SIEMENS

SIMATIC NET

PROFINET
IO-Base user programming interface

Programming Manual

02/2017

C79000-G8976-C195-20

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

Document order number: C79000-G8976-C195
℗ 02/2017 Subject to change

Preface

SIMATIC NET – IO-Base 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 IO-Base 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 IO-Base 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 IO controller or an IO device (CP 1604/CP 1616 from V2.0 to V2.6) with the IO-Base user programming interface.

Changes compared with the previous version

Description of the functions for the CP 1626.

Overview of the documentation

We recommend that you familiarize yourself with the following documentation before you create an IO-Base 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	Getting Started guides 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.

Name of the document	Why you should read it
Readme file • for "SIMATIC NET PC Software" DVD • for CD "DK-16xx PN IO"	Here, you will find the latest information on the SIMATIC NET PC software products.
Installation Manual for "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).
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 when you create an IO-Base user program.
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 experience with the STEP 7 Professional (TIA Portal) or NCM PC configuration tools

Scope of the manual

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

- CP 1626
- CP 1616
- CP 1604
- SOFTNET PN IO

- DK-16xx PN IO
- PN driver

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).

See also

http://support.automation.siemens.com/ (http://support.automation.siemens.com/)

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:

(http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo2&aktprim=99&lang=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: (https://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&lang=en&objid=38718979&caller=view)
- 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: (http://www.automation.siemens.com/partner/guiwelcome.asp?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:

(www.siemens.com/sitrain)

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 only form one element of such a concept.

Customer is responsible to prevent unauthorized access to its plants, systems, machines and networks. Systems, machines and components should only be connected to the enterprise network or the internet if and to the extent necessary and with appropriate security measures (e.g. use of firewalls and network segmentation) in place.

Additionally, Siemens' guidance on appropriate security measures should be taken into account. For more information about industrial security, please visit:
http://www.siemens.com/industrialsecurity

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends to apply product updates as soon as available and to always use the latest product versions. Use of product versions that are no longer supported, and failure to apply latest updates may increase customer's 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/zh/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)

Table of contents

Quick start with IO controller functionality 1

This chapter presents a recommended step-by-step procedure for creating a user program in the C/C++ programming language based on the IO-Base user programming interface.

In the first step, familiarize yourself with the basics of PROFINET IO. Gradually work through the steps and complete them by writing your IO-Base controller user program.

1.1 Procedure

Description

By following the steps below, you can create an IO-Base controller 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 basic characteristics of the IO-Base user programming interface. You can do this by reading the section "Overview of the IO-Base user programming interface for IO controllers (Page 21)" 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 IO-Base user programming interface: – "pniousrx.h" with the functions and data structures. – "pnioerrx.h" with the error and return codes • The "pniousrx.lib" or "libpniousr" library for linking to your IO-Base controller user program. • Sample programs and corresponding configuration.
4	Work through the source text of the sample program "PnioEasy" in the "..\Examples\easy" or "../examples/easy" subfolder and check out the functions and data structures in Chapter Description of the functions and data types for IO controllers (Page 37).
5	Based on your system configuration, find out which data needs to be sent and received (IO data, data records, alarms) and which IO devices are involved. Read the user documentation of the IO devices used.
6	Complete the STEP 7 or NCM PC configuration and download it to the devices involved.

Step	Description
7	Modify the supplied sample program "PnioEasy" 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 include the modified sample program and test it on the system you have available.
8	Now create your IO-Base controller user program covering the entire functionality.

1.2 Overview of the functionality of the products

Description

The table below provides you with an overview of the IO controller functionality that can be used in the various SIMATIC NET products.

Function groups and names	Products		
	SOFTNET PN IO (RT)	CP 1616 / CP 1604 (RT + IRT) with DK-16xx as of V2.0	PN Driver
Management functions			
PNIO_controller_open()	x	x	x
PNIO_register_cbf()	x	x	x
PNIO_controller_close()	x	x	x
PNIO_interface_open()	–	–	x
PNIO_interface_register_cbf()	–	–	x
PNIO_interface_close()	–	–	x
PNIO_interface_set_ip_and_nos()	–	–	x
Functions relating to the mode			
PNIO_set_mode()	x	x	x
Callback event PNIO_CBE_MODE_IND	x	x	x
PNIO_device_activate()	x	x	x
Callback event PNIO_CBE_DEV_ACT_CONF	x	x	x
Functions for reading IO data			
PNIO_data_read()	x	x	x
PNIO_output_data_read()	–	x	–
PNIO_data_read_cache_refresh()	–	x	–
PNIO_data_read_cache()	–	x	–
Functions for writing IO data			
PNIO_data_write()	x	x	x
PNIO_data_write_cache()	–	x	–
PNIO_data_write_cache_flush()	–	x	–
Interface for read data record			

Function groups and names	Products		
PNIO_rec_read_req()	x	x	x
Callback event PNIO_CBE_REC_READ_CONF	x	x	x
PNIO_interface_rec_read_req()	–	–	x
PNIO_CBE_IFC_REC_READ_CONF callback event	–	–	x
Interface for write data record			
PNIO_rec_write_req()	x	x	x
Callback event PNIO_CBE_REC_WRITE_CONF	x	x	x
Alarm interface			
Callback event PNIO_CBE_ALARM_IND	x	x	x
PNIO_CBE_IFC_ALARM_IND callback event	–	–	x
Diagnostics interface			
PNIO_ctrl_diag_req()	–	x	x
Callback event PNIO_CBE_CTRL_DIAG_CONF	–	x	–
Station signaling with LEDs			
Callback event PNIO_CBE_START_LED_FLASH_IND	–	x	–
Callback event PNIO_CBE_STOP_LED_FLASH_IND	–	x	–
General			
Callback event PNIO_CBE_CP_STOP_REQ	–	x	–
Interface for isochronous real-time mode (IRT)			
PNIO_CP_register_cbf()	–	x	–
Callback event PNIO_CP_CBE_STARTOP_IND	–	x	–
Callback event PNIO_CP_CBE_OPFAULT_IND	–	x	–
PNIO_CP_set_opdone()	–	x	–
Interface for PROFIenergy (PE)			
PNIO_register_pe_cbf()	x	x	–
PNIO_pe_cmd_req()	x	x	–
Product-specific functions for the PN driver			
SERV_CP_init()	–	–	x
SERV_CP_undo_init()	–	–	x
SERV_CP_get_network_adapters()	–	–	x
SERV_CP_startup()	–	–	x
SERV_CP_shutdown()	–	–	x
SERV_CP_set_trace_level()	–	–	x
PNIO_IOS_RECONFIG_ MODE()	–	–	x

Note

Partial access to modules and submodules is available only using the communications processors CP 1604 and CP 1616.

Overview of the real-time modes 2

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

The IO-Base user programming interface supports the following modes:

- RT (Real Time)
- IRT (Isochronous Real Time)

2.1 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

In isochronous mode (IRT mode), real-time frames are transferred cyclically with high priority during a fixed period (deterministic interval).

IRT

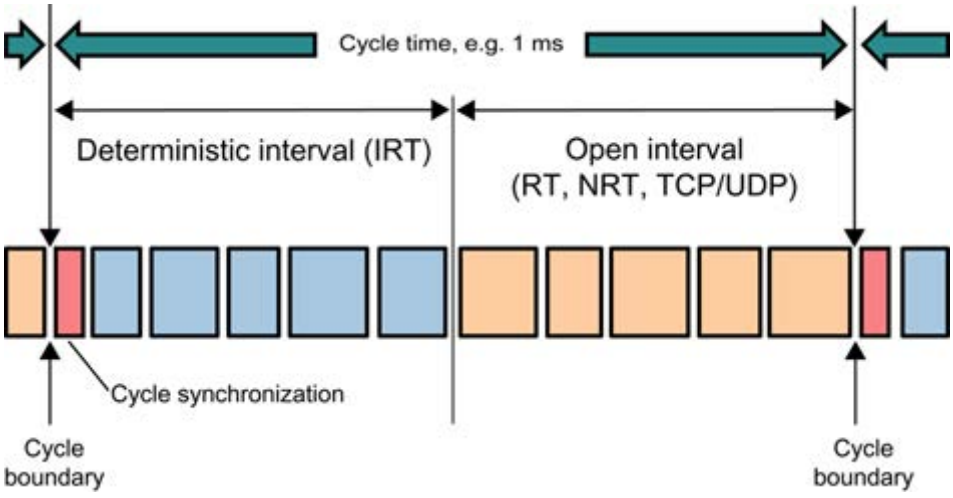
The end of the transfer interval is signaled to the IO-Base user program. This signal allows the IO-Base user program to be synchronized with the bus clock. During the open interval, RT, NRT (**N**on **R**eal **T**ime) and TCP/UDP frames are transferred.

If the IO-Base user program accesses the IRT data only outside the deterministic interval, this is known as isochronous real-time mode.

If the IRT data can be accessed at any time, the IO-Base user program is running in non-isochronous mode.

The IO-Base user programming interface provides other functions that need to be called in the initialization, operational and termination phases. The times and intervals are specified in the configuration and transferred to the IO controllers, IO devices and switches.

The topology must be planned extra.


Cycle time, e.g. 1 ms
Deterministic interval (IRT)
Open interval
(RT, NRT, TCP/UDP)
Cycle synchronization
Cycle
boundary
Cycle
boundary

Overview of the IO-Base user programming interface for IO controllers

3

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

Function calls and data access are described in detail in the Section "Description of the functions and data types for IO controllers (Page 37)".

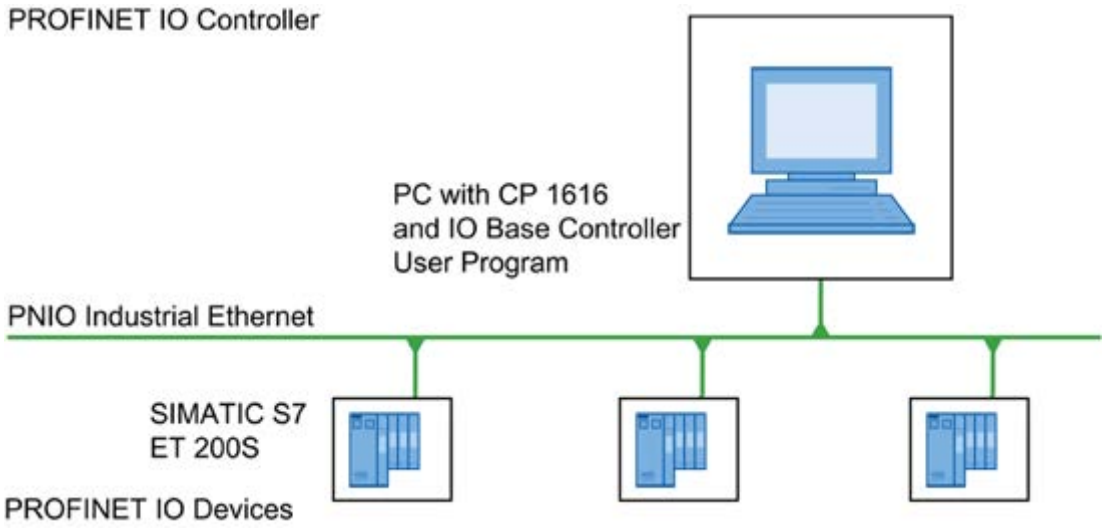
3.1 Typical use of the IO-Base user programming interface for IO controllers

Description of the application example

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

The IO-Base controller user program and the IO-Base functions of the SIMATIC NET product run on the PC. The data exchange with several PROFINET IO devices, for example ET 200S, is handled over a PROFINET communications processor or a standard Ethernet network adapter on Industrial Ethernet.

This configuration is referred to again in the supplied sample program.

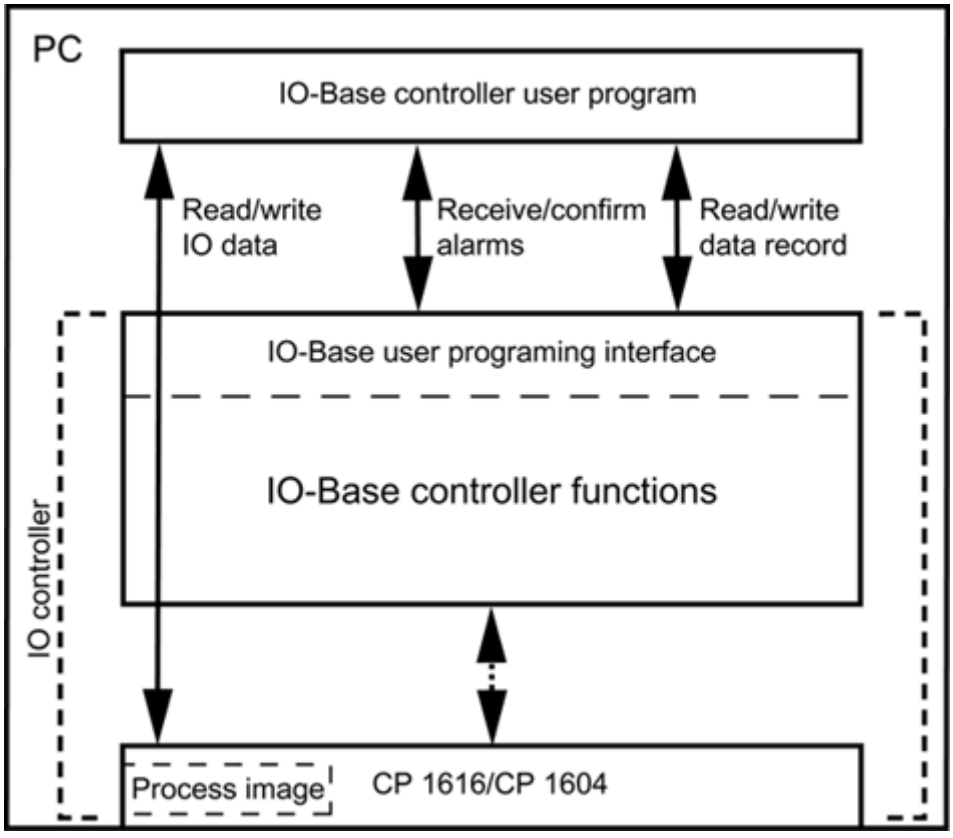


3.2 Software architecture in a PC

Description of the PROFINET I/O architecture in conjunction with a CP 1616 / CP 1604

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

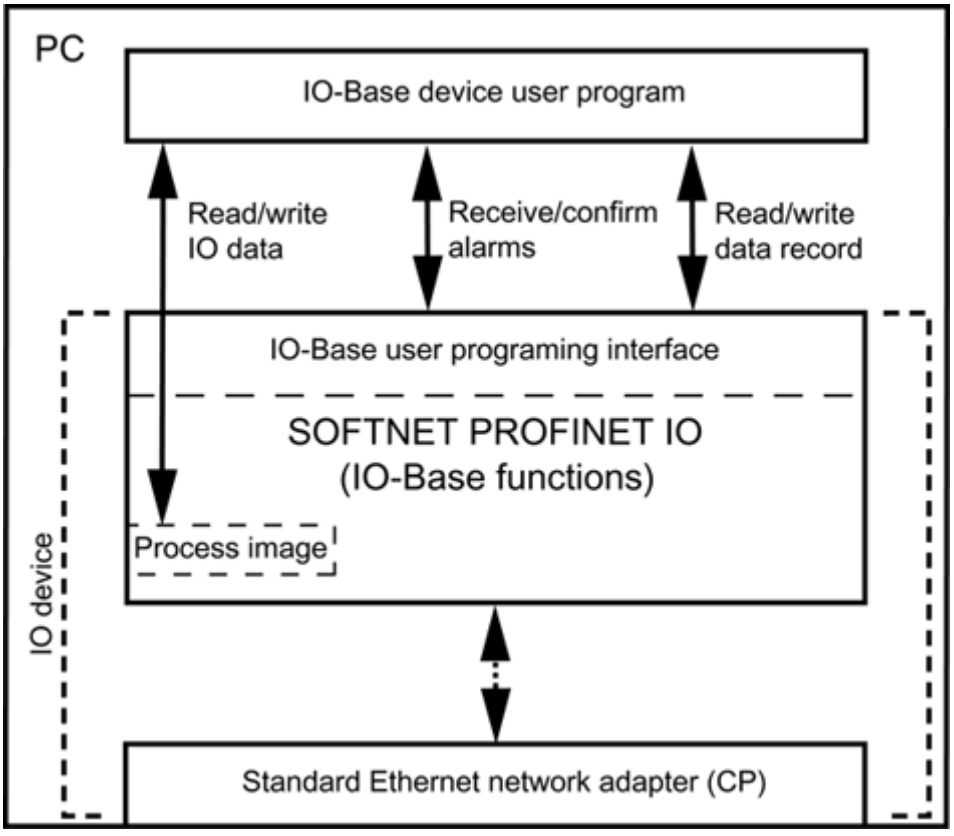
These are the IO-Base functions read/write IO data, send alarms and receive alarm confirmations and read/write data records. The PROFINET IO software is connected to the process over a PROFINET IO communications processor. The communication with the IO controller is configured with STEP 7 or NCM PC.



Description of the SOFTNET PROFINET IO architecture

With the IO-Base user programming interface, SOFTNET PROFINET IO provides all the functions that a C user program requires to communicate with PROFINET IO devices.

These are the IO-Base functions read/write IO data, receive/confirm alarms and read/write data records. In the process direction, the SOFTNET PROFINET IO software is connected to the Industrial Ethernet network over a standard Ethernet adapter (CP - communications processor). The connections to the I/O devices are configured with the STEP 7 or NCM PC software.



DLLs, header files, import libraries and sample programs

The IO-Base programming interface is a C programming interface consisting of a file. This is installed during setup.

To use the IO-Base programming interface, you require the following files:

File type	File name	Purpose
Header file	pniobase.h	This contains global structures and definitions for the IO-Base user programming interface.
Header file	pniousrx.h	Contains data types, constants and function declarations for the IO controller functionality of the IO-Base user programming interface.
Header file	pnioerrx.h	Contains the error codes
Library	pniousrx.lib or libpniousr	IO-Base programming interface for linking to your IO-Base controller user program.

3.3 Typical sequence of an IO-Base controller user program on the IO controller

Overview

The typical sequence of an IO-Base controller user program can be divided into 3 phases.

- Initialization phase
- Productive operation
- Completion phase

These are explained in detail below.

3.3.1 Initialization phase

Description

The initialization phase is divided into five steps:

Step	Action	Purpose
1	PNIO_controller_open( )	• Register the IO controller with the IO-Base user programming interface. • Register callback functions for data records and alarms.
2	PNIO_register_cbf( ) with parameter PNIO_CBE_MODE_IND	Register callback function for mode changes.
3	PNIO_CP_register_cbf( )	If you want support of the IRT configuration and therefore isochronous real-time mode, the callbacks PNIO_CP_CBE_OPFAULT_IND( ) and PNIO_CP_CBE_STARTOP_IND( ) must be registered. From this time onwards, these callbacks are signaled.
4	PNIO_set_mode( ) with parameter PNIO_MODE_OPERATE	Set controller to OPERATE mode.
5	Wait until callback event PNIO_CBE_MODE_IND occurs a second time.	The mode achieved should be OPERATE otherwise there is an error.

3.3.2 Productive operation

Overview

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

- Read/write IO data
- Receive and confirm alarms
- Read/write data record

Data exchange is explained in detail below.

Read/write IO data

The following functions allow process data to be read and written:

- PNIO_data_read( )
- PNIO_output_data_read( )
- PNIO_data_read_cache_refresh( )
- PNIO_data_read_cache( )
- PNIO_data_write( )
- PNIO_data_write_cache( )
- PNIO_data_write_cache_flush( )

Access to the IO data of PROFINET IO on the PC.

The IO-Base interface communicates with the IO devices cyclically as specified in the configuration.

For more information, refer to the section "Non-isochronous access to cyclic IO data (RT) (Page 27)".

Receive and confirm alarms

A received alarm (event PNIO_CBE_ALARM_IND) leads to the corresponding callback function being called.

For more information, refer to the section "Callback mechanism (Page 34)".

Read/write data record

The following functions allow a data record to be read or written by accessing the IO device, for example parameter assignment:

- PNIO_rec_read_req( )
- PNIO_rec_write_req( )

The requested data record or the confirmation of the sent data record is made available by the callback mechanism.

For more information, refer to the section "Callback mechanism (Page 34)".

Sequence of a read data record job

Step	Action	Purpose
1	PNIO_rec_read_req( )	Send read data record job.
2	Callback function (event PNIO_CBE_REC_READ_CONF)	Signals confirmation of the read data record job and receipt of the data record. The pointer to the data record is attached to this callback function and is therefore available for the user program.

Sequence of a write data record job

Step	Action	Purpose
1	PNIO_rec_write_req( )	Send write data record job.
2	Callback function (event PNIO_CBE_REC_WRITE_CONF)	Evaluate confirmation.

3.3.3 Completion phase

Description

The completion phase of the IO controller covers three steps:

Step	Action	Purpose
1	PNIO_set_mode( ) (OFFLINE)	Set the IO controller to the OFFLINE mode.
2	Wait until OFFLINE is achieved.	Wait until callback function (event PNIO_CBE_MODE_IND) signals the PNIO_MODE_OFFLINE mode. From this point, communication with the IO devices is discontinued.
3	PNIO_controller_close( )	Deregister the IO controller with the IO-Base user programming interface.

3.4 Basic data exchange of the IO-Base functions

Description

The IO-Base functions have three basic methods of exchanging data:

- Cyclic non-isochronous real-time IO data traffic (RT)
  - Write RT IO data
  - Read RT IO data

  The IO data exchange also includes status information.

  This special feature is described in the following section.
- Cyclic isochronous real-time IO data traffic (IRT)
  These access functions are the same as those described above. The access must relate only to IRT data and take place between the PNIO_CP_CBE_STARTOP_IND callback event and the PNIO_CP_set_opdone( ) function call.
- Acyclic IO data exchange
  - Writing and reading data records
  - Receiving alarms

  For more information, refer to the section "Callback mechanism (Page 34)".

3.5 Non-isochronous access to cyclic IO data (RT)

Non-isochronous access to cyclic IO data (RT)

Two types of access are available for cyclic data traffic:

- Direct unbuffered access to the process image
- Fast buffered access to the process image

Direct unbuffered access to the process image

With direct unbuffered access to the process image, the IO data is always written directly to the process image memory of the CP.

Direct unbuffered access to the process image is handled with the following functions:

Function	Description
PNIO_data_read( )	Immediate reading of the input data from the process image
PNIO_output_data_read( )	Immediate reading of the output data from the process image
PNIO_data_write( )	Immediate writing of the output data into the process image

As soon as one of these functions is called, data is transferred immediately to the process image.

Note

You should always use one of these types of access when only small amounts of output data change within a cycle.

Fast buffered access to the process image

With fast buffered access to the process image, access is always over a cache on the host computer. The entire process image is always transferred when the refresh and flush functions are used. Due to the full access, transmission mechanisms were used to optimize speed.

You handle fast buffered access to the process image with the following functions:

For reading
Reading involves two steps:

Step	Function	Description
1	PNIO_data_read_cache_refresh( )	Transfer of all input data and remote status of the output data from the process image to the read cache of the host computer.
2	PNIO_data_read_cache( )	Read individual input data per submodule from the read cache of the host computer.

For writing
Writing involves two steps:

Step	Function	Description
1	PNIO_data_write_cache( )	Writing the individual output data per submodule to the write cache of the host computer.
2	PNIO_data_write_cache_flush( )	Transfer of all output data and the local status of the input data from the write cache of the host computer to the process image.

Note

Buffered access is not supported with SOFTNET PN IO.

Note

To achieve the maximum benefits in terms of speed when using the cache, note the following:

- In the project engineering of a device, first arrange all submodules with input data and then all submodules with output data.
- Configure few submodules with larger data lengths instead of a lot of submodules with little data.
- Avoid submodules containing both input and output data.
- First read the input data and then write the output data.

Data transfer to the IO device

Both when writing and reading, the IO-Base functions access the process image of PROFINET IO but not the IO devices themselves.

Data exchange between the process image and the IO devices is handled automatically and cyclically by the underlying IO-Base functions.
The details of this data exchange are specified in the configuration.

Note

The IO-Base controller user program does not need to write or read RT IO data cyclically.

Note

There is no practical use in the IO-Base controller user program accessing the process image more often than the configured cycle.

IO data and data status

The quality of the IO data is described by the data status that can have the value GOOD or BAD.

Both when writing and reading, two data statuses are exchanged:

- Local status (status of your IO-Base controller user program)
- Remote status (status of the communications partner)

3.5.1 Cyclic writing with status

Sequence of the PNIO_data_write( ) and PNIO_data_write_cache( ) functions

With the PNIO_data_write( ) and PNIO_data_write_cache( ) function, not only output data for the communications partner but also the corresponding local status of this data is written. The remote status of this output data is also read from the communications partner. There are therefore two statuses involved in cyclic writing.

Communication direction	Values
to the communications partner	• Output data • Local status
from the communications partner	Remote status

Local status

Normally, the local status is set to GOOD by the IO-Base controller user program.

If the output data is bad or invalid, the IO-Base controller user program should set the local status to BAD.

The communications partner could then, for example, output configured substitute values.

Note

If there is a device failure, the IO-Base library automatically sets the data status to "BAD". When a return alarm is signaled, the user then has to initialize the IO data with the status "GOOD".

The status for the slots without data (for example the head module and with IRT the ports) is automatically set to "GOOD" by the IO-Base library.

Remote status of the communications partner

With the remote status, the communications partner signals whether it was able to process the output data successfully "good" or whether there was a problem.

This status relates to output data transferred in the last cycle from the same module and not to the write job that has just been started.

3.5.2 Cyclic reading with status

Sequence of the PNIO_data_read( ) and PNIO_data_read_cache( ) functions

With the PNIO_data_read( ) and PNIO_data_read_cache( ) functions, not only the input data is read from the communications partner but also the corresponding remote data status.

The local status of this input data is also sent to the communications partner.

There are therefore two statuses involved in cyclic reading.

Communication direction	Values
from the communications partner	• Input data • Remote status
to the communications partner	Local status

Remote status of the communications partner

With the remote status, the communications partner reports the quality of the input data (GOOD or BAD).

If the communications partner reports BAD, the IO-Base controller user program can, for example, continue processing with substitute values.

Note

If there is a device failure, the IO-Base library automatically sets the data status to "BAD". When a return alarm is signaled, the user then has to initialize the IO data with the status "GOOD".

The status for the slots without data (for example the head module and with IRT the ports) is automatically set to "GOOD" by the IO-Base library.

Local status

Normally, the local status is set to GOOD by the IO-Base controller user program.

If, however, the IO-Base controller user program cannot process the returned data, it is possible to set the local status to BAD for a read job.

As soon as it receives this status, the communications partner can recognize whether the data it has sent was processed correctly.

3.6 Isochronous real-time and non-isochronous access to cyclic IO data (IRT)

Description

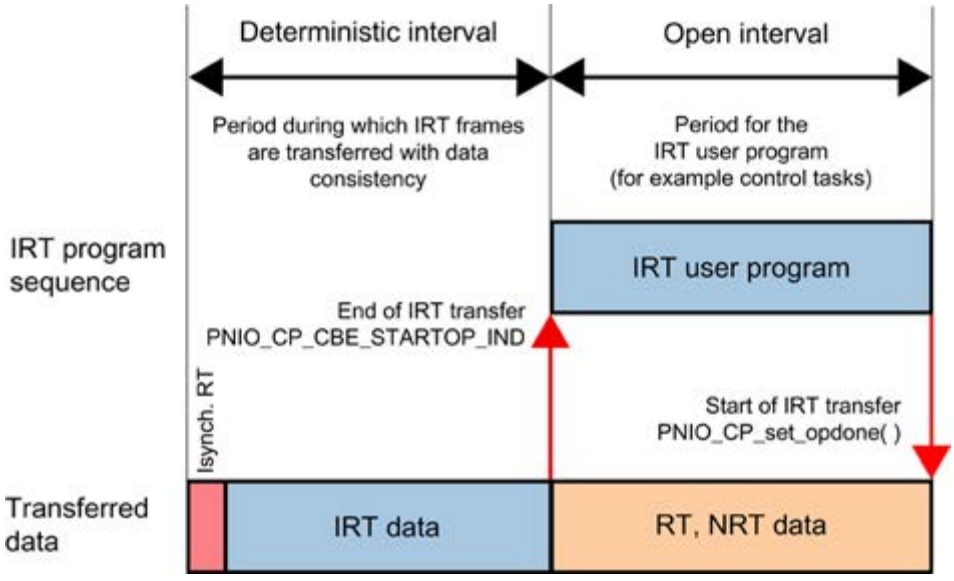
In isochronous real-time mode, the IRT user program processes and writes die IRT data between two cyclically repeated IRT times:

- IRT transfer end
- IRT transfer start

As soon as the IRT transfer end is reached, the IRT input data is transferred from the process image memory to the host memory by DMA and the IRT user program is informed by the PNIO_CP_CBE_STARTOP_IND callback event. As of this point in time, all IRT data is

in the host memory and processing of the data can begin. The first PNIO_CP_CBE_STARTOP_IND callback events are signaled after the PNIO_CP_register_cbf( ) function call.

Once the IRT user program has completed processing the data, it calls the PNIO_CP_set_opdone( ) function. This copies the IRT output data to the process image memory using DMA and informs the communications processor of the validity of the data, so that the data can be transferred in the next bus cycle.



Violation of isochronous real-time mode

If completion of the processing of the IRT data is signaled too late, the DMA transfer cannot be completed in time and the IRT output data cannot be marked as valid.

As a result, the communications processor transfers the IRT data as invalid in the next bus cycle. The user program is informed of this with the PNIO_CP_CBE_OPFAULT_IND callback event.

Consistency of the isochronous real-time IO data (IRT data)

To give the user program more time to process the data, IRT data is transferred between host memory and process image memory by DMA. As a result, IRT data is consistent only between the IRT transfer end and the IRT transfer start.

This means that your user program can only access consistent IRT data between the PNIO_CP_CBE_STARTOP_IND callback event and PNIO_CP_set_opdone( ).

When the PNIO_CP_CBE_OPFAULT_IND callback event arrives, the data is no longer consistent. This situation remains until the next PNIO_CP_CBE_STARTOP_IND callback event.

3.6.1 Access to IRT data

Isochronous real-time access to the IRT data

If IRT data is accessed between the PNIO_CP_CBE_STARTOP_IND callback event and PNIO_CP_set_opdone( ), this is known as isochronous real-time access to the IRT data. The functions for accessing IRT data are the same as for access to non-isochronous real-time cyclic RT data.

Non-isochronous real-time access to the IRT data

If IRT data is accessed outside the PNIO_CP_CBE_STARTOP_IND callback event and PNIO_CP_set_opdone( ), this is known as asynchronous access to the IRT data. Such access returns inconsistent data and this is indicated by the PNIO_WARN_IRT_INCONSISTENT warning as return value.

Identifying which data is IRT data

To allow your IO controller user program to identify the logical addresses that reference IRT data, you can use the PNIO_ctrl_diag_req( ) function with the PNIO_CTRL_DIAG_CONFIG_SUBMODULE_LIST diagnostics service.

Isochronous real-time data and access functions optimized for speed

If your IO controller user program uses access functions that are optimized for speed to access the IRT data, these accesses are forwarded to the uncached function. This makes the access functions optimized for speed for RT data transparent for IRT data.

You can therefore use the speed-optimized access functions in your isochronous real-time processing routine to access both IRT data and RT data without having to make a distinction. To maintain performance, you should only access IRT data in the isochronous real-time processing routine.

The PNIO_data_read_cache_refresh( ) and PNIO_data_write_cache_flush( ) functions affect only RT data and not IRT data.

3.7 Addressing PROFINET IO devices

Address space

In a PROFINET IO system, there is an address space for read access and one for write access that are common to all devices. Both are specified in the configuration.

Refer to the configuration for the addresses of the slots of the devices and modules (module or submodule) with which your IO-Base controller user program communicates.

3.8 Callback mechanism

How it works

Callback functions are programmed in the IO-Base controller user program.

A callback function can be given any name.

A callback event is an asynchronous event that is called by the IO-Base interface. It interrupts the execution of the IO-Base controller user program and starts the callback function in a separate thread. This means that synchronization techniques are necessary.

Each callback event is accompanied by a callback type. The callback event type defines the callback event.

Callback functions on the IO controller

The callback events and event types on the IO controller are listed in the following table. The table also shows what to use to register a callback function and what triggers a callback event:

Callback event (asynchronous)	Callback event type	Registered by ...	Triggered by ...
Alarm arrived	PNIO_CBE_ALARM_IND	PNIO_controller_open( )	... IO device
Result of the read data record job arrived	PNIO_CBE_ REC_READ_CONF	PNIO_controller_open( )	... PNIO_rec_read_req( )
Result of the write data record job arrived	PNIO_CBE_REC_WRITE_ CONF	PNIO_controller_open( )	... PNIO_rec_write_req( )
The local mode has changed.	PNIO_CBE_MODE_IND	PNIO_register_cbf( )	... PNIO_set_mode( )
The status of the connection to the IO device has changed.	PNIO_CBE_ DEV_ACT_CONF	PNIO_register_cbf( )	... PNIO_device_activate( )
Activate flashing mode for LED	PNIO_CBE_START_LED_ FLASH_IND	PNIO_register_cbf( )	... Arrival of a request for identification.
Deactivate flashing mode for LED	PNIO_CBE_STOP_LED_F LASH_IND	PNIO_register_cbf( )	... Arrival of a request to end identification.
Signal remote download request (firmware update or reconfiguration)	PNIO_CBE_CP_STOP_RE Q	PNIO_register_cbf( )	... Arrival of a firmware update or reconfiguration request
Start of isochronous real-time data processing	PNIO_CP_CBE_ STARTOP_IND	PNIO_CP_register_cbf( )	... the end of the IRT transfer phase; IRT data in host memory.
IRT cycle violation	PNIO_CP_ OPFAULT_IND	PNIO_CP_register_cbf( )	... PNIO_CP_set_opdone( ); PNIO_CP_set_opdone( ) was called too late by the user program.

Note

Include multithreading standard libraries when you compile your user program.

Note

Within a callback function, the only functions that can be called are those that access IO data.

These are:

- PNIO_data_read( )
- PNIO_output_data_read( )
- PNIO_data_read_cache( )
- PNIO_data_read_cache_refresh( )
- PNIO_data_write( )
- PNIO_data_write_cache( )
- PNIO_data_write_cache_flush( )
- PNIO_CP_set_opdone( )

Coordinating the sequence of callbacks

A callback function can interrupt the IO-Base controller user program at any time. Callback functions for different events can also interrupt each other. A callback function must therefore be designed for simultaneous, multiple execution (reentrant) since it can be called from different threads. In practical terms, this means that writing and reading of shared variables must be protected by synchronization mechanisms.

Avoid waiting in callback functions, particularly when entering critical sections. A further call for this callback function by the same callback event would be blocked. Instead, you should, where possible, use a separate database.

A separate callback function can be registered for each callback event. On the other hand, it is also possible to group several callback events in one callback function

Description of the functions and data types for IO controllers

4

This chapter describes the individual functions of the IO-Base user programming interface for the IO controller in detail and the data types used.

The chapter is primarily intended as a source of reference when you are writing your IO-Base controller user programs.

4.1 Management functions

Overview

The IO controller recognizes the following management functions:

- PNIO_controller_open( )
- PNIO_register_cbf( )
- PNIO_controller_close( )

Other management functions are available for IRT mode as described in the section "Interface for isochronous real-time mode (IRT) (Page 76)".

4.1.1 PNIO_controller_open( ) (register device as IO controller)

Description

Using this function, the IO-Base controller user program registers an IO controller with the IO-Base functions.

Callback functions that process alarm events and read/write data record events can also be registered or partial access to submodules (cyclic data) can be enabled with the PNIO_CEP_SLICE_ACCESS ExtPar flag.

The following applies only to SOFTNET PN IO:

After starting the PC, the PNIO_controller_open( ) function is permitted only on completion of the SIMATIC NET configuration phase.

If this phase is not completed, the error code PNIO_ERR_INTERNAL is returned and the call must be repeated.

Instead, you can also evaluate the "SimaticNetPcStationUpEvent" event. For a detailed description, refer to the "Commissioning PC Stations" manual.

Syntax

```
PNIO_UINT32  PNIO_controller_open(
    PNIO_UINT32      CpIndex,                    //in
    PNIO_UINT32      ExtPar,                     //in
    PNIO_CBF         cbf_rec_read_conf,          //in
    PNIO_CBF         cbf_rec_write_conf,         //in
    PNIO_CBF         cbf_alarm_ind,              //in
    PNIO_UINT32     *Handle                      //out
);
```

Parameter

Name	Description
CpIndex	Module index - Used to uniquely identify the communications module. Refer to the configuration for this parameter ("module index" of the CP).
ExtPar	Each of the 32 bits of this parameter can be used for a parameter assignment. For the significance of the individual bits, refer to the section "ExtPar (extended parameter) (Page 100)". Undefined bits are "reserved" and must not be set.
cbf_rec_read_conf	Address of the callback function that is started after the arrival of the read data record job with the callback event type PNIO_CBE_REC_READ_CONF.
	Note The function pointer must not be NULL.
cbf_rec_write_conf	Address of the callback function that is started after the arrival of the write data record job with the callback event type PNIO_CBE_REC_WRITE_CONF.
	Note The function pointer must not be NULL.
cbf_alarm_ind	Address of the callback function that is started after the arrival of an alarm with the callback event type PNIO_CBE_ALARM_IND.
	Note The function pointer can be NULL.
Handle	Handle assigned to the registered IO controller that is returned to the IO-Base controller user program if successful. This must be included in all further function calls.

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_CONFIG_IN_UPDATE
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_INTERNAL
- PNIO_ERR_MAX_REACHED

- PNIO_ERR_NO_CONFIG
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_NO_LIC_SERVER (SOFTNET PNIO only)
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_PRM_EXT_PAR
- PNIO_ERR_PRM_HND

4.1.2 PNIO_register_cbf( ) (registration of callback functions)

Description

This function registers a callback function.

Note

Callback functions can only be registered in offline mode.

Syntax

```
PNIO_UINT32  PNIO_register_cbf(
    PNIO_UINT32      Handle,                      //in
    PNIO_CBE_TYPE    CbeType,                     //in
    PNIO_CBF         Cbf                          //in
);
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )
CbeType	Callback event type for which the callback function "Cbf" will be registered; see the section PNIO_CBE_TYPE (callback event type) (Page 99).
Cbf	Address of the callback function to be started after arrival of the callback event "CbeType".
	Note The function pointer must not be NULL.
	Note If a callback function is already registered for a callback event type, a further callback function can only be registered after PNIO_controller_close( ).

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

Programming example

The following example shows that a callback function can also be used for several callback event types.

