

Syntax

```
typedef struct {
 PNIO_UINT32 CpHandle;
 PNIO_CYCLE_INFO CycleInfo;
} ATTR_PACKED PNIO_CP_CBE_PRM_OPFAULT_IND;
```

Parameter

Name	Description
CpHandle	Handle from"PNIOD_init_opne_sync()"
CycleInfo	Information on the current cycle

Return values

No return values.

4.7.5.5 PNIO_CP_CBE_BUSCYCLE_IND

Description

The "PNIO_CP_CBE_BUSCYCLE_IND" callback event signals the start of a new bus clock rate to the I-device user program. The event is only signaled when it was registered using "PNIO_CP_register_cbf( )".

The following syntax shows the specific parameters for this event as part of the "union" from the "PNIO_CP_CBE_PRM" structure; see the section "PNIO_CP_CBE_PRM (Page 86)".

Syntax

```
typedef struct {
 PNIO_UINT32 CpHandle;
 PNIO_CYCLE_INFO CycleInfo;
} ATTR_PACKED PNIO_CP_CBE_PRM_BUSCYCLE_IND;
```

Parameter

Name	Description
CpHandle	Handle from "PNIOD_init_open_sync()"
CycleInfo	Information on the current cycle

Return values

No return values.

4.7.5.6 PNIO_CP_CBE_PRM

Description

The various callbacks have the same data type "PNIO_CP_CBE_PRM" that groups the various parameters of the individual callbacks using a "union".

Syntax

```
typedef struct {
 PNIO_CP_CBE_TYPE CbeType;
 PNIO_UINT32 CpIndex;
union
{ ...
}u;
} ATTR_PACKED PNIO_CP_CBE_PRM;
```

Parameter

Name	Description
CbeType	Callback event
CpIndex	"CpIndex" from "PNIOD_init_open_sync()"

Return values

No return values.

4.8 Interface for support of the Asset Management Record

You can find information on Asset Management data records in the following document:

Profile Guidelines Part 1

Identification & Maintenance Functions

Guideline for PROFIBUS and PROFINET

Version V2.1 Date: May 2016

Order No.: 3.502

4.8.1 PNIOD_insert_asset_block()

Description

With this function, the user program provides an Asset Management block for forming the Asset Management data record. The Asset Management data record can consist of multiple blocks, as long as the maximum length for data records is not overwritten (PNIO-MAX_REC_LEN in the file pniousrx.h). After the Asset Management data record is formed (Index =0xF880), it can be read by the connected PNIO controller. The Asset Management data record can be expanded during operation by calling this function or reduced by removing Asset Management blocks with the function "PNIOD_remove_asset_block". The Asset Management data record cannot be written by the PROFINET IO controller. The data record read job for the Asset Management data record is answered by the PNIO library and not forwarded to the user program.

Syntax

```
PNIO_UINT32 PNIOD_insert_asset_block (
 PNIO_UINT32 DevHndl,
 AM_BLOCK_TYPE *pBlock,
 AM_BLOCK_HANDLE *pBlockHdl);
```

Parameters

Name	Description
DevHdl	"DevHndl" from "PNIOD_init_open_sync()"
pBlock	Pointer to a block of the type "AM_BLOCK_TYPE"
pBlockHdl	Pointer to the block handle; is used to remove the block.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_PRM_LEN
- PNIO_ERR_OS_RES
- PNIO_ERR_PRM_INVALIDARG
- PNIO_ERR __UNIQUE_IDENTIFIER_NOT_SET,
- PNIO_ERR __LOCATION_FORMAT_INVALID,
- PNIO_ERR __LOCATION_LEVEL_INVALID,
- PNIO_ERR __ANNOTATION_TOO_LARGE,
- PNIO_ERR __ORDER_ID_TOO_LARGE,
- PNIO_ERR __BOTH_HARDWARE_VERSIONS_SET,
- PNIO_ERR __HARDWARE_VERSION_LONG_TOO_LARGE,
- PNIO_ERR __BOTH_SOFTWARE_VERSIONS_SET,
- PNIO_ERR __SOFTWARE_VERSION_LONG_TOO_LARGE,
- PNIO_ERR __SERIAL_NUMBER_TOO_LARGE

4.8.2 PNIOD_remove_asset_block()

Description

With this function, a block created previously with the "PNIOD_insert_asset_block()" function is removed from the Asset Management data record.

Syntax

```
PNIO_UINT32 PNIO_CODE_ATTR PNIOD_remove_asset_block(
 PNIO_UINT32 DevHndl,
 AM_BLOCK_HANDLE BlockHdl);
```

Parameters

Name	Description
DevHdl	"DevHndl" from "PNIOD_init_open_sync()"
BlockHdl	Handle to a block which was transferred by the "PNIOD_insert_asset_block" function.

Return values

If successful, "PNIO_OK" is returned.

If an error occurs, the following values are possible (for the meaning, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_PRM_HND

4.8.3 Data type "AM_BLOCK_TYPE"

Syntax

```
typedef struct am_block_tag {
 PNIO_UINT8 UniqueIdentifier[AM_UNIQUE_IDENTIFIER_SIZE];
 AM_LOCATION_TYPE Location;
 PNIO_UNICODE_STRING_8* Annotation;
 PNIO_UNICODE_STRING_8* OrderId;
 AM_REVISION_TYPE Revision;
 PNIO_UNICODE_STRING_8* SerialNumber;
 PNIO_UINT16 DeviceSubId;
 PNIO_UINT16 DeviceId;
 PNIO_UINT16 VendorId;
 PNIO_UINT16 OrganizationId;
 PNIO_UINT16 TypeIdentification;
} AM_BLOCK_TYPE;
```

For the definition of the other data types required for the "AM_BLOCK_TYPE" data type, see header file "pniousrd.h".