```
PNIO_CBF  callback1( );    // for alarms and MODE changes
PNIO_CBF  callback2( );    // for data record confirmations
PNIO_controller_open(
    CpIndex,
    ExtPar,
    callback2,             // for READ events
    callback2,             // also for WRITE events
    callback1,             // for ALARM events
    &Handle
);
PNIO_register_cbf(
    Handle,
    PNIO_CBE_MODE_IND,
    callback1             // for MODE changes
);
```

4.1.3 PNIO_controller_close( ) (deregister)

Description

Using this function, the IO-Base controller user program deregisters an IO controller.

All registered callback functions are deregistered (including the callback functions registered with PNIO_register_cbf( )). On completion of the PNIO_controller_close( ) function, no further callback functions will be called for the deregistered IO controller.

Note

While the PNIO_controller_close( ) function executes, callback functions can also be executed.

Note

After successfully deregistering, the handle on the CP is no longer valid and can no longer be used by the IO-Base controller user program.

Note

Before exiting the IO-Base controller user program, the PNIO_controller_close( ) function must be called.

Note

If the CP is operated as an IO controller and IO device at the same time in the same user program, before deregistering the IO controller/IO device, the writing and reading of IO data of the IO device/IO controller must also be stopped.

Syntax

```
PNIO_UINT32  PNIO_controller_close(
    PNIO_UINT32     Handle              //in
);
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )

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

4.1.4 PNIO_iosystem_reconfig

Description

This function supports configuration control for IO systems. The product CD contains an example of an implementation of this function.

Syntax

```
PNIO_UINT32 PNIO_CODE_ATTR PNIO_iosystem_reconfig(
 PNIO_UINT32 Handle,
 PNIO_IOS_RECONFIG_MODE Mode,
 PNIO_UINT32 DeviceCnt,
 PNIO_ADDR *DeviceList,
 PNIO_UINT32 PortInterconnectionCnt
 PNIO_ADDR *PortInterconnectionList,
);
```

Elements

Name	Description
Handle	Handle from "PNIO_controller_open()"
Mode	Specifies the mode to be executed. • PNIO_IOS_RECONFIG_MODE_DEACT: All IO devices are deactivated to avoid diagnostics messages • PNIO_IOS_RECONFIG_MODE_TAILOR: Starts "tailoring" with the parameters specified in the "DeviceList" and "PortInterconnectionList" and then activates all IO devices that are absolutely necessary or optional and also included in the "DeviceList".
DeviceCnt	Number of entries contained in the "DeviceList"
DeviceList	List of logical addresses of the optional IO devices that exist in the real setup.
PortInterconnectionCnt	Number of entries contained in the"PortInterconnectionList"
PortInterconnectionList	List of logical addresses of the port interconnections. Each port interconnection is made up or a pair of logical addresses (logical address device1/port2, logical address device2/port1)

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_MODE_VALUE
- PNIO_ERR_TAILOR_INV_STATION
- PNIO_ERR_TAILOR_INV_PORT
- PNIO_ERR_NOT_POSSIBLE
- PNIO_ERR_WRONG_HND

4.2 Functions relating to the mode

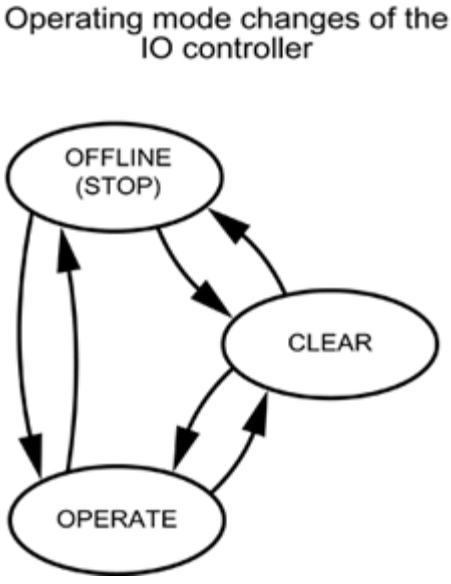
Overview

An IO controller is in one of the following modes:

- OFFLINE (STOP)
- CLEAR
- OPERATE

Mode changes are possible in the direction shown by the arrows.

They are initiated by calling the PNIO_set_mode( ) function.



OFFLINE (STOP)

The OFFLINE mode is the state of the IO controller after the PNIO_controller_open( ) is called. (The OFFLINE mode corresponds to STOP mode.)

It is not possible to transfer IO data, data records, or alarms in this mode.

Note

Callback functions can only be registered in OFFLINE mode.

Note

If the IO controller is in OFFLINE mode, it is not possible to activate or deactivate IO devices.

Note

If the IO controller is in OFFLINE mode, all the IO data written to the process image by PROFINET IO is lost including the local status of this data ("IOlocState").

After changing to OPERATE mode, all IO data of all IO devices must therefore be written again.

CLEAR

This status of the IO controller is supported only by certain specific PROFINET IO devices, for example SIMATIC NET IE/PB Link PN IO. For details of the functionality, refer to the manufacturer's information.

OPERATE (normal mode)

In OPERATE mode (normal operation), the current IO data is transferred.

Writing and reading data records and processing of alarms is possible if the corresponding callback functions were registered.

4.2.1 PNIO_set_mode( ) (set operating mode)

Description

With this function you can change to a new mode.

The function returns immediately. The actual status change is signaled with the PNIO_CBE_MODE_IND event if the function returned PNIO_OK. The user may only call PNIO_set_mode( ) again when a previous PNIO_set_mode( ) job was confirmed with PNIO_CBE_MODE_IND( ).

Syntax

```
PNIO_UINT32 PNIO_set_mode(
 PNIO_UINT32 Handle,
 PNIO_MODE_TYPE Mode                //in
) ;                                 //in
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )
Mode	New mode to be set; see section "PNIO_MODE_TYPE (operating mode type) (Page 97)".

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_MODE_VALUE
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SET_MODE_NOT_ALLOWED
- PNIO_ERR_WRONG_HND
- PNIO_ERR_SEQUENCE

4.2.2 Callback event PNIO_CBE_MODE_IND (signal change of own operating mode)

Description

The PNIO_CBE_MODE_IND callback event reports a change in the mode of the IO controller; see the section "PNIO_CBE_TYPE (callback event type) (Page 99)".

The callback event must already have been registered with the PNIO_register_cbf( ) function.

Note

This callback event must be registered before the PNIO_set_mode call.

Note

On completion of PNIO_controller_open( ), the IO controller is OFFLINE. This mode change cannot, however, be signaled because at this time the callback function cannot have been registered!

Syntax

The following syntax shows the specific parameters for this event as part of the "union" from the PNIO_CBE_PRM structure; see the section "PNIO_CBE_PRM (callback event parameter) (Page 99)".

```
typedef struct
{
    PNIO_MODE_TYPE           Mode;                        //in
} ATTR_PACKED PNIO_CBE_PRM_MODE_IND;
```

Parameter

Name	Description
Mode	New mode; see the section "PNIO_MODE_TYPE (operating mode type) (Page 97)" and header file PNIOUSRX.H.

4.2.3 PNIO_device_activate( ) (activate/deactivate IO device)

Meaning of activate/deactivate

There is a connection to an activated IO device and there is no connection to a deactivated IO device.

After the operating mode of the IO controller changes to the "OPERATE" mode, the configured I/O devices that can be reached are signaled as activated by a station return alarm if they were activated previously.

Description of the function

With this function, an IO device is activated or deactivated by the IO controller.

The function returns immediately. If the function has returned "PNIO_OK", the "PNIO_CBE_DEV_ACT_CONF" event is used to signal whether or not the IO device could be informed of the activation or deactivation.

In addition to this, the actual activation or deactivation of an IO device is also signaled to the IO controller with a station return alarm (PNIO_ALARM_DEV_RETURN) or station failure alarm (PNIO_ALARM_DEV_FAILURE).

The sequence of the events "PNIO_CBE_DEV_ACT_CONF" and the station failure or return alarms over time is not predictable.

Note

Parallel jobs with DN Driver

Up to eight jobs can be executed at the same time.

Note

Note that a deactivated IO device may only be disconnected or turned off after arrival of the callback for "PNIO_ALARM_DEV_FAILURE" and the "PNIO_CBE_DEV_ACT_CONF" callback.

Syntax

```
PNIO_UINT32 PNIO_device_activate(

 PNIO_UINT32 Handle, //in
```

```
 PNIO_ADDR *pAddr, //in
 PNIO_DEF_ACT_TYPE DeviceMode //in
);
```

Parameter

Name	Description
Handle	Handle from "PNIO_controller_open()"
pAddr	Any address of the IO device to which the job is sent. **Explanation** Each IO device can have several modules and therefore also have several addresses. It is adequate to address one of these IO device addresses to activate or deactivate the entire IO device. **Note** With the CP 1626 it is not the address of the module that is used but the hardware identifier of the STEP 7 configuration.
DeviceMode	The "DeviceMode" parameter can adopt two states: • PNIO_DA_FALSE deactivates the IO device. • PNIO_DA_TRUE activates the IO device.

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_DEV_ACT_NOT_ALLOWED
- PNIO_ERR_INTERNAL
- PNIO_ERR_MODE_VALUE
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_DEV_STATE
- PNIO_ERR_WRONG_HND

4.2.4 Callback event PNIO_CBE_DEV_ACT_CONF (signal connection status to IO device)

Description

The callback event PNIO_CBE_DEV_ACT_CONF signals the status of the connection to the IO device (activated or deactivated) to your IO-Base controller user program.

The callback event must already have been registered with the PNIO_register_cbf( ) function.

Syntax

The following syntax shows the specific parameters for this event as part of the "union" from the PNIO_CBE_PRM structure; see the section "PNIO_CBE_PRM (callback event parameter) (Page 99)".

```
typedef struct
{
    PNIO_ADDR              *pAddr;
    PNIO_DEV_ACT_TYPE       Mode;
    PNIO_UINT32             Result;
} ATTR_PACKED PNIO_CBE_PRM_DEV_ACT_CONF;
```

Parameter

Name	Description
pAddr	Pointer to the address of the IO device to which the job was sent.
Mode	Current status of the connection to the device
Result	(error code) The "Result" parameter indicates that activation or deactivation was successful with PNIO_OK.

4.3 Functions for reading IO data

Overview

The following functions are available for reading IO data:

- Direct unbuffered access to the process image
  - PNIO_data_read( )
  - PNIO_output_data_read( )
- Fast buffered access to the process image
  - PNIO_data_read_cache_refresh( )
  - PNIO_data_read_cache( )

4.3.1 PNIO_data_read( ) (read input data)

Description

This function reads input data from the process image. The function then returns immediately. The process image is kept up to date by the IO controller cyclically regardless of this read job.

The basic mechanisms for this are described in the section "Non-isochronous access to cyclic IO data (RT) (Page 27)".

Note

The PNIO_data_read( ) function reads the data of a single submodule. As a result, data consistency beyond the limits of the submodule can only be achieved for IRT data accessed in isochronous real-time mode.

Note

This function is not reentrant. Make sure that access is only from a thread context or that a locking strategy is used.

Syntax

```
PNIO_UINT32 PNIO_data_read(
 PNIO_UINT32 Handle,
 PNIO_ADDR *pAddr,
 PNIO_UINT32 BufLen,
```

```
 PNIO_UINT32 *pDataLen,
 PNIO_UINT8 *pBuffer,
 PNIO_IOXS IOlocState,
 PNIO_IOXS *pIOremState
);
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )
pAddr	Logical address of the submodule. With partial access, this can be any address of the submodule. Example: 4 bytes input. Address 100 to 103 Valid addresses: 100, 101, 102, 103
	Note On the IO controller, "pAddr" is the address of the remote submodule from which data will be read.
BufLen	Length of the data buffer made available (in bytes). This parameter specifies how many bytes will be read. Less bytes than available can be read (in other words, the number of bytes from the specified address to the end of the module).
pDataLen	Length of the read data buffer (in bytes) If the length of the IO device data is less than or equal to "BufLen", "*pDataLen" contains the length of the IO data that was read. If the length of the IO device data is greater than the data buffer available, as much data as possible is read into the data buffer. "*pDataLen“, however, contains the length of the data buffer that would have been necessary to read all IO data.
pBuffer	Pointer to the data buffer of the read IO data.
IOlocState	Local status of the read IO data (GOOD or BAD).
	Note The IO-Base controller user program specifies the local status using "IOlocState". "IOlocState" is sent by the underlying IO-Base functions to the remote device in the next cycle. The BAD status informs the communications partner that the IO-Base controller user program could not process the received IO data. Partial access: With each partial access, the transferred status ("IOlocState") applies to the entire submodule.
pIOremState	Remote status of the read IO data (GOOD or BAD). Indicates the quality/validity of the read data.

Example: PNIO_data_read( ) partial access

As an example, we will assume two input modules:

- 2 bytes, I address: 0 ... 1
- 4 bytes, I address: 2 ... 5

I address	BufLen	*pDataLen OUTPUT	Return Value
2	8	4	OK
2	2	4	OK
3	2	3	OK
5	1	1	OK
5	8	1	OK
1	2	1	OK
1	1	1	OK

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_CONNECTION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_IO_TYPE
- PNIO_ERR_PRM_LEN
- PNIO_ERR_PRM_LOC_STATE
- PNIO_ERR_PRM_RSTATE
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_WRONG_HND
- PNIO_WARN_IRT_INCONSISTENT

4.3.2 PNIO_output_data_read( ) (read output data)

Description

This function reads output data from the process image. The function then returns immediately. The process image is kept up to date by the IO controller cyclically regardless of this read job.

The basic mechanisms for this are described in the section "Non-isochronous access to cyclic IO data (RT) (Page 27)".

Note

The PNIO_output_data_read( ) function reads the data of a single submodule. As a result, data consistency beyond the limits of the submodule can only be achieved for IRT data accessed in isochronous real-time mode.

Syntax

```
PNIO_UINT32  PNIO_output_data_read(
    PNIO_UINT32        Handle,                          //in
    PNIO_ADDR         *pAddr,                           //in
    PNIO_UINT32        BufLen,                          //in
    PNIO_UINT32       *pDataLen,                        //out
    PNIO_UINT8        *pBuffer,                         //out
    PNIO_IOXS         *pIOlocState,                     //out
    PNIO_IOXS         *pIOremState                      //out
);
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )
pAddr	Logical address of the submodule. With partial access, this can be any address of the submodule. Example: 4 bytes output. Address 100 to 103 Valid addresses: 100, 101, 102, 103
	Note On the IO controller, "pAddr" is the address of the remote submodule from which data will be read.
BufLen	Length of the data buffer made available (in bytes). This parameter specifies how many bytes will be read. Less bytes than available can be read (in other words, the number of bytes from the specified address to the end of the module).
pDataLen	Length of the read data buffer (in bytes) If the length of the IO device data is less than or equal to "BufLen", "*pDataLen" contains the length of the IO data that was read. If the length of the IO device data is greater than the data buffer available, as much data as possible is read into the data buffer. "*pDataLen“, however, contains the length of the data buffer that would have been necessary to read all IO data.
pBuffer	Pointer to the data buffer of the read IO data.

Name	Description
pIOlocState	Local status of the read IO data (GOOD or BAD).
	Note The IO-Base controller user program specifies the local status using "IOlocState". "IOlocState" is sent by the underlying IO-Base functions to the remote device in the next cycle. The BAD status informs the communications partner that the IO-Base controller user program could not process the received IO data. Partial access: With each partial access, the transferred status ("IO-locState") applies to the entire submodule.
pIOremState	Remote status of the read IO data (GOOD or BAD). Indicates the quality/validity of the read data.

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_CONNECTION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_IO_TYPE
- PNIO_ERR_PRM_LEN
- PNIO_ERR_PRM_LOC_STATE
- PNIO_ERR_PRM_RSTATE
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_WRONG_HND

4.3.3 PNIO_data_read_cache_refresh( ) (transfer IO data to the read cache)

Description

With this function, your IO-Base controller user program initiates the transfer of the following data from the process image on the CP to the read cache:

- RT IO input data and its remote status
- The remote status of the RT IO output data

The basic mechanisms for this are described in the section "Non-isochronous access to cyclic IO data (RT) (Page 27)".

This function does not transfer any IRT IO data to the read cache; see also the section "Isochronous real-time and non-isochronous access to cyclic IO data (IRT) (Page 31)".

Syntax

```
PNIO_UINT32  PNIO_data_read_cache_refresh(
    PNIO_UINT32     Handle                      //in
);
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )

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_WRONG_HND

4.3.4 PNIO_data_read_cache( ) (read IO data from the read cache)

Description

This function reads RT input data from the read cache and writes the local status of the RT output data to the write cache. The function then returns immediately. The process image is kept up to date by the IO controller cyclically regardless of this read job.

The basic mechanisms for this are described in the section "Non-isochronous access to cyclic IO data (RT) (Page 27)".

This function does not transfer any IRT IO data to the read cache; see also the section "Isochronous real-time and non-isochronous access to cyclic IO data (IRT) (Page 31)".

Note

The PNIO_data_read_cache( ) reads the data of a single submodule. As a result, data consistency beyond the limits of the submodule can only be achieved for IRT data accessed in isochronous real-time mode.

Syntax

```
PNIO_UINT32  PNIO_data_read_cache(
    PNIO_UINT32        Handle,                    //in
    PNIO_ADDR         *pAddr,                     //in
    PNIO_UINT32        BufLen,                    //in
    PNIO_UINT32       *pDataLen,                  //out
    PNIO_UINT8        *pBuffer,                   //out
    PNIO_IOXS          IOlocState,                //in
    PNIO_IOXS         *pIOremState                //out
);
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )
pAddr	Logical address of the submodule. With partial access, this can be any address of the submodule. Example: 4 bytes input. Address 100 to 103 Valid addresses: 100, 101, 102, 103
	Note On the IO controller, "pAddr" is the address of the remote submodule from which data will be read.
BufLen	Length of the data buffer made available (in bytes). This parameter specifies how many bytes will be read. Less bytes than available can be read (in other words, the number of bytes from the specified address to the end of the module).
pDataLen	Length of the read data buffer (in bytes) If the length of the IO device data is less than or equal to "BufLen", "*pDataLen" contains the length of the IO data that was read. If the length of the IO device data is greater than the data buffer available, as much data as possible is read into the data buffer. "*pDataLen“, however, contains the length of the data buffer that would have been necessary to read all IO data.
pBuffer	Pointer to the data buffer of the read IO data
IOlocState	Local status of the read IO data (GOOD or BAD)
	Note The IO-Base controller user program specifies the local status using "IOlocState". "IOlocState" is sent by the underlying IO-Base functions to the remote device in the next cycle. The BAD status informs the communications partner that the IO-Base controller user program could not process the received IO data. Partial access: With each partial access, the transferred status ("IO-locState") applies to the entire submodule.
pIOremState	Remote status of the read IO data (GOOD or BAD). Indicates the quality/validity of the read data.

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_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_IO_TYPE
- PNIO_ERR_PRM_LEN
- PNIO_ERR_PRM_LOC_STATE
- PNIO_ERR_PRM_RSTATE
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND

4.4 Functions for writing IO data

Overview

The following functions are available for writing IO data:

- Direct unbuffered access to the process image
  - PNIO_data_write( )
- Fast buffered access to the process image
  - PNIO_data_write_cache( )
  - PNIO_data_write_cache_flush( )

4.4.1 PNIO_data_write( ) (write IO data)

Description

With this function, your IO-Base controller user program writes IO data to the process image. The function then returns immediately. The IO controller transfers the IO data to the IO device in the next cycle.

The basic mechanisms for this are described in the section "Non-isochronous access to cyclic IO data (RT) (Page 27)".

Note

The PNIO_data_write( ) function writes the data of a single submodule. As a result, data consistency beyond the limits of the submodule can only be achieved for IRT data accessed in isochronous real-time mode.

Note

This function is not reentrant. Make sure that access is only from a thread context or that a locking strategy is used.

Syntax

```
PNIO_UINT32 PNIO_data_write(
 PNIO_UINT32 Handle,
 PNIO_ADDR *pAddr,
 PNIO_UINT32 BufLen,
 PNIO_UINT8 *pBuffer,
 PNIO_IOXS IOlocState,
 PNIO_IOXS *pIOremState
);
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )
pAddr	Logical address of the submodule.
	Note On the IO controller, "pAddr" is the address of the remote submodule to which data will be written. With partial access, this can be any logical address of the submodule. The data length (BufLen) must be adapted accordingly. Maximum up to end of module (see section "ExtPar (extended parameter) (Page 100)").
BufLen	Length of the data buffer (in bytes). This parameter specifies how many bytes will be written. Less bytes can be written than are available from the specified address up to the end of the module.

Name	Description
	Note Partial access not enabled: "BufLen" must match the length of the output data of the addressed module exactly, otherwise an error is reported in the return code. Partial access enabled: "BufLen" can be less than or equal to the length of the output data of the addressed module. The maximum data length is the number of bytes available from the specified address to the end of the module.
pBuffer	Pointer to the data buffer of the IO data to be written.
IOlocState	Local status of the IO data to be written (GOOD or BAD). Partial access enabled: The last transferred status ("IOlocState") applies to the entire submodule. In other words, if a partial status has the BAD status, all data of the module have the BAD status.
pIOremState	Remote status, the returned status of the written IO data (GOOD or BAD).
	Note This status relates to IO data written earlier and not to the write job that has just been sent.
	Note With BAD, the remote device can signal to the IO-Base controller user program that it could not process the previously received IO data.

Example: PNIO_data_write( ) partial access

As an example, we will assume three output modules:

- 2 bytes, Q address: 0 ... 1
- 4 bytes, Q address: 2 ... 5
- 4 bytes, Q address: 10 ... 13

Q address	BufLen	Return Value
2	8	PNIO_ERR_VALUE_LEN
2	2	PNIO_OK
1	2	PNIO_ERR_VALUE_LEN
1	1	PNIO_OK
3	3	PNIO_OK
3	1	PNIO_OK
3	4	PNIO_ERR_VALUE_LEN
6	4	PNIO_ERR_UNKNOWN_ADDR
10	1	PNIO_OK
11	1	PNIO_OK
9	1	PNIO_ERR_UNKNOWN_ADDR

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_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_IO_TYPE
- PNIO_ERR_PRM_LEN
- PNIO_ERR_PRM_LOC_STATE
- PNIO_ERR_PRM_RSTATE
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND
- PNIO_WARN_IRT_INCONSISTENT

4.4.2 PNIO_data_write_cache( ) (write IO data to the write cache)

Description

With this function, your IO-Base controller user program writes RT output data to the write cache and reads the remote status of the RT input data from the read cache. The data remains in the write cache until the next PNIO_data_write_cache_flush( ) function call.

The basic mechanisms for this are described in the section "Non-isochronous access to cyclic IO data (RT) (Page 27)".

This function is transparent for IRT IO data; see also the section "Isochronous real-time and non-isochronous access to cyclic IO data (IRT) (Page 31)".

Note

The PNIO_data_write_cache( ) functions always relate to the data of one submodule. As a result, data consistency beyond the limits of the submodule can only be achieved for IRT data accessed in isochronous real-time mode.

Syntax

```
PNIO_UINT32  PNIO_data_write_cache(
    PNIO_UINT32     Handle,                          //in
    PNIO_ADDR      *pAddr,                           //in
    PNIO_UINT32     BufLen,                          //in
    PNIO_UINT8     *pBuffer,                         //in
    PNIO_IOXS       IOlocState,                      //in
    PNIO_IOXS      *pIOremState                      //out
);
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )
pAddr	Logical address of the submodule.
	Note On the IO controller, "pAddr" is the address of the remote submodule to which data will be written. With partial access, this can be any logical address of the submodule. The data length (BufLen) must be adapted accordingly. Maximum up to end of module (see section "ExtPar (extended parameter) (Page 100)").
BufLen	Length of the data buffer (in bytes). This parameter specifies how many bytes will be written. Less bytes can be written than are available from the specified address up to the end of the module.
	Note Partial access not enabled: "BufLen" must match the length of the output data of the addressed module exactly, otherwise an error is reported in the return code. Partial access enabled: "BufLen" can be less than or equal to the length of the output data of the addressed module. The maximum data length is the number of bytes available from the specified address to the end of the module.
pBuffer	Pointer to the data buffer of the IO data to be written.
IOlocState	Local status of the IO data to be written (GOOD or BAD). Partial access enabled: The last transferred status ("IOlocState") applies to the entire submodule. In other words, if a partial status has the BAD status, all data of the module has the BAD status.
pIOremState	Remote status, the returned status of the written IO data (GOOD or BAD).
	Note This status relates to IO data written earlier and not to the write job that has just been sent.

Name	Description
	Note With BAD, the remote device can signal to the IO-Base controller user program that it could not process the previously received IO data.
	Note This status is transferred only after the PNIO_data_read_cache_refresh( ) call.

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_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_IO_TYPE
- PNIO_ERR_PRM_LEN
- PNIO_ERR_PRM_LOC_STATE
- PNIO_ERR_PRM_RSTATE
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND

4.4.3 PNIO_data_write_cache_flush( ) (write IO data from the write cache to the process image)

Description

With this function, your IO-Base controller user program initiates the transfer of the following data from the write cache to the process image on the CP:

- RT IO output data and its local status.
- The local status of the RT IO input data.

The basic mechanisms for this are described in the section "Non-isochronous access to cyclic IO data (RT) (Page 27)".

This function does not transfer any IRT IO data from the write cache; see also the section "Isochronous real-time and non-isochronous access to cyclic IO data (IRT) (Page 31)".

Syntax

```
PNIO_UINT32  PNIO_data_write_cache_flush(
    PNIO_UINT32     Handle,                          //in
);
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )

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_WRONG_HND

4.5 Interface for read data record

Description

The IO controller sends a read data record job with the PNIO_rec_read_req( ) function. The read data record job is sent to the IO device.

The IO device responds to this read data record job by sending the data record to the IO controller.

The arrival of this data record at the IO controller is signaled by the callback event PNIO_CBE_REC_READ_CONF; the registered callback function is started.

The basic mechanisms for this are described in the section "Non-isochronous access to cyclic IO data (RT) (Page 27)".

Diagnostic data records supported by the IO device

The following table lists data record numbers of diagnostics data records that are supported by the IO device and can therefore be read by the IO controller:

Data record number (record index)	Content and meaning
0x800A	Channel diagnostics for a submodule slot
0x800B	Channel diagnostics and vendor-specific diagnostics information for a submodule slot
0x800C	Channel diagnostics and vendor-specific diagnostics information for a submodule slot
0xC00A	Channel diagnostics for a slot

Data record number (record index)	Content and meaning
0xC00B	Channel diagnostics and vendor-specific diagnostics information for a slot
0xC00C	Channel diagnostics and vendor-specific diagnostics information for a slot
0xE002	Deviation of the expected and actual configuration of an IO device assigned to an IO controller.
0xE00A	Channel diagnostics for an AR
0xE00B	Channel diagnostics and vendor-specific diagnostics information for an AR
0xE00C	Channel diagnostics and vendor-specific diagnostics information for an AR
0xF00A	Channel diagnostics for an API
0xF00B	Channel diagnostics and vendor-specific diagnostics information for an API
0xF00C	Channel diagnostics and vendor-specific diagnostics information for an API

You will find other supported data records in the documentation of the relevant IO devices.

4.5.1 PNIO_rec_read_req( ) (send read data record job)

Description

With this function, the IO controller sends a read data record job. The result of the job is signaled by the "PNIO_CBE_REC_READ_CONF" callback event if the function returned "PNIO_OK".

With this function, you can also read diagnostics data of an IO device stored in data records. Please read the description of the relevant IO devices.

Note

Parallel jobs with DN Driver

Up to 16 jobs can be executed at the same time.

Syntax

```
PNIO_UINT32 PNIO_rec_read_req(
 PNIO_UINT32 Handle, //in
 PNIO_ADDR *pAddr, //in
 PNIO_UINT32 RecordIndex, //in
 PNIO_REF ReqRef, //in
 PNIO_UINT32 Length //in
);
```

Parameter

Name	Description
Handle	Handle from "PNIO_controller_open()"
pAddr	Address of the module of the IO device for which the read data record job is intended.
	Notice With the CP 1626 and PN Driver it is not the address of the module that is used but the hardware identifier of the STEP 7 configuration.
	Note If the address (pAddr) is a diagnostics address, the parameter "IO-DataType" must be set to Input.
	Note The "IODataType" parameter is described in the section "PNIO_ADDR (address structure) (Page 98)".
RecordIndex	Data record number
ReqRef	Reference assigned by the IO-Base controller user program.
	Note The "ReqRef" reference allows the IO-Base controller user program to distinguish different jobs. This parameter can be assigned with any value by the IO-Base controller user program and is returned with the callback event "PNIO_CBE_REC_READ_CONF".
Length	Data record length of 4096 bytes. The pointer to the data record and its real length are returned by the callback event "PNIO_CBE_REC_READ_CONF".

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_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_REC_INDEX
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND

4.5.2 Callback event PNIO_CBE_REC_READ_CONF (signal result of a read data record job)

Description

The PNIO_CBE_REC_READ_CONF callback event reports the result of a read data record job sent previously with a PNIO_rec_read_req call; see the section "PNIO_CBF (PNIO callback function) (Page 100)".

The callback event must already have been registered with the PNIO_controller_open( ) function.

Syntax

The following syntax shows the specific parameters for this event as part of the "union" from the PNIO_CBE_PRM structure; see the section "PNIO_CBE_PRM (callback event parameter) (Page 99)".

```
typedef struct
{
    PNIO_UINT32           Length;                  //in
    const PNIO_UINT8     *pBuffer;                 //in
    PNIO_ADDR            *pAddr;                   //in
    PNIO_UINT32           RecordIndex;             //in
    PNIO_REF              ReqRef;                  //in
    PNIO_ERR_STAT         Err;                     //in
} ATTR_PACKED PNIO_CBE_PRM_REC_READ_CONF;
```

Parameter

Name	Description
Length	Length of the transferred data record (in bytes) to which "pBuffer" points.
pBuffer	Pointer to the data buffer that, if successful, contains the requested data record.
	Note The data buffer is valid only within the callback function and becomes invalid once the function completes!
	Note The data buffer must not be changed during processing of the callback function.
pAddr	Address of the module of the IO device that replied to the read data record job.
RecordIndex	Data record number

Name	Description
ReqRef	Reference assigned by the IO-Base controller user program.
	Note Here, the reference specified with the PNIO_rec_read_req( ) call is returned. This allows the IO-Base controller user program to distinguish different jobs.
Err	Error code on the execution of the job

4.6 Interface for write data record

Description

The IO controller sends a write data record job with the PNIO_rec_write_req( ) function. The write data record job is sent along with the data record to the IO device.

The IO device receives this write data record job and confirms it.

The arrival of the write data record confirmation at the IO controller is signaled by the callback event PNIO_CBE_REC_WRITE_CONF; the registered callback function is started.

4.6.1 PNIO_rec_write_req( ) (send write data record job)

Description

With this function, the IO controller sends a write data record job. The result of the job is signaled by the PNIO_CBE_REC_WRITE_CONF callback event if the function returned PNIO_OK.

Note

Parallel jobs with DN Driver

Up to 16 jobs can be executed at the same time.

Syntax

```
PNIO_UINT32 PNIO_rec_write_req(
 PNIO_UINT32 Handle, //in
 PNIO_ADDR *pAddr, //in
 PNIO_UINT32 RecordIndex, //in
 PNIO_REF ReqRef //in
```

```
 PNIO_UINT32 Length, //in
 PNIO_UINT8 *pBuffer //in
);
```

Parameter

Name	Description
Handle	Handle from "PNIO_controller_open()"
pAddr	Address of the module of the IO device for which the read data record job is intended.
	Notice With the CP 1626 and PN Driver it is not the address of the module that is used but the hardware identifier of the STEP 7 configuration.
	Note If the address (pAddr) is a diagnostics address, the parameter "IODataType" must be set to Input.
	Note The "IODataType" parameter is described in the section "PNIO_ADDR (address structure) (Page 98)".
RecordIndex	Data record number
ReqRef	Reference assigned by the IO-Base controller user program.
	Note The "ReqRef" reference allows the IO-Base controller user program to distinguish different jobs. This parameter can be assigned with any value by the IO-Base controller user program and is returned in the callback event "PNIO_CBE_REC_WRITE_CONF".
Length	Length of the data buffer (in bytes) The maximum length depends on the version and the product used. You can find it in the "readme.txt" file.
pBuffer	Data buffer containing the data record.
	Note The caller of the "PNIO_rec_write_req()" function must make the "pBuffer" data buffer available that contains the data record to be written.

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_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_REC_INDEX

- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND

4.6.2 Callback event PNIO_CBE_REC_WRITE_CONF (signal result of a write data record job)

Description

The PNIO_CBE_REC_WRITE_CONF callback event reports the result of a write data record job sent previously with a PNIO_rec_write_req call; see the section "PNIO_CBF (PNIO callback function) (Page 100)".

The callback event must already have been registered with the PNIO_controller_open( ) function.

Syntax

The following syntax shows the specific parameters for this event as part of the "union" from the PNIO_CBE_PRM structure; see the section "PNIO_CBE_PRM (callback event parameter) (Page 99)".

```
typedef struct
{
    PNIO_ADDR         *pAddr;                          //in
    PNIO_UINT32        RecordIndex;                    //in
    PNIO_REF           ReqRef;                         //in
    PNIO_ERR_STAT      Err;                            //in
} ATTR_PACKED PNIO_CBE_PRM_REC_WRITE_CONF;
```

Parameter

Name	Description
pAddr	Address of the module of the IO device for which the read data record job is intended.
RecordIndex	Data record number
ReqRef	Reference assigned by the IO-Base controller user program.
	Note The "ReqRef" reference specified with PNIO_rec_write_req( ) is returned. This allows the IO-Base controller user program to distinguish different jobs.
Err	Error code on the execution of the job.
	Note "Err" is the PROFINET IO error code transferred by the IO device for the error that occurred when this read data record job executed on the IO device.

4.7 Alarm interface

Description

Alarms are generated automatically and signaled using the callback event PNIO_CBE_ALARM_IND.

4.7.1 Callback event PNIO_CBE_ALARM_IND (signal alarm)

Description

The PNIO_CBE_ALARM_IND callback event reports an incoming alarm from an IO device to the IO controller; see the section "PNIO_CBF (PNIO callback function) (Page 100)".

If you want to use this callback event in the IO-Base controller user program, it must be registered with the PNIO_controller_open( ) function.

Note

Registration of this callback event with PNIO_controller_open( ) is optional.

If this callback event is not registered, an alarm is confirmed internally shortly after it arrives.

Note

Alarms are confirmed to the IO device internally after this callback function is exited.

Note

Until the callback function belonging to this callback event is exited, no further alarms are signaled to the IO-Base controller user program for this IO controller. They are signaled later.

Note

After the device return alarm (PNIO_ALARM_DEV_RETURN) or the plug alarm (PNIO_ALARM_PLUG) arrives, invalid IO data with the BAD status is transferred to the IO device. For this reason, the IO-Base controller user program must write all output data with valid values and the status GOOD after the return or plug alarm. This corresponds to an initialization of the output data.

Syntax

The following syntax shows the specific parameters for this event as part of the `union` from the PNIO_CBE_PRM structure; see the section "PNIO_CBE_PRM (callback event parameter) (Page 99)".

```
typedef struct
{
    PNIO_ADDR                       *pAddr;            //in
    PNIO_REF                         ReqRef;           //in
    const PNIO_CTRL_ALARM_DATA      *pAlarmData;       //in
} ATTR_PACKED PNIO_CBE_PRM_ALARM_IND;
```

Parameter

Name	Description	
pAddr	Address of the device from which the alarm originates. For example, the following addresses are transferred:	
	Failure of a ...	**Meaning of "pAddr"**
	IO device (station failure alarm)	Base address of the IO device (diagnostics address)
	module (pull alarm)	Module address
IndRef	Internally assigned reference (reserved)	
pAlarmData	Structure containing information on the alarms.	
	Note "pAlarmData" points to data in the PNIO_CTRL_ALARM_DATA structure. This is valid only within the callback function and becomes invalid once the function completes! The data must not be changed during processing of the callback function.	
	Note The PNIO_CTRL_ALARM_DATA structure is defined in the pniousrx.h header file and contains, among other things, the parameters "Alarm type" and "Alarm specifier". For more detailed information on alarms, refer to the following documentation: • STEP 7 documentation, for example in the STEP 7 online help on SFB 54. • Excerpt from the PROFINET IO standard in the SIMATIC NET document "Alarm ASE.pdf" on the "SIMATIC NET, PC Software" CD. • Documentation of the IO device	

Return values

No return values

4.8 Reading out the configuration

Overview

The diagnostics interface for reading out the configuration provides the IO controller user program with diagnostics request services.

To request the diagnostics services, the diagnostics interface makes the PNIO_ctrl_diag_req( ) function and the PNIO_CBE_CTRL_DIAG_CONF callback event available:

The following diagnostics request service is available:

Diagnostics service	Description
PNIO_CTRL_DIAG_CONFIG_SUBMODULE_LIST	List of all configured submodules

4.8.1 PNIO_ctrl_diag_req( ) (trigger diagnostics request)

Description

This function triggers a diagnostics request on the IO controller. The result of the job is signaled by the PNIO_CBE_CTRL_DIAG_CONF callback event if the function returned PNIO_OK.

If the IO-Base controller user program wants to use this function, it must first register the PNIO_CBE_CTRL_DIAG_CONF callback event with the PNIO_register_cbf( ) function.

Syntax

```
PNIO_UINT32 PNIO_ctrl_diag_req(
 PNIO_UINT32 Handle,
 PNIO_CTRL_DIAG *pDiagReq
);
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )
pDiagReq	Diagnostics request - Pointer to a filled PNIO_CTRL_DIAG structure containing the diagnostics request job; for the structure of the diagnostics request, refer to the section "PNIO_CTRL_DIAG (diagnostics request) (Page 102)".
	Note The memory to which "pDiagReq" points remains occupied by the IO-Base interface until the confirmation is received. This is signaled by the PNIO_CBE_CTRL_DIAG_CONF( ) callback event. As soon as the diagnostics request result is signaled with the PNIO_CBE_CTRL_DIAG_CONF callback event, the memory is returned to the IO-Base controller user program.

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
- PNIO_ERR_PRM_POINTER
- PNIO_ERR_PRM_INVALIDARG
- PNIO_ERR_PRM_ADD

4.8.2 Callback event PNIO_CBE_CTRL_DIAG_CONF (signal result of the diagnostics request)

Description

The PNIO_CBE_CTRL_DIAG_CONF callback event reports the result of the diagnostics request to the IO controller; see also the section "PNIO_ctrl_diag_req( ) (trigger diagnostics request) (Page 71)".

If you want to use this callback event in the IO-Base user program, it must be registered with the PNIO_register_cbf( ) function.

Syntax

The following syntax shows the specific parameters for this event as part of the "union" from the PNIO_CBE_PRM structure; see the section "PNIO_CBE_PRM (callback event parameter) (Page 99)".

```
typedef struct
{
    PNIO_CTRL_DIAG      *pDiagData;                  //in
    PNIO_UINT32          DiagDataBufferLen;          //in
    PNIO_UINT8          *pDiagDataBuffer;            //in
    PNIO_UINT32          ErrorCode;                  //in
} ATTR_PACKED PNIO_CBE_CTRL_DIAG_CONF;
```

Parameter

Name	Description
pDiagData	Pointer to the diagnostics request - points to the diagnostics request transferred to the PNIO_ctrl_diag_req( ) function; for the structure of the diagnostics request, refer to the section "PNIO_CTRL_DIAG (diagnostics request) (Page 102)".
DiagDataBufferLen	Length of the diagnostics reply in "pDiagDataBuffer"
pDiagDataBuffer	Pointer to the buffer with the diagnostics response - Interpret the diagnostics response according to the diagnostics service PNIO_CTRL_DIAG_ENUM; see the section "PNIO_CTRL_DIAG_ENUM (diagnostics service) (Page 103)".
	Note This buffer is valid only within the callback. The structure of this buffer depends on the diagnostics request.
ErrorCode	Error code that occurred when executing the diagnostics service.

Return values

No return values.

4.9 Station signaling with LEDs

Overview

This interface consists of the following callbacks events:

- PNIO_CBE_START_LED_FLASH_IND
- PNIO_CBE_STOP_LED_FLASH_IND

4.9.1 Callback event PNIO_CBE_START_LED_FLASH_IND (activate flash mode for LED)

Description

The PNIO_CBE_START_LED_FLASH_IND callback event signals the IO-Base controller user program that a request for identification has been received.

The IO-Base controller user program can then take suitable measures to attract the attention of the operator, for example by lighting up a diode. The callback event must already have been registered with the PNIO_register_cbf( ) function.

The syntax below shows the specific parameters for this event as part of the "union" from the PNIO_CBE_PRM structure; see the section "PNIO_CBE_PRM (callback event parameter) (Page 99)".

Note

The PNIO_CBE_START_LED_FLASH_IND and PNIO_CBE_STOP_LED_FLASH_IND callbacks occur alternately every three seconds as long as the module is expected to make itself noticeable.

When PNIO_CBE_START_LED_FLASH_IND( ) arrives, the LED should flash at the frequency specified for the "Frequency" parameter.

Syntax

```
typedef struct
{
    PNIO_UINT32       Frequency;                    //in
} ATTR_PACKED PNIO_CBE_PRM_START_LED_FLASH_IND;
```

Parameter

Name	Description
Frequency	Specified frequency for signaling in hertz

4.9.2 Callback event PNIO_CBE_STOP_LED_FLASH_IND (deactivate flash mode for LED)

Description

The PNIO_CBE_STOP_LED_FLASH_IND callback event signals the IO-Base controller user program that a request to stop identification has been received.

The callback event must already have been registered with the PNIO_register_cbf( ) function.

The following syntax shows the specific parameters for this event as part of the "union" from the PNIO_CBE_PRM structure; see the section "PNIO_CBE_PRM (callback event parameter) (Page 99)".

Syntax

This event type has no other specific parameters as part of the "union" from the PNIO_CBE_PRM structure; see the section "PNIO_CBE_PRM (callback event parameter) (Page 99)".

4.10 Callback Event PNIO_CBE_CP_STOP_REQ (signal remote download request)

Description

The PNIO_CBE_CP_STOP_REQ callback event signals your IO-Base controller user program that a remote download request (firmware update or reconfiguration) was received over the network. The callback event must already have been registered with the PNIO_register_cbf( ) function.

On receiving this callback, the user program must send a PNIO_controller_close( ). Following this, the user program can once again call PNIO_controller_open( ).

The functions PNIO_controller_close( ) and PNIO_controller_open( ), however, must not execute within the function belonging to the callback event PNIO_CBE_CP_STOP_REQ.

During the download, the PNIO_controller_open( ) function returns with the error code "PNIO_ERR_CONFIG_IN_UPDATE" or with the error code "PNIO_ERR_NO_FW_COMMUNICATION".

Following a successful download, the PNIO_controller_open( ) function returns with the "PNIO_OK" error code.

Note

If a user program does not register this callback, there can be no remote download request or reconfiguration as long as the user program is active.

Syntax

This event type has no other specific parameters as part of the "union" from the PNIO_CBE_PRM structure; see the section "PNIO_CBE_PRM (callback event parameter) (Page 99)".

4.11 Interface for isochronous real-time mode (IRT)

Overview

This interface consists of the following functions and callback events:

- PNIO_CP_register_cbf( ) function
- Callback event PNIO_CP_CBE_STARTOP_IND
- Callback event PNIO_CP_CBE_OPFAULT_IND
- PNIO_CP_set_opdone( ) function

4.11.1 PNIO_CP_register_cbf( ) (register callbacks)

Description

This function registers a callback function.

Note

Callback functions can only be registered by the IO controller user program in OFFLINE mode.

Note

The PNIO_CP_CBE_OPFAULT_IND callback event type must be registered before the PNIO_CP_CBE_STARTOP_IND callback event type.

Syntax

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

Parameter

Name	Description
CpHandle	Handle from PNIO_controller_open( ) or PNIO_device_open( )
CbeType	Callback event type for which the callback function "Cbf" will be registered; see the section "PNIO_CP_CBE_TYPE (callback event type) (Page 112)".
Cbf	Callback function to be started after arrival of the callback event "CbeType".
	Note The function pointer must not be NULL.

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_INVALID_CONFIG
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

4.11.2 Callback event PNIO_CP_CBE_STARTOP_IND (start of isochronous real-time data processing)

Description

The PNIO_CP_CBE_STARTOP_IND callback event informs your IO-Base user program of 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.

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

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 (callback event parameter) (Page 113)".

Syntax

```
typedef struct
{
    PNIO_UINT32          CpHandle;             //in
    PNIO_CYCLE_INFO      CycleInfo;            //in
} ATTR_PACKED PNIO_CP_CBE_PRM_STARTOP_IND;
```

Parameter

Name	Description
CpHandle	Handle from PNIO_controller_open( ) or PNIO_device_open( )
CycleInfo	Information on the current cycle

Note

During the PNIO_CP_CBE_STARTOP_IND event, you must not call any functions that might endanger the real-time capability of your IO-Base user program. 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 IO-Base user interface.

4.11.3 PNIO_CP_set_opdone( ) (end of processing the isochronous real-time data)

Description

By calling this function, the IO-Base user program signals the IO-Base interface that it has completed isochronous real-time processing of the IRT IO data. The IO-Base 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            //in
    PNIO_CYCLE_INFO     *CycleInfo;          //out
);
```

Parameter

Name	Description
CpHandle	Handle from PNIO_controller_open( ) or PNIO_device_open( )
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

4.11.4 Callback event PNIO_CP_CBE_OPFAULT_IND (violation of isochronous real-time mode)

Description

The PNIO_CP_CBE_OPFAULT_IND callback event signals a violation of the isochronous real-time mode to your IO-Base user program. This means that your user program called PNIO_CP_set_opdone( ) too late after receiving the PNIO_CP_CBE_STARTOP_IND callback event.

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 (callback event parameter) (Page 113)".

Syntax

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

Parameter

Name	Description
CpHandle	Handle from PNIO_controller_open( ) or PNIO_device_open( )
CycleInfo	Information on the current cycle

4.12 Interface for signaling the start of a new bus cycle

Description

This interface consists of the following callback event:

PNIO_CP_CBE_NEWCYCLE_IND

4.12.1 Callback event PNIO_CP_CBE_NEWCYCLE_IND (new bus cycle)

Description

The PNIO_CP_CBE_NEWCYCLE_IND callback event signals the start of a new bus cycle to the IO-Base 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 (callback event parameter) (Page 113)".

Syntax

```
typedef struct
{
    PNIO_UINT32           CpHandle;           //in
    PNIO_CYCLE_INFO       CycleInfo;          //in
} ATTR_PACKED PNIO_CP_CBE_PRM_NEWCYCLE_IND;
```

Parameter

Name	Description
CpHandle	Handle from PNIO_controller_open( ) or PNIO_device_open( )
CycleInfo	Information on the current cycle

4.13 PROFIenergy programming interface

4.13.1 Overview of the PROFIenergy programming interface

Note

In the following text, "PROFIenergy" is abbreviated to "PE".

IO devices that support PE functions can be switched to an energy saving mode by the IO controller during longer breaks in operation and returned to the normal operating state again. With IO devices that have a large current consumption, this can lead to considerable savings in energy. The functions for this are as follows:

- PNIO_register_pe_cbf()

  This function is responsible for registering the callback function. The callback function for processing the responses must be implemented by the user.
- PNIO_pe_cmd_req()
  Function for PE jobs. The PE jobs are selected based on job IDs.
- cbf_pe_cmd_conf()
  The PE job results are signaled to the user by a common callback function.

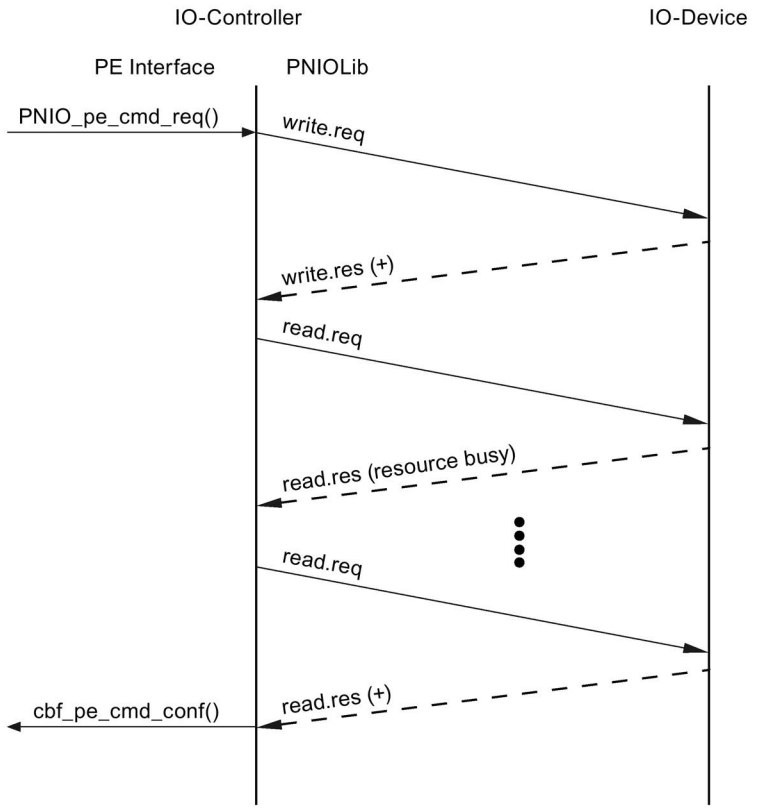
These functions are made available to the user program by the PNIOLib. The PNIOLib handles the PE jobs using write record and read record communication with the IO device. Each PE job consists of a request/response pair. With write record, a request is sent to an IO device. With read record, a response is received. Since the response will not be immediately available, read record must be repeated (polled) until the response is available. The maximum wait time is defined by the constant PNIO_PE_SERVICE_REQ_LIFETIME_DEFAULT and is 10 seconds. The PE job results are made available to the user program again by the PNIOLib.

Note

ET200S PE configuration

Before you can use the PE functions, you will need to configure the ET200S PE appropriately. Refer to the section "PROFIenergy as device (CP 1604 / CP 1616) (Page 95)".

The following figure shows the sequence of the functions:



Description of the figure: write.req is a data record job sent by the PNIOLib to the IO device. The PE job is embedded in the data record data. write.res(+) is a positive response from the IO device that signals that the data record job was received. The results of the PE job are fetched from the IO device with the data record job read.req. If the results are not available, the IO device responds with an error "resource busy". The read record job read.req is repeated until the results are available. (maximum 10 seconds).

4.13.2 Description of the PROFIenergy functions

4.13.2.1 PNIO_register_pe_cbf()

Description

This function registers a PE confirmation callback function.

Note

This callback function can be registered in any IO controller operating mode. The registration of the callback function must, however, take place after the PNIO_controller_open( ) and before the first PNIO_pe_cmd_req() function call.

If the registration does not take place, the PNIO_pe_cmd_req() function returns the error: PNIO_ERR_SEQUENCE

Syntax

```
PNIO_UINT32 PNIO_register_pe_cbf(
 PNIO_UINT32 Handle,
 PNIO_PE_CBF cbf_pe_cmd_conf,
);
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )
cbf_pe_cmd_conf	Address of the callback function to be called after arrival of the PE confirmation from the PNIOLib. **Note** The function pointer must not be NULL.

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_WRONG_HND
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_ALREADY_DONE

4.13.2.2 PNIO_pe_cmd_req()

Description

This function is used to send PE jobs to an IO device. It is a group function for all PE requests of a controller application. The results (confirmations) of the PE jobs are signaled by the PE group callback function cbf_pe_cmd_conf.

If the IO-Base controller user program wants to use this function (in other words the PROFIenergy functionality), it must already have registered the callback function with the PNIO_register_pe_cbf() call (PNIO_register_pe_cbf() (Page 83)).

You will find an overview of all possible PE jobs in Table 4-1 List of all possible PE jobs (Page 88).

Syntax

```
PNIO_UINT32 PNIO_pe_cmd_req(
 PNIO_UINT32 Handle,
 PNIO_ADDR *pAddr,
 PNIO_REF ReqRef,
 PNIO_PE_REQ_PRM *pPeReqPrm,
);
```

Parameter

Name	Description
Handle	Handle from PNIO_controller_open( )
pAddr	Logical address of the head module of the IO device for which the job is intended.
ReqRef	Job reference assigned by the IO-Base controller user program. **Note** The "ReqRef" reference allows the IO-Base controller user program to distinguish different PE jobs. This parameter can be assigned any value by the IO-Base controller user program and is returned in the callback event.
pPeReqPrm	Pointer to the parameter structure "PNIO_PE_REQ_PRM" with the corresponding "PNIO_PE_PRM_xxx_REQ" parameter structures. With jobs that do not require any parameters, only the CmdId and CmdModifier are evaluated. Refer to the description of the "PNIO_PE_REQ_PRM" structure (PE job parameter); PNIO_PE_REQ_PRM (Page 90) **Note** The content of the memory to which "pPeReqPrm" points is copied, in other words on the return from the function, the memory can be used again.

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
- PNIO_ERR_SEQUENCE

4.13.2.3 Callback event PNIO_PE_CBF (PE job result)

Description

The PNIO_PE_CBF callback event reports the result of a PE job sent previously with a PNIO_pe_cmd_req call. The callback function must be implemented by the user and registered by the PNIO_register_pe_cbf() call. See section 5.13.2.3

The callback function can be given any name by the user. In this manual, for example, the name "cbf_pe_cmd_conf" is used.

Syntax

Prototype (defined in pniousrx.h):

```
typedef void (* PNIO_PE_CBF)(
 PNIO_PE_CBE_PRM * pPeCbfPrm
);
```

Definition (for implementation in the application):

```
void cbf_pe_cmd_conf(PNIO_PE_CBE_PRM *pPeCbfPrm)
{
 //PE confirmation processing …
 …
}
```

Parameter

Name	Description
pPeCbfPrm	Pointer to PE confirmation parameter The various callback events have the same data type PNIO_PE_CBE_PRM (see PNIO_PE_CBE_PRM (callback event parameter) (Page 86)) that groups the various parameters of the individual PE callback events using a "union". **Note** After exiting this callback function, the "pPeCbfPrm" parameters are invalid. If necessary, these must be saved prior to this.

Return values

No return values.

4.13.3 Description of the PROFIenergy data types

4.13.3.1 PNIO_PE_CBE_PRM (callback event parameter)

Description

The various PE callback events have the same data type PNIO_CBE_PRM that groups the various response parameters of the individual PE jobs using a "union".

Syntax

```
typedef struct {
 PE_CBE_HDR pe_hdr;
 union {
 PNIO_PE_PRM_GENERIC_CONF_NEG RespNegative;
 PNIO_PE_PRM_START_PAUSE_CONF StartPause;
 PNIO_PE_PRM_END_PAUSE_CONF EndPause;
 PNIO_PE_PRM_PE_IDENTIFY_CONF PeIdentify;
 PNIO_PE_PRM_PEM_STATUS_CONF PemStatus;
 PNIO_PE_PRM_Q_MODE_LIST_ALL_CONF ModeList;
 PNIO_PE_PRM_Q_MODE_GET_MODE_CONF ModeGet;
 PNIO_PE_PRM_Q_MSRMT_GET_LIST_CONF MeasurementList;
 PNIO_PE_PRM_Q_MSRMT_GET_VAL_CONF MeasurementValue;
 } pe_prm;
} ATTR_PACKED PNIO_PE_CBE_PRM;
```

Parameter

Name	Description
pe_hdr	The pe_hdr structure contains parameters that are shared by all PE callback events. Structure as in section "PE_CBE_HDR (PROFIenergy callback event header) (Page 87)"
pe_prm	"union" of the job-specific parameter structures. The selection of the correct parameter structure is based on the elements CmdId and struct_id from the pe_hdr structure.

The following table shows the dependency of the parameter structures of CmdId and struct_id:

pe_prm parameter structures	CmdId struct_id
PNIO_PE_PRM_GENERIC_CONF_NEG	CmdId not valid struct_id not valid
PNIO_PE_PRM_START_PAUSE_CONF	PNIO_PE_CMD_START_PAUSE PE_CNF_STRUCT_ID_UNIQUE
PNIO_PE_PRM_END_PAUSE_CONF	PNIO_PE_CMD_END_PAUSE PE_CNF_STRUCT_ID_UNIQUE

pe_prm parameter structures	CmdId struct_id
PNIO_PE_PRM_PE_IDENTIFY_CONF	PNIO_PE_CMD_PE_IDENTIFY PE_CNF_STRUCT_ID_UNIQUE
PNIO_PE_PRM_PEM_STATUS_CONF	PNIO_PE_CMD_PEM_STATUS PE_CNF_STRUCT_ID_UNIQUE
PNIO_PE_PRM_Q_MODE_LIST_ALL_CONF	PNIO_PE_CMD_Q_MODES PE_CNF_STRUCT_ID_QMODE_LIST_ALL
PNIO_PE_PRM_Q_MODE_GET_MODE_CONF	PNIO_PE_CMD_Q_MODES PE_CNF_STRUCT_ID_QMODE_GET_MODE
PNIO_PE_PRM_Q_MSRMT_GET_LIST_CONF	PNIO_PE_CMD_Q_MSRMT PE_CNF_STRUCT_ID_QMSRMT_LIST_ALL
PNIO_PE_PRM_Q_MSRMT_GET_VAL_CONF	PNIO_PE_CMD_Q_MSRMT PE_CNF_STRUCT_ID_QMSRMT_GET_VAL

4.13.3.2 PE_CBE_HDR (PROFIenergy callback event header)

Description

This structure describes parameters of the "cbf_pe_cmd_conf" callback function that are shared by all PE callback events.

Syntax

```
typedef struct {
 PNIO_PE_CMD_ENUM CmdId;
 PNIO_UNIT32 Handle;
 PNIO_ADDR Addr;
 PNIO_REF Ref;
 PNIO_ERR_STAT Err;
 PNIO_UINT16 state;
 PNIO_UINT16 struct_id;
 PNIO_UINT32 length;
} ATTR_PACKED PE_CBE_HDR;
```

Parameter

Name	Description
CmdId	PE job IDs (PNIO_PE_CMD_ENUM (Page 88))
Handle	"Handle", transferred with PNIO_pe_cmd_req()
Addr	Address of the head module from the job
Ref	Returned ReqRef of the PE job
Err	Error code on the execution of the job **Note** "Err" is the PROFINET IO error code transferred by the IO device for the error that occurred when this read data record job executed on the IO device.

Name	Description
state	Job result: 1= ok (job successful) 2=err (job could not be executed)
struct_id	Structure ID of the job result: Depending on the CmdId, the struct_id precisely defines the type of the response parameters. This distinction is necessary with "group jobs", for example PNIO_PE_CMD_Q_MODES. With these jobs, the CmdId is not unique. Possible combinations are described in the section PNIO_PE_CBE_PRM; PNIO_PE_CBE_PRM (callback event parameter) (Page 86)
length	Length in bytes of the following parameters

4.13.3.3 PNIO_PE_CMD_ENUM

Description

This enumeration type lists the PE jobs. This means both requests and confirmations.

Syntax

```
typedef enum {
 PNIO_PE_CMD_RESERVED =0,
 PNIO_PE_CMD_START_PAUSE =1,
 PNIO_PE_CMD_END_PAUSE =2,
 PNIO_PE_CMD_Q_MODES =3,
 PNIO_PE_CMD_PEM_STATUS =4,
 PNIO_PE_CMD_PE_IDENTIFY =5,
 PNIO_PE_CMD_Q_MSRMT =16
} PNIO_PE_CMD_ENUM;
```

Parameter

Table 4- 1 List of all possible PE jobs

PE job	Description
PNIO_PE_CMD_START_PAUSE	Start energy saving mode
PNIO_PE_CMD_END_PAUSE	End energy saving mode
PNIO_PE_CMD_Q_MODES	Returns information about the energy saving mode depending on the CmdModifier
PNIO_PE_CMD_PEM_STATUS	Returns the current energy saving mode
PNIO_PE_CMD_PE_IDENTIFY	Returns a list of all supported PE jobs
PNIO_PE_CMD_Q_MSRMT	Depending on the CmdModifier, returns information about the energy consumption

4.13.3.4 PNIO_PE_CMD_MODIFIER_ENUM

Description

This enumeration type lists the PE command modifiers required to define the precise job type with "group jobs". PE command modifiers are used with requests. PE job modifier is an element of the PNIO_PE_REQ_PRM structure. See section "PNIO_PE_REQ_PRM (Page 90)"

With confirmations, this function is handled by the data structure identifier (struc_id). See section "PNIO_PE_CBE_PRM (callback event parameter) (Page 86)"

Note

Group jobs are the PNIO_PE_CMD_Q_MODES (query modes) and PNIO_PE_CMD_Q_MSRMT (query measurement) jobs.

Syntax

```
typedef enum {
 PE_CMD_MODIF_NOT_USED = 0,
 PE_CMD_MODIF_Q_MODE_LIST_ALL = 1,
 PE_CMD_MODIF_Q_MODE_GET_MODE = 2,
 PE_CMD_MODIF_Q_MSRMT_LIST_ALL = 1,
 PE_CMD_MODIF_Q_MSRMT_GET_VAL = 2,
} PNIO_PE_CMD_MODIFIER_ENUM;
```

Parameter

PE job modifier	Description
PE_CMD_MODIF_NOT_USED	For all unique jobs. These do not require a modifier.
PE_CMD_MODIF_Q_MODE_LIST_ALL	With PNIO_PE_CMD_Q_MODES job. Returns a list of all available energy saving modes.
PE_CMD_MODIF_Q_MODE_GET_MODE	With PNIO_PE_CMD_Q_MODES job. Returns data of an energy saving mode.
PE_CMD_MODIF_Q_MSRMT_LIST_ALL	With PNIO_PE_CMD_Q_MSRMT job. Returns a list of all available measured values (list of measured value IDs)
PE_CMD_MODIF_Q_MSRMT_GET_VAL	With PNIO_PE_CMD_Q_MSRMT job. Returns data of a measured value.

4.13.3.5 PNIO_PE_REQ_PRM

Description

The PE job parameters for the individual PE jobs are transferred in suitable parameter structures. All these parameter structures are grouped together in a "union" in the PNIO_PE_REQ_PRM structure. The individual structures are described in detail in the following sections.

Syntax

```
typedef struct {
 PNIO_PE_CMD_ENUM CmdId;
 PNIO_PE_CMD_MODIFIER_ENUM CmdModifier;
 union {
```

"union" of the various PE job parameter structures (details in the relevant sections)

```
 } rq;
} ATTR_PACKED PNIO_PE_REQ_PRM;
```

Parameter

Name	Description
CmdId	PE job ID
CmdModifier	PE job modifier
rq	"union" of all PE job parameter structures

4.13.3.6 PNIO_PE_PRM_START_PAUSE_REQ

Description

Parameter for the PNIO_PE_CMD_START_PAUSE PE job that triggers the start of a break in operation.

Syntax

```
typedef struct {
 PNIO_UINT32 time_ms;
} ATTR_PACKED PNIO_PE_PRM_START_PAUSE_REQ;
```

Parameter

Name	Description
time_ms	Specifies the expected duration of the break in milliseconds. If this time is too short for an IO device, this IO device does not pause operation.

4.13.3.7 PNIO_PE_CMD_START_PAUSE_CONF

Description

Parameter of the callback event that confirms the start of the break in operation.

Syntax

```
typedef struct {
 PNIO_UINT32 pe_mode_id;
} ATTR_PACKED PNIO_PE_PRM_START_PAUSE_CONF;
```

Parameter

Name	Description	
pe_mode_id	Mode reached	
	0x00	Energy saving mode
	0x01…0xFE	Vendor-specific
	0xFF	ready to function

4.13.3.8 PNIO_PE_PRM_END_PAUSE_CONF

Description

Parameter of the callback event that confirms the end of the break in operation.

Syntax

```
typedef struct {
 PNIO_UINT32 time_ms;
} ATTR_PACKED PNIO_PE_PRM_END_PAUSE_CONF;
```

Parameter

Name	Description
time_ms	Time required in milliseconds to reach the ready to operate status.

4.13.3.9 PNIO_PE_PRM_PE_IDENTIFY_CONF

Description

Parameter of the callback event that returns a list of all supported PE services.

Syntax

```
typedef struct {
 PNIO_UINT8 numServices;
 PNIO_UINT8 serviceId[254];
} ATTR_PACKED PNIO_PE_PRM_PE_IDENTIFY_CONF;
```

Parameter

Name	Description
numServices	Number of valid parameters in the following serviceId field
serviceId[254]	Field of supported PE job IDs

4.13.3.10 PNIO_PE_PRM_PEM_STATUS_CONF

Description

Parameter of the callback event that describes the current energy saving mode.

Syntax

```
typedef struct {
 PNIO_UINT8 PE_Mode_ID_Source;
 PNIO_UINT8 PE_Mode_ID_Destination;
 PNIO_UINT32 Time_to_operate;
 PNIO_UINT32 Remaining_time_to_destination;
 float Mode_Power_Consumption;
 float Energy_Consumption_to_Destination;
 float Energy_Consumption_to_operate;
} ATTR_PACKED PNIO_PE_CMD_PEM_STATUS_CONF;
```

Parameter

Name	Description	
PE_Mode_ID_Source;	Original energy saving mode	
	0x00	Energy saving mode (power off)
	0x01…0xFE	Vendor-specific
	0xFF	ready for operation
PE_Mode_ID_Destination	Energy saving mode to be reached	
	0x00	Energy saving mode
	0x01…0xFE	Vendor-specific
	0xFF	ready for operation
Time_to_operate	Maximum time before the ready to operate status is reached	
Remaining_time_to_destination	Time remaining until reaching the target status	

Name	Description
Mode_Power_Consumption	Energy consumption in the current energy saving mode in [kW] If unknown: = 0.0 (Float 32)
Energy_Consumption_to_Destination	Energy consumption of the current status change in [kWh]
Energy_Consumption_to_operate	Energy consumption from the current energy saving status until reaching the ready to operate status in [kWh]

4.13.3.11 PNIO_PE_PRM_Q_MODE_LIST_ALL_CONF

Description

Parameter of the callback event that returns a list of all supported energy saving modes.

Syntax

```
typedef struct {
 PNIO_UINT8 numModeIds;
 PNIO_UINT8 peModeId[254];
} ATTR_PACKED PNIO_PE_PRM_Q_MODE_LIST_ALL_CONF;
```

Parameter

Name	Description	
numModeIds;	Number of valid parameters in the following peModeId[254] field	
peModeId[254]	Field with IDs of all supported energy saving modes Energy saving mode IDs:	
	0	Energy saving mode (power off)
	1-0xFE	Vendor-specific
	0xFF	ready for operation

4.13.3.12 PNIO_PE_PRM_Q_MODE_GET_MODE_REQ

Description

Parameter for the request to signal all details of a certain energy saving mode.

Syntax

```
typedef struct {
 PNIO_UINT8 peModeId;
 PNIO_UINT8 reserved;
} ATTR_PACKED PNIO_PE_PRM_Q_MODE_GET_MODE_REQ;
```

Parameter

Name	Description
peModeId;	Energy saving mode ID
reserved;	reserved

4.13.3.13 PNIO_PE_PRM_Q_MODE_GET_MODE_CONF

Description

Parameter of the callback event that returns details of a specific energy saving mode.

Syntax

```
typedef struct {
 PNIO_UINT8 peModeId;
 PNIO_UINT8 PE_Mode_Attributes;
 PNIO_UINT32 Time_min_Pause;
 PNIO_UINT32 Time_to_Pause;
 PNIO_UINT32 Time_to_operate;
 PNIO_UINT32 Time_min_length_of_stay;
 PNIO_UINT32 Time_max_length_of_stay;
 float Mode_Power_Consumption;
 float Energy_Consumption_to_pause;
 float Energy_Consumption_to_operate;
} ATTR_PACKED PNIO_PE_PRM_Q_MODE_GET_MODE_CONF;
```

Parameter

Name	Description
peModeId	Energy saving status ID
PE_Mode_Attributes	Bit 0; Possible values: 0 = time and performance/energy information is static 1 = time and performance/energy information is dynamic Bit 1 to 7: reserved
Time_min_Pause	Minimum duration of break in operation. Is the sum in milliseconds of the three following parameters: • Time_to_Pause • Time_to_operate • Time_min_length_of_stay
Time_to_Pause	Expected duration before the energy saving mode is reached. Can only be static.
Time_to_operate	Expected duration before reaching the ready to operate status. Can be either static or dynamic.
Time_min_length_of_stay	Minimum duration of energy saving mode.
Time_max_length_of_stay	Maximum duration of energy saving mode.

Name	Description
Mode_Power_Consumption	Energy consumption in the current energy saving mode in [kW] If undefined: = 0.0 (Float 32)
Energy_Consumption_to_pause	Energy consumption of the changeover to the energy saving mode in [kWh]
Energy_Consumption_to_operate	Energy consumption from the current energy saving status until reaching the ready to operate status in [kWh]

4.13.4 Configuring PROFIenergy device

4.13.4.1 PROFIenergy as device (CP 1604 / CP 1616)

CP 1604 / CP1616 as PROFIenergy device

As of firmware version 2.6, the communications processors CP 1604 and CP 1616 support the energy management profile "PROFIenergy". This makes it possible to turn off the host PC in which these communications processors are installed using the PROFIenergy command "Start Pause" or to turn it on again with the PROFIenergy command "End Pause". To allow this, the CP 1616 requires an external power supply and must be hardware version 10. The CP 1604 also requires an external power supply and must be hardware version 8.

To be able to do this, the following must be taken into account in the user program:

If you want to use the PROFIenergy features, you also need to set the "PNIO_CEP_PE_MODE_ENABLE" flag in the "ExtPar" function parameter (second function parameter) when calling the "PNIO_device_open()" function. This is defined in the header file "pniolib/iobase/inc/pniobase.h" as follows: #define PNIO_CEP_PE_MODE_ENABLE 0x00000020

The PROFIenergy commands are sent by writing data records to the IO device. The reply of the device is obtained by reading data records. The callback functions that are called when using the IO device application, are "cbf_rec_write" (when the IO controller sends a PROFIenergy command) and "cbf_rec_read" (when the IO controller queries the reply of the IO device to the command). The IO controller always has to query the reply from the IO device. This means that every PROFIenergy command is handled by writing and reading data records.

You will find further information on the PROFIenergy mechanisms in the sample program "dev_certfify_energy".

PE configuration data record for ET200S PE

Description

The structure of the PE configuration data record is vendor-specific.

Before using the PE functions, the IO device needs to be configured appropriately. The configuration is based on a PE configuration data record and the "PNIO_rec_write_req( )" request (write record job).

Note

The power modules (PM) of an ET200S PE device are configured with the data record described below.

The configuration parameters are transferred to the ET200S PE with the write record function "PNIO_rec_write_req( )" with "RecordIndex" = 3.

Syntax

```
#define PNIO_PE_CFG_DEV_NUM_SLOTS_MAX 8
typedef struct {
 PNIO_UINT8 slot_num;
 PNIO_UINT8 power_mode;
} ATTR_PACKED PNIO_PE_CFG_SLOT;

typedef struct {
 PNIO_UINT8 reserved1;
 PNIO_UINT8 num_cfg_slots;
 PNIO_PE_CFG_SLOT slot[PNIO_PE_CFG_DEV_NUM_SLOTS_MAX];
} ATTR_PACKED PNIO_PE_PRM_CONFIG_DEVICE_REQ;
```

Parameter

Name	Description
slot_num	Slot of the power module to be turned off 1 to 62 0: An entry in this field is not relevant
power_mode	0: Continue working (do not turn off module) 1: Turn off
reserved1	Not used
num_cfg_slots	Number of Slot-Config blocks
slot [num_cfg_slots]	Field of "Slot-Config blocks" ((PNIO_PE_CFG_SLOT)) with length "num_cfg_slots". Max: PNIO_PE_CFG_DEV_NUM_SLOTS_MAX 8

4.14 Data types

The data types described below are used in the IO-Base user programming interface:

- Basic data types
- Special data types for the IO-Base user programming interface

4.14.1 Basic data types

The following basic data types are used:

File type	Definition
PNIO_UINT8	unsigned char (8 bits)
PNIO_UINT16	unsigned short (16 bits)
PNIO_UINT32	unsigned long (32 bits)
PNIO_REF	unsigned long (32 bits)

4.14.2 PNIO_MODE_TYPE (operating mode type)

Description

The PNIO_MODE_TYPE data type contains the IDs for the supported modes.

Syntax

```
typedef enum
{
    PNIO_MODE_OFFLINE,
    PNIO_MODE_CLEAR,
    PNIO_MODE_OPERATE
} PNIO_MODE_TYPE;
```

Elements

Name	Description
PNIO_MODE_OFFLINE	OFFLINE mode ("Stop")
PNIO_MODE_CLEAR	CLEAR mode
PNIO_MODE_OPERATE	OPERATE mode

4.14.3 PNIO_IO_TYPE (direction type)

Description

Identifier for the data direction of an address.

Syntax

```
typedef enum
{
    PNIO_IO_IN,
    PNIO_IO_OUT,
    PNIO_IO_INOUT
} PNIO_IO_TYPE;
```

Elements

Name	Description
PNIO_IO_IN	Identifier for the input data direction
PNIO_IO_OUT	Identifier for the output data direction
PNIO_IO_INOUT	Identifier for bi-directional data direction

4.14.4 PNIO_ADDR (address structure)

Description

The PNIO_ADDR address structure is used for addressing with all functions described here.

Syntax

```
typedef struct
{
    PNIO_ADDR_TYPE      AddrType;
    PNIO_IO_TYPE        IODataType;
    union
    {
        PNIO_UINT32     Addr;
        PNIO_UINT32     Reserved[5];
    } u;
} ATTR_PACKED PNIO_ADDR;
```

Parameter

Name	Description
AddrType	Must always have the value PNIO_ADDR_LOG for IO controllers.
IODataType	Input or output
Addr	Address
Reserved	Reserved for future expansion

4.14.5 PNIO_CBE_TYPE (callback event type)

Description

The IO-Base user programming interface recognizes the following callback event types:

Callback event type	Callback event
PNIO_CBE_ALARM_IND	Alarm arrived. *
PNIO_CBE_REC_READ_CONF	Result of the read data record job arrived. *
PNIO_CBE_REC_WRITE_CONF	Result of the write data record job arrived. *
PNIO_CBE_MODE_IND	The local mode has changed.
PNIO_CBE_DEV_ACT_CONF	Result of activating/deactivating the IO device arrived.
PNIO_CBE_CTRL_DIAG_CONF	Signal result of diagnostics request
PNIO_CBE_CP_STOP_REQ	Signal remote firmware download or reconfiguration request
PNIO_CBE_START_LED_FLASH	Activate flashing mode for LED
PNIO_CBE_STOP_LED_FLASH	Deactivate flashing mode for LED
PNIO_CBE_REMA_READ_CONF	???
PNIO_CBE_IOSYSTEM_RECONFIG	???
PNIO_CBE_IFC_SET_ADDR_CONF	???
PNIO_CBE_IFC_REC_READ_CONF	???
PNIO_CBE_IFC_ALARM_IND	???

* *This callback event type is registered with the "PNIO_controller_open()" function and cannot be registered in the "PNIO_register_cbf()" function.

4.14.6 PNIO_CBE_PRM (callback event parameter)

Description

The various callback events have the same data type PNIO_CBE_PRM that groups the various parameters of the individual callback events using a "union".

Syntax

```
typedef struct
{
    PNIO_CBE_TYPE     CbeType;                //in
    PNIO_UINT32       Handle;                 //in
    union
    {
        ...         /*Union over the various
        ...           callback event parameters
                      (details in relevant sections)*/
    };
} ATTR_PACKED PNIO_CBE_PRM;
```

Parameter

Name	Description
CbeType	Callback event
Handle	Handle from PNIO_controller_open( ) or from PNIO_device_open( ).

4.14.7 PNIO_CBF (PNIO callback function)

Description

Using the callback event parameter PNIO_CBE_PRM, a general PNIO callback function of the type PNIO_CBF is declared.

Syntax

```
typedef void (* PNIO_CBF)(
    PNIO_CBE_PRM       *pCbfPrm
);
```

4.14.8 ExtPar (extended parameter)

Description

The bits of "ExtPar" are used for parameter assignment.

Of the 32 possible bits, only the PNIO_CEP_MODE_CTRL and PNIO_CEP_SLICE_ACCESS bits are currently defined.

The bits are set when the PNIO_controller_open( ) function is called.

Parameter

Identifier	Description
PNIO_CEP_MODE_CTRL	The IO-Base user program that sets this bit is given the right to change its own mode. This bit must always be set.
PNIO_CEP_SLICE_ACCESS	This bit activates partial access, in other words, it is possible to write to or read from any subareas of a submodule. This ID is not supported by all products. You will find more detailed information in the readme of the relevant product.

Example 1

The following example sets the PNIO_CEP_MODE_CTRL bit:

```
PNIO_controller_open(
    ..., PNIO_CEP_MODE_CTRL, ...
);
```

Example 2

The following example sets the PNIO_CEP_MODE_CTRL and PNIO_CEP_SLICE_ACCESS bit:

```
PNIO_controller_open(
    ..., PNIO_CEP_MODE_CTRL | PNIO_CEP_SLICE_ACCESS, ...
);
```

Example of partial access

A 4-byte output module has start address 2. 2 bytes are written starting at address 3.

4.14.9 PNIO_DEV_ACT_TYPE (type for activating all deactivating an IO device)

Description

Identifiers for deactivating and activating an IO device.

Syntax

```
typedef enum
{
    PNIO_DA_FALSE,
    PNIO_DA_TRUE
} PNIO_DEV_ACT_TYPE;
```

Elements

Name	Description
PNIO_DA_FALSE	Disable
PNIO_DA_TRUE	Activate

4.14.10 PNIO_IOXS (status of the IO data)

Description

Identifiers to describe the status of the IO data.

Syntax

```
typedef enum
{
    PNIO_S_GOOD,
    PNIO_S_BAD
} PNIO_IOXS;
```

Elements

Name	Description
PNIO_S_GOOD	IO data is valid.
PNIO_S_BAD	IO data is invalid.

4.14.11 PNIO_CTRL_DIAG (diagnostics request)

Description

This structure is used to set up a diagnostics request. It is used with the PNIO_ctrl_diag_req( ) function and with the PNIO_CBE_CTRL_DIAG_CONF callback event.

Syntax