Sample programs 5

Sample program "dev_rw_digital_io"

In this example, the basic steps are taken to initialize the I-device and to start the cyclic RT data exchange.

You have option of writing and reading data.

Sample program "dev_rw_im"

In this sample program apart from the cyclic RT data exchange the receipt or sending of I&M data records is also shown.

Received I&M data is stored persistently and are preset automatically the next time you start the program. You have the option of resetting the stored I&M data back to the default values.

Sample program "dev_handle_alerts"

In this sample program apart from the cyclic RT data exchange the handling of I-device interrupts is also shown.

You have option of triggering a diagnostic interrupt, an extended diagnostic interrupt or a process interrupt or reset it again.

Sample program "dev_rw_irt_io"

In this example, the basic steps are taken to initialize the I-device and to start the cyclic IRT data exchange.

The values of the output modules are transferred to the input modules. At the same time you have the option of displaying the statistics of the data exchange.

Sample program "dev_certify_energy"

This sample program is used for PNIO certification.

All the functions relevant for the certification are supported: Startup RT or IRT, hot restart, read and write I&M data and handling PROFIenergy.

Overview of the changes from the device interface to the I-device interface

6

The following changes in the functions of the device interface to the I-device interface have taken place:

Device interface	what changes	I-device interface
PNIO_device_open()	is replaced by	PNIOD_init_open_sync() PNIOD_open_sync()
PNIO_device_open_ext()	omitted	-
PNIO_set_appl_state_ready()		
PNIO_device_ar_abort()		
PNIO_device_close()	is replaced by	PNIOD_close_sync()
PNIO_device_start()	is replaced by	PNIOD_start_sync()
PNIO_device_stop()	is replaced by	PNIOD_stop_async()
PNIO_set_dev_state()	omitted The available IO modules of the I-device are specified by the configuation. Since the I-device of the CP 1626 / CP 1616/ CP 1604 has no IO modules that can be plugged or pulled, these functions are omitted.	-
PNIO_api_add()		
PNIO_api_add_ext()		
PNIO_api_remove()		
PNIO_mod_pull()		
PNIO_sub_pull()		
PNIO_sub_plug()		
PNIO_sub_plug_ext()		
PNIO_sub_plug_ext_IM()		
PNIO_mod_plug()		
PNIO_build_channel_properties()	is replaced by	PNIOD_build_chan_prop_sync()
PNIO_diag_channel_add()	is replaced by	PNIOD_init_diag_channel_add_sync() PNIOD_diag_channel_add_sync()
PNIO_diag_channel_remove()	is replaced by	PNIOD_diag_remove_sync()
PNIO_diag_generic_add()	is replaced by	PNIOD_init_diag_generic_add_sync() PNIOD_diag_generic_add_sync()
PNIO_diag_generic_remove()	omitted	-
PNIO_diag_ext_channel_add()		
PNIO_diag_ext_channel_remove()		
PNIO_process_alarm_send() PNIO_diag_alarm_send()	are replaced by	PNIOD_alarm_send_async() PNIOD_init_alarm_send_async()
PNIO_ret_of_sub_alarm_send()	omitted	-
PNIO_initiate_data_read()	is replaced by	PNIOD_trigger_data_read_sync()
PNIO_initiate_data_read_ext()		

Device interface	what changes	I-device interface
PNIO_initiate_data_write()	is replaced by	PNIOD_trigger_data_write_sync()
PNIO_initiate_data_write_ext()		
-	new	PNIOD_get_config_sync()
		PNIOD_rec_write_async_rsp()
		PNIOD_rec_read_async_rsp()
		PNIOD_indata_async_rsp()
		PNIOD_irt_init_inputs_async_rsp()
		PNIOD_prm_end_async_rsp()
		PNIOD_ownership_async_rsp()
		PNIOD_connect_async_rsp()

Note

You will find further details in this programming manual and in the programming manual "IO-Base user programming interface".

The following changes in the callback functions of the device interface to the I-device interface have taken place:

Device interface	what changes	I-device interface
cbf_data_write()	remains the same	cbf_data_write()
cbf_data_read()	remains the same	cbf_data_read()
cbf_rec_read()	is replaced by	cbf_async_rec_read()
cbf_rec_write()	is replaced by	cbf_async_rec_write()
cbf_alarm_done()	is replaced by	cbf_sync_alarm_done()
cbf_check_ind()	is replaced by	cbf_async_connect_ind()
cbf_ar_check_ind()	is replaced by	cbf_async_ownership_ind()
cbf_ar_info_ind()	omitted	-
cbf_ar_indata_ind()	is replaced by	cbf_async_indata_ind()
cbf_ar_abort_ind()	is replaced by	cbf_sync_disconnect_ind()
cbf_ar_offline_ind()		
cbf_apdu_status_ind()	is replaced by	cbf_sync_data_status_ind()
cbf_prm_end_ind()	is replaced by	cbf_async_prm_end_ind()
cbf_cp_stop_req()	is replaced by	cbf_sync_cp_stop_req()
cbf_device_stopped()	is replaced by	cbf_sync_device_stopped()
-	new	cbf_async_irt_init_inputs()
cbf_start_led_flash()	is replaced by	cbf_sync_start_led_flash()
cbf_stop_led_flash()	is replaced by	cbf_sync_stop_led_flash()
cbf_pull_plug_conf()	omitted	-

Note

You will find further details in this programming manual and in the programming manual "IO-Base user programming interface".