```
typedef struct
{
    PNIO_CTRL_DIAG_ENUM         DiagService;
    union
    {
        PNIO_UINT32             Reserved1[8];
        PNIO_ADDR               Addr;
    }u;
    PNIO_REF                    ReqRef;
    PNIO_UINT32                 Reserved2;
} ATTR_PACKED PNIO_CTRL_DIAG;
```

Elements

Name	Description
DiagService	Interpret the diagnostics reply - Diagnostics service according to the PNIO_CTRL_DIAG_ENUM diagnostics service; see the section "PNIO_CTRL_DIAG_ENUM (diagnostics service) (Page 103)".
Reserved1	Reserved for future expansion
Addr	Address of a submodule for the diagnostics query of the status of an IO device; applies to DiagService = PNIO_CTRL_DIAG_DEVICE_STATE.
ReqRef	Unique reference supplied by the user for this diagnostics request.
RESERVED2	Reserved for future expansion

4.14.12 PNIO_CTRL_DIAG_ENUM (diagnostics service)

Description

This enumeration type lists the diagnostics services. The supported services are described.

Syntax

```
typedef enum{
 PNIO_CTRL_DIAG_RESERVED = 0,
 PNIO_CTRL_DIAG_CONFIG_SUBMODULE_LIST = 1,
 PNIO_CTRL_DIAG_DEVICE_STATE = 2,
 PNIO_CTRL_DIAG_CONFIG_IOROUTER_PRESENT = 3,
 PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE_LIST= 4,
 #ifndef PNIO_SOFTNET /* not supported for PNIO SOFTNET */
 PNIO_CTRL_DIAG_CONFIG_NAME_ADDR_INFO = 5 /* FW V2.5.2.0 and higher */
 #endif /* PNIO_SOFTNET */
 PNIO_CTRL_DIAG_GET_COMM_COUNTER_DATA = 6,
 PNIO_CTRL_DIAG_RESET_COMM_COUNTERS = 7 /* FW V2.5.2.0 and greater */
 PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC = 8
} PNIO_CTRL_DIAG_ENUM;
```

Elements

Diagnostics service	Description
PNIO_CTRL_DIAG_CONFIG_SUBMODULE_LIST	Supplies a list of all configured submodules - The diagnostics reply buffer "pDataBuffer" in the PNIO_CBE_CTRL_DIAG_CONF callback event contains elements of the type PNIO_CTRL_DIAG_CONFIG_ SUBMODULE; see the section "PNIO_CTRL_DIAG_CONFIG_SUBMODULE (submodule configuration information) (Page 105)".
PNIO_CTRL_DIAG_DEVICE_STATE	The diagnostics query provides information on the status of an IO device. The diagnostics reply buffer "pDataBuffer" in the PNIO_CBE_CTRL_DIAG_CONF callback event contains an element of the type PNIO_CTRL_DIAG_DEVICE; see the section "PNIO_CTRL_DIAG_DEVICE_STATE (Page 115)". Is not supported by the PN Driver
PNIO_CTRL_DIAG_CONFIG_ IOROUTER_PRESENT	The diagnostics request provides information on whether there are output bit slices reserved for IO routing; see the section "PNIO_CTRL_DIAG_CONFIG_IOROUTER_PRESENT (diagnostics request about IO routers) (Page 107)". Is not supported by the PN Driver
PNIO_CTRL_DIAG_CONFIG_ OUTPUT_SLICE_LIST	The diagnostics request provides information on the size of the output bit slices remaining for IO routing; see the section "PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE_LIST (diagnostics query about IO router) (Page 107)". Is not supported by the PN Driver
PNIO_CTRL_DIAG_CONFIG_NAME_ADDR_INFO	The diagnostic query returns the configured device name, device type and the IP address parameters (IP address, mask and default router). The results are returned in the structure "PNIO_CTRL_DIAG_CONFIG_NAME_ADDR_INFO_DATA"; see the section "PNIO_CTRL_DIAG_CONFIG_NAME_ADDR_INFO_DATA (Page 109)".
PNIO_CTRL_DIAG_GET_COMM_COUNTER_DATA	This diagnostics query returns information on the quality of the data transfer in the form of statistical data of the Ethernet interface. Refer to the explanations of the structure "PNIO_CTRL_COMM_PORT_COUNTER_DATA and PNIO_CTRL_COMM_COUNTER_DATA (Page 110)". Is not supported by the PN Driver

Diagnostics service	Description
PNIO_CTRL_DIAG_RESET_COMM_COUNTERS	This diagnostics query returns information on the quality of the data transfer in the form of statistical data of the Ethernet interface. Refer to the explanations of the structure "PNIO_CTRL_COMM_PORT_COUNTER_DATA and PNIO_CTRL_COMM_COUNTER_DATA (Page 110)". In addition, the counters for the statistical data are reset to 0 after reading out the data. Is not supported by the PN Driver
PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC	The diagnostics request returns detailed information for an AR abort. The result is returned with the structure "PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC"; see section "PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC (Page 109)".

4.14.13 PNIO_CTRL_DIAG_CONFIG_SUBMODULE (submodule configuration information)

Description

The structure provides information about a configured submodule. It is returned for the diagnostics request PNIO_CTRL_DIAG_CONFIG_SUBMODUL_LIST.

Syntax

```
typedef struct{
 PNIO_ADDR Address;
 PNIO_UINT32 DataLength;
 PNIO_DATA_TYPE DataType;
 PNIO_COM_TYPE ComType;
 PNIO_UINT32 Api;
 PNIO_UINT32 ReductionFactor;
 PNIO_UINT32 Phase;
 PNIO_UINT32 CycleTime;
 PNIO_UINT32 HwIdentifier;
 PNIO_UINT8 AddressValid;
 PNIO_UINT8 Reserved1[3];
 PNIO_UINT32 StatNo;
 PNIO_UINT32 Slot;
 PNIO_UINT32 Subslot;
 PNIO_UINT32 Reserved2[2];
```

```
} ATTR_PACKED PNIO_CTRL_DIAG_CONFIG_SUBMODULE;
```

Elements

Name	Description
Address	Logical address of the submodule
DataLength	Data length of the submodule
	Note Logical addresses of submodules with the input length 0 are diagnostics addresses. For the meaning of the diagnostics addresses, refer to the STEP 7 documentation.
DataType	Specifies the type of IO data.
ComType	Specifies whether or not direct data exchange is used for the submodule.
Api	Specifies the Api to which the submodule belongs.
ReductionFactor	The "ReductionFactor" specifies how often data is sent. With, for example, a ReductionFactor of 2, data is sent every 2 * CycleTime.
Phase	If there is a "ReductionFactor" > 1, this specifies the send time within a cycle; Range of values: 0 to ReductionFactor – 1
CycleTime	Send clock in multiples of 31.25 µs
HwIdentifier	"Hardware Identifier" of the submodule
AddressValid	The structure element "Address" may only be used when "AddressValid" has a value other than 0.
StatNo	Number of the station in which the submodule is inserted.
Slot	Module number from the perspective of the device
Subslot	Submodule number from the perspective of the device
Reserved1	Reserved for future expansions
Reserved2	Reserved for future expansions

Note

The structure elements "HwIdentifier" and "AddressValid" only apply to the products CP1626 and PN Driver.

4.14.14 PNIO_CTRL_DIAG_CONFIG_IOROUTER_PRESENT (diagnostics request about IO routers)

Description

The PNIO_CTRL_DIAG_CONFIG_IOROUTER_PRESENT diagnostics request provides information about whether output bits of submodules are assigned to an external IO controller.

The notification comes from the callback function with callback event type PNIO_CBE_TYPE = PNIO_CBE_CTRL_DIAG_CONF registered in PNIO_register_cbf( ) (see also PNIO_CTRL_DIAG and PNIO_CTRL_DIAG_ENUM).

The parameters "DiagDataBuffer" and "DiagDataBufferLen" in the callback event structure PNIO_CBE_CTRL_DIAG_CONF (see the section "Callback event PNIO_CBE_CTRL_DIAG_CONF (signal result of the diagnostics request) (Page 72)") return the result of the request.

Parameter

DiagDataBufferLen	Description
0	IO router is not configured, no restriction for output bits.
Not equal to 0	Call the PNIO_ctrl_diag_req( ) function with the parameter `pDiagReq -> DiagService = PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE_LIST` that informs you of the output bit slices available for the local IO controller user program.

4.14.15 PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE_LIST (diagnostics query about IO router)

Description

If output bit slices are assigned to an external IO controller due to IO routing, the local IO controller user program can no longer write to these output bit slices.

The PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE_LIST diagnostics query provides information about the output bits that the local IO controller user program can write to. It is an extension of the PNIO_CTRL_DIAG_CONFIG_IOROUTER_PRESENT diagnostics request described in the section "PNIO_CTRL_DIAG_CONFIG_IOROUTER_PRESENT (diagnostics request about IO routers) (Page 107)".

The notification comes from the callback function with callback event type PNIO_CBE_TYPE = PNIO_CBE_CTRL_DIAG_CONF registered in PNIO_register_cbf( ) (see also PNIO_CTRL_DIAG and PNIO_CTRL_DIAG_ENUM).

The output bit slices available for the local IO controller user program are reported in the form of the PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE data structure described below. The data structures of several output bit slices follow one after the other.

The parameters "DiagDataBuffer" and "DiagDataBufferLen" in the callback event structure PNIO_CBE_CTRL_DIAG_CONF (see the section "Callback event PNIO_CBE_CTRL_DIAG_CONF (signal result of the diagnostics request) (Page 72)") return the result of the request.

Note

In contrast to the output bits that can only be assigned to a single IO controller, input bits can always be read by both IO controllers. There is therefore no need for an information function.

Syntax

```
typedef struct
{
    PNIO_ADDR        Address;
    PNIO_UINT16      BitOffset;
    PNIO_UINT16      BitLength;
} ATTR_PACKED PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE;
```

Elements

Name	Description
Address	Logical address of the submodule
BitOffset	Specified as a number of bits
BitLength	Specified as a number of bits

Example 1

IO routing will make output bit slice 2 to 5 of a submodule with a length of 8 bits available to an external IO controller for output.

The diagnostics request returns the structure shown above for the two remaining output bit slices (bits 0, 1 and bits 6, 7) twice with the same logical address of the submodule and the following content:

Remaining output bit slice	BitOffset	BitLength
1	0	2
2	6	2

Example 2

A non routed submodule with a length of 8 bits is completely available to the local IO controller user program. The diagnostics request returns "BitOffset" = 0 and "BitLength" = 8.

4.14.16 PNIO_CTRL_DIAG_CONFIG_NAME_ADDR_INFO_DATA

Description

This structure contains the PROFINET network parameters of the controller such as device name, device type and
the address parameters (IP address, mask and default router). These parameters are transferred to the controller by configuring with STEP 7.

Syntax

```
typedef struct {
 PNIO_UINT8 name[256];
 PNIO_UINT8 TypeOfStation[256];
 PNIO_UINT32 ip_addr;
 PNIO_UINT32 ip_mask;
 PNIO_UINT32 default_router;
} ATTR_PACKED PNIO_CTRL_DIAG_CONFIG_NAME_ADDR_INFO_DATA;

 PNIO_UINT32 PNIO_CODE_ATTR SERV_CP_set_type_of_station (PNIO_UINT32 CpIndex, char*
TypeName);
```

Parameter

Parameter	Description
name[256]	PROFINET station name of the module
TypeOfStation[256]	PROFINET device type
ip_addr	IP address of the module
ip_mask	IP subnet mask of the module
default_router	IP default gateway of the module

4.14.17 PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC

Description

The "PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC" structure returns diagnostics information and the status of an IO device.

To execute the request, the logical address of an IO device submodule must be specified in the "u.Addr" parameter with the function call of the function PNIO_ctrl_diag_req().

The result of the request is returned when the callback event "PNIO_CBE_CTRL_DIAG_CONF" is called. The parameter "DiagDataBuffer" contains the structure "PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC". The length is specified with the "DiagDataBufferLen".

Syntax

```
typedef struct {
 PNIO_DEV_ACT_TYPE Mode;
 PNIO_UINT16 ErrorCause;
 PNIO_UINT8 ReasonCode;
 PNIO_UINT8 AdditionalInfo[10];
} ATTR_PACKED PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC_DATA;
```

Elements

Name	Description
Mode	The current status of the IO device PNIO_DEV_ACT_TYPE; see the section "PNIO_DEV_ACT_TYPE (type for activating all deactivating an IO device) (Page 101)".
ErrorCause	detailed IO device diagnostics
ReasonCode	detailed IO device diagnostics
AdditionalInfo[10]	detailed IO device diagnostics

4.14.18 PNIO_CTRL_COMM_PORT_COUNTER_DATA and PNIO_CTRL_COMM_COUNTER_DATA

Description

These structures return information on the quality of the data transfer in the form of statistical data of the Ethernet interface.

Syntax

```
typedef struct {
 PNIO_UINT32 SndNoError;
 PNIO_UINT32 SndCollision;
 PNIO_UINT32 SndOtherError;
 PNIO_UINT32 RcvNoError;
 PNIO_UINT32 RcvResError;
 PNIO_UINT32 RcvRejected;
} ATTR_PACKED PNIO_CTRL_COMM_PORT_COUNTER_DATA;

typedef struct {
 PNIO_CTRL_COMM_PORT_COUNTER_DATA Port[4];
 PNIO_UINT16 NumberOfPorts;
} ATTR_PACKED PNIO_CTRL_COMM_COUNTER_DATA;
```

Note

The content of Port[3] is not relevant. if NumberOfPorts = 3.

Parameter (PNIO_CTRL_COMM_PORT_COUNTER_DATA)

Parameter	Description
SndNoError	Frames sent free of error
SndCollision	Failed attempt to send (collision)
SndOtherError	Failed attempts to send
RcvNoError	Frames received free of error
RcvResError	Incorrectly received frames
RcvRejected	Rejected frames

Parameter (PNIO_CTRL_COMM_COUNTER_DATA)

Parameter	Description
Port[4]	Statistical data of ports 1/-4
NumberOfPorts	Number of Ethernet interfaces (can be 3 (industrial PC) or 4 (CP 1616 / CP 1604))

4.14.19 PNIO_DATA_TYPE (IO data type)

Description

This enumeration type specifies the types of data of a submodule. It is used in the PNIO_CTRL_DIAG_CONFIG_SUBMODULE structure.

Syntax

```
typedef enum
{
    PNIO_DATA_RT        = 0;
    PNIO_DATA_IRT       = 1
}PNIO_DATA_TYPE
```

Elements

Name	Description
PNIO_DATA_RT	RT data
PNIO_DATA_IRT	IRT data

4.14.20 PNIO_COM_TYPE (type of transfer)

Description

This enumeration type specifies the communication type of a submodule. It is used in the PNIO_CTRL_DIAG_CONFIG_SUBMODULE structure.

Syntax

```
typedef enum
{
    PNIO_COM_UNICAST              = 0x0,
    PNIO_COM_DIRECT_DATA_EX       = 0x1
}PNIO_COM_TYPE
```

Elements

Name	Description
PNIO_COM_UNICAST	Transfer between IO device and IO controller
PNIO_COM_DIRECT_DATA_EX	Direct data exchange transfer

4.14.21 PNIO_CP_CBE_TYPE (callback event type)

Description

The IO-Base user programming interface recognizes the following callback event types:

Callback event type	Signaled Information
PNIO_CP_CBE_STARTOP_IND	Start of isochronous real-time data processing
PNIO_CP_CBE_OPFAULT_IND	Violation of isochronous real-time mode
PNIO_CP_CBE_NEWCYCLE_IND	Start of a new bus cycle

These callback events can only be registered by the IO controller or the IO device.

If you want to implement an IO controller and IO device at the same time, you will need to address both devices in a common IO-Base user program.

4.14.22 PNIO_CP_CBE_PRM (callback event parameter)

Description

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

Syntax

```
typedef struct
{
    PNIO_CP_CBE_TYPE    CbeType;
    PNIO_UINT32         CpIndex;
    union
    {
        ...          /*Union over the various
        ...            callback event parameters
                       (details in relevant sections)*/
    }u;
} ATTR_PACKED PNIO_CP_CBE_PRM;
```

Parameter

Name	Description
CbeType	Callback event
CpIndex	CpIndex from PNIO_controller_open( ) or from PNIO_device_open( )

4.14.23 PNIO_CP_CBF (PNIO callback function)

Description

Using the callback event parameter PNIO_CP_CBE_PRM, a general PNIO callback function of the type PNIO_CP_CBF is declared.

Syntax

```
typedef void (* PNIO_CP_CBF)(
    PNIO_CP_CBE_PRM       *pCbfPrm
);
```

4.14.24 PNIO_CYCLE_INFO (information on current cycle)

Description

The structure contains information on the current cycle and allows the following to be identified:

- Highly accurate counter ("ClockCount”) with a resolution of 10 ns
- Loss of interrupts ("CycleCount" does not change by the same value in every cycle)
- "CountSinceCycleStart" returns the time between the start of the cycle and the call for the function that returns the PNIO_CYCLE_INFO structure.
- Time between the PNIO_CP_CBE_STARTOP_IND callback event and the PNIO_CP_set_opdone( ) function call (difference between the input value "CountSinceCycleStart” of the PNIO_CP_CBE_STARTOP_IND callback event and the return value of the PNIO_CP_set_opdone( ) function)

It is used as the input parameter for the callback events:

- PNIO_CP_CBE_STARTOP_IND
- PNIO_CP_CBE_OPFAULT_IND
- PNIO_CP_CBE_NEWCYCLE_IND

The structure serves as a return value for the function PNIO_CP_set_opdone( ).

Syntax

```
typedef struct
{
    PNIO_UINT32     CycleCount;
    PNIO_UINT32     ClockCount;
    PNIO_UINT32     CountSinceCycleStart;
} ATTR_PACKED PNIO_CYCLE_INFO;
```

Parameter

Name	Description
CycleCount	The "CycleCount" value is incremented in each cycle by the value corresponding to the cycle duration based on 31.25 µs. The range of values is 0 to 216-1; the data type UINT32 is intended for future expansions. **Example** With a cycle of 1 ms, the value is incremented each time by 32.
ClockCount	Free-running hardware counter on the CP with a resolution of 10 ns (32-bits wide).
CountSinceCycleStart	The "CountSinceCycleStart" value specifies the time since the start of the last cycle based on 10 ns. The value is 32 bits wide.

4.14.25 PNIO_CTRL_DIAG_DEVICE_STATE

Description

The PNIO_CTRL_DIAG_DEVICE_STATE structure provides information on the status of an IO device.

To start the request, the logical address of an IO device submodule must be transferred with the PNIO_ctrl_diag_req( ) function call in the "u.Addr" element of the PNIO_CTRL_DIAG structure to which "pDiagReq" points.

The "DiagDataBuffer" and "DiagDataBufferLen" parameters in the PNIO_CBE_CTRL_DIAG_CONF callback event (see section "Callback event PNIO_CBE_CTRL_DIAG_CONF (signal result of the diagnostics request) (Page 72)") return the result of the request - the "PNIO_CTRL_DIAG_DEVICE" structure.

Syntax

```
typedef struct{
 PNIO_DEV_ACT_TYPE Mode;
 PNIO_UINT32 DiagState;
 PNIO_UINT32 Reason;
 PNIO_UINT32 Reserved1[12];
} ATTR_PACKED PNIO_CTRL_DIAG_DEVICE;
```

Elements

Name	Description
Mode	Current status of the IO device "PNIO_DEV_ACT_TYPE"; see the section "PNIO_DEV_ACT_TYPE (type for activating all deactivating an IO device) (Page 101)".
DiagState	Current status of the connection establishment to this IO device - The possible values are as follows: • PNIO_CTRL_DEV_DIAG_OFFLINE • PNIO_CTRL_DEV_DIAG_ADDR_INFO_OK • PNIO_CTRL_DEV_DIAG_AR_CONNECTING • PNIO_CTRL_DEV_DIAG_AR_IN_DATA You will find the meaning of the values in the header file "pniousrx.h".
Reason	If there are errors in connection establishment, the cause of the error is entered here. You will find the possible values in the "PNIO_CTRL_REASON_..." constant in the "pniousrx.h" header file.

Quick start with IO device functionality 5

This chapter recommends a step-by-step procedure for creating a user program in the C/C++ programming language based on the IO-Base 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 writing your IO-Base device user program.

5.1 Procedure

Description

By following the steps below, you can create an IO-Base 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 IO-Base device user programming interface for the IO device. You can do this by reading the section "Overview of the IO-Base user programming interface for IO devices (Page 121)" 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 IO-Base device user programming interface: – "pniousrd.h" with the functions and data structures. – "pnioerrx.h" with the error and return codes – "pniobase.h" global definitions of the IO-Base • The "libpniousr" libraries for linking to your IO-Base device user program. • Sample programs and corresponding databases.
4	Work through the source text of the sample program "DevEasy" in the "../Examples/deveasy" subdirectory and check out the functions and data structures in the section "Overview of the IO-Base user programming interface for IO devices (Page 121)".
5	Based on your system configuration and your GSDML file, find out which data needs to be sent and received (IO data, data records, alarms) and which IO devices and IO controllers are involved.
6	Complete the STEP 7 or NCM configuration and download it to the devices involved.

Step	Description
7	Modify the supplied sample program "DevEasy" 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 include the modified sample program and test it on the system you have available.
8	Now create your IO-Base device user program covering the entire functionality.

5.2 Overview of the functionality of the products

Description

The table below provides you with an overview of the IO device functionality that can be used in the various SIMATIC NET products.

Function groups and names	SIMATIC NET products	
	SOFTNET PN IO (RT)	CP 1616 / CP 1604 (RT + IRT) with DK-16xx as of V2.0 to V2.6
Management functions		
PNIO_device_open()	–	x
PNIO_device_close()	–	x
PNIO_device_start()	–	x
PNIO_device_stop()	–	x
PNIO_CBF_DEVICE_STOPPED() callback function	–	x
Interface for IO device configuration		
PNIO_api_add()	–	x
PNIO_api_remove()	–	x
PNIO_CBF_PULL_PLUG_CONF() callback function	–	x
PNIO_mod_plug()	–	x
PNIO_mod_pull()	–	x
PNIO_sub_plug()	–	x
PNIO_sub_pull()	–	x
PNIO_sub_plug_ext()	–	x
PNIO_sub_plug_ext_IM()	–	x
Interface for write IO data on the IO device		
PNIO_initiate_data_write()	–	x
PNIO_CBF_DATA_WRITE() callback function	–	x
PNIO_initiate_data_write()	–	x
Interface for read IO data on the IO device		
PNIO_initiate_data_read()	–	x
PNIO_initiate_data_read_ext()	–	x

Function groups and names	SIMATIC NET products	
PNIO_CBF_DATA_READ() callback function	–	x
Interface for read/write data record on the IO device		
PNIO_CBF_REC_READ( ) callback function	–	x
PNIO_CBF_REC_WRITE() callback function	–	x
Interface for managing diagnostics data on the IO device		
PNIO_build_channel_properties()	–	x
PNIO_diag_channel_add()	–	x
PNIO_diag_channel_remove()	–	x
PNIO_diag_generic_add()	–	x
PNIO_diag_generic_remove()	–	x
Alarm interface for the IO device		
PNIO_process_alarm_send()	–	x
PNIO_diag_alarm_send()	–	x
PNIO_ret_of_sub_alarm_send()	–	x
PNIO_CBF_REQ_DONE() callback function	–	x
Interface for connection establishment and termination of the IO device		
PNIO_device_ar_abort()	–	x
PNIO_set_appl_state_ready()	–	x
PNIO_CBF_CHECK_IND() callback function	–	x
PNIO_CBF_AR_CHECK_IND() callback function	–	x
PNIO_CBF_AR_INFO_IND() callback function	–	x
PNIO_CBF_AR_INDATA_IND() callback function	–	x
PNIO_CBF_AR_ABORT_IND() callback function	–	x
PNIO_CBF_AR_OFFLINE_IND() callback function	–	x
PNIO_CBF_APDU_STATUS_IND() callback function	–	x
PNIO_CBF_CBF_PRM_END_IND() callback function	–	x
Station signaling with LEDs		
Callback function PNIO_CBF_START_LED_FLASH()	–	x
Callback function PNIO_CBF_STOP_LED_FLASH()	–	x
General		
Callback function PNIO_CBF_CP_STOP_REQ()	–	x
Interface for isochronous real-time mode (IRT)		
PNIO_CP_register_cbf()	–	x
PNIO_CP_CBE_STARTOP_IND() callback event	–	x
PNIO_CP_CBE_OPFAULT_IND() callback event	–	x
PNIO_CP_set_opdone()	–	x

Overview of the IO-Base user programming interface for IO devices

6

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

Function calls and data access are described in detail in the section "Description of the functions and data types for IO devices (Page 145)".

Note

The IO data directions are always described from the perspective of the IO controller. As a result, an IO device writes input data (data goes to the IO controller) and reads output data (data comes from the IO controller).

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

Note

The IO device functionality can be used only in conjunction with the CP 1616 / CP 1604 communications processors as of V2.0 to V2.6. As of V2.7, see document "I-device user programming interface" on the Support pages (https://support.industry.siemens.com/cs/ww/en/view/109737176).

With the "DK ERTEC 400/200" product, we have the same interface as for ERTEC 200 and 400.

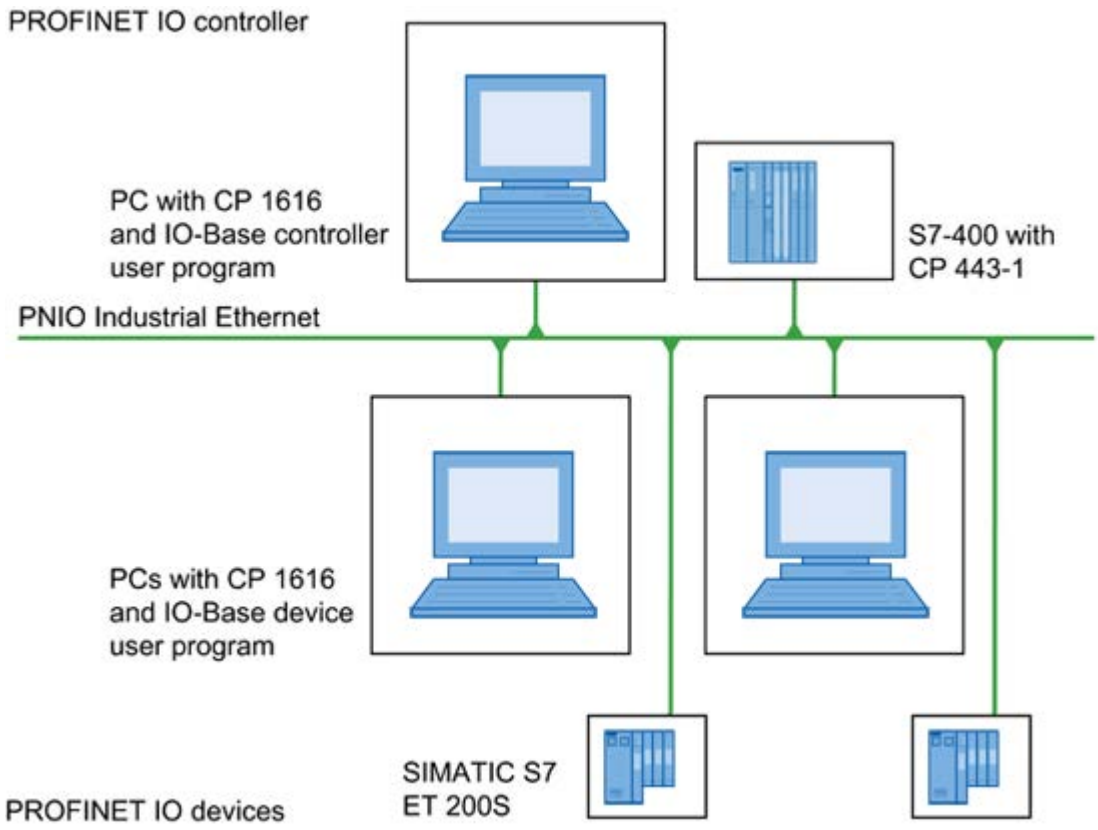
6.1 Typical use of the IO-Base device user programming interface

Description

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

The IO-Base device user program and the IO-Base device functions of the SIMATIC NET PC software product run on the PC. Data traffic is handled over a PROFINET IO communications processor with a SIMATIC S7 PROFINET CP (PLC) or alternatively with a PC with an IO-Base controller user program over Industrial Ethernet.

This configuration is referred to again in the supplied sample program.

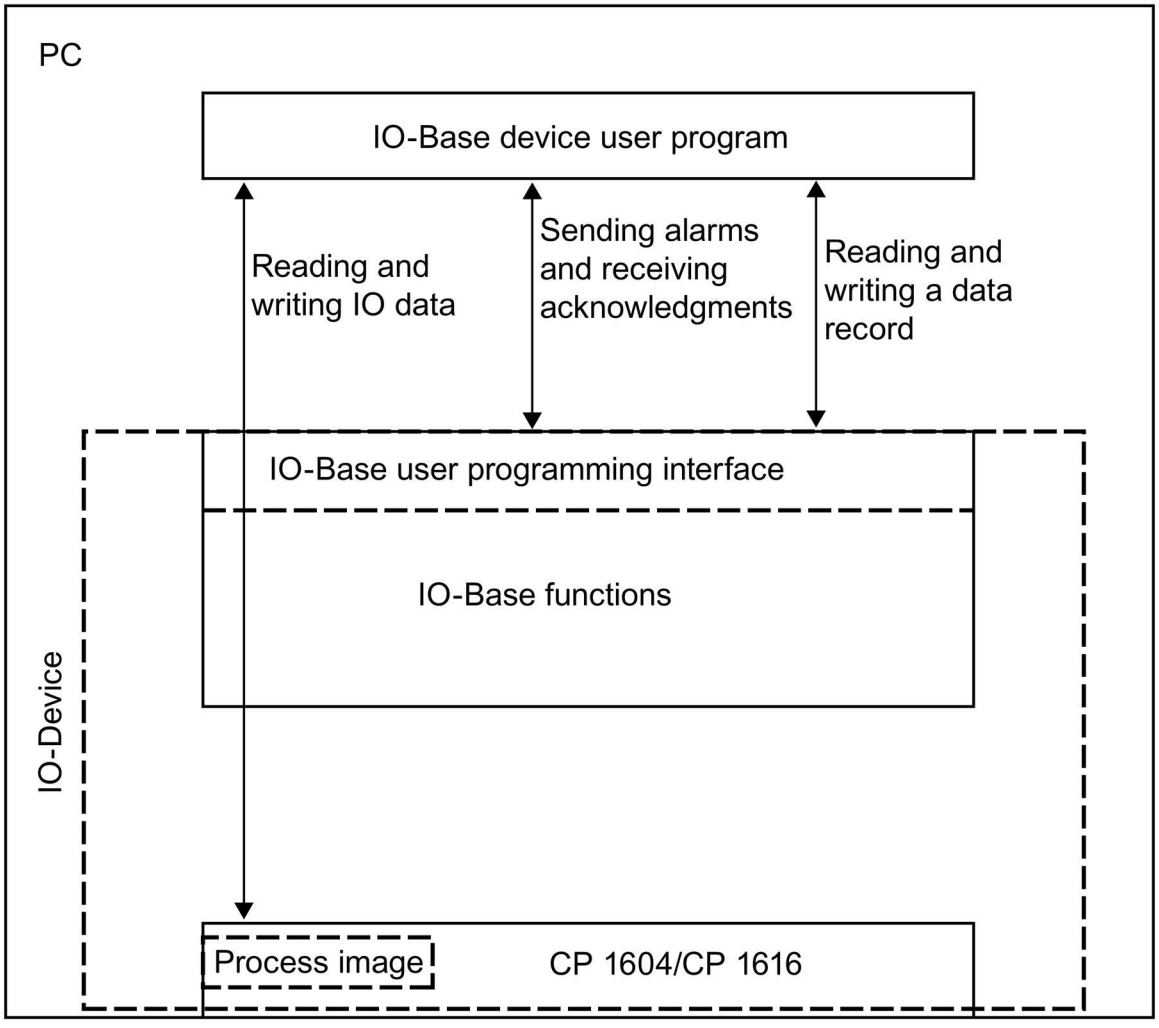


6.2 Software architecture on a PC with PROFINET IO

Description

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

These are the IO-Base functions read/write IO data, send alarms and receive alarm confirmations and read/write data records. The PROFINET IO software is connected to the process over a PROFINET IO communications processor. Communication with the IO controller is configured with STEP 7 or NCM PC.



Header files, libraries and sample programs

The following files are required for IO device support:

File type	File name	Purpose
Header file	pniobase.h	Contains global structures and definitions for IO controller and IO device.
Header file	pniousrd.h	Contains data types, constants, and function declarations.
Header file	pnioerrx.h	Contains the error codes.
Library	libpniousr	To link to your IO-Base device user program.

6.3 Typical sequence of an IO-Base device user program

Overview

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

- Initialization phase
- Productive operation
- Completion phase

These are explained in detail below.

6.3.1 Initialization phase

Description

The initialization phase is divided into several steps. A distinction must be made between function calls activated by the IO-Base device user program and callback calls activated by the IO-Base interface.

Note

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

Step	Action	Purpose
1	PNIO_device_open( )	• Register with CP. • Register callback functions for process data, data records and alarms.
2	PNIO_api_add( )	Register API (Application Process Identifier).
3	PNIO_mod_plug( )	• If FW >= V2.6 Insert head module (DAP) in slot 1 Insert corresponding submodule in slot 1, subslot 1 • If FW < V2.6 Insert head module (DAP) in slot 0 Insert corresponding submodule in slot 0, subslot 1 Wait for the PNIO_CBF_PULL_PLUG_CONF( ) call if this callback was registered.
4	PNIO_sub_plug_ext_IM( )	Insert associated submodule 0 (corresponds to head station). Wait for the PNIO_CBF_PULL_PLUG_CONF( ) call if this callback was registered.

Step	Action	Purpose
5	PNIO_mod_plug( )	Insert further modules. Wait for the PNIO_CBF_PULL_PLUG_CONF( ) call if this callback was registered.
6	PNIO_sub_plug_ext_IM( )	Insert associated submodules. Wait for the PNIO_CBF_PULL_PLUG_CONF( ) call if this callback was registered.
7	PNIO_CP_register_cbf( )	If you want support of the IRT configuration and therefore isochronous real-time mode, the callbacks PNIO_CP_CBE_OPFAULT_IND and PNIO_CP_CBE_STARTOP_IND must be registered. Otherwise, the connection establishment is finally agreed to with the PNIO_set_appl_state_ready( ) function (see Step 17). From this time onwards, these callbacks are signaled.
8	PNIO_device_start( )	Activates the IO device; from this point onwards, the IO device is visible for the IO controller.
9	Wait for call of the PNIO_CBF_AR_ CHECK_IND( ) callback function.	This callback is called by the IO-Base interface as soon as an IO controller has established a connection to the IO-Base device user program and has transferred its expected configuration for the IO-Base device user program. By calling this callback, application relation global parameters are transferred to the IO-Base device user program for checking. The IO-Base device user program can abort the establishment of the application relation if there are errors in the application relation global parameters; see also PNIO_device_ar_abort( ). The connection establishment is finally agreed to with the PNIO_set_appl_state_ready( ) function (see Step 17).
10	React to any call of the PNIO_CBF_CHECK_ IND( ) callback function.	This callback is called for every submodule of the IO device whose configuration does not match that of the IO controller. This gives the user program the opportunity to change the submodule configuration or to mark the submodule as compatible or bad. If the configurations match, this callback is not called.
11	Wait for call of the PNIO_CBF_AR_ INFO_IND( ) callback function.	This callback is called by the IO-Base interface as soon as the application relation to the IO controller is established. By calling this callback, the IO-Base device user program is informed of the modules and submodules operating in this application relation.
12	React to call of the PNIO_CBF_REC_ WRITE( ) callback function.	This callback is called by the IO-Base interface if an IO controller transfers a parameter assignment data record for a submodule. By calling this callback, any parameter assignment data is transferred per submodule to the IO-Base device user program.
13	Wait for call of the PNIO_CBF_PRM_ END_IND( ) callback function.	This callback is called by the IO-Base interface as soon as an IO controller signals the end of the parameter assignment phase.

Step	Action	Purpose
14	PNIO_initiate_data_ write( )	With this call, the IO-Base device user program initiates the call for the PNIO_CBF_DATA_WRITE( ) callback, so that the IO-Base device user program can initialize the input data of the functional submodules and set the local status to "GOOD". The local status must be set to "BAD" for all non-functional submodules.
		Notice The PROFINET IO standard requires that all output data of all functional submodules is set to valid values and the local status is set to GOOD prior to sending the application ready (Step 17).
15	PNIO_initiate_data_ read( )	With this call, the IO-Base device user program initiates the call of the PNIO_CBF_DATA_READ( ) callback so that it can set the local status to "GOOD" for all output data of the functional submodules. The local status must be set to "BAD" for all non-functional submodules.
		Notice The PROFINET IO standard requires that the local status of all functional submodules is set to GOOD before the application ready (Step 17) is sent.
16	Wait for the arrival of the PNIO_CP_CBE_ STARTOP_IND event	If you want support for IRT configuration, the program must wait for the PNIO_CP_CBE_STARTOP_IND callback event. This event informs the IO-Base device user program that the IRT data can now be processed (start of isochronous real-time data processing). **Isochronous real-time mode** By calling the functions PNIO_initiate_data_read_ext( ) and PNIO_initiate_data_write_ext( ), the IO-Base device user program can command the IO-Base programming interface to call the PNIO_CBF_DATA_READ( ) or PNIO_CBF_DATA_WRITE( ) callbacks so that the user program can initialize the IRT submodules with the "GOOD“ status. **Non-isochronous mode** The functions listed above under isochronous real-time mode can be executed outside this callback event. They must simply be executed before Step 18 that is described below. **Completion of Step 17** Before initiating the callback event, PNIO_CP_set_opdone( ) must be called.
17	PNIO_set_appl_ state_ready( )	With this, the IO-Base device user program reports a list of non-functional submodules to the IO controller and signals that it can begin data exchange.
18	Wait for call of the PNIO_CBF_AR_ INDATA_IND( ) callback function.	This callback is called by the IO-Base interface as soon as an IO controller has transferred the IO data the first time. Signaling of the start of cyclic data exchange.

Integration of an IO device in cyclic operation (minimum requirement)

The following steps at least must be taken to integrate an IO device in cyclic operation:

Step	Action	Purpose
1	PNIO_device_open( )	• Register with CP. • Register callback functions for process data, data records and alarms.
2	PNIO_api_add( )	Register API (Application Process Identifier) - PNIO_api_add( ) with the corresponding API must be executed successfully for every configured profile. Remember that PNIO_api_add( ) with API=0 is mandatory for the head station.

6.3.2 Productive operation

Overview

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

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

Data exchange 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	PNIO_initiate_data_read( ) or PNIO_initiate_data_read_ext( )	Read request signaled to the IO-Base interface. This causes the IO-Base interface to execute step 2. This call returns only after step 2 has been processed.
2	PNIO_CBF_DATA_READ( )	The IO-Base interface calls this callback for every submodule with output data and, among other things, transfers to it the pointer to a data buffer with the output data received from the IO controller.

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	PNIO_initiate_data_write( ) or PNIO_initiate_data_write_ext( )	Signal write request to the IO-Base interface. This causes the IO-Base interface to execute step 2. This call returns only after step 2 has been processed.
2	PNIO_CBF_DATA_WRITE( )	The IO-Base interface calls this callback for every submodule with input data and 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	Wait for the PNIO_CP_CBE_STARTOP_IND event.	With this event, the IO-Base interface signals the start of isochronous real-time data processing to the IO device user program.
2	PNIO_initiate_data_read_ ext(PNIO_ACCESS_IRT_ WITHOUT_LOCK)	Read request signaled to the IO-Base interface. This causes the IO-Base interface to execute step 3. This call returns only after step 3 has been processed.
3	PNIO_CBF_DATA_READ( )	The IO-Base interface calls this callback for every IRT submodule with output data. A pointer to a data buffer is transferred to the callback and contains the output data received from the IO controller.
4	PNIO_initiate_data_write_ ext(PNIO_ACCESS_IRT_ WITHOUT_LOCK)	Signal write request to the IO-Base interface. This causes the IO-Base interface to execute step 5. This call returns only after step 5 has been processed.
5	PNIO_CBF_DATA_WRITE( )	The IO-Base interface calls this callback for every IRT submodule with input data. A pointer to a data buffer is transferred to the callback. The input data to be sent to the IO controller must be copied to this buffer by the IO-Base user program.
6	PNIO_CP_set_opdone( )	The IO device user program uses this call to inform the IO-Base interface of the end of the isochronous real-time data processing.

In non-isochronous mode, there is no coupling with the bus cycle, in other words, Step 6 is executed immediately following Step 1; Steps 2 to 5 are asynchronous to the PNIO_CP_CBE_STARTOP_IND event, in other words, they can execute at any point in time.

Responding to a read data record job

Responding to a read data record job consists of one step:

Step	Action	Purpose
1	Wait until the PNIO_CBF_REC_READ callback is called.	As soon as the IO-Base interface receives a read data record job from the IO controller, it calls the PNIO_CBF_REC_READ callback registered by the IO-Base device user program.

Responding to a write data record job

Responding to a write data record job consists of one step:

Step	Action	Purpose
1	Wait until the PNIO_CBF_REC_WRITE callback is called.	As soon as the IO-Base interface receives a write data record job from the IO controller, it calls the PNIO_CBF_REC_WRITE callback registered by the IO-Base device user program.

Sending alarms and receiving alarm confirmations

Alarms are processed in two steps as described below:

Step	Action	Purpose
1	Call one of the following functions: PNIO_process_alarm_send( ) PNIO_diag_alarm_send( ) PNIO_ret_of_sub_alarm_send( )	Send alarm
2	Wait until the PNIO_CBF_REQ_DONE callback is called.	Receive confirmation

Callback events when establishing and terminating the connection on the IO device

When the connection is established, information is made available by the IO-Base interface using callbacks.

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

Callback name	Signaled Information
PNIO_CBF_AR_CHECK_IND( )	Application relation information
PNIO_CBF_CHECK_IND( )	Check indication - Actual module configuration does not match the configuration in the project engineering.
PNIO_CBF_AR_INFO_IND( )	Expected configuration of the IO device
PNIO_CBF_PRM_END_IND( )	End of parameter assignment by IO controller
PNIO_CP_CBE_STARTOP_ IND( )	Start of isochronous real-time data processing

Callback name	Signaled Information
PNIO_CP_ OPFAULT_IND( )	Violation of the isochronous real-time mode; PNIO_CP_set_opdone( ) was called too late by the user program.
PNIO_CBF_AR_INDATA_IND( )	Application relation InData - First data exchange with IO controller completed.
PNIO_CBF_AR_ABORT_IND( )	Abort event - Connection abort before data has been exchanged.
PNIO_CBF_AR_OFFLINE_IND( )	Offline event - Connection abort after data has been exchanged.
PNIO_CBF_APDU_STATUS_IND( )	Status of the IO controller

Note

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

6.3.3 Completion phase

Description

The completion phase of the IO device covers four steps:

Step	Action	Purpose
1	PNIO_device_stop( )	The IO device is deactivated and is therefore no longer available for the IO controller. The acknowledgment of PNIO_device_stop() comes with the PNIO_CBF_DEVICE_STOPPED( ) callback. If data is exchanged between IO controller and IO device (PNIO_CBF_AR_INDATA_IND( ) callback was called by the interface), the IO-Base interface calls the PNIO_CBF_AR_OFFLINE_IND( ) callback. If there has not yet been any data exchange between the IO controller and IO device (PNIO_CBF_AR_INDATA_IND( ) callback not yet called by the interface), the IO-Base interface calls the PNIO_CBF_AR_ABORT_IND( ) and the PNIO_CBF_DEVICE_STOPPED( ) callbacks.
2	PNIO_mod_pull( )	All modules including their submodules are removed from the IO-Base interface. Wait for the PNIO_CBF_PULL_PLUG_CONF( ) call if this callback was registered.
3	PNIO_api_remove( )	Removal of the application process identifier.
4	PNIO_device_close( )	Deregister with the communications processor.

6.4 Basic data exchange of the IO-Base device functions

Description

The IO-Base device functions have the following basic methods of exchanging data:

- Non-isochronous access to cyclic IO data (RT)
  - Write IO data
  - Read IO data

  The IO data exchange also includes status information.

  This special feature is described in the following section.
- Isochronous real-time and non-isochronous access to cyclic IO data (IRT)

  The access functions are the same as those described above. Access must only relate to IRT data.

  Access to IRT data between the PNIO_CP_CBE_STARTOP_IND event and the PNIO_CP_set_opdone( ) function call is known as isochronous real-time access.

  Access to IRT data outside the PNIO_CP_CBE_STARTOP_IND event and the PNIO_CP_set_opdone( ) function call is known as non-isochronous real-time access.

  For more information, refer to the section "Isochronous real-time and non-isochronous access to cyclic IO data (IRT) (Page 134)".
- Access to acyclic data
  - Writing and reading data records
  - Send alarms and receive their confirmations

  For more information, refer to the section "Callback mechanism (Page 142)".

6.5 Non-isochronous access to cyclic IO data (RT)

How it works in principle

Both when the IO-Base device user program writes and reads the IO data, the local process image on the CP is written or read. No data whatsoever is sent via the network.

The underlying IO-Base device functions or the hardware handle data exchange between the local process image and the IO controller separately and cyclically.

The details of this data exchange are specified in the configuration.

Note

It is not necessary for the IO-Base device user program to write or read IO data in every bus cycle.

Note

It is not necessary for the IO-Base device user program to access the process image more often than the configured cycle.

IO data and data status

The quality of the IO data is described by the data status that can have the value GOOD or BAD.

Both when writing and reading, two data statuses are exchanged:

- Local status (status of your IO-Base device user program)
- Remote status (status of the communications partner)

6.5.1 Cyclic reading with status

Sequence of the PNIO_CBF_DATA_READ( ) function

The IO-Base device user program initiates reading by calling the PNIO_initiate_data_read( ) or PNIO_initiate_data_read_ext( ) function. The IO-Base interface then calls the PNIO_CBF_DATA_READ( ) callback function for each submodule put into operation by the IO controller. By calling the callback function, the output data and the corresponding remote data status are read out from the local process image by the communications partner.

In addition to this, the local status of this output data is written to the local process image for the communications partner.

There are therefore two statuses involved in cyclic reading.

Communication direction	Values
From the IO controller	• Output data of the IO controller • Remote status
To the IO controller	Local status

Remote status of the communications partner

With the remote status, the communications partner reports the quality of the output data (GOOD or BAD).

If the communications partner reports BAD, the IO-Base controller user program can, for example, continue processing with substitute values.

Local status

Normally, the local status is set to GOOD by the IO-Base device user program.

If, however, the IO-Base device user program cannot process the supplied output data further, the local status should be set to BAD during the read job.

As soon as it receives this status, the communications partner can recognize whether the output data it has sent was processed correctly.

6.5.2 Cyclic writing with status

Sequence of writing to the process image

The IO-Base device user program initiates writing by calling the PNIO_initiate_data_write( ) or PNIO_initiate_data_write_ext( ) function. The IO-Base interface then calls the PNIO_CBF_DATA_WRITE( ) callback for each submodule put into operation by the IO controller. By calling the PNIO_CBF_DATA_WRITE( ) callback function, the input data and the corresponding local data status of this data is written to the local process image for the communications partner.

The communications partner also reads the remote status of this input data from the local process image.

There are therefore two statuses involved in cyclic writing.

Communication direction	Values
From the IO controller	• Input data of the IO controller • Local status
To the IO controller	Remote status

Local status

Normally, the local status is set to GOOD by the IO-Base device user program.

If the input data is bad or invalid, the IO-Base device user program should set the local status to BAD.

The communications partner could then, for example, output configured substitute values.

Remote status of the communications partner

With the remote status, the communications partner signals whether it was able to process the input data successfully "good" or whether there was a problem.

This status relates to input data transferred earlier from the same submodule and not to the write job that has just been started.

6.6 Isochronous real-time and non-isochronous access to cyclic IO data (IRT)

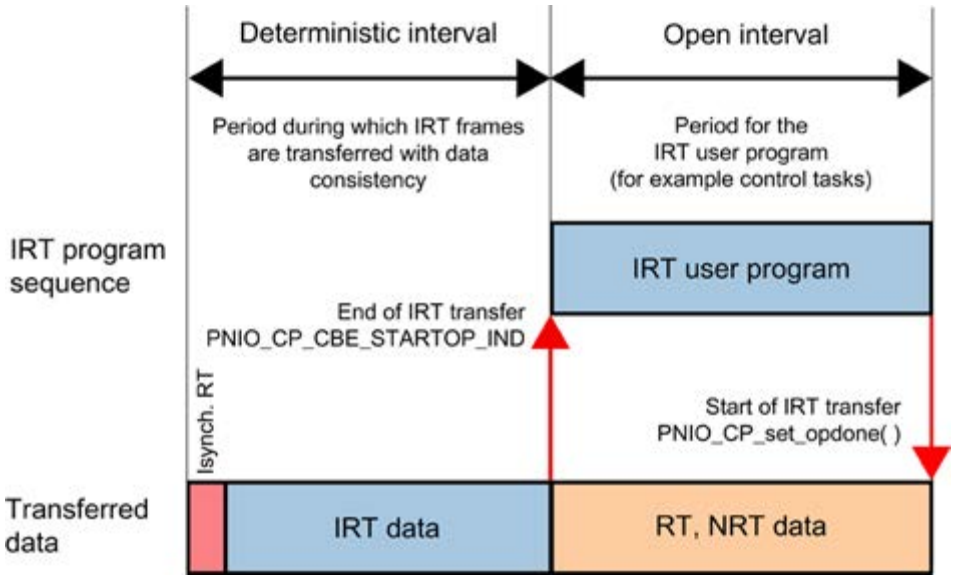
Description

In isochronous real-time mode, the user program processes and writes die IRT data between two cyclically repeated IRT times:

- IRT transfer end
- IRT transfer start

As soon as the IRT transfer end is reached, the IRT data is transferred from the process image memory to the host memory by DMA and the IRT user program is informed by the PNIO_CP_CBE_STARTOP_IND callback event. As of this point in time, all IRT data is in the host memory and processing of the data can begin. The first PNIO_CP_CBE_STARTOP_IND callback events are signaled after the PNIO_CP_register_cbf( ) function call.

Once the IRT user program has completed processing the data, it signals this by calling the PNIO_CP_set_opdone( ) function. This copies the IRT input data (from the perspective of the IO controller) to the process image memory using DMA and informs the communications processor of the validity of the data, so that the data can be transferred in the next bus cycle.



Violation of isochronous real-time mode

If completion of the processing of the IRT data is signaled too late, the DMA transfer cannot be completed in time and the IRT input data (from the perspective of the IO controller) cannot be marked as valid.

As a result, the communications processor transfers the IRT data as invalid in the next bus cycle. The user program is informed of this with the PNIO_CP_CBE_OPFAULT_IND callback event.

Consistency of the isochronous real-time IO data (IRT data)

To give the user program more time to process the data, IRT data is transferred between host memory and process image memory by DMA. As a result, IRT data is consistent only between the IRT transfer end and the IRT transfer start.

This means that your user program can only access consistent IRT data between the PNIO_CP_CBE_STARTOP_IND callback event and PNIO_CP_set_opdone( ).

When the PNIO_CP_CBE_OPFAULT_IND callback event arrives, the data is no longer consistent. This situation remains until the next PNIO_CP_CBE_STARTOP_IND callback event.

6.6.1 Access to IRT data

Isochronous real-time access to the IRT data

IRT data is accessed using the following functions and corresponding callback functions:

- Read
  - PNIO_initiate_data_read_ ext(PNIO_ACCESS_IRT_ WITHOUT_LOCK)
  - PNIO_CBF_DATA_READ( )
- Write
  - PNIO_initiate_data_write_ ext(PNIO_ACCESS_IRT_ WITHOUT_LOCK)
  - PNIO_CBF_DATA_WRITE( )

If IRT data is accessed between the PNIO_CP_CBE_STARTOP_IND callback event and PNIO_CP_set_opdone( ), this is known as isochronous real-time access to the IRT data.

Non-isochronous real-time access to the IRT data

If IRT data is accessed outside the PNIO_CP_CBE_STARTOP_IND callback event and PNIO_CP_set_opdone( ), this is known as asynchronous access to the IRT data.

These accesses are not permitted and no PNIO_WARN_IRT_INCONSISTENT error code is returned.

Identifying which submodules contain IRT data

Your IO-Base device user program uses the transfer parameter "pAR" of the PNIO_CBF_AR_INFO_IND( ) callback to identify the submodules that contain IRT data and that must therefore be processed in isochronous real-time mode.

As an alternative, your IO-Base device user program can call the PNIO_initiate_data_read_ext(PNIO_ACCESS_IRT_WITHOUT_LOCK) and PNIO_initiate_data_write_ext(PNIO_ACCESS_IRT_WITHOUT_LOCK) functions the first time the PNIO_CP_CBE_STARTOP_IND callback is called and set up a list of the IO devices in the PNIO_CBF_DATA_READ( ) and PNIO_CBF_DATA_WRITE( ) callbacks.

6.7 Access to acyclic data

How it works in principle

Acyclic data exchange includes the response to read data record requests and write data record requests as well as the sending of alarms and receipt of their acknowledgments.

6.7.1 Processing data record jobs

Description

The IO device user program must register the two following callback functions with PNIO_device_open( ):

- PNIO_CBF_REC_READ( )
- PNIO_CBF_REC_WRITE( )

These callbacks are then always called by the IO-Base interface as soon as a read data record or a write data record request is received from the IO controller. To allow your IO device user program to process the read data record or write data record job, these callbacks contain the data record number and the submodule number as parameters. Refer to IEC 61158 to see which records are standardized and which you can use freely for your purposes.

6.7.2 Send alarms and receive their confirmations

Description

The IO-Base interface provides your IO device user program with functions for sending alarms to the IO controller or informing it of unexpected events.

For each alarm sent by the IO device, the IO-Base device user program receives an alarm confirmation by the IO-Base interface calling the PNIO_CBF_REQ_DONE( ) callback.

The assignment of confirmation to alarm is made with the user handle. This user handle is assigned by the IO-Base device user program when the alarm is sent.

The PNIO_CBF_REQ_DONE( ) callback includes the user handle as a transfer parameter.

Note

Alarms can only be sent by the IO-Base device user program after calling PNIO_set_appl_state_ready( ).

Session key for PROFINET IO

The session key is a value that is incremented each time the application relation is established. This must be stored separately for every connection ("ArNumber") and is used along with the "ArNumber" parameter in all subsequent function calls.

The session key is transferred to the application during connection establishment in the `pAr -> SessionKey` parameter by calling the PNIO_CBF_AR_CHECK_IND( ) callback function.

It must be included with every alarm. The IO controller is then able to discard out-of-date alarms and confirmations of read data record and write date record jobs.

Example

In PROFINET IO, there is practically a race between alarms and the reestablishment of the application relation between the IO controller and IO device. This is best illustrated by the following example:

The following is assumed:

- There is an application relation between IO controller and IO device.
- The IO device wants to send a diagnostics alarm to the IO controller because a submodule is faulty for a short time due to a bad contact.
- Before the IO device can send this alarm over the network, however, the IO controller terminates the existing application relation, for example as a result of a restart, and then re-establishes the application relation.
- After reestablishing the application relation of the IO controller and the IO device, however, the bad contact on the submodule has already disappeared.

If the alarm for the previously existing application relation now reaches the IO controller, the IO controller would incorrectly assume that the bad contact on the submodule still existed.

PROFINET IO solves such problems by using the session key as described above.

6.8 Managing diagnostics data

Description

In PROFINET IO, there are two different diagnostics procedures.

- Channel diagnostics data
- Manufacturer-specific diagnostics data

Note

In IEC 61158-6 PROFINET IO "Application Layer Protocol Specification" Version 1.0, there is a strong recommendation to use the "channel diagnostics data" as the diagnostics procedure.

6.8.1 Channel diagnostics data

In PROFINET IO, diagnostics data is coded as "channel diagnostics data". The "channel properties" are part of the "channel diagnostics data". These can be generated by the IO-Base device user program using the PNIO_build_channel_properties( ) function. For a precise definition of the "channel diagnostics data", refer to the PROFINET IO standard.

In PROFINET IO, a submodule can consist of several channels. There may be a variety of channel diagnostics data per channel. This can be stored on the submodule by the IO-Base device user program using the PNIO_diag_channel_add( ) function.

If channel diagnostics data is no longer valid, the IO-Base device user program must remove it from the submodule with the PNIO_diag_channel_remove( ) function.

The following functions are available to manage channel diagnostics data:

Function	Purpose
PNIO_build_channel_properties( )	Generate channel properties
PNIO_diag_channel_add( )	Store channel diagnostics data in the subslot
PNIO_diag_channel_remove( )	Remove channel diagnostics data from the subslot

Setting channel diagnostics data

Channel diagnostics data is set in four steps:

Step	Action	Purpose
1	PNIO_build_channel_properties( )	Generate the channel properties with the "entering state" parameter
2	PNIO_diag_channel_add( )	Store channel diagnostics data in the subslot.
3	PNIO_diag_alarm_send( )	Signal the channel error entering state as a diagnostics alarm to the IO controller.
4	PNIO_CBF_REQ_DONE( )	Evaluate confirmation.

Removing channel diagnostics data

Channel diagnostics data is removed in four steps:

Step	Action	Purpose
1	PNIO_build_channel_properties( )	Generate the channel properties with the "exiting state" parameter.
2	PNIO_diag_channel_add( )	Remove channel diagnostics data from the sub-module.
3	PNIO_diag_alarm_send( )	Send channel error exiting state as diagnostics alarm to the IO controller.
4	PNIO_CBF_REQ_DONE( )	Evaluate confirmation.

6.8.2 Manufacturer-specific diagnostics data

Manufacturer-specific diagnostics data allows the IO-Base device user program to store manufacturer-specific diagnostics data for a submodule. No fixed structure is specified for manufacturer-specific diagnostics data.

The channel properties must also be specified for manufacturer-specific diagnostics data. For precise details, refer to the PROFINET IO standard.

The following functions are available to manage manufacturer-specific diagnostics data:

Function	Purpose
PNIO_build_channel_properties( )	Generate channel properties
PNIO_diag_generic_add( )	Store manufacturer-specific diagnostics data in the subslot
PNIO_diag_generic_remove( )	Remove manufacturer-specific diagnostics data from the subslot

Setting manufacturer-specific diagnostics data

Manufacturer-specific diagnostics data is set in four steps:

Step	Action	Purpose
1	PNIO_build_channel_properties( )	Generate the channel properties with the "entering state" parameter
2	PNIO_diag_generic_add( )	Store vendor-specific diagnostics data in the subslot.
3	PNIO_diag_alarm_send( )	Signal the channel error entering state as a diagnostics alarm to the IO controller.
4	PNIO_CBF_REQ_DONE( )	Evaluate confirmation.

Removing manufacturer-specific diagnostics data

Manufacturer-specific diagnostics data is removed in four steps:

Step	Action	Purpose
1	PNIO_build_channel_properties( )	Generate the channel properties with the "exiting state" parameter
2	PNIO_diag_generic_remove( )	Remove channel diagnostics data from the submodule.
3	PNIO_diag_alarm_send( )	Signal the channel error exiting state as a diagnostics alarm to the IO controller
4	PNIO_CBF_REQ_DONE( )	Evaluate confirmation.

6.9 Points to note when pulling and plugging modules in productive operation

Pull alarm when removing a module/submodule

The IO-Base interface generates a PROFINET IO pull alarm as soon as the IO-Base device user program pulls a module or submodule by calling the following functions:

- PNIO_mod_pull( )
- PNIO_sub_plug_ext_IM

Note

If you pull a module, all the submodules it contains are automatically removed.

Note

The PROFINET standard specifies that IO data from submodules that are to be pulled must first be written with the BAD status. This is only possible if the submodules were registered during connection establishment.

Note

No modules must be pulled or plugged between the PNIO_CBF_AR_CHECK_IND( ) callback and PNIO_CBF_PRM_END_IND( ).

Note

Pulling and plugging during productive operation is supported only if the PNIO_CBF_PULL_PLUG_CONF( ) callback was registered with PNIO_device_open( ).

Note

If you did not register the PNIO_CBF_PULL_PLUG_CONF( ) callback and want to change the module configuration, you will have to stop the IO device with the PNIO_device_stop( ) call to terminate the application relation before pulling or plugging modules or submodules.

After plugging modules and submodules, start the IO device again with PNIO_device_start( ). Then repeat the steps starting at step 8 in the section "Initialization phase (Page 124)".

Plug alarm when inserting a module/submodule

The IO-Base interface generates a PROFINET IO plug alarm as soon as the IO-Base device user program inserts a module or submodule by calling the following functions.

- PNIO_mod_plug( )
- PNIO_sub_plug_ext_IM

Reassignment of parameters after plugging

Following each PNIO_sub_plug_ext_IM, the corresponding submodule is assigned new parameters by the IO controller. This means that the IO-Base interface calls the PNIO_CBF_REC_WRITE( ) callback for every parameter assignment data record transferred by the IO controller.

Completion of the parameter assignment is signaled by the IO-Base interface by calling the PNIO_CBF_PRM_END_IND( ) callback.

Following parameter assignment, the IO-Base device user program checks whether the inserted submodule can operate with the transferred parameter settings.

- If the answer is "YES", the IO-Base device user program sets the input data to be sent and the local status for the inputs and outputs of this submodule to GOOD. The IO-Base device user program then calls the PNIO_set_appl_state_ready( ) function with an empty submodule list.

  This completes the plugging procedure.

- If the answer is "NO", the IO-Base device user program must leave the local status for the inputs and outputs of this submodule set to BAD (BAD was set prior to pulling; see note above). In this case, the IO-Base device user program must call the PNIO_set_appl_state_ready( ) function specifying the submodule in the submodule list.

  Plugging is then ended with an error.

6.9.1 Points to note with "return of submodule"

Description

If there is a functional problem on an inserted module, the IO-Base device user program may set the local status of the input and output data to BAD. On the assigned IO controller, this means that the input and output data is no longer valid for the user program on the IO controller.

When the submodule is functional again, the IO-Base device user program must set the local status of the input and output data to GOOD. The IO-Base device user program must then signal the change from BAD to GOOD to the IO controller by calling the PNIO_ret_of_sub_alarm_send( ) function. Due to the return of submodule alarm, the IO controller does not send new parameter settings to the submodule.

The submodule is then functional again.

6.10 Callback mechanism

How it works

Callback functions are programmed in the IO-Base device user program. A callback function can be given any name.

A callback event is an asynchronous event that is started by the IO-Base interface. It interrupts the execution of the IO-Base controller user program and starts the callback function in a separate thread. This means that synchronization techniques are necessary.

Callback functions on the IO device

The callback events and event types on the IO device are listed in the following table. The table also shows what to use to register a callback function and what triggers a callback event:

Callback event (asynchronous)	Callback event type	Registered by ...	Triggered by ...
Read data	PNIO_CBF_ DATA_READ	PNIO_device_open( )	... User program with call for: • PNIO_initiate_data_ read( ) • PNIO_initiate_data_read_ext( )
Write data	PNIO_CBF_ DATA_WRITE	PNIO_device_open( )	... User program with call for: • PNIO_initiate_data_ write( ) • PNIO_initiate_data_ write_ext( )
Read data record	PNIO_CBF_REC_READ	PNIO_device_open( )	... IO controller
Write data record	PNIO_CBF_REC_WRITE	PNIO_device_open( )	... IO controller
Confirmation for a send alarm job	PNIO_CBF_REQ_DONE	PNIO_device_open( )	... User program with call for: • PNIO_process_alarm_send( ) • PNIO_diag_alarm_send( ) • PNIO_ret_of_sub_alarm_send( )
Comparison of expected configuration	PNIO_CBF_CHECK_IND	PNIO_device_open( )	IO-Base interface
Connection establishment	PNIO_CBF_AR_INFO_IND	PNIO_device_open( )	IO controller
Start of data exchange	PNIO_CBF_AR_INDATA_IND	PNIO_device_open( )	IO controller
Connection termination without previous data exchange	PNIO_CBF_AR_ABORT_IND	PNIO_device_open( )	IO controller, user program
Connection termination after previous data exchange	PNIO_CBF_AR_OFFLINE_IND	PNIO_device_open( )	IO controller, user program
Status change of the IO controller	PNIO_CBF_APDU_STATUS_IND	PNIO_device_open( )	IO controller
Firmware download, update or configuration download	PNIO_CBF_CP_STOP_REQ	PNIO_device_open( )	Configuration station
Activate flashing mode for LED	PNIO_CBF_START_LED_FLASH	PNIO_device_open( )	IO controller
Deactivate flashing mode for LED	PNIO_CBF_STOP_LED_FLASH	PNIO_device_open( )	IO controller

Callback event (asynchronous)	Callback event type	Registered by ...	Triggered by ...
Confirm pull and plug	PNIO_CBF_PULL_PLUG_CONF	PNIO_device_open( )	User program by calling: • PNIO_mod_plug( ) • PNIO_sub_plug_ext_IM • PNIO_mod_pull( ) • PNIO_sub_pull( )
Start of isochronous real-time data processing	PNIO_CP_CBE_STARTOP_IND	PNIO_CP_register_cbf( )	IRT transfer phase is completed and data is in host memory
IRT cycle violation	PNIO_CP_CBE_OPFAULT_IND	PNIO_CP_register_cbf( )	PNIO_CP_set_opdone( ) was called too late by the application.

Note

Include multithreading standard libraries when you compile your user program.

Note

Within a callback function, the only functions that can be called are those that access IO data. These are:

- PNIO_initiate_data_read( )
- PNIO_initiate_data_read_ext( )
- PNIO_initiate_data_write( )
- PNIO_initiate_data_write_ext( )
- PNIO_CP_set_opdone( )

Coordinating the sequence of callbacks

A callback function can interrupt the IO-Base device user program at any time. Callback functions for different events can also interrupt each other. A callback function must therefore be designed for simultaneous, multiple execution (reentrant) since it can be called from different threads.

In practical terms, this means that writing and reading of shared variables must be protected by synchronization mechanisms.

Avoid wait times in callback functions, particularly when entering critical sections. A new call for this and other callback functions may otherwise be blocked. Instead, you should, where possible, use a separate database.

Description of the functions and data types for IO devices

7

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

The chapter is primarily intended as a source of reference when you are writing your IO-Base device user programs.

7.1 Management functions for an IO device

Overview

The IO device recognizes the following management functions:

- PNIO_device_open( )
- PNIO_device_close( )
- PNIO_device_start( )
- PNIO_device_stop( )

These are described in detail in the following sections.

Other management functions are available for IRT mode as described in the section "Interface for isochronous real-time mode (IRT) (Page 199)".

7.1.1 PNIO_device_open( ) (register device as IO device)

Description

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

The callback functions used are registered using a function table.

Note

When the return value "PNIO_ERR_CONFIG_IN_UPDATE" is signaled, this means that the IO device has not yet been assigned an IP address by the PNIO controller. In this case, the application needs to call the "PNIO_device_open" until this return value is no longer signaled.

Syntax

```
PNIO_UINT32  PNIO_device_open(
    PNIO_UINT32          CpIndex,          //in
    PNIO_UINT32          ExtPar,           //in
    PNIO_UINT16          VendorId,         //in
    PNIO_UINT16          DeviceId,         //in
    PNIO_UINT16          InstanceId,       //in
    PNIO_UINT32          MaxAr,            //in
    PNIO_ANNOTATION     *pDevAnnotation,   //in
    PNIO_CFB_FUNCTIONS  *pCbf,             //in
    PNIO_UINT32         *pDevHndl          //out
```

Parameter

Name	Description
CpIndex	Module index - Used for unique identification of the communications module; must be 1!
ExtPar	PNIO_CEP_MODE_CTRL must always be transferred!
VendorId	Vendor ID for the IO device - assigned by the PROFINET user organization. This corresponds to the "VendorID" from the GSDML file.
DeviceId	IO device ID - This must be unique within the PROFINET IO products of a manufacturer. This corresponds to the "DeviceID" from the GSDML file.
InstanceId	The "Instance ID" is a reserved identifier for the IO device with the value = 1.
MaxAR	Depends on the required mode, • If only real-time mode is to be supported, "MaxAR" must always be 1. • If both real-time and isochronous real time modes need to be supported, "MaxAR" must be 2. • If real-time and isochronous real-time mode as well as supervisor connections need to be supported, "MaxAR" must be 3.
pDevAnnotation	Pointer to structure for version identification of the module; see the section "PNIO_ANNOTATION (version ID structure) (Page 209)".
pCbf	Pointer to the function table of the callback functions - If 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.
pDevHandle	Pointer to handle assigned to the IO device. This must be included in all further function calls.

Additional information on the "ExtPar" parameter

Name of the flag of the "ExtPar" parameter with the "PNIO_device_open( )" management function	Description
PNIO_CEP_SET_MRP_ROLE_OFF	With the "PNIO_CEP_SET_MRP_ROLE_OFF" flag, the MRP role is set to the value "MRP role OFF". This means that the MRP role can be turned off via the IO Base interface. This flag can be linked with other flags.
PNIO_CEP_PE_MODE_ENABLE	If this flag is set, it is possible to shut down and start up the host PC using jobs from the PNIO controller to save energy.

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_CONFIG_IN_UPDATE
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_OS_RES
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_PRM_DEV_ANNOTATION
- PNIO_ERR_PRM_EXT_PAR
- PNIO_ERR_PRM_HND
- PNIO_ERR_PRM_MAX_AR_VALUE
- PNIO_ERR_PRM_PCBF
- PNIO_ERR_SEQUENCE

7.1.2 PNIO_device_close( ) (deregister device as IO device)

Description

With this function, the IO-Base device user program deregisters an IO device from the IO-Base interface that was previously registered with "PNIO_device_open".

Note

The PNIO_device_close( ) function may only be called when all jobs have been acknowledged by the IO device with asynchronous callbacks.

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

Note

The PNIO_device_close( ) function may only be called after receiving the successful confirmation of PNIO_device_stop( ).

Note

All previously notified callback functions are deregistered. On completion of the PNIO_device_close( ) function, no further callback functions will be called for the deregistered IO device.

Note

After successfully deregistering, the handle is no longer valid and can no longer be used by the IO-Base device user program.

Syntax

```
PNIO_UINT32  PNIO_device_close(
    PNIO_UINT32        DevHndl             //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )

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

7.1.3 PNIO_device_start( ) (start IO device)

Description

An IO controller can only address the IO device after this function has been called.

Syntax

```
PNIO_UINT32  PNIO_device_start(
    PNIO_UINT32          DevHndl             //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )

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

7.1.4 PNIO_device_stop( ) (stop IO device)

Description

After calling this function, the IO device can no longer be addressed over the network. The IO device is deactivated and is therefore no longer available for the IO controller.

If data is exchanged between IO controller and IO device (PNIO_CBF_AR_INDATA_IND( ) callback was called by the interface), the IO-Base interface calls the PNIO_CBF_AR_OFFLINE_IND( ) callback.

If no data has been exchanged between IO controller and IO device (PNIO_CBF_AR_INDATA_IND( ) callback has not yet been called by the interface), the IO-Base interface calls the PNIO_CBF_AR_ABORT_IND( ) callback.

The acknowledgment of PNIO_device_stop() is made with the PNIO_CBF_DEVICE_STOPPED( ) callback.

Syntax

```
PNIO_UINT32  PNIO_device_stop(
    PNIO_UINT32          DevHndl             //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )

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

7.1.5 Callback function PNIO_CBF_DEVICE_STOPPED( ) (signal device stop)

Description

The PNIO_CBF_DEVICE_STOPPED( ) callback function is called by the IO-Base interface as a result of the PNIO_device_stop( ) function being called. It informs the IO-Base device user program that the IO device can no longer be addressed over the network.

Syntax

```
typedef void (* PNIO_CBF_DEVICE_STOPPED)(
    PNIO_UINT32     DevHndl,                //in
    PNIO_UINT32     ErrorCode               //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
ErrorCode	Error messages

Return values

No return values.

7.2 Interface for IO device configuration

Overview

With the function PNIO_CBF_PULL_PLUG_CONF( ), an IO device signals the application process identifier and status/status change.

7.2.1 Callback function PNIO_CBF_PULL_PLUG_CONF( ) (confirm pull/plug)

Description

The PNIO_CBF_PULL_PLUG_CONF( ) callback function is called by the IO-Base interface to confirm the pulling or plugging of a module or a submodule.

Syntax

```
typedef PNIO_IOXS (*PNIO_CBF_PULL_PLUG_CONF)(
    PNIO_UINT32        DevHndl,               //in
    PNIO_UINT32        API,                   //in
    PNIO_UINT32        Action,                //in
    PNIO_DEV_ADDR     *pAddr,                 //in
    PNIO_UINT32        ErrorCode              //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application Process Identifier
Action	Specifies whether or not there is a confirmation for the plugging or pulling of a module/submodule. Possible values are as follows: • PNIO_MOD_PULL • PNIO_SUB_PULL • PNIO_MOD_PLUG • PNIO_SUB_PLUG
pAddr	Pointer to the address of the module/submodule
ErrorCode	Error message about pulling or plugging

Return values

No return values.

7.3 Pull and plug functions

Overview

The pull and plug functions are called by the IO-Base device user program to inform the IO-Base interface of the current configuration. The following functions are available:

Function	Description
PNIO_sub_plug_ext_IM( )	Plug in submodule; even submodules that are not currently configured.
PNIO_sub_pull	Pull submodule.

Exists for reasons of compatibility

The IO-Base user programming interface was expanded by the PNIO_sub_plug_ext_IM( ) function. It replaces the following two previous functions that still exist for reasons of compatibility.

Previous functions	Current functions
PNIO_sub_plug( )	PNIO_sub_plug_ext_IM( )
PNIO_sub_plug_ext( )	PNIO_sub_plug_ext_IM( )

Note

Do not use the previous functions any longer.

7.3.1 PNIO_sub_plug_ext_IM( ) (extended plugging of a submodule)

Description

The function must be called by the IO-Base device user program in the initialization phase, see the section "Initialization phase (Page 124)". It reports the current configuration to the IO-Base interface.

It can also be called during operation if the current configuration changes, for example if a failed or removed submodule becomes functional again. In this case, a plug alarm is sent to the IO controller automatically by the IO-Base interface.

This function can inform the IO controller that a new submodule has been plugged in that does not correspond to the configuration. This case, the "Plug Wrong Submodule Alarm" is sent to the IO controller.

PNIO_sub_plug_ext_IM( ) is an asynchronous function. The return value provides information on whether or not the job was accepted by the interface. Confirmation of plugging is provided by the PNIO_CBF_PULL_PLUG_CONF( ) callback function. In contrast to the previous functions PNIO_sub_plug() and PNIO_sub_plug_ext(), the submodule is configured for reading and writing I&M data records.

Note

If the IO controller is informed of the change to the IO device with "Plug Alarm" or "Plug Wrong Submodule Alarm", no callback function is triggered with an IO controller.

Note

No "plug alarms" are sent to the IO controller before the start of the cyclic data exchange that is signaled to the IO-Base device user program with the PNIO_CBF_AR_INDATA_IND( ) callback function.

Note

Whenever a submodule is inserted in a module, a "plug alarm" is sent to the IO controller.

If a module is inserted, no alarm is triggered.

Note

Write/read queries of an engineering station for I&M data records are always reported to the IO-Base device user program regardless of which function (PNIO_sub_plug_ext_IM( ), PNIO_sub_plug( ) or PNIO_sub_plug_ext( )) was used to insert a submodule.

Syntax

```
PNIO_UINT32  PNIO_CODE_ATTR(
    PNIO_UINT32             DevHndl,              //in
    PNIO_UINT32             API,                  //in
    PNIO_DEV_ADDR          *pAddr,                //in
    PNIO_UINT32             SubIdent              //in
    PNIO_UINT32             AlarmType             //in
    PNIO_PLUG_IM0_BITS      IM0_bits              //in
);
```

Parameter

Name	Description	
DevHndl	IO device handle from PNIO_device_open( )	
API	Application process identifier for the module - This must correspond to one of the API parameters transferred with the PNIO_api_add( ) function call.	
pAddr	Pointer to the address of the subslot in which the submodule is inserted.	
	Note Only the value 1 to "MaxSubslots" may be specified in the IO device address for the subslot number. The value for "MaxSubslots" is specified with the PNIO_api_add( ) call.	
SubIdent	Submodule ID from the relevant GSDML file.	
AlarmType	The "AlarmType" parameter decides whether a "Plug Alarm" or a "Plug Wrong Submodule Alarm" is sent to the IO controller after inserting a new submodule. If the new submodule has the wrong submodule ID and is also incompatible with the submodule in the project engineering, then according to the standard, the "Plug Wrong Submodule Alarm" must be sent.	
	If you want to send a ... ,	**set the "AlarmType" parameter to ...**
	Plug alarm	PNIO_ALARM_TYPE_PLUG
	Plug wrong submodule alarm	PNIO_ALARM_TYPE_PLUG_WRONG
IM0_bits	Submodules with the parameter "IM0_bits" set are listed in the "I&M0FilterData" data record. The transferred values are described below in the paragraph "IM0_bits".	

IM0_bits

The transferred values of the "IM0_bits" parameter have the following meaning:

Name	Description
PNIO_PLUG_IM0_BITS_NOTHING	The submodule does not contain any I&M data.
PNIO_PLUG_IM0_BITS_ SUBMODULE_REP_MOD_DEV	The submodule contains the I&M data of the substitute module and the IO device. Select this value when only the head module, in other words, the submodule in slot 0 and subslot 1, contains I&M data.

Name	Description
PNIO_PLUG_IM0_BITS_ SUBMODULE_REP_MODULE	The submodule contains the I&M data of the substitute module. Select this value when only one submodule per slot is plugged in that contains I&M data. Select this value for the first submodule when several submodules with I&M data are plugged into a slot.
PNIO_PLUG_IM0_BITS_SUBMODULE	The submodule contains I&M data. Select this value for the second and all other submodules when several submodules with I&M data are inserted in a slot.

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_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.3.2 PNIO_sub_pull( ) (pull a submodule)

Description

The function is called by the IO-Base device user program when an inserted submodule fails or is removed. In this case, a pull alarm is sent to the IO controller automatically by the IO-Base interface.

PNIO_sub_pull( ) is an asynchronous function. The return value provides information on whether or not the job was accepted by the interface. Confirmation of the removal is provided by the PNIO_CBF_PULL_PLUG_CONF( ) callback function.

Note

If the IO controller is informed of the change to the IO device with a "Plug Alarm" or "Plug Wrong Submodule Alarm", no callback function is triggered with an IO controller with a CP 16xx.

Note

No "plug alarms" are sent to the IO controller before the start of cyclic data exchange that is signaled to the IO-Base device user program with the PNIO_CBF_AR_INDATA_IND( ) callback function.

Note

Whenever a submodule is removed, a "pull alarm" is generated.

If a module is removed, that does not have a submodule, no alarm is generated.

Syntax

```
PNIO_UINT32 PNIO_sub_pull(
    PNIO_UINT32          DevHndl,           //in
    PNIO_UINT32          API,               //in
    PNIO_DEV_ADDR       *pAddr              //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application process identifier for the module - This must correspond to one of the API parameters transferred with the PNIO_api_add( ) function call.
pAddr	Pointer to the address of the subslot from which the submodule was removed.

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"):

- PNIOERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.3.3 PNIO_sub_plug( ) (plug a submodule, not for new development)

Description

Note

This function is obsolete. You should no longer use it.

Instead, work with the function PNIO_sub_plug_ext_IM( ).

The function must be called by the IO-Base device user program in the initialization phase. It reports the current configuration to the IO-Base interface.

It can also be called during operation if the current configuration changes, for example if a failed or removed submodule becomes functional again. In this case, a plug alarm is sent to the IO controller automatically by the IO-Base interface.

PNIO_sub_plug( ) is an asynchronous function. The return value provides information on whether or not the job was accepted by the interface. Confirmation of plugging is provided by the PNIO_CBF_PULL_PLUG_CONF( ) callback function.

Note

If the IO controller is informed of the change to the IO device with "Plug Alarm" or "Plug Wrong Submodule Alarm", no callback function is triggered with an IO controller.

Note

No "plug alarms" are sent to the IO controller before the start of the cyclic data exchange that is signaled to the IO-Base device user program with the PNIO_CBF_AR_INDATA_IND( ) callback function.

Note

Whenever a submodule is plugged into a module, a "plug alarm" is sent to the IO controller.

If a module is inserted, no alarm is triggered.

Syntax

```
PNIO_UINT32 PNIO_sub_plug(
    PNIO_UINT32            DevHndl,          //in
    PNIO_UINT32            API,              //in
    PNIO_DEV_ADDR         *pAddr,            //in
    PNIO_UINT32            SubIdent          //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application process identifier for the submodule - This must correspond to one of the API parameters transferred with the PNIO_api_add( ) function call.
pAddr	Pointer to the address of the subslot in which the submodule is inserted.
	Note Only the value 1 to "MaxSubslots" may be specified in the IO device address for the subslot number. The value for "MaxSubslots" is specified with the PNIO_api_add( ) call.

Name	Description
SubIdent	Submodule ID - corresponds to the "SubmoduleIdentNumber" from the GSDML file.

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_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.3.4 PNIO_sub_plug_ext( ) (extended plugging of a submodule, not for new development)

Description

Note

This function is obsolete. You should no longer use it.

Instead, work with the function PNIO_sub_plug_ext_IM( ).

The function is called during startup by the IO-Base device user program. It reports the current configuration to the IO-Base interface.

It can also be called during operation if the current configuration changes, for example if a failed or removed submodule becomes functional again. In this case, a plug alarm is sent to the IO controller automatically by the IO-Base interface.

PNIO_sub_plug_ext( ) can inform the IO controller that a new submodule has been plugged in that does not correspond to the configuration. This means that the new submodule ID neither corresponds to the old submodule ID nor is compatible with the old submodule. This case, the "Plug Wrong Submodule Alarm" is sent to the IO controller.

PNIO_sub_plug_ext( ) is an asynchronous function. The return value provides information on whether or not the job was accepted by the interface. Confirmation of plugging is provided by the PNIO_CBF_PULL_PLUG_CONF( ) callback function.

Note

If the IO controller is informed of the change to the IO device with "plug alarm" or "plug wrong submodule alarm", no callback function is triggered with an IO controller with CP 16xx.

Note

No "plug alarms" are sent to the IO controller before the start of the cyclic data exchange that is signaled to the IO-Base device user program with the PNIO_CBF_AR_INDATA_IND( ) callback function.

Note

Whenever a submodule is plugged into a module, a "plug alarm" is sent to the IO controller.

If a module is inserted, no alarm is triggered.

Syntax

```
PNIO_UINT32 PNIO_sub_plug_ext(
    PNIO_UINT32             DevHndl,            //in
    PNIO_UINT32             API,                //in
    PNIO_DEV_ADDR          *pAddr,              //in
    PNIO_UINT32             SubIdent            //in
    PNIO_UINT32             AlarmType           //in
);
```

Parameter

Name	Description	
DevHndl	IO device handle from PNIO_device_open( )	
API	Application process identifier for the module - This must correspond to one of the API parameters transferred with the PNIO_api_add( ) function call.	
pAddr	Pointer to the address of the subslot in which the submodule is inserted.	
	Note Only the value 1 to "MaxSubslots" may be specified in the IO device address for the subslot number. The value for "MaxSubslots" is specified with the PNIO_api_add( ) call.	
SubIdent	Submodule ID from the relevant GSDML file.	
Alarm type	The "AlarmType" parameter decides whether a "Plug Alarm" or a "Plug Wrong Submodule Alarm" is sent to the IO controller after inserting a new submodule. If the new submodule has the wrong submodule ID and is also incompatible with the submodule in the project engineering, then according to the standard, the "Plug Wrong Submodule Alarm" must be sent.	
	If you want to send a ... ,	**set the "AlarmType" parameter to ...**
	Plug alarm	PNIO_ALARM_TYPE_PLUG
	Plug wrong submodule alarm	PNIO_ALARM_TYPE_PLUG_WRONG

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_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.4 Interface for write IO data on the IO device

On the IO device, data exchange for "write IO data" from the IO-Base device user program to the IO-Base interface is initiated with the following functions:

- PNIO_initiate_data_write( )
- PNIO_initiate_data_write_ext( )

The IO-Base interface calls the PNIO_CBF_DATA_WRITE( ) callback function for each subslot that has input data and meets the selection criteria.

Note

The PNIO_initiate_data_write( ) and PNIO_initiate_data_write_ext( ) functions are synchronous; in other words, the functions return only when all PNIO_CBF_DATA_WRITE( ) callbacks are completed.

Note

The PNIO_initiate_data_write( ) and PNIO_initiate_data_read( ) functions can return the error codes PNIO_WARN_NO_SUBMODULES and PNIO_WARN_IRT_INCONSISTENT if they are called in the PNIO_CBF_PRM_END_IND() function.

These return values can be ignored.

7.4.1 PNIO_initiate_data_write( ) (initiate writing RT data)

Description

This function triggers the writing of IO data to the IO-Base interface.

The IO-Base interface then calls the PNIO_CBF_DATA_WRITE( ) callback function for submodules with input data for which an I/O application relation with an IO controller exists. This requests the IO-Base device user program to write the relevant IO data to the process image.

Syntax

```
PNIO_UINT32  PNIO_initiate_data_write(
    PNIO_UINT32          DevHndl,              //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )

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_SEQUENCE
- PNIO_ERR_WRONG_HND

7.4.2 PNIO_initiate_data_write_ext( ) (trigger writing of selective data)

Description

This function initiates selective writing of IO data on the IO-Base interface once.

The IO-Base interface then calls the PNIO_CBF_DATA_WRITE( ) callback function for every submodule with input data that corresponds to the transferred criteria. This is only possible when there is an I/O application relation to an IO controller.

The PNIO_CBF_DATA_WRITE( ) callback function requests the IO-Base device user program to write the appropriate IO data to the process image.

Compared with PNIO_initiate_data_write( ), PNIO_initiate_data_write_ext( ) has the following advantages:

- RT and IRT data can be written.
- The submodules for which PNIO_CBF_DATA_WRITE( ) is called by the IO-Base interface can be restricted.
- The access mode for RT modules can be selected freely.

Syntax

```
PNIO_UINT32  PNIO_initiate_data_write_ext(
    PNIO_UINT32          DevHndl,          //in
    PNIO_DEV_ADDR       *pAddr,            //in
    PNIO_ACCESS_ENUM     AccessType        //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
pAddr	Address of the module/submodule for which PNIO_CBF_DATA_WRITE( ) will be called. If NULL is transferred with "pAddr", PNIO_CBF_DATA_WRITE( ) will be called for each submodule of the IO device. If an address that points to a module is transferred with "pAddr", in other words pAddr -> Subslot is equal to 0, a PNIO_CBF_DATA_WRITE( ) is called for each of the submodules of the specified module. If an address with a slot and a subslot that is not equal to 0 is specified with "pAddr", a PNIO_CBF_DATA_WRITE( ) will be called only for the specified submodule.
AccessType	Selects the type of IO data and the type of access; see also the section "PNIO_APPL_READY_LIST_TYPE (application ready) (Page 210)".

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_ACCESS_TYPE
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_WARN_IRT_INCONSISTENT
- PNIO_WARN_NO_SUBMODULES

7.4.3 Callback function PNIO_CBF_DATA_WRITE( ) (write data)

Description

The PNIO_CBF_DATA_WRITE( ) callback function is called by the IO-Base interface after the IO-Base device user program has initiated "write data".

The IO-Base device user program must read the physical inputs of the addressed submodule and copy them to the data buffer specified by the IO-Base interface.

The local and remote status are also exchanged.

Syntax

```
typedef PNIO_IOXS (* PNIO_CBF_DATA_WRITE)(
    PNIO_UINT32        DevHndl,             //in
    PNIO_DEV_ADDR     *pAddr,               //in
    PNIO_UINT32        BufLen,              //in
    PNIO_UINT8        *pBuffer,             //out
    PNIO_IOXS          IOremState           //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
pAddr	Pointer to the address of the submodule
BufLen	Length of the data buffer made available in bytes
	Note The length specified with "BufLen" must not be exceeded under any circumstances. This is the responsibility of the user.
pBuffer	Pointer to the data buffer for the IO data to be written.
IOremState	Remote status

Return values

Local status: PNIO_S_GOOD or PNIO_S_BAD

7.5 Interface for read IO data on the IO device

Description

On the IO device, data exchange for "read IO data" from the IO-Base device user program to the IO-Base interface is initiated with the following functions:

- PNIO_initiate_data_read( )
- PNIO_initiate_data_read_ext( )

The IO-Base interface calls the PNIO_CBF_DATA_READ( ) callback function for each subslot that has output data and meets the selection criteria.

Note

The PNIO_initiate_data_read( ) and PNIO_initiate_data_read_ext( ) functions are synchronous; in other words, the functions return only when all PNIO_CBF_DATA_READ( ) callbacks are completed.

The PNIO_initiate_data_write( ) and PNIO_initiate_data_read( ) functions can return the error codes PNIO_WARN_NO_SUBMODULES and PNIO_WARN_IRT_INCONSISTENT if they are called in the PNIO_CBF_PRM_END_IND() function.

These return values can be ignored.

7.5.1 PNIO_initiate_data_read( ) (initiate reading RT data)

Description

This function reads the RT IO output data from the IO-Base interface to the IO-Base device user program once (data transfer direction from IO controller to IO device).

The IO-Base interface then calls the PNIO_CBF_DATA_READ ( ) callback function for submodules with RT output data for which an IO application relation with an IO controller exists. This requests the IO-Base device user program to read the relevant RT IO data from the process image.

Syntax

```
PNIO_UINT32  PNIO_initiate_data_read(
    PNIO_UINT32          DevHndl,              //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )

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_SEQUENCE
- PNIO_ERR_WRONG_HND

7.5.2 PNIO_initiate_data_read_ext( ) (initiate reading selective data)

Description

This function initiates selective reading of IO data on the IO-Base interface once.

The IO-Base interface then calls the PNIO_CBF_DATA_READ( ) callback function for each submodule with output data that correspond to the transferred criteria. This is only possible when there is an I/O application relation to an IO controller.

The PNIO_CBF_DATA_READ( ) callback function requests the IO-Base device user program to read the appropriate IO data from the process image.

Compared with PNIO_initiate_data_read), PNIO_initiate_data_read_ext( ) has the following advantages:

- RT and IRT data can be read.
- The submodules for which PNIO_CBF_DATA_READ( ) is called by the IO-Base interface can be restricted.
- The access mode for RT modules can be selected freely.

Syntax

```
PNIO_UINT32 PNIO_initiate_data_read_ext(
    PNIO_UINT32          DevHndl,          //in
    PNIO_DEV_ADDR       *pAddr,            //in
    PNIO_ACCESS_ENUM     AccessType        //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
pAddr	Address of the module/submodule for which PNIO_CBF_DATA_READ( ) will be called. If NULL is transferred with "pAddr", PNIO_CBF_DATA_READ( ) will be called for each submodule of the IO device. If an address that points to a module is transferred with "pAddr", in other words pAddr -> Subslot is equal to 0, a PNIO_CBF_DATA_READ( ) is called for each of the submodules of the specified module. If an address with a slot and a subslot that is not equal to 0 is specified with "pAddr", a PNIO_CBF_DATA_READ( ) will be called only for the specified submodule.
AccessType	Selects the type of IO data and the type of access; see also the section "PNIO_APPL_READY_LIST_TYPE (application ready) (Page 210)".

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_ACCESS_TYPE
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_WARN_IRT_INCONSISTENT
- PNIO_WARN_NO_SUBMODULES

7.5.3 Callback function PNIO_CBF_DATA_READ( ) (read data)

Description

The callback function is called by the IO-Base interface after the IO-Base device user program has initiated a "read data" with PNIO_initiate_data_read( ) or PNIO_initiate_data_read_ext( ).

The IO-Base device user program must read the IO data stored in the data buffer and write it to the physical outputs of the addressed submodule.

The remote and local status are also exchanged.

Syntax

```
typedef PNIO_IOXS (* PNIO_CBF_DATA_READ)(
    PNIO_UINT32        DevHndl,             //in
    PNIO_DEV_ADDR     *pAddr,               //in
    PNIO_UINT32        BufLen,              //in
    PNIO_UINT8        *pBuffer,             //in
    PNIO_IOXS          IOremState           //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
pAddr	Pointer to the address of the submodule
BufLen	Length of the data buffer made available in bytes
pBuffer	Pointer to the data buffer for the IO data to be read
IOremState	Remote status

Return values

Local status: PNIO_S_GOOD or PNIO_S_BAD

7.6 Interface for read/write data record on the IO device

Overview

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

- PNIO_CBF_REC_READ( )
- PNIO_CBF_REC_WRITE( )

7.6.1 Callback function PNIO_CBF_REC_READ( ) (process read data record job)

Description

The PNIO_CBF_REC_READ( ) callback function is called by the IO-Base interface when the controller sends a read data record request to the IO device.

A data record is addressed by the submodule address and the data record number. The IO-Base device user program reads the requested data from the subslot and writes this to the address area specified with "pBuffer".

Syntax

```
typedef void (* PNIO_CBF_REC_READ)(
    PNIO_UINT32           DevHndl,             //in
    PNIO_UINT32           API,                 //in
    PNIO_UINT16           ArNumber,            //in
    PNIO_UINT16           SessionKey,          //in
    PNIO_UINT32           SequenceNum,         //in
    PNIO_DEV_ADDR        *pAddr,               //in
    PNIO_UINT32           RecordIndex,         //in
    PNIO_UINT32          *pBufLen,             //in,out
    PNIO_UINT8           *pBuffer,             //out
    PNIO_ERR_STAT        *pPnioState           //out
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application Process Identifier

Name	Description
ArNumber	Application relation number
SessionKey	Session key for the current application relation
SequenceNum	Consecutive number of the data record
pAddr	Pointer to the local module address for this job.
	Note There are also read data record jobs that are not assigned to a submodule, see IEC 61158, for example data record number 0xEC00 to 0xEFFF. In these cases, pAddr points to module addresses in which no module was inserted!
RecordIndex	Data record number
pBufLen	In: Maximum permitted the data record length in bytes Out: Actual written data record length in bytes
	Note The length specified with "pBufLen" must not be exceeded under any circumstances. This is the responsibility of the user.
pBuffer	Pointer to data buffer to which the IO-Base device user program will copy the read data.
pPnioState	Error code on the execution of the job

Return values

No return values.

7.6.2 Callback function PNIO_CBF_REC_WRITE( ) (process write data record job)

Description

The PNIO_CBF_REC_WRITE( ) callback function is called by the IO-Base interface when the controller sends a write data record request to the IO device.

A data record is addressed by the local slot/subslot address and the data record number. The IO-Base device user program accepts the data record data starting at the address specified with "pBuffer" and writes it to the subslot.

Syntax

```
typedef void (* PNIO_CBF_REC_WRITE)(
    PNIO_UINT32           DevHndl,             //in
    PNIO_UINT32           API,                 //in
    PNIO_UINT16           ArNumber,            //in
    PNIO_UINT16           SessionKey,          //in
    PNIO_UINT32           SequenceNum,         //in
    PNIO_DEV_ADDR        *pAddr,               //in
    PNIO_UINT32           RecordIndex,         //in
    PNIO_UINT32          *pBufLen,             //in
    PNIO_UINT8           *pBuffer,             //in
    PNIO_ERR_STAT        *pPnioState           //out
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application Process Identifier
ArNumber	Application relation number
SessionKey	Session key for the current application relation
SequenceNum	Consecutive number of the data record
pAddr	**Note** There are also write data record jobs that are not assigned to a submodule, see IEC 61158, for example data record number 0xEC00 - 0xEFFF. In these cases, "pAddr" points to module addresses in which no module was insert-ed!
RecordIndex	Data record number
pBufLen	Amount of data in bytes
pBuffer	Pointer to record data
pPnioState	Error code on the execution of the job

Return values

No return values.

7.7 Interface for managing diagnostics data on the IO device

Overview

These functions are available for an IO device for diagnostics purposes:

- PNIO_build_channel_properties( )
- PNIO_diag_channel_add( )

- PNIO_diag_ext_channel_add( )
- PNIO_diag_channel_remove( )
- PNIO_diag_ext_channel_remove( )
- PNIO_diag_generic_add( )
- PNIO_diag_generic_remove( )

7.7.1 PNIO_build_channel_properties( ) (generate channel properties)

Description

With this function, the IO-Base device user program can generate a packed bit structure for channel diagnostics data records. This bit structure can be downloaded to a subslot with the PNIO_diag_channel_add( ) function.

Syntax

```
PNIO_UINT16 PNIO_build_channel_properties(
    PNIO_UINT32    DevHndl,                //in
    PNIO_UINT16    Type,                   //in
    PNIO_UINT16    Spec,                   //in
    PNIO_UINT16    Dir                     //in
);
```

Parameter

Name	Description	
DevHndl	IO device handle from PNIO_device_open( )	
Type	Packing size for the channel diagnostics 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	
Spec	Meaning of the channel diagnostics property:	
	Value	**Description**
	PNIO_DIAG_CHAN_SPEC_ERROR_APPEARS	Error entering state.

Name	Description	
	PNIO_DIAG_CHAN_SPEC_ERROR_DISAPP_FREE	Error exiting state, no further problems.
	PNIO_DIAG_CHAN_SPEC_ERROR_DISAPP_REMAIN	Problem exiting state, further problems exist.
Dir	Data direction of the module for which the diagnostics data will be created.	
	Value	**Description**
	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.

7.7.2 PNIO_diag_channel_add( ) (store channel diagnostic data in subslot)

Description

This function downloads a channel diagnostics data record to a subslot.

Note

Several diagnostics data records can be downloaded to a subslot. These are referenced by DiagTag.

Syntax

```
PNIO_UINT32  PNIO_diag_channel_add(
    PNIO_UINT32          DevHndl,             //in
    PNIO_UINT32          API,                 //in
    PNIO_DEV_ADDR       *pAddr,               //in
    PNIO_UINT16          ChannelNum,          //in
    PNIO_UINT16          ChannelProp,         //in
    PNIO_UINT32          ChannelErrType,      //in
    PNIO_UINT16          DiagTag              //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application process identifier - Must correspond to the API parameter to which the submodule belongs.
pAddr	Pointer to the local subslot address to which the diagnostics data record will be downloaded.
ChannelNum	Channel number. Range of values 0x8000 (see IEC 61158)
ChannelProp	The "ChannelProp" parameter is generated by the PNIO_build_channel_properties( ) function. The PNIO_build_channel_properties( ) function must be called with the following parameters: • Type = PNIO_DIAG_CHANPROP_TYPE_SUBMOD • Dir = PNIO_DIAG_CHAN_DIRECTION_MANUFACTURE (vendor-specific) • Spec = PNIO_DIAG_CHAN_SPEC_ERROR_APPEARS (error entering state) Diagnostics data - Return value of the PNIO_build_channel_properties( ) function
ChannelErrType	Channel error type - possible values with the prefix "PNIO_DIAG_CHAN_ERROR_" can be found in the header files.
DiagTag	Unique reference specified by the user to the diagnostics data record (!=0)

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_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.7.3 PNIO_diag_ext_channel_add( ) (store extended channel diagnostics data in subslot)

Description

This function downloads an extended channel diagnostics data record to a subslot.

Note

Several diagnostics data records can be downloaded to a subslot. These are referenced by "DiagTag".

Syntax

```
PNIO_UINT32  PNIO_diag_ext_channel_add(
    PNIO_UINT32          DevHndl,            //in
    PNIO_UINT32          API,                //in
    PNIO_DEV_ADDR       *pAddr,              //in
    PNIO_UINT16          ChannelNum,         //in
    PNIO_UINT16          ChannelProp,        //in
    PNIO_UINT16          ChannelErrType,     //in
    PNIO_UINT16          ExChannelErrType,   //in
    PNIO_UINT32          ExChannelAddValue,  //in
    PNIO_UINT16          DiagTag             //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application process identifier - Must correspond to the API parameter to which the submodule belongs.
pAddr	Pointer to the local subslot address to which the diagnostics data record will be downloaded.
ChannelNum	Channel number - range of values 0x8000 (see IEC 61158)

Name	Description
ChannelProp	The "ChannelProp" parameter is generated by the PNIO_build_channel_properties( ) function. The PNIO_build_channel_properties( ) function must be called with the following parameters: • Type = PNIO_DIAG_CHANPROP_TYPE_SUBMOD • Dir = PNIO_DIAG_CHAN_DIRECTION_MANUFACTURE (vendor-specific) • Spec = PNIO_DIAG_CHAN_SPEC_ERROR_APPEARS (error entering state) Diagnostics data - Return value of the PNIO_build_channel_properties( ) function
ChannelErrType	Channel error type - possible values with the prefix "PNIO_DIAG_CHAN_ERROR_" can be found in the header files.
ExtChannelErrType	The "ExtChannelErrorType" parameter can be used to transfer additional information on the channel error; for possible values, refer to the PROFINET IO standard IEC 61158.
ExtChannelAddValue	The "ExtChannelAddValue" parameter can be used to transfer additional information over and above the type of channel error; for possible values, refer to the PROFINET IO standard IEC 61158.
DiagTag	Unique reference specified by the user to the diagnostics data record (!=0)

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_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.7.4 PNIO_diag_channel_remove( ) (remove diagnostics data from subslot)

Description

With this function, a channel diagnostics data record loaded on a subslot can be deleted.

Syntax

```
PNIO_UINT32  PNIO_diag_channel_remove(
    PNIO_UINT32             DevHndl,          //in
    PNIO_UINT32             API,              //in
    PNIO_DEV_ADDR          *pAddr,            //in
    PNIO_UINT16             DiagTag           //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application process identifier - Must correspond to the API parameter to which the submodule belongs.
pAddr	Pointer to the local subslot address from which the diagnostics data record will be deleted.
DiagTag	• DiagTag ≠ 0 - Unique reference to the diagnostics data record to be deleted as specified by the user. • DiagTag = 0 - All channel diagnostics data records of the submodule are deleted.
	Note If the PNIO_diag_channel_remove( ) function is called with an invalid "Diag-Tag", there is no error message.

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_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.7.5 PNIO_diag_ext_channel_remove( ) (remove extended channel diagnostics data from subslot)

Description

With this function, an extended channel diagnostics data record loaded on a subslot can be deleted.

Syntax

```
PNIO_UINT32  PNIO_diag_channel_remove(
    PNIO_UINT32             DevHndl,           //in
    PNIO_UINT32             API,               //in
    PNIO_DEV_ADDR          *pAddr,             //in
    PNIO_UINT16             DiagTag            //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application process identifier - Must correspond to the API parameter to which the submodule belongs.
pAddr	Pointer to the local subslot address from which the diagnostics data record will be deleted.
DiagTag	• DiagTag ≠ 0 - Unique reference to the diagnostics data record to be deleted as specified by the user. • DiagTag = 0 - All extended channel diagnostics data records of the submodule are deleted.
	Note If the PNIO_diag_ext_channel_remove( ) function is called with an invalid "DiagTag", there is no error message.

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_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.7.6 PNIO_diag_generic_add( ) (store vendor-specific diagnostics data in the subslot)

Description

This function downloads a manufacturer-specific diagnostics data record to a subslot.

Note

Several manufacturer-specific diagnostics data records can be downloaded to a submodule. These are referenced by "DiagTag".

Syntax

```
PNIO_UINT32  PNIO_diag_generic_add(
    PNIO_UINT32        DevHndl,            //in
    PNIO_UINT32        API,                //in
    PNIO_DEV_ADDR     *pAddr,              //in
    PNIO_UINT16        ChannelNum,         //in
    PNIO_UINT16        ChannelProp,        //in
    PNIO_UINT16        DiagTag,            //in
    PNIO_UINT16        UserStructIdent,    //in
    PNIO_UINT8        *pInfoData,          //in
    PNIO_UINT32        InfoDataLen         //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application process identifier - Must correspond to the API parameter to which the submodule belongs.
pAddr	Pointer to the local subslot address to which the diagnostics data record will be downloaded.
ChannelNum	Channel number - According to the PROFINET IO standard, the"ChannelNumber" for vendor-specific diagnostics is always 0x0 to 0x7FFF (see IEC 61158)!
ChannelProp	Diagnostics data - Return value of the PNIO_build_channel_properties( ) function
DiagTag	Unique reference specified by the user to the diagnostics data record (!=0)
UserStructIdent	Structure of the diagnostics data; see IEC 61158, User structure identifier with range of values 0x0 to 0x7FFF.
pInfoData	Pointer to the diagnostics data in network format (big endian)
InfoDataLen	Length of the diagnostics data in bytes, max. 1404.

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_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.7.7 PNIO_diag_generic_remove( ) (remove vendor-specific diagnostics data from the submodule)

Description

With this function, a vendor-specific diagnostics data record loaded on a submodule can be deleted.

Syntax

```
PNIO_UINT32  PNIO_diag_generic_remove(
    PNIO_UINT32             DevHndl,          //in
    PNIO_UINT32             API,              //in
    PNIO_DEV_ADDR          *pAddr,            //in
    PNIO_UINT16             DiagTag           //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application process identifier - Must correspond to the API parameter to which the submodule belongs.

Name	Description
pAddr	Pointer to the local submodule address from which the diagnostics data record will be deleted.
DiagTag	• DiagTag ≠ 0 - Unique reference to the diagnostics data record to be deleted as specified by the user. • DiagTag = 0 - All vendor-specific diagnostics data records of the submodule are deleted
	Note If the PNIO_diag_generic_remove ( ) function is called with an invalid "DiagTag", there is no error message.

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_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.8 Alarm interface for the IO device

Overview

The following functions are available for handling alarms between an IO device and an IO controller:

- PNIO_process_alarm_send( )
- PNIO_diag_alarm_send( )
- PNIO_ret_of_sub_alarm_send( )
- PNIO_CBF_REQ_DONE( )

7.8.1 PNIO_process_alarm_send( ) (send process alarm)

Description

This function sends a submodule-specific process alarm from the IO device to the IO controller. The result of the asynchronous job is signaled with the PNIO_CBF_REQ_DONE( ) callback function.

The alarm data is not stored on the submodule. As an option, the user of the function can add user-specific alarm data in pData.

Syntax

```
PNIO_UINT32  PNIO_process_alarm_send(
    PNIO_UINT32       DevHndl,              //in
    PNIO_UINT32       API,                  //in
    PNIO_UINT16       ArNumber,             //in
    PNIO_UINT16       SessionKey,           //in
    PNIO_DEV_ADDR    *pAddr,                //in
    PNIO_UINT8       *pData,                //in
    PNIO_UINT32       DataLen,              //in
    PNIO_UINT16       UserStructIdent,      //in
    PNIO_UINT32       UserHndl              //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application process identifier - Must correspond to the API parameter to which the submodule belongs.
ArNumber	Application relation number from PNIO_CBF_AR_CHECK_IND( ) - For a description of the callback, see the section "Callback function PNIO_CBF_AR_CHECK_IND( ) (signal application relation check indication) (Page 191)"
SessionKey	Session key from PNIO_CBF_AR_CHECK_IND( ) - For a description of the callback, see the section "Callback function PNIO_CBF_AR_CHECK_IND( ) (signal application relation check indication) (Page 191)"
pAddr	Pointer to the local submodule address at which the alarm was generated.
pData	Data buffer containing the alarm data in the network format (big endian).
DataLen	Length of data in bytes - The IO device application must expect the rejection of the service as of a "DataLen" of 172 bytes depending on the capabilities of the controller. If the IO controller is a CP 1616 or a CP 1604, "DataLen" is a maximum of 172 bytes.
UserStructIdent	For the structure of the diagnostic data, see IEC 61158 / User structure identifier.
UserHndl	Handle assigned by the IO-Base device user program to identify the alarm when the confirmation is received with PNIO_CBF_REQ_DONE( ).

Return values

If successful, PNIO_OK is returned asynchronously. If an error occurs, the following values are possible as synchronous or asynchronous return values (for the meanings, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN
- PNIO_ERR_SEQUENCE
- PNIO_ERR_SESSION
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND

7.8.2 PNIO_diag_alarm_send( ) (send diagnostics alarm)

Description

This function sends a submodule-specific diagnostics alarm from the IO device to the IO controller. The result of the asynchronous job is signaled with the PNIO_CBF_REQ_DONE( ) callback function.

Diagnostics alarms can be entering or exiting state. Before an entering state alarm is sent, the relevant diagnostics data record must be transferred to the IO-Base interface with PNIO_diag_channel_add( ), PNIO_diag_ext_channel_add( ) or PNIO_diag_generic_add( ) to allow the alarm data to be read out later.

If the cause of an alarm is no longer present, the data record must first be removed with PNIO_diag_channel_remove( ), PNIO_diag_ext_channel_remove( ) or PNIO_diag_generic_remove( ) and then the corresponding exiting state alarm must be triggered.

Syntax

```
PNIO_UINT32  PNIO_diag_alarm_send(
    PNIO_UINT32          DevHndl,            //in
    PNIO_UINT32          API,                //in
    PNIO_UINT16          ArNumber,           //in
    PNIO_UINT16          SessionKey,         //in
    PNIO_UINT32          AlarmState,         //in
    PNIO_DEV_ADDR       *pAddr,              //in
    PNIO_UINT8          *pData,              //in
    PNIO_UINT32          DataLen,            //in
    PNIO_UINT16          UserStructIdent,    //in
    PNIO_UINT32          UserHndl            //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application process identifier - Must correspond to the API parameter to which the submodule belongs.
ArNumber	Application relation number from PNIO_CBF_AR_CHECK_IND( ) - For a description of the callback, see the section "Callback function PNIO_CBF_AR_CHECK_IND( ) (signal application relation check indication) (Page 191)".
SessionKey	Session key from PNIO_CBF_AR_CHECK_IND( ) - For a description of the callback, see the section "Callback function PNIO_CBF_AR_CHECK_IND( ) (signal application relation check indication) (Page 191)".
AlarmState	Indicates whether the diagnostics alarm is entering or exiting state; possible values are: • PNIO_STATE_ALARM_APPEARS • PNIO_STATE_ALARM_DISAPPEARS
pAddr	Pointer to the local submodule address at which the alarm was generated.
	Note If the "Slot" or "Subslot" structure element is specified incorrectly, there is no error message in the return value of this function.

Name	Description	
pData	Pointer to alarm data in network format (big endian) - The diagnostics data for the diagnostics alarm must be transferred with the "pData“ pointer. This diagnostics data must correspond to the diagnostics data that was transferred earlier either with the PNIO_diag_generic_add( ) function or with the PNIO_diag_channel_add( ) function. If you transferred the diagnostics data with the PNIO_diag_generic_add( ) function, the diagnostics data corresponds to the data to which "pInfoData" points. If you transferred the diagnostics data with the PNIO_diag_channel_add( ) function, the diagnostics data must be transferred as a field consisting of the following values: • ChannelNum (big endian) • ChannelProp (big endian) • ChannelErrorType (big endian) If you transfer "pData = NULL", this means that all previously set alarms exit the alarm state.	
	Note The "pData" pointer must point to alarm data after the "user structure identifier” data element. The alarm data is described in the "Alarm ASE.pdf" document. You will find the document in the "doc" folder on the product CD "DK-16xx PN IO".	
DataLen	Length of the data in bytes that was transferred with "pData".	
	Note When all alarms are exiting the state, transfer DataLen = 0. Depending on the capabilities of IO controller, the IO device application must expect the service to be denied as of a "DataLen" of 172. If the IO controller is a CP 1616 or a CP 1604, "DataLen" is a maximum of 172 bytes. (The maximum value of the parameter "DataLen" is limited by the maximum length of the "AlarmNotification" PDU. According to the PROFINET IO standard, the maximum length of the PDU on the network depends on the IO controller and is between 200 and 1432 bytes including the PDU header. The PDU header is 28 bytes long per PDU sent; in other words, the IO device application must expect the service to be denied as of a "DataLen" of 172.)	
UserStructIdent	User Structure Identifier - Provides information on the data to which "pData" points.	
	0x0000 to 0x7FFF	Vendor-specific data in "pData", see also "PNIO_diag_generic_add( ) (store vendor-specific diagnostics data in the subslot) (Page 177)" function.
	0x8000	Channel diagnostics data in "pData"
		Note When creating the channel properties with PNIO_build_channel_properties, the value PNIO_DIAG_CHANPROP_TYPE_ SUBMOD must be transferred for "Type" and the value PNIO_DIAG_CHAN_DIRECTION_ MANUFACTURE for "Dir".
	0x8001	This value must not be set.

Name	Description	
		Note It is not possible to send multiple diagnostics alarms to the IO controller within one alarm notification PDU. This means that the "UserStructIdent" parameter must not be set to the value 0x8001.
	0x8002	Extended channel diagnostics data
UserHndl	Handle assigned by the IO-Base device user program to identify the alarm when the confirmation is received with PNIO_CBF_REQ_DONE( ).	

Return values

If successful, PNIO_OK is returned asynchronously. If an error occurs, the following values are possible as synchronous or asynchronous return values (for the meanings, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN
- PNIO_ERR_SEQUENCE
- PNIO_ERR_SESSION
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND

7.8.3 PNIO_ret_of_sub_alarm_send( ) (send return of submodule alarm)

Description

With this function, a submodule-specific return of submodule alarm is sent by the IO device to the IO controller if the status of the value sent with the user data IOPS/IOCS changes from the BAD to the GOOD state. The reaction on the IO controller is similar to that when there is a module insert alarm, however in this case, the submodule is not assigned new parameter settings.

The result of the asynchronous job is signaled with the PNIO_CBF_REQ_DONE( ) callback function.

Syntax

```
PNIO_UINT32  PNIO_ret_of_sub_alarm_send(
    PNIO_UINT32        DevHndl,           //in
    PNIO_UINT32        API,               //in
    PNIO_UINT16        ArNumber,          //in
    PNIO_UINT16        SessionKey,        //in
    PNIO_DEV_ADDR     *pAddr,             //in
    PNIO_UINT32        UserHndl           //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application process identifier - Must correspond to the API parameter to which the submodule belongs.
ArNumber	Application relation number from PNIO_CBF_AR_CHECK_IND( ) - For a description of the callback, see the section "Callback function PNIO_CBF_AR_CHECK_IND( ) (signal application relation check indication) (Page 191)".
SessionKey	Session key from PNIO_CBF_AR_CHECK_IND( ) - For a description of the callback, see the section "Callback function PNIO_CBF_AR_CHECK_IND( ) (signal application relation check indication) (Page 191)".
pAddr	Pointer to the local submodule address at which the alarm was generated.
	Note If the "Slot" or "Subslot" structure element is specified incorrectly, there is no error message in the return value of this function.
UserHndl	Handle assigned by the IO-Base device user program to identify the alarm when the confirmation is received with PNIO_CBF_REQ_DONE( ).

Return values

If successful, PNIO_OK is returned asynchronously. If an error occurs, the following values are possible as synchronous or asynchronous return values (for the meanings, refer to the comments in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_SESSION
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_WRONG_HND

7.8.4 Callback function PNIO_CBF_REQ_DONE( ) (signal alarm confirmation)

Description

With the PNIO_CBF_REQ_DONE( ) callback function, the IO controller confirms an alarm signaled by the IO device.

Syntax

```
typedef void (* PNIO_CBF_REQ_DONE)(
    PNIO_UINT32        DevHndl,            //in
    PNIO_UINT32        UserHndl,           //in
    PNIO_UINT32        Status,             //in
    PNIO_ERR_STAT     *pPnioState          //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
UserHndl	Handle assigned by the IO-Base device user program when the alarm was called.
Status	If successful, PNIO_OK is returned. If an error occurs, refer to the header file "pnioerrx.h".
pPnioState	PROFINET IO error code transferred with the alarm confirmation.

Return values

No return values.

7.9 Interface for connection establishment and termination for the IO device

Overview

The following functions are available for connection establishment and termination and for operating states of an IO device:

- PNIO_device_ar_abort( )
- PNIO_set_appl_state_ready( )
- PNIO_CBF_CHECK_IND( )
- PNIO_CBF_AR_INFO_IND( )
- PNIO_CBF_AR_INDATA_IND( )

- PNIO_CBF_AR_ABORT_IND( )
- PNIO_CBF_AR_OFFLINE_IND( )
- PNIO_CBF_APDU_STATUS_IND( )
- PNIO_CBF_PRM_END_IND( )

7.9.1 PNIO_device_ar_abort( ) (force connection abort)

Description

This function aborts the application relation. As a result, the connection between the IO controller and the IO device is terminated.

If data is exchanged between IO controller and IO device (PNIO_CBF_AR_INDATA_IND( ) callback was called by the interface), the IO-Base interface calls the PNIO_CBF_AR_OFFLINE_IND( ) callback.

If no data has been exchanged between IO controller and IO device (PNIO_CBF_AR_INDATA_IND( ) callback has not yet been called by the interface), the IO-Base interface calls the PNIO_CBF_AR_ABORT_IND( ) callback.

Syntax

```
PNIO_UINT32  PNIO_device_ar_abort(
    PNIO_UINT32          DevHndl,                //in
    PNIO_UINT16          ArNumber,               //in
    PNIO_UINT16          SessionKey              //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
ArNumber	Application relation number from PNIO_CBF_AR_CHECK_IND( ) - For a description of the callback, see the section "Callback function PNIO_CBF_AR_CHECK_IND( ) (signal application relation check indication) (Page 191)".
SessionKey	Session key from PNIO_CBF_AR_CHECK_IND( ) - For a description of the callback, see the section "Callback function PNIO_CBF_AR_CHECK_IND( ) (signal application relation check indication) (Page 191)".

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_SESSION
- PNIO_ERR_WRONG_HND

7.9.2 PNIO_set_appl_state_ready( ) (signal ready for data exchange)

Description

This function transfers an "application ready" and all non functional submodules to the IO controller. The user program informs the IO-Base interface that it is ready to exchange data.

Syntax

```
PNIO_UINT32 PNIO_set_appl_state_ready(
    PNIO_UINT32                  DevHndl,         //in
    PNIO_UINT16                  ArNumber,        //in
    PNIO_UINT16                  SessionKey,      //in
    PNIO_APPL_READY_LIST_TYPE    *pList           //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
ArNumber	Application relation number from PNIO_CBF_AR_CHECK_IND( ) - For a description of the callback, see the section "Callback function PNIO_CBF_AR_CHECK_IND( ) (signal application relation check indication) (Page 191)".
SessionKey	Session key from PNIO_CBF_AR_CHECK_IND( ) - For a description of the callback, see the section "Callback function PNIO_CBF_AR_CHECK_IND( ) (signal application relation check indication) (Page 191)".
pList	Chained list with nonfunctional submodules; see the section "PNIO_APPL_READY_LIST_TYPE (application ready) (Page 210)".

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

7.9.3 Callback function PNIO_CBF_CHECK_IND( ) (signal check indication)

Description

The PNIO_CBF_CHECK_IND( ) callback function is called by the IO-Base interface if a difference is detected between the expected and actual configuration on the IO device. In this case, the IO-Base device user program must provide specific information on the module/submodule IDs and module/submodule status of the modules that are actually inserted.

PNIO_CBF_CHECK_IND( ) is not called for missing modules and submodules. These are automatically reported as missing. This means that the values PNIO_MOD_STATE_NO_MODULE and PNIO_SUB_CHECK_NO_SUBMODULE are not required for PNIO_CBF_CHECK_IND( ).

The following values are returned as the module/submodule value:

Value	Module/submodule status
PNIO_MOD_STATE_PROPER_MODULE	When a configured module is inserted.
PNIO_MOD_STATE_SUBSTITUTED_ MODULE	When a substitute module is inserted.
PNIO_MOD_STATE_WRONG_MODULE	When an incorrect module is inserted.
PNIO_SUB_CHECK_PROPER_MODULE	When a configured submodule is inserted.
PNIO_SUB_CHECK_SUBSTITUTED_ MODULE	When a substitute submodule is inserted.
PNIO_SUB_CHECK_WRONG_ MODULE	When an incorrect submodule is inserted.

Syntax

```
typedef void (* PNIO_CBF_CHECK_IND)(
    PNIO_UINT32        DevHndl,            //in
    PNIO_UINT32        API,                //in
    PNIO_UINT16        ArNumber,           //in
    PNIO_UINT16        SessionKey,         //in
    PNIO_DEV_ADDR     *pAddr,              //in
    PNIO_UINT32       *pModIdent,          //out
    PNIO_UINT16       *pModState,          //out
    PNIO_UINT32       *pSubIdent,          //out
    PNIO_UINT16       *pSubState           //out
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
API	Application Process Identifier
ArNumber	Application relation number
SessionKey	Session key for the current application relation
pAddr	Pointer to the address of the subslot in which no suitable submodule is inserted.
pModIdent	Pointer to the buffer in which the IO-Base device user program must write the module identification of the module actually inserted. Module identification corresponds to the "ModulIdentNumber" from the GSDML file.
pModState	Pointer to the buffer in which the IO-Base device user program must write the status of the module. The following values are possible: • PNIO_MOD_STATE_WRONG_MODULE • PNIO_MOD_STATE_SUBSTITUTED_MODULE • PNIO_MOD_STATE_PROPER_MODULE
pSubIdent	Pointer to the buffer in which the IO-Base device user program will write the module identification of the submodule actually inserted.
pSubState	Pointer to the buffer in which the IO-Base device user program will write the status of the submodule. The following values are possible: • PNIO_SUB_CHECK_WRONG_SUBMODULE • PNIO_SUB_CHECK_SUBSTITUTED_SUBMODULE • PNIO_SUB_CHECK_PROPER_SUBMODULE

Return values

No return values.

7.9.4 Callback function PNIO_CBF_AR_CHECK_IND( ) (signal application relation check indication)

Description

This callback is called by the IO-Base interface to inform the IO-Base device user program of the properties of the application relation.

This callback function transfers the new values for the `pAr -> ArNumber` and `pAr -> SessionKey` parameters to the user application and must be used by the application in diagnostics and alarm functions.

Session key must be stored separately for every connection ("ArNumber") and is used along with the "ArNumber" parameter in all subsequent function calls.

Note

The module list that exists in "pAr" along with other values is always empty!

Syntax

```
typedef void (* PNIO_CBF_AR_CHECK_IND)(
    PNIO_UINT32       DevHndl,                //in
    PNIO_UINT32       HostIp,                 //in
    PNIO_UINT16       ArType,                 //in
    PNIO_UUID_TYPE    ArUUID,                 //in
    PNIO_UINT32       ArProperties,           //in
    PNIO_UUID_TYPE    CmiObjUUID,             //in
    PNIO_UINT16       CmiStationNameLength,   //in
    PNIO_UINT8       *pCmiStationName,        //in
    PNIO_AR_TYPE     *pAr                     //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
HostIp	IP address of the IO controller
ArType	Type of connection establishment
ArUUID	Application relation UUID
ArProperties	Properties of the application relation
CmiObjUUID	UUID of the connection establishment initiator, for example the IO controller.
CmiStationName Length	Length of the station name in bytes to which the next field points.

Name	Description
pCmiStationName	Station name of the connection establishment initiator, for example the IO controller.
pAr	Pointer to the PNIO_AR_TYPE structure - The application must store the `pAr -> ArNumber` and `pAr -> SessionKey` parameters.

Return values

No return values.

7.9.5 Callback function PNIO_CBF_AR_INFO_IND( ) (signal application relation information)

Description

With the PNIO_CBF_AR_INFO_IND( ) callback function, the IO-Base interface informs the IO-Base device user program which modules and submodules are operating in this application relation. With the "pAR" transfer parameter, a pointer to a chained structure is transferred. The structure contains all the important information, for example which modules and submodules the established application relation contains.

Note

The IO-Base device user program must expect that several ARs will be established. An engineering station can, for example, query I&M data records from the IO device via a second supervisor AR.

Syntax

```
typedef void (* PNIO_CBF_AR_INFO_IND)(
    PNIO_UINT32     DevHndl,                //in
    PNIO_UINT16     ArNumber,               //in
    PNIO_UINT16     SessionKey,             //in
    PNIO_AR_TYPE   *pAR                     //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
ArNumber	Application relation number
SessionKey	Session key for the current application relation
pAr	Pointer to a chained application relation structure; see the section "PNIO_AR_TYPE (application relation) (Page 216)".

Return values

No return values.

7.9.6 Callback function PNIO_CBF_AR_INDATA_IND( ) (signal application relation InData)

Description

With the PNIO_CBF_AR_INDATA_IND( ) callback function, the IO-Base interface informs the IO-Base device user program that the cyclic data exchange has started.

Note

The PNIO_CBF_AR_INDATA_IND( ) callback function can be called by the IO-Base programming interface before the PNIO_set_appl_ready( ) function has returned a result.

Syntax

```
typedef void (* PNIO_CBF_AR_INDATA_IND)(
    PNIO_UINT32     DevHndl,                //in
    PNIO_UINT16     ArNumber,               //in
    PNIO_UINT16     SessionKey              //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
ArNumber	Application relation number
SessionKey	Session key for the current application relation

Return values

No return values.

7.9.7 Callback function PNIO_CBF_AR_ABORT_IND( ) (signal abort event)

Description

With the PNIO_CBF_AR_ABORT_IND( ) callback function, the IO-Base interface informs the IO-Base device user program when an existing connection was interrupted or terminated by the IO controller before application relation InData was signaled.

Syntax

```
typedef void (* PNIO_CBF_AR_ABORT_IND)(
    PNIO_UINT32     DevHndl,              //in
    PNIO_UINT16     ArNumber,             //in
    PNIO_UINT16     SessionKey,           //in
    PNIO_AR_REASON  ReasonCode            //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
ArNumber	Application relation number
SessionKey	Session key for the current application relation
ReasonCode	Reason for the connection abort, see the section PNIO_APPL_READY_LIST_TYPE (application ready) (Page 210).

Return values

No return values.

7.9.8 Callback function PNIO_CBF_AR_OFFLINE_IND( ) (signal offline event)

Description

With the PNIO_CBF_AR_OFFLINE_IND( ) callback function, the IO-Base interface informs the IO-Base device user program of an offline event when an existing connection was interrupted or terminated by the IO controller after application relation InData was signaled.

Syntax

```
typedef void (*PNIO_CBF_AR_OFFLINE_IND)(
    PNIO_UINT32     DevHndl,                //in
    PNIO_UINT16     ArNumber,               //in
    PNIO_UINT16     SessionKey,             //in
    PNIO_AR_REASON  ReasonCode              //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
ArNumber	Application relation number
SessionKey	Session key for the current application relation
ReasonCode	Reason for the connection abort, see the section PNIO_APPL_READY_LIST_TYPE (application ready) (Page 210).

Return values

No return values.

7.9.9 Callback function PNIO_CBF_APDU_STATUS_IND( ) (signal status of the IO controller)

Description

With the PNIO_CBF_APDU_STATUS_IND( ) callback function, the IO-Base interface informs the IO-Base device user program that the status of the IO controller has changed.

Syntax

```
typedef void (* PNIO_CBF_APDU_STATUS_IND)(
    PNIO_UINT32             DevHndl,            //in
    PNIO_UINT16             ArNumber,           //in
    PNIO_UINT16             SessionKey,         //in
    PNIO_APDU_STATUS_IND    ApduStatus          //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
ArNumber	Application relation number

Name	Description
SessionKey	Session key for the current application relation
ApduStatus	New status of the IO controller, see the section "PNIO_APDU_STATUS_IND (IO controller status) (Page 218)"

Return values

No return values.

7.9.10 Callback function PNIO_CBF_PRM_END_IND( ) (signal end of parameter assignment by IO controller)

Description

With the PNIO_CBF_PRM_END_IND( ) callback function, the IO-Base interface informs the IO-Base device user program that the IO controller has completed parameter assignment.

Syntax

```
typedef void (* PNIO_CBF_PRM_END_IND)(
    PNIO_UINT32     DevHndl,            //in
    PNIO_UINT16     ArNumber,           //in
    PNIO_UINT16     SessionKey,         //in
    PNIO_UINT32     API,                //in
    PNIO_UINT32     SlotNum,            //in
    PNIO_UINT32     SubslotNum          //in
);
```

Parameter

Name	Description	
DevHndl	IO device handle from PNIO_device_open( )	
ArNumber	Application relation number	
SessionKey	Session key for the current application relation	
API	Application process identifier - Valid only when SubslotNr does not equal 0.	
SlotNum	Slot number (valid only when SubslotNr does not equal 0.)	
SubslotNum	Subslot number	
	0	End of parameter assignment for all submodules, for example, at connection establishment.
	otherwise	End of parameter assignment of this submodule, for example when inserting a submodule later.

Return values

No return values.

7.10 Station signaling with LEDs

Overview

This interface consists of the following callbacks:

- PNIO_CBF_START_LED_FLASH( )
- PNIO_CBF_STOP_LED_FLASH( )

7.10.1 PNIO_CBF_START_LED_FLASH( ) (activate flashing mode for LEDs)

Description

The PNIO_CBF_START_LED_FLASH( ) callback function signals the arrival of a request for identification to your IO-Base device user program.

The IO-Base device user program can then take suitable measures to attract the attention of the operator) for example by lighting up a diode. The callback function must already have been registered with the PNIO_device_open( ) function.

Note

The callback functions PNIO_CBF_START_LED_FLASH( ) and PNIO_CBF_STOP_LED_FLASH( ) alternate cyclically every 3 seconds as long as the module is required to identify itself.

When PNIO_CBF_START_LED_FLASH( ) arrives, the LED should flash at the frequency specified in the "Frequency" parameter.

Syntax

```
typedef void (* PNIO_CBF_START_LED_FLASH)(
    PNIO_UINT32    DevHndl,                  //in
    PNIO_UINT32    Frequency                 //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )
Frequency	Specified frequency for signaling in hertz

Return values

No return values.

7.10.2 PNIO_CBF_STOP_LED_FLASH( ) (deactivate flashing mode for LEDs)

Description

The identification of the IO device by PNIO_CBF_START_LED_FLASH( ) is reset by this callback function.

Syntax

```
typedef void (* PNIO_CBF_STOP_LED_FLASH)(
    PNIO_UINT32   DevHndl                   //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )

Return values

No return values.

7.11 General callback PNIO_CBF_CP_STOP_REQ( ) (remote download and reconfiguration request)

Description

This callback is called by the IO-Base user programming interface as soon as a firmware update or reconfiguration request is received over the network.

On receiving this callback, the user program must send a PNIO_device_close( ). Following this, the user program can once again call PNIO_device_open( ).

The functions PNIO_device_close( ) and PNIO_device_open( ), however, must not execute within the function belonging to the callback event PNIO_CBE_CP_STOP_REQ.

During the download, the PNIO_device_open( ) function returns with the error code "PNIO_ERR_CONFIG_IN_UPDATE" or "PNIO_ERR_NO_FW_COMMUNICATION". Following a successful download, the PNIO_device_open( ) function returns with the "PNIO_OK" error code.

Note

If a user program does not register this callback, no reconfiguration is possible as long as the user program is active.

Syntax

```
typedef void (*PNIO_CBF_CP_STOP_REQ)(
    PNIO_UINT32   DevHndl                  //in
);
```

Parameter

Name	Description
DevHndl	IO device handle from PNIO_device_open( )

Return values

No return values.

7.12 Interface for isochronous real-time mode (IRT)

Overview

This interface consists of the following functions and callback events:

- PNIO_CP_register_cbf( )
- Callback event PNIO_CP_CBE_STARTOP_IND
- Callback event PNIO_CP_CBE_OPFAULT_IND
- PNIO_CP_set_opdone( )

7.12.1 PNIO_CP_register_cbf( ) (register callbacks)

Description

This function registers a callback function.

Note

Callback functions can only be registered by the IO device user program prior to PNIO_device_start( ).

Note

The PNIO_CP_CBE_OPFAULT_IND callback event type must be registered before the PNIO_CP_CBE_STARTOP_IND callback event type.

Syntax

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

Parameter

Name	Description
CpHandle	IO device handle from PNIO_device_open( )
CbeType	Callback event type for which the callback function "Cbf" will be registered; see the section "PNIO_CP_CBE_TYPE (callback event type) (Page 224)".
Cbf	Callback function to be started after arrival of the callback event "CbeType".
	Note The function pointer must not be NULL.

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_INVALID_CONFIG
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_TYPE

- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.12.2 Callback event PNIO_CP_CBE_STARTOP_IND (start of isochronous real-time data processing)

Description

The PNIO_CP_CBE_STARTOP_IND callback event informs your IO-Base user program of the end of the IRT data transfer phase and the transfer of the IRT output data (from the perspective of the IO controller) from the process image to the host memory using DMA. As of this point in time, all IRT output data (from the perspective of the IO controller) is in the host memory. This allows your user program to access consistent IRT data.

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 (callback event parameter) (Page 225)".

Note

The PNIO_CP_CBE_STARTOP_IND callback event is called at the end of each IRT data transfer phase. This occurs even if there is no connection established to the IO controller and there is therefore no IRT data for the IO device.

If functions for IO data access are called due to the PNIO_CP_CBE_STARTOP_IND callback event, although there is no IRT data available, this causes the warning return value PNIO_WARN_NO_SUBMODULES that can be ignored in this case.

Syntax

```
typedef struct
{
    PNIO_UINT32           CpHandle;            //in
    PNIO_CYCLE_INFO       CycleInfo;           //in
} ATTR_PACKED PNIO_CP_CBE_PRM_STARTOP_IND;
```

Parameter

Name	Description
CpHandle	Handle from PNIO_controller_open( ) or PNIO_device_open( )
CycleInfo	Information on the current cycle

Note

During the PNIO_CP_CBE_STARTOP_IND event, you must not call any functions that might endanger the real-time capability of your user program. 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 your situation.

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

7.12.3 Callback event PNIO_CP_CBE_OPFAULT_IND (violation of isochronous real-time mode)

Description

The PNIO_CP_CBE_OPFAULT_IND callback event signals a violation of the isochronous real-time mode to your IO-Base user program. This means that your user program called PNIO_CP_set_opdone( ) too late after receiving the PNIO_CP_CBE_STARTOP_IND callback event.

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 (callback event parameter) (Page 225)".

Syntax

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

Parameter

Name	Description
CpHandle	Handle from PNIO_controller_open( ) or PNIO_device_open( )
CycleInfo	Information on the current cycle

7.12.4 PNIO_CP_set_opdone( ) (end of processing the isochronous real-time data)

Description

By calling this function, the IO-Base user program signals the IO-Base interface that it has completed isochronous real-time processing of the IRT IO data. The IO-Base interface then initiates the transfer of the IRT input data (from the perspective of the controller) from the host memory to the process image using DMA.

Syntax

```
PNIO_UINT32  PNIO_CP_set_opdone(
    PNIO_UINT32          CpHandle          //in
    PNIO_CYCLE_INFO     *CycleInfo;        //out
);
```

Parameter

Name	Description
CpHandle	Handle from PNIO_controller_open( ) or PNIO_device_open( )
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

7.13 Interface for signaling the start of a new bus cycle

Description

This interface consists of the following callback event:

PNIO_CP_CBE_NEWCYCLE_IND

7.13.1 Callback event PNIO_CP_CBE_NEWCYCLE_IND (new bus cycle)

Description

The PNIO_CP_CBE_NEWCYCLE_IND callback event signals the start of a new bus cycle to your IO-Base 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 (callback event parameter) (Page 225)".

Syntax

```
typedef struct
{
    PNIO_UINT32          CpHandle;             //in
    PNIO_CYCLE_INFO      CycleInfo;            //in
} ATTR_PACKED PNIO_CP_CBE_PRM_NEWCYCLE_IND;
```

Parameter

Name	Description
CpHandle	Handle from PNIO_controller_open( ) or PNIO_device_open( )
CycleInfo	Information on the current cycle

7.14 I&M data records (identification and maintenance)

7.14.1 Overview

Description

If, for example, STEP 7 queries the I&M data of an IO device or an IO module, the read data record callback function of the IO-Base device user program is called. The data record number transferred must be interpreted according to the table in the section "Assigning the data records".

Reading and writing data records

There are I&M data records I&M0 to I&M4.

These are assigned the following data record numbers according to the standard for PROFINET IO.

Data record number	Type	Description in section
0xAFF0	PNIO_IM0_TYPE	"PNIO_IM0_TYPE (I&M0 data record) (Page 227)"
0xAFF1	PNIO_IM1_TYPE	"PNIO_IM1_TYPE (I&M1 data record) (Page 229)"
0xAFF2	PNIO_IM2_TYPE	"PNIO_IM2_TYPE (I&M2 data record) (Page 230)"
0xAFF3	PNIO_IM3_TYPE	"PNIO_IM3_TYPE (I&M3 data record) (Page 231)"
0xAFF4	PNIO_IM4_TYPE	"PNIO_IM4_TYPE (I&M4 data record) (Page 232)"

The IO device user program delivers a corresponding I&M data record by writing the data to "pBuffer" of the PNIO_CBF_REC_READ callback.

The IO device user program takes an I&M data record from "pBuffer" of the PNIO_CBF_REC_WRITE callback by copying the data to the corresponding I&M records.

Further information on I&M data records

You will find more information in the documentation provided by the PROFIBUS User Organization e. V. (http://www.profibus.com):

- Profile Guidelines Part 1: Identification & Maintenance Functions. Version 1.1.1. March 2005. Order No: 3.502
- Test Specifications for I&M Functions", Version 1.0.0, June 2005.
- GSDML Specification for PROFINET IO, Version 2.10, August 2006, Order No: 2.352
- IEC 61158

7.14.2 Instructions for I&M writes

Description

In the IO controller user program, you execute a write data record job on one of the I&M data records as follows:

Step	Description
1	Convert the "pBuffer" parameter from the PNIO_CBF_REC_READ( ) callback function to one of the data types from the table "Converting data records" in the section "Overview (Page 204)".
2	Save the following elements retentively, for example in a file: • IM_Tag_Function (section "PNIO_IM1_TYPE (I&M1 data record) (Page 229)") • IM_Tag_Location (section "PNIO_IM1_TYPE (I&M1 data record) (Page 229)") • IM_Date (section "PNIO_IM2_TYPE (I&M2 data record) (Page 230)") • IM_Descriptor (section "PNIO_IM3_TYPE (I&M3 data record) (Page 231)") • IM_Signature (section "PNIO_IM4_TYPE (I&M4 data record) (Page 232)")
3	Increment the "IM_Revision_Counter" element. After an overflow, "IM_Revision_Counter" does not start to count at 0x0000, but at 0x0001.

Note

Write data record jobs for data record "I&M0" are always rejected. The "I&M0" data record does not contain any elements that can be written.

Error code

The following table shows the error code elements that are always returned by the PNIO_CBF_REC_READ( ) callback function.

If no error occurs, the values are "0", if an error occurs, the result is as described in the table below:

Name	Value in the event of an error
ErrorCode	= 0xDF (IODWriteRes)
ErrorDecode	= 0x80 (PNIORW)
ErrorCode1	Is filled in according to the table in the standard for PROFINET IO "Coding of ErrorCode1 with ErrorDecode PNIORW".
ErrorCode2	Is filled in user-specific according to the standard for PROFINET IO.

7.14.3 Instructions for I&M reads

Description

A read data record job is handled as follows:

Step	Description
1	Convert the "pBuffer" parameter from the PNIO_CBF_REC_READ( ) callback function to one of the data types from the table "Converting data records" in the section "Overview (Page 204)".
2	Set the I&M parameters in the data buffer according to the descriptions in the sections "PNIO_IM0_TYPE (I&M0 data record) (Page 227)" to "PNIO_IM1_TYPE (I&M1 data record) (Page 229)". The following elements are read from retentive memory: • IM_Tag_Function • IM_Tag_Location • IM_Date • IM_Descriptor • IM_Signature As default, the elements have the value zero and are set by an engineering station.

Note

All elements of the I&M data records with a data length of two or four bytes have the big endian format.

Note

The entry "Writeable_IM_Records" must be included in the GSDML file for each submodule that supports writeable I&M data records.

Note

Engineering station jobs for reading the information data record "I&M0FilterData" with data record number 0xF840 are handled internally by the PROFINET IO functions and not passed on to the PROFINET IO device user program.

Error code elements

The following table describes the error code elements and their possible values without and with errors from the PNIO_CBF_REC_WRITE( ) callback function, "pPnioState" parameter (see also the section "Callback function PNIO_CBF_REC_WRITE( ) (process write data record job) (Page 168)"):

Name	Value if no error occurs	Value if error occurs
ErrorCode	= 0	= 0xDF (IODWriteRes)
ErrorDecode	= 0	= 0x80 (PNIORW)
ErrorCode1	= 0	Is filled in according to the table in the standard for PROFINET IO "Coding of ErrorCode1 with ErrorDecode PNIORW".
ErrorCode2	= 0	Is filled in user-specific according to the standard for PROFINET IO.

7.15 Data types

Description

The following data types are used in the IO-Base user programming interface:

- PNIO_ANNOTATION (version ID structure) (Page 209)
- PNIO_APPL_READY_LIST_TYPE (application ready) (Page 210)
- PNIO_AR_REASON (reason for connection abort) (Page 213)
- PNIO_AR_TYPE (application relation) (Page 216)
- PNIO_CFB_FUNCTIONS (callback registration structure) (Page 216)
- PNIO_DEV_ADDR (IO device address type) (Page 217)
- PNIO_APDU_STATUS_IND (IO controller status) (Page 218)
- PNIO_IOCR_TYPE (application relation) (Page 219)
- PNIO_IOCS_TYPE (application relation) (Page 221)
- PNIO_MODULE_TYPE (application relation) (Page 221)
- PNIO_SUBMOD_TYPE (application relation) (Page 222)
- PNIO_ACCESS_ENUM (type of access) (Page 223)
- PNIO_CP_CBE_TYPE (callback event type) (Page 224)
- PNIO_CP_CBE_PRM (callback event parameter) (Page 225)
- PNIO_CP_CBF (general PNIO callback function) (Page 225)

7.15.1 PNIO_ANNOTATION (version ID structure)

Description

The version ID must be transferred with the PNIO_device_open( ) function call. This contains the IO device type, the order number, the hardware and software version.

The parameters from the PNIO_ANNOTATION structure are displayed in online diagnostics of the "HW Config" tool and allow the user (maintenance personnel) to identify the module type and the version of the device (or version of the control application running on the device).

The order number ("orderId") can be useful when ordering replacement devices.

The user application should initialize this parameter with a useful value since this information is important for maintenance of the plant and for device identification.

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 PNIO_ANNOTATION;
```

Elements

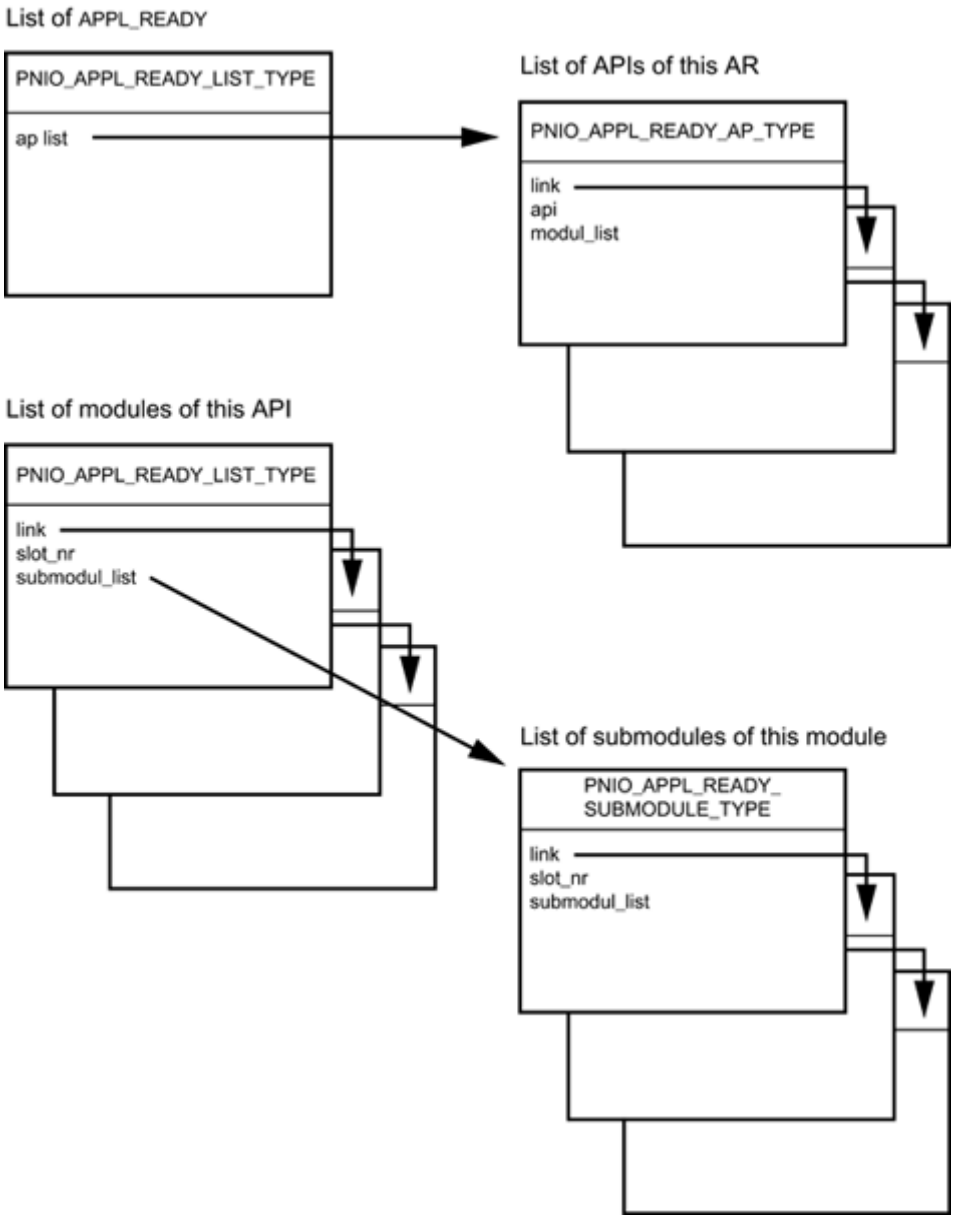
Name	Description
deviceType	Identifier for the IO device - Corresponds to the "ProductFamily" from the GSDML file.
orderId	Order number for the IO device - Corresponds to the "OrderNumber Value" from the GSDML file.
hwRevision	Version number of the hardware - The "hwRevision" corresponds to the "HardwareRelease Value" from the GSDML file.
SwRevisionPrefix	Prefix for the software version - The "swRevisionPrefix" corresponds to the prefix character in the "SoftwareRelease Value" from the GSDML file. **Example** From the line `<SoftwareRelease Value="Z1.2.3" />` in the GSDML file, it follows that "swRevisionPrefix" is Z.

Name	Description
SwRevision1	First numeric value of the software version – "swRevision1" corresponds to the first numeric value in "SoftwareRelease Value" from the GSDML file. **Example** From the line `<SoftwareRelease Value="Z1.2.3" />` in the GSDML file, it follows that "swRevision1" is 1.
SwRevision2	Second numeric value of the software version – "swRevision2" corresponds to the second numeric value in "SoftwareRelease Value" from the GSDML file. **Example** From the line `<SoftwareRelease Value="Z1.2.3" />` in the GSDML file, it follows that "swRevision2" is 2.
SwRevision3	Third numeric value of the software version – "swRevision3" corresponds to the third numeric value in "SoftwareRelease Value" from the GSDML file. **Example** From the line `<SoftwareRelease Value="Z1.2.3" />` in the GSDML file, it follows that "swRevision3" is 3.

7.15.2 PNIO_APPL_READY_LIST_TYPE (application ready)

Description

Hierarchically structured list (AP/module/submodule, no duplicates) with submodules. In this list, the IO-Base device user program must enter all modules/submodules that have not reached application ready. The hierarchical structure of this list is illustrated in the following figure.


List of APPL_READY
PNIO_APPL_READY_LIST_TYPE
ap list
List of APIs of this AR
PNIO_APPL_READY_AP_TYPE
link
api
modul_list
List of modules of this API
PNIO_APPL_READY_LIST_TYPE
link
slot_nr
submodul_list
List of submodules of this module
PNIO_APPL_READY_
SUBMODULE_TYPE
link
slot_nr
submodul_list

Syntax

```
typedef struct pnio_appl_ready_tag
{
   PNIO_LIST_ENTRY_TYPE    ap_list;
} PNIO_APPL_READY_LIST_TYPE;

typedef struct pnio_appl_ready_ap_tag
{
   PNIO_LIST_ENTRY_TYPE    link;
   PNIO_UINT32             api;

/* with list from type PNIO_APPL_READY_MODULE_TYPE  */
   PNIO_LIST_ENTRY_TYPE    module_list;
} PNIO_APPL_READY_AP_TYPE;

typedef struct pnio_appl_ready_module_tag
{
   PNIO_LIST_ENTRY_TYPE    link;
   PNIO_UINT16             slot_nr;

/*with list from type PNIO_APPL_READY_SUBMODULE_TYPE*/
   PNIO_LIST_ENTRY_TYPE    submodule_list;
} PNIO_APPL_READY_MODULE_TYPE;

typedef struct pnio_appl_ready_submodule_tag
{
   PNIO_LIST_ENTRY_TYPE    link;
   PNIO_UINT16             subslot_nr;
} PNIO_APPL_READY_SUBMODULE_TYPE;

/* device control */
typedef struct pnio_list_entry_tag
{
   struct pnio_list_entry_tag   *Flink; /* forward link */
   struct pnio_list_entry_tag   *Blink; /* backward link */
} PNIO_LIST_ENTRY_TYPE;
```

Elements

Name	Description
ap_list	List of supported APIs of this application relation
Api	Supported API of this list entry
module_list	List of all modules of this API
slot_nr	Module number of this list entry
submodul_list	List of all submodules of this module

Name	Description
submodul_nr	Submodule number of this list entry
link	For each list entry there is a link element that references the next list element. The list is at the end when the reference to the next list element is NULL.

7.15.3 PNIO_AR_REASON (reason for connection abort)

Description

List of the possible connection abort conditions transferred to the IO-Base device user program with the PNIO_CBF_AR_ABORT_IND( ) callback function.

Syntax

```
typedef enum
{
    PNIO_AR_REASON_NONE,
    PNIO_AR_REASON_RESERVED1,
    PNIO_AR_REASON_RESERVED2,
    PNIO_AR_REASON_MEM,
    PNIO_AR_REASON_FRAME,
    PNIO_AR_REASON_MISS,
    PNIO_AR_REASON_TIMER,
    PNIO_AR_REASON_ALARM,
    PNIO_AR_REASON_ALSND,
    PNIO_AR_REASON_ALACK,
    PNIO_AR_REASON_ALLEN,
    PNIO_AR_REASON_ASRT,
    PNIO_AR_REASON_RPC,
    PNIO_AR_REASON_ABORT,
    PNIO_AR_REASON_RERUN,
    PNIO_AR_REASON_REL,
    PNIO_AR_REASON_PAS,
    PNIO_AR_REASON_RMV,
    PNIO_AR_REASON_PROT,
    PNIO_AR_REASON_NARE,
    PNIO_AR_REASON_BIND,
    PNIO_AR_REASON_CONNECT,
    PNIO_AR_REASON_READ,
    PNIO_AR_REASON_WRITE,
    PNIO_AR_REASON_CONTROL,
    PNIO_AR_REASON_PULLPLUG,
    PNIO_AR_REASON_AP_RMV,
    PNIO_AR_REASON_LNK_DWN,
    PNIO_AR_REASON_MMAC,
    PNIO_AR_REASON_SYNC,
    PNIO_AR_REASON_TOPO,
    PNIO_AR_REASON_DCP_NAME,
    PNIO_AR_REASON_DCP_RESET,
    PNIO_AR_REASON_PRM,
    PNIO_AR_REASON_IRDATA,
    PNIO_AR_REASON_MAX,
}PNIO_AR_REASON;
```

Elements

Name	Description from the PROFINET IO standard
PNIO_AR_REASON_NONE	none
PNIO_AR_REASON_RESERVED1	internal use
PNIO_AR_REASON_RESERVED2	internal use
PNIO_AR_REASON_MEM	out of memory
PNIO_AR_REASON_FRAME	Add provider or consumer failed

Name	Description from the PROFINET IO standard
PNIO_AR_REASON_MISS	consumer miss
PNIO_AR_REASON_TIMER	cmi timeout
PNIO_AR_REASON_ALARM	alarm-open failed
PNIO_AR_REASON_ALSND	alarm-send
PNIO_AR_REASON_ALACK	alarm-ack-send
PNIO_AR_REASON_ALLEN	alarm-data too long
PNIO_AR_REASON_ASRT	alarm indication err
PNIO_AR_REASON_RPC	rpc-client call
PNIO_AR_REASON_ABORT	ar-abort
PNIO_AR_REASON_RERUN	re-run abort existing
PNIO_AR_REASON_REL	got release
PNIO_AR_REASON_PAS	device passivated
PNIO_AR_REASON_RMV	device/ar removed
PNIO_AR_REASON_PROT	protocol violation
PNIO_AR_REASON_NARE	NARE error
PNIO_AR_REASON_BIND	rpc-bind error
PNIO_AR_REASON_CONNECT	rpc-connect error
PNIO_AR_REASON_READ	rpc-read error
PNIO_AR_REASON_WRITE	rpc-write error
PNIO_AR_REASON_CONTROL	rpc-control error
PNIO_AR_REASON_PULLPLUG	forbidden pull or plug after check.rsp and before in-data.ind
PNIO_AR_REASON_AP_RMV	deprecated! AP removed
PNIO_AR_REASON_LNK_DWN	link "down"
PNIO_AR_REASON_MMAC	could not register multicast-mac
PNIO_AR_REASON_SYNC	not synchronized (cannot start companion-ar)
PNIO_AR_REASON_TOPO	wrong topology (cannot start companion-ar)
PNIO_AR_REASON_DCP_NAME	dcp, station-name changed
PNIO_AR_REASON_DCP_RESET	dcp, reset to factory-settings
PNIO_AR_REASON_PRM	deprecated! cannot start companion-AR because a 0x8ipp submodule in the first AR • has appl-ready-pending (erroneous parameterization) • is locked (no parameterization) • is wrong or pulled (no parameterization)
PNIO_AR_REASON_IRDATA	no irdata record yet
PNIO_AR_REASON_MAX	maximum value

7.15.4 PNIO_AR_TYPE (application relation)

Description

The structure is sent with the PNIO_CBF_AR_INFO_IND( ) callback to inform the user program which modules and submodules the IO controller wants to to control.

This structure is sent with the PNIO_CBF_AR_CHECK_IND( ) callback to inform the user program of the properties of the application relation (the pointer "pModList" is always NULL and "NumOfMod=0"! This means that no module list is transferred).

Syntax

```
typedef struct
{
    void                *pReserved;
    PNIO_UINT16          ArNumber;
    PNIO_UINT16          SessionKey;
    PNIO_UINT32          NumOfIocr;
    PNIO_IOCR_TYPE      *pIoCrList;
    PNIO_UINT32          NumOfMod;
    PNIO_MODULE_TYPE    *pModList;
} ATTR_PACKED PNIO_AR_TYPE;
```

Elements

Name	Description
ArNumber	Application relation number
SessionKey	Session key for the current application relation
NumOfIocr	Number of IO-CR entries in "pIoCrList"
pIoCrList	Pointer to IO-CR list
NumOfMod	Number of modules in "pModList"
pModList	Pointer to module list
pReserved	Reserved for future expansion

7.15.5 PNIO_CFB_FUNCTIONS (callback registration structure)

Description

The structure is used with PNIO_device_open( ) to register the callbacks.

Syntax

```
typedef struct
{
    PNIO_UINT32    size          ;/* Size of this structure */
    PNIO_CBF_DATA_WRITE           cbf_data_write;
    PNIO_CBF_DATA_READ            cbf_data_read;
    PNIO_CBF_REC_READ             cbf_rec_read;
    PNIO_CBF_REC_WRITE            cbf_rec_write;
    PNIO_CBF_REQ_DONE             cbf_alarm_done;
    PNIO_CBF_CHECK_IND            cbf_check_ind;
    PNIO_CBF_AR_CHECK_IND         cbf_ar_check_ind;
    PNIO_CBF_AR_INFO_IND          cbf_ar_info_ind;
    PNIO_CBF_AR_INDATA_IND        cbf_ar_indata_ind;
    PNIO_CBF_AR_ABORT_IND         cbf_ar_abort_ind;
    PNIO_CBF_AR_OFFLINE_IND       cbf_ar_offline_ind;
    PNIO_CBF_APDU_STATUS_IND      cbf_apdu_status_ind;
    PNIO_CBF_PRM_END_IND          cbf_prm_end_ind;
    PNIO_CBF_CP_STOP_REQ          pnio_cbf_cp_stop_req;
    PNIO_CBF_DEVICE_STOPPED       pnio_cbf_device_stopped;
    PNIO_CBF_START_LED_FLASH      cbf_start_led_flash;
    PNIO_CBF_STOP_LED_FLASH       cbf_stop_led_flash;
    PNIO_CBF_PULL_PLUG_CONF       cbf_pull_plug_conf;
} ATTR_PACKED PNIO_CFB_FUNCTIONS;
```

Elements

Name	Description
size	Size of the structure - always "=sizeof(PNIO_CFB_FUNCTIONS)". Used to version this structure.
Remaining fields	Pointers to the corresponding callback functions All callbacks must be specified in the PNIO_CFB_FUNCTIONS structure except: • PNIO_CBF_START_LED_FLASH( ) • PNIO_CBF_STOP_LED_FLASH( ) • PNIO_CBF_PULL_PLUG_CONF( ) PNIO_CBF_PULL_PLUG_CNF( ) must be registered if you want removal and insertion of modules/submodules during productive operation to be supported.

7.15.6 PNIO_DEV_ADDR (IO device address type)

Description

The PNIO_DEV_ADDR address structure is used for addressing with all IO-Base interface functions described here.

Syntax

```
typedef struct
{
    PNIO_ADDR_TYPE            AddrType;
    PNIO_IO_TYPE              IODataType;
    union
    {
        PNIO_UINT32     RESERVED;
        struct
        {
            PNIO_UINT32 RESERVED1[2];
            PNIO_UINT32 Slot;
            PNIO_UINT32 Subslot;
            PNIO_UINT32 RESERVED2[1];
        } Geo;
    }u;
} ATTR_PACKED PNIO_DEV_ADDR;
```

Elements

Name	Description
AddrType	For the IO device must always be PNIO_ADDR_GEO!
IODataType	Specifies whether input or output type. Possible values: • PNIO_IO_IN for input data • PNIO_IO_OUT for output data
addr.Geo.Slot	Slot number (geographical address)
addr.Geo.Subslot	Subslot number (geographical address)
Reserved	Reserved for future expansion
Geo. RESERVED1 [2]	Reserved for future expansion
Geo. RESERVED2	Reserved for future expansion

7.15.7 PNIO_APDU_STATUS_IND (IO controller status)

Description

Description of the IO controller status contained in the APDU.

Syntax

```
typedef enum
{
    PNIO_EVENT_APDU_STATUS_STOP,
    PNIO_EVENT_APDU_STATUS_RUN,
    PNIO_EVENT_APDU_STATUS_STATION_OK,
    PNIO_EVENT_APDU_STATUS_STATION_PROBLEM,
    PNIO_EVENT_APDU_STATUS_PRIMARY,
    PNIO_EVENT_APDU_STATUS_BACKUP
} PNIO_APDU_STATUS_IND;
```

Elements

Name	Description
PNIO_EVENT_APDU_STATUS_STOP	IO controller in STOP
PNIO_EVENT_APDU_STATUS_RUN	IO controller in RUN
PNIO_EVENT_APDU_STATUS_STATION_OK	Station O. K.
PNIO_EVENT_APDU_STATUS_STATION_PROBLEM	Station not O. K.
PNIO_EVENT_APDU_STATUS_PRIMARY	Status Primary
PNIO_EVENT_APDU_STATUS_BACKUP	Status Backup

7.15.8 PNIO_IOCR_TYPE (application relation)

Description

This structure is used by PNIO_AR_TYPE and provides information on the type of input/output data

Syntax

```
typedef struct
{
    PNIO_UINT32          CycleTime;
    PNIO_UINT32          Direction;
    PNIO_UINT32          IOCRProperties;
    PNIO_UINT32          SendClock;
    PNIO_UINT32          ReductioFactor;
    PNIO_UINT32          Phase;
    PNIO_UINT32          NumOfIoCs;
    PNIO_IOCS_TYPE      *pIoCsList;
    PNIO_UINT32          reserved[3];
} ATTR_PACKED PNIO_IOCR_TYPE;
```

Elements

Name	Description
CycleTime	Cycle time in microseconds for the IO connection relation - According to the standard, this corresponds to the formula: SendClock x 31.25 µs x ReductioFactor
Direction	See PNIO_IOCR_TYPE_ENUM
IOCRProperties	See PNIO_IOCR_PROP_ENUM
SendClock	Send cycle duration divided by time base (31.25 µs).
ReductioFactor	Reduction factor
Phase	Phase, in which the data of the submodule is transferred for this IOCR.
NumOfIoCs	Number of entries in "pIOCsList"
pIoCsList	List of submodules that belong to this IOCR.
Reserved	Reserved for future expansion.

PNIO_IOCR_TYPE_ENUM elements

Name	Description
PNIO_IOCR_TYPE_RESERVED_0	Reserved
PNIO_IOCR_TYPE_INPUT	INPUT FRAME (from IO controller perspective)
PNIO_IOCR_TYPE_OUTPUT	OUTPUT FRAME (from IO controller perspective)
PNIO_IOCR_TYPE_MULTICAST_ PROVIDER	Output data of this submodule is distributed as multicast provider connection relations (INPUT FRAME).
PNIO_IOCR_TYPE_MULTICAST_ CONSUMER	Input data of this submodule is received as multicast consumer connection relations (OUTPUT FRAME).

PNIO_IOCR_PROP_ENUM elements

Name	Description
PNIO_IOCR_PROP_RT_CLASS_MASK	Mask element
PNIO_IOCR_PROP_RT_CLASS_1	Real-time frame
PNIO_IOCR_PROP_RT_CLASS_2	Real-time frame
PNIO_IOCR_PROP_RT_CLASS_3	Isochronous real-time frame
PNIO_IOCR_PROP_RT_CLASS_1_UDP	RT over UDP

Note

In the user program, make sure that unknown codes for the call of PNIO_device_ar_abort( ) lead to a connection abort.

7.15.9 PNIO_IOCS_TYPE (application relation)

Description

This structure is used by PNIO_IOCR_TYPE and provides information about which submodule belongs to IOCR.

Syntax

```
typedef struct
{
    PNIO_UINT32          SlotNum;
    PNIO_UINT32          SubslotNum;
    PNIO_UINT32          reserved;
} ATTR_PACKED PNIO_IOCS_TYPE;
```

Elements

Name	Description
SlotNum	Module number
SubslotNum	Submodule number
Reserved	Reserved for future expansion

7.15.10 PNIO_MODULE_TYPE (application relation)

Description

Structure is used by PNIO_AR_TYPE. Describes the configuration of a module.

Syntax

```
typedef struct
{
    PNIO_UINT32          API;
    PNIO_UINT32          SlotNum;
    PNIO_UINT32          NumOfSubmod;
    PNIO_UINT32          ModProperties;
    PNIO_UINT32          ModIdent;
    PNIO_SUBMOD_TYPE    *pSubList;
} ATTR_PACKED PNIO_MODULE_TYPE;
```

Elements

Name	Description
API	Application process identifier for specified module
SlotNum	Slot number of the module
NumOfSubmod	Number of submodules in "pSubList"
ModProperties	Reserved - must always be 0!
ModIdent	Module identification of the specified module - corresponds to the "ModulI-dentNumber" from the GSDML file.
pSubList	Pointer to submodule list

7.15.11 PNIO_SUBMOD_TYPE (application relation)

Description

Structure is used by PNIO_MODULE_TYPE. Describes the configuration of a submodule.

Syntax

```
typedef struct
{
    PNIO_UINT32         SlotNum;
    PNIO_UINT32         SubSlotNum;
    PNIO_UINT32         SubMProperties;
    PNIO_UINT32         SubMIdent;
    PNIO_UINT32         InDatLen;
    PNIO_UINT32         InIopsLen;
    PNIO_UINT32         InIocsLen;
    PNIO_UINT32         OutDatLen;
    PNIO_UINT32         OutIopsLen;
    PNIO_UINT32         OutIocsLen;
    PNIO_UINT32         reserved[8];
} ATTR_PACKED PNIO_SUBMOD_TYPE;
```

Elements

Name	Description
SlotNum	Module number to which the submodule belongs.
SubSlotNum	Subslot number of the submodule
SubMProperties	Properties of the submodule – see PNIO_SUB_PROPERTIES_ENUM.
SubMIdent	Submodule identification from the GSDML file
InDatLen	Length of the input data to the IO controller for this submodule
InIopsLen	Length of the local status for the input data
InIocsLen	Length of the remote status for the input data

Name	Description
OutDatLen	Length of the output data from the IO controller for this submodule
OutIopsLen	Length of the local status for the output data (from the perspective of the IO controller)
OutIocsLen	Length of the remote status for the output data (from the perspective of the IO controller)
Reserved	Reserved for later expansions

PNIO_SUB_PROPERTIES_ENUM elements

Name	Description
PNIO_SUB_PROP_ TYPE_MASK	Mask for selecting the following IO data types.
	Note This mask is ANDed with the "SubMProperties" field to allow it to be compared with the following constants. This is necessary since the remaining bits are marked as reserved according to the standard in "SubMProperties".
PNIO_SUB_PROP_ TYPE_NO_DATA	Module without data
PNIO_SUB_PROP_ TYPE_INPUT	Module with input data
PNIO_SUB_PROP_ TYPE_OUTPUT	Module with output data
PNIO_SUB_PROP_ TYPE_INPUT_OUTPUT	Module with input and output data

Note

In the user program, make sure that unknown constants for the call of the PNIO_device_ar_abort( ) callback lead to a connection abort.

7.15.12 PNIO_ACCESS_ENUM (type of access)

Description

This enumeration type lists the possible types of access.

The values are supplied as parameters with the PNIO_initiate_data_read_ext( ) and PNIO_initiate_data_write_ext( ) functions. They restrict the submodules and the type of access for which the PNIO_DATA_READ_CBF( ) and PNIO_DATA_WRITE_CBF( ) callbacks are called.

Syntax

```
typedef enum
{
   PNIO_ACCESS_ALL_WITHOUT_LOCK,
   PNIO_ACCESS_RT_WITH_LOCK,
   PNIO_ACCESS_RT_WITHOUT_LOCK,
   PNIO_ACCESS_IRT_WITHOUT_LOCK
} PNIO_ACCESS_ENUM;
```

Elements

Name	Description
PNIO_ACCESS_ALL_WITHOUT_LOCK	Both RT and IRT data will be accessed. Consistency of the RT data is not guaranteed. Consistency of the IRT data only if access is isochronous real-time access.
PNIO_ACCESS_RT_WITH_LOCK	Only RT data will be accessed. Consistency of the data is guaranteed.
PNIO_ACCESS_RT_WITHOUT_LOCK	Only RT data will be accessed. Consistency of the data is not guaranteed.
PNIO_ACCESS_IRT_WITHOUT_LOCK	Only IRT data will be accessed. Consistency of the IRT data is only guaranteed if access is isochronous real-time access.

7.15.13 PNIO_CP_CBE_TYPE (callback event type)

Description

The IO-Base user programming interface recognizes the following callback event types:

Callback event type	Callback event
PNIO_CP_CBE_STARTOP_IND	Start of isochronous real-time data processing
PNIO_CP_CBE_OPFAULT_IND	Violation of isochronous real-time mode
PNIO_CP_CBE_NEWCYCLE_IND	Start of a new bus cycle

These callback events can only be registered by the IO controller or the IO device.

If you want to implement an IO controller and IO device at the same time, you will need to implement both devices in a common IO-Base user program.

7.15.14 PNIO_CP_CBE_PRM (callback event parameter)

Description

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

Syntax

```
typedef struct
{
    PNIO_CP_CBE_TYPE    CbeType;            //in
    PNIO_UINT32         Handle;             //in
    union
    {
        ...       /*Union over the various
        ...         callback event parameters
                    (details in relevant sections)*/
    };
} ATTR_PACKED PNIO_CP_CBE_PRM;
```

Parameter

Name	Description
CbeType	Callback event type
Handle	Handle from PNIO_controller_open( ) or from PNIO_device_open( )

7.15.15 PNIO_CP_CBF (general PNIO callback function)

Description

Using the callback event parameter PNIO_CP_CBE_PRM, a general PNIO callback function of the type PNIO_CP_CBF is declared.

Syntax

```
typedef void (* PNIO_CP_CBF)(
    PNIO_CP_CBE_PRM     *pCbfPrm
);
```

7.15.16 PNIO_CYCLE_INFO (information on current cycle)

Description

The structure contains information on the current cycle and allows the following to be identified:

- Highly accurate counter ("ClockCount”) with a resolution of 10 ns
- Loss of interrupts ("CycleCount" does not change by the same value in every cycle)
- "CountSinceCycleStart" returns the time between the start of the cycle and the call for the function that returns the PNIO_CYCLE_INFO structure.
- Time between the PNIO_CP_CBE_STARTOP_IND callback event and the PNIO_CP_set_opdone( ) function call (difference between the input value "CountSinceCycleStart” of the PNIO_CP_CBE_STARTOP_IND callback event and the return value of the PNIO_CP_set_opdone( ) function)

It is used as the input parameter for the callback events:

- PNIO_CP_CBE_STARTOP_IND
- PNIO_CP_CBE_OPFAULT_IND
- PNIO_CP_CBE_NEWCYCLE_IND

The structure serves as a return value for the function PNIO_CP_set_opdone( ).

Syntax

```
typedef struct
{
    PNIO_UINT32     ClockCount;
    PNIO_UINT32     CycleCount;
    PNIO_UINT32     CountSinceCycleStart;
} ATTR_PACKED PNIO_CYCLE_INFO;
```

Parameter

Name	Description
ClockCount	Free-running hardware counter on the CP with a resolution of 10 ns (32-bits wide).
CycleCount	The "CycleCount" value is incremented in each cycle by the value corre-sponding to the cycle duration based on 31.25 µs. The value is 16 bits wide. **Example** With a cycle of 1 ms, the value is incremented each time by 32.
CountSinceCycleStart	The "CountSinceCycleStart" value specifies the time since the start of the last cycle based on 10 ns. The value is 32 bits wide.

7.15.17 PNIO_BLOCK_HEADER

Description

The PNIO_BLOCK_HEADER structure is part of the I&M data record structures described below:

- PNIO_IM0_Type
- PNIO_IM1_TYPE
- PNIO_IM2_TYPE
- PNIO_IM3_TYPE
- PNIO_IM4_TYPE

Syntax

```
typedef struct
{
    PNIO_UINT16     BlockType;
    PNIO_UINT16     BlockLength;
    PNIO_UINT8      BlockVersionHigh;
    PNIO_UINT8      BlockVersionLow;
} ATTR_PACKED PNIO_BLOCK_HEADER;
```

Elements

The values of the elements of PNIO_BLOCK_HEADER depend on the I&M data record structures. You will find these values in the descriptions of the I&M data records below (PNIO_IM0 to PNIO_IM4).

7.15.18 PNIO_IM0_TYPE (I&M0 data record)

Description

The PNIO_IM0_TYPE structure is used to inform an IO controller of general information about the module or device.

Syntax

```
typedef struct
{
    PNIO_BLOCK_HEADER   BlockHeader;
    PNIO_UINT8          VendorIDHigh;
    PNIO_UINT8          VendorIDLow;
    PNIO_UINT8          OrderID[20];
    PNIO_UINT8          IM_Serial_Number[16];
    PNIO_UINT16         IM_Hardware_Revision (Big Endian);
    PNIO_UINT8          IM_Software_Revision[4];
    PNIO_UINT16         IM_Revision_Counter (Big Endian);
    PNIO_UINT16         IM_Profile_ID (Big Endian);
    PNIO_UINT16         IM_Profile_Specific_Type (Big Endian);
    PNIO_UINT8          IM_Version_Major;
    PNIO_UINT8          IM_Version_Minor;
    PNIO_UINT16         IM_Supported (Big Endian);
} ATTR_PACKED PNIO_IM0_TYPE;
```

Parameters of "BlockHeader"

Name	Value
BlockType	0x0020
BlockLength	0x0038
BlockVersionHigh	0x01
BlockVersionLow	0x00

The PNIO_BLOCK_HEADER structure is described in the section "PNIO_BLOCK_HEADER (Page 227)".

Remaining elements

Name	Description	Value
VendorIDHigh	First byte of the vendor ID	**Example** 0x00 (Siemens AG)
VendorIDLow	Second byte of the vendor ID	**Example** 0x2A (Siemens AG)
OrderId	Serial number filled with blanks (to be assigned by the vendor).	**Example** 6GK1 161-6AA00
IM_Serial_Number	Serial number filled with blanks (to be assigned by the vendor).	**Example** 000-000-000-0001
IM_Hardware_Revision	Hardware revision of the device (to be assigned by the vendor).	**Example** 0x01

Name	Description	Value
IM_Software_Revision	Software revision (to be assigned by the vendor) - Consists of: • SwRevisionPrefix • IM_SWRevision_ Functional_Enhancement • IM_SWRevision_Bug_Fix • IM_SWRevision_Internal_Change	**Example** 'V', 1, 0, 0
IM_Revision_Counter	Revision counter - Increment this after every hardware change or a reassignment of parameters of the IO device.	• 1: Factory settings • 2 to 0xFFFF: Following reassignment of parameters
IM_Profile_ID	Profile ID - Corresponds to the definition in: (http://www.profibus.com/IM/Profile_ID_Table.xml)	**Example** Example for a generic PROFINET IO device: 0xF600
IM_Profile_Specific_Type	Profile-specific device type - according to : (http://www.profibus.com/IM/Profile_specific_type_table_6102.xml)	**Example** for a communications module: 0x0004
IM_Version_Major	I&M version major	0x01
IM_Version_Minor	I&M version minor	0x01
IM_Supported	Bit 0 always has the value 0. IM0 is always supported. Bits 1 to 4 specify which I&M data record is supported. Bit 1 = 1 - IM1 is supported. Bit 2 = 1 - IM2 is supported. Bit 3 = 1 - IM3 is supported. Bit 4 = 1 - IM4 is supported.	**Example 1** 0x0002 means: IM0 and IM1 are supported. **Example 2** 0x0006 means: IM0, IM1 and IM2 are supported.

7.15.19 PNIO_IM1_TYPE (I&M1 data record)

Description

The PNIO_IM1_TYPE structure is used to inform an IO controller of the function and location of the module or device.

Syntax

```
typedef struct
{
    PNIO_BLOCK_HEADER   BlockHeader;    /* BlockType: 0x0021 */
    PNIO_UINT8          IM_Tag_Function[32];
    PNIO_UINT8          IM_Tag_Location[22];
} ATTR_PACKED PNIO_IM1_TYPE;
```

Parameters of "BlockHeader"

Name	Value
BlockType	0x0021
BlockLength	0x0038
BlockVersionHigh	0x01
BlockVersionLow	0x00

The PNIO_BLOCK_HEADER structure is described in the section "PNIO_BLOCK_HEADER (Page 227)".

Remaining elements

Name	Description	Value
IM_Tag_Function	Description of the function or purpose of the module or device.	**Example** [1] "Pressure sensor 7 "
IM_Tag_Location	Description of the location.	**Example** [1] "Production plant 3 "

[1] The string is padded to the maximum length.

7.15.20 PNIO_IM2_TYPE (I&M2 data record)

Description

The PNIO_IM2_TYPE structure is used to inform an IO controller of the installation date of the module or device.

Syntax

```
typedef struct
{
    PNIO_BLOCK_HEADER   BlockHeader;
    PNIO_UINT8          IM_Date[16];
} ATTR_PACKED PNIO_IM2_TYPE;
```

Parameters of "BlockHeader"

Name	Value
BlockType	0x0022
BlockLength	0x0038
BlockVersionHigh	0x01
BlockVersionLow	0x00

The PNIO_BLOCK_HEADER structure is described in the section "PNIO_BLOCK_HEADER (Page 227)".

"IM_Date" element

Description	Value
Installation date in the format: YYYY-MM-DD hh:mm	Example "2008-05-04 10:15" The string is padded to the maximum length.

7.15.21 PNIO_IM3_TYPE (I&M3 data record)

Description

The PNIO_IM3_TYPE structure is used to inform an IO controller of individual additional information about the module or device.

Syntax

```
typedef struct
{
    PNIO_BLOCK_HEADER   BlockHeader;    /* BlockType: 0x0023 */
    PNIO_UINT8          IM_Descriptor[54];
} ATTR_PACKED PNIO_IM3_TYPE;
```

Parameters of "BlockHeader"

Name	Value
BlockType	0x0023
BlockLength	0x0010
BlockVersionHigh	0x01
BlockVersionLow	0x00

The PNIO_BLOCK_HEADER structure is described in the section "PNIO_BLOCK_HEADER (Page 227)".

"IM_Signature" element

Description	Value
Individual additional information about the module or device	Example "Replaced on 2008-05-04 10:15" The string is padded to the maximum length.

7.15.22 PNIO_IM4_TYPE (I&M4 data record)

Description

The PNIO_IM4_TYPE structure is used to inform an IO controller of a security code of the module or device.

Syntax

```
typedef struct
{
    PNIO_BLOCK_HEADER   BlockHeader;    /* BlockType: 0x0024*/
    PNIO_UINT8          IM_Signature[54];
} ATTR_PACKED PNIO_IM4_TYPE;
```

Parameters of "BlockHeader"

Name	Value
BlockType	0x0024
BlockLength	0x0010
BlockVersionHigh	0x01
BlockVersionLow	0x00

The PNIO_BLOCK_HEADER structure is described in the section "PNIO_BLOCK_HEADER (Page 227)".

"IM_Descriptor" element

Contains the security code.

7.15.23 "pData" parameter of the PNIO_diag_alarm_send( ) function

Note

These structures are not defined in the header file. These structures are only intended to show how the "pData" parameter is structured according to IEC 61158-6 PROFINET IO.

Description

"pData" is the pointer to alarm data in network format (big endian).

The alarm data are the diagnostic data that were stored earlier as a packed structure in network format (big endian) on the IO device.

From the following table, you can see how "pData" needs to be structured depending on the function used:

Function	Structure
PNIO_diag_generic_add( )	Structure 1
PNIO_diag_channel_add( )	Structure 2
PNIO_diag_ext_channel_add( )	Structure 3

Note

When the alarm exits the state, transfer the same data in "pData" as was transferred in the entering state alarm except for the value of "ChannelProp".

The PNIO_build_channel_properties( ) function returns the value "ChannelProp".

To form the return value for the "Spec" parameter, use the value 2 if there is no other problem and the value 3 if there are other problems.

The "ChannelProp" value is formed by the PNIO_build_channel_properties( ) function. Use the "Spec" parameter with the PNIO_build_channel_properties( ) function as follows:

2 - Problem exiting state, no further problems.

3 - Problem exiting state, further problems exist.

Note

If all alarms are exiting state, transfer a NULL pointer in "pData".

Structure 1

The diagnostics data was transferred with PNIO_diag_generic_add( ); "pData" points to the packed structure:

```
Struct
{
   PNIO_UINT16 ChannelNum;              /* ChannelNum parameter as for
                                        PNIO_diag_generic_add( )in big
                                        endian */
   PNIO_UINT16 ChannelProp;             /* ChannelProp parameter from
                                        PNIO_build_channel_properties
                                        ( ) in big endian */
   PNIO_UINT8 InfoData[InfoDataLen];    /* Diagnostics data to which the
                                        parameter pInfoData from
                                        PNIO_diag_generic_add( )
                                        points*/
};
```

Structure 2

The diagnostics data was transferred with PNIO_diag_channel_add( ); "pData" points to the packed structure:

```
Struct
{
   PNIO_UINT16 ChannelNum;              /* ChannelNum parameter as for
                                        PNIO_diag_ channel_add( ) */
   PNIO_UINT16 ChannelProp;             /* ChannelProp parameter from
                                        PNIO_build_channel_properties
                                        ( ) */
   PNIO_UINT16 ChannelErrorType;        /* ChannelErrorType parameter
                                         as for PNIO_diag_channel_add
                                        ( ) */
};
```

Structure 3

The diagnostics data was transferred with PNIO_diag_ext_channel_add( ): "pData" points to the packed structure:

```
Struct
{
   PNIO_UINT16 ChannelNum;       /* ChannelNum parameter as for
                                 PNIO_diag_ext_channel_add( ) */
   PNIO_UINT16 ChannelProp;      /* ChannelProp parameter from
                                 PNIO_diagbuild_channel_properties( ).*/
   PNIO_UINT16 ChannelErrorType;       /* ChannelErrorType parameter as
                                        for PNIO_diag_ext_channel_add( )
                                       */
   PNIO_UINT16 ExtChannelErrorType;    /* ExtChannelErrorType parameter
                                       as for PNIO_diag_ext_channel_
                                       add( ) */
   PNIO_UINT32 ExtChannelAddValue;     /* ExtChannelAddValue parameter
                                       as for PNIO_diag_ext_channel_
                                       add( ) */
};
```

Functions specific to the products CP 1626 / CP 1616 / CP 1604

8

Overview

The following additional functions are available for the CP 1604, CP 1616 and CP 1626 both as IO controller and as IO device:

Function name	SIMATIC NET products	
	CP 1604 / CP 1616	CP 1626
PNIO_CP_set_appl_watchdog()	x	x
PNIO_CP_trigger_watchdog()	x	x
PNIO_CBF_APPL_WATCHDOG()	x	x
SERV_CP_get_fw_info()	x	–
SERV_CP_download()	x	x
SERV_CP_download_config_pwd()	--	x
SERV_CP_upload()	–	x
SERV_CP_download_omsstore()	–	x
SERV_CP_backup()	–	x
SERV_CP_restore()	–	x
SERV_CP_set_type_of_station()	x	x
SERV_CP_set_name_and_ip()	–	x
SERV_CP_get_fw_info_extended()	–	x
SERV_CP_reset()	x	x
SERV_CP_info programming interface	x	x
SERV_CP_info_open( )	x	x
SERV_CP_info_close( )	x	x
SERV_CP_INFO_PRM	x	x
SERV_CP_info_register_cbf( )	x	x
SERV_CP_INFO_TYPE	x	x
PNIO_CBE_CP_STOP_REQ callback event	x	x
SERV_CP_INFO_PDEV_DATA callback event	x	x
SERV_CP_INFO_PDEV_INIT_DATA callback event	x	x
SERV_CP_INFO_LED_STATE callback event	x	x

8.1 PNIO_CP_set_appl_watchdog( ) (user program watchdog)

Description

With this function, the user program registers for time monitoring (watchdog). The user program specifies the timeout interval in "wdTimeOutInMs". After activating the watchdog, the user program must call the PNIO_CP_trigger_watchdog( ) function at least once during the monitoring interval to signal to the IO-Base interface that the user program is functioning. The monitoring is activated only after PNIO_CP_trigger_watchdog() is called the first time. If the user program does not call the PNIO_CP_trigger_watchdog( ) in good time, monitoring stops, the registered callback is called and the CP does not send any more IO data. Following this, no other function calls are permitted except PNIO_contoller_close() and PNIO_device_close(). To allow the CP to send IO data again, the user program must first call PNIO_controller_close() or PNIO_device_close() and then call PNIO_contoller_open() or PNIO_controller_open() again.

Note

Only one user program can ever register for the user program timeout monitoring.

Note

The watchdog is only started when the PNIO_CP_trigger_watchdog function has been called at least once.

Note

If the timeout interval of a user program is exceeded, the PROFINET IO communication of other user programs over the same CP is also stopped.

Note

To restart the watchdog when the timeout interval has elapsed, the previous watchdog must be deregistered and then registered again.

Note

If the watchdog is registered and unregistered repeatedly, a wait time should be maintained between the individual function calls. This must be at least 20 ms.

Syntax

```
PNIO_UINT32 PNIO_CP_set_appl_watchdog(
 PNIO_UINT32 CPIndex,
 PNIO_UINT32 wdTimeOutInMs,
PNIO_CBF_APPL_WATCHDOG pnio_appl_wd_cbf;
```

Parameter

Name	Description
CpIndex	Module index - Used to uniquely identify the communications module. Refer to the configuration for this parameter ("module index" of the CP).
wdTimeOutInMs	Timeout interval in ms. To deregister the user program watchdog, the value must be set to 0.
pnio_appl_wd_cbf	Callback to be called in the timeout interval is exceeded. To deregister the user program watchdog, the value must be set to 0.

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_CREATE_INSTANCE
- PNIO_ERR_INTERNAL
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_PRM_CP_ID

8.2 PNIO_CP_trigger_watchdog( ) (user program watchdog)

Description

This function is used to retrigger the user program watchdog.

Syntax

```
PNIO_UINT32  PNIO_CP_trigger_watchdog(
    PNIO_UINT32     CPIndex               //in
);
```

Parameter

Name	Description
CpIndex	CP index for which the user program watchdog will be retriggered.

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_CP_ID

8.3 PNIO_CBF_APPL_WATCHDOG( ) (user program watchdog)

Description

This callback is called by the IO-Base interface as soon as user program watchdog has responded. After calling this callback, all user programs that have called PNIO_controller_open( ) must call PNIO_controller_close( ) and all user programs that have called PNIO_device_open( ) must send PNIO_device_close( ).

Syntax

```
typedef void (*PNIO_CBF_APPL_WATCHDOG)(
    PNIO_UINT32     CPIndex                //in
);
```

Parameter

Name	Description
CpIndex	CP index for which the user program watchdog was registered.

Return values

No return values.

8.4 SERV_CP_get_fw_info()

Description

This function returns various firmware configuration parameters. There is a detailed description of the returned parameters in the description of the output structure.

Syntax

```
PNIO_UINT32 SERV_CP_get_fw_info (
 PNIO_UINT32 CpIndex,
 SERV_CP_FW_INFO_TYPE *p_info);
```

Parameter

Parameter	Description
CpIndex	Input parameter used to identify the CP from which the information is queried. The values are in the range <1, n> (n = MAX_CP16XX_DEVICES). Currently only a CP 1604/CP 1616/CP 1626 is supported in the PC. "MAX_CP16XX_DEVICES" is defined in "host_linwin\driver\cp16xx_base.h".
SERV_CP_FW_INFO_TYPE* p_info	Pointer to the output structure in which the CP configuration parameters are returned by the firmware. The structure is made available (allocated) by the caller (user). The data is copied to the structure by "Servlib".

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_PRM_CP_ID
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_DRIVER_IOCTL_FAILED
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_OS_RES
- PNIO_ERR_NO_RESOURCE

If an error occurs, the values in the output structure are invalid.

8.4.1 Description of the structure SERV_FW_VERS_TYPE

Syntax

```
typedef struct{
 PNIO_UINT16 v1;
 PNIO_UINT16 v2;
 PNIO_UINT16 v3;
 PNIO_UINT16 v4;
 PNIO_UINT16 v5;
```

```
 PNIO_UINT16 reserved;
} ATTR_PACKED SERV_FW_VERS_TYPE;
```

Name	Description
V1	1st place in the version identifier (e.g. 1 in V1.0.0.0.0)
V2	2nd place in the version identifier
V3	3rd place in the version identifier
V4	4th place in the version identifier
V5	5th place in the version identifier
reserved	reserved

8.4.2 Description of the structure SERV_CP_FW_INFO_TYPE

Syntax

```
typedef struct serv_cp_fw_info_s {
 SERV_FW_VERS_TYPE fw_version;
 PNIO_UNIT8 cp_ser_nr[16];
 PNIO_UNIT8 mlfb[32];
 PNIO_UINT16 hw_version;
 PNIO_UNIT8 mac[6];
 PNIO_UINT32 ip_addr;
 PNIO_UINT32 ip_mask;
 PNIO_UINT32 default_router;
 PNIO_UINT8 name[256];
 PNIO_UINT8 TypeOfStation[256];
};
```

Name	
fw_version	Version of the firmware, see "SERV_FW_VERS_TYPE" structure
cp_ser_nr	Serial number of the communications processor (null terminated ASCII string)
mlfb	Order number of the communications processor (null terminated ASCII string)
hw_version	Hardware version of the communications processor
mac	First Ethernet address of the communications processor (every communications processor has multiple Ethernet addresses). There is a further Ethernet address for each LAN socket of the communications processor.
ip_addr	IP address of the communications processor (IE/PNIO parameter)
ip_mask	IP subnet mask of the communications processor
default_router	IP default gateway of the communications processor
Name	PROFINET station name of the communications processor
TypeOfStation	PROFINET device type

8.5 SERV_CP_download() (download firmware or configuration)

Description

This function is used to load firmware of a configuration or its own Web server certificate on the CP1616/1626. It may only be called when no IO controller or IO device application is registered. After downloading firmware, the communications processor is restarted automatically. The data is stored retentively on the CP1616/1626.

Note

This function requires exclusive access to the communications processor. If a different application (PROFINET IO controller, PROFINET IO device, …) is registered, this function is terminated with the error code PNIO_ERR_MAX_REACHED.

Syntax

```
PNIO_UINT32 SERV_CP_download (
 PNIO_UINT32 CpIndex,
 PNIO_CP_DLD_TYPE DataType,
 PNIO_UINT8 *pData,
 PNIO_UINT32 DataLen
);
```

Parameter

Name	Description
CpIndex	CP index to identify the communications processor. Only a CP 1616/CP 1626 is supported in the PC, therefore always 1.
DataType	Specifies which data type should be loaded.
pData	CP 1616. Pointer to the memory area containing the binary data of the configuration or the firmware. CP 1626: String (null terminated) with the path to the source file.
DataLen	CP 1616. Length of the memory area containing the binary data of the configuration or the firmware. CP 1626: Length of the string (null terminated) pData.

Supported data types

DataType	CP 1616	CP 1626
PNIO_CP_DLD_CONFIG	Loading a configuration in the XDB format.	Loading a configuration in the OCF format.
PNIO_CP_DLD_FIRMWARE	Loading firmware in the FWL format.	Loading firmware in the UPD format.
PNIO_CP_DLD_SSL_PUBLIC	Not supported	Loading a Web server certificate in the PEM format.
PNIO_CP_DLD_SSL_PRIVATE	Not supported	Loading the private key for the Web server certificate in the PEM 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_CORRUPTED_DATA
- PNIO_ERR_DRIVER_IOCTL_FAILED
- PNIO_ERR_FLASH_ACCESS
- PNIO_ERR_OS_RES
- PNIO_ERR_PRM_HND
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_ALREADY_DONE
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SESSION
- PNIO_ERR_INTERNAL

8.6 SERV_CP_download_config_pwd()

Description

This function is used to load a password-protected OCF file on the CP 1626 which is generated in "STEP 7 Professional (TIA Portal)".

Syntax

```
PNIO_UINT32 SERV_CP_download_config_pwd (
 PNIO_UINT32 CpIndex,
 PNIO_UINT8 *pData,
 PNIO_UINT32 DataLen,
 PNIO_UINT8 *pKey,
 PNIO_UINT32 KeyLen
);
```

Parameter

Name	Description
CpIndex	CP index to identify the communications processor. Only a 1626 is supported in the PC, therefore always 1.
pData	String with the path to the OCF file.
DataLen	Length of the string pData.
pKey	Pointer to the string to decrypt the OCF file.
KeyLen	Length of the string of pKey.

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_PRM_BUF
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_SESSION
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_ALREADY_DONE

8.7 SERV_CP_upload()

Description

With this function firmware(upd) and configuration(omstore) can be exported from the CP 1626 to the host computer.

Syntax

```
PNIO_UINT32 SERV_CP_upload(
 PNIO_UINT32 CpIndex,
 SERV_CP_DLD_TYPE DataType,
 const PNIO_UINT8 *const TargetPath
);
```

Parameter

Name	Description
CpIndex	CP index to identify the communications processor. Only a 1626 is supported in the PC, therefore always 1.
DataType	Selection of the type of installation (enum SERV_CP_DLD_TYPE in servusrx.h). The following types of installation are permitted: • SERV_CP_DLD_FIRMWARE: exports the installed UPD file • PNIO_CP_DLD_CONFIG: exports the configuration as a directory tree "OMSSTORE"
const TargetPath	Destination path for the export (e.g. "C:\TMP\")

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_PRM_BUF
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_NO_RESOURCE

8.8 SERV_CP_download_omsstore()

Description

This function is used to transfer a directory tree containing the configuration of the CP 1626 from "STEP 7 Professional (TIA Portal)".

Syntax

```
PNIO_UINT32 SERV_CP_download_omsstore (
 PNIO_UINT32 CpIndex,
 PNIO_UINT8 *pData,
 PNIO_UINT32 DataLen
);
```

Parameter

Name	Description
CpIndex	CP index to identify the communications processor. Only a 1626 is supported in the PC, therefore always 1.
pData	Path to the directory tree on the host computer.
DataLen	Length of the string "pData".

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_PRM_BUF
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_SESSION
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_ALREADY_DONE

8.9 SERV_CP_backup()

Description

This function creates a local backup of the current CP 1626. The backup contains the firmware, the configuration from "STEP 7 Professional (TIA Portal)" and the local adaptations with series machines.

The paths to the backup directories:

- Windows: <public documents>\Siemens\SIMATIC.NET\CP1626
- Linux: /etc/cp1626_config

Syntax

```
PNIO_UINT32 SERV_CP_backup(
 PNIO_UINT32 CpIndex
);
```

Parameter

Name	Description
CpIndex	CP index to identify the communications processor. Only a 1626 is supported in the PC, therefore always 1.

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_PRM_BUF
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_NO_RESOURCE

8.10 SERV_CP_restore()

Description

This function restores the status of a CP 1626 with a previously backed up firmware and configuration. You must first back up this status using the "SERV_CP_backup" function.

The paths to the backup directories:

- Windows: <public documents>\Siemens\SIMATIC.NET\CP1626
- Linux: /etc/cp1626_config

Syntax

```
PNIO_UINT32 SERV_CP_restore(
 PNIO_UINT32 CpIndex
);
```

Parameter

Name	Description
CpIndex	CP index to identify the communications processor. Only a 1626 is supported in the PC, therefore always 1.

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_CORRUPTED_DATA
- PNIO_ERR_DRIVER_IOCTL_FAILED
- PNIO_ERR_FLASH_ACCESS
- PNIO_ERR_OS_RES
- PNIO_ERR_PRM_HND
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_ALREADY_DONE
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_NO_FW_COMMUNICATION

- PNIO_ERR_SESSION
- PNIO_ERR_INTERNAL

8.11 SERV_CP_set_type_of_station()

Description

This function sets a new device type in the firmware. This function must be called before calling "PNIO_controller_open() / PNIO_device_open".

Changing the device type only takes effect after resetting the communications processor. This means that there needs to be a firmware reset after calling "SERV_CP_set_type_of_station()". Following this, you need to restart your PC.

If you "reset to factory settings", the device type is reset to the default value "S7 PC". Following this then needs to be a reset (restart) of the firmware before the "reset" takes effect.

Note

Naming the device type

In "STEP 7 Professional (TIA Portal)", only 19 characters are recognized. It is therefore advisable to avoid using longer names as well as umlauts (ä, ö etc.) or special characters for the device type.

Syntax

```
PNIO_UINT32 SERV_CP_set_type_of_station(
 PNIO_UINT32 CpIndex,
 char *TypeName
);
```

Parameter

Parameter	Description
CpIndex	CP index to identify the communications processor. Only a communications processor is supported in the PC, therefore always 1.
TypeName	Name of the device type of the communications processor. Character string (null terminated). The device type to be set. Note: The maximum length is 255 characters. A longer string is truncated.

Return values

If successful, PNIO_OK is returned. If an error occurs, the following value is possible (for the meaning, refer to the comment in the header file "pnioerrx.h"):

- PNIO_ERR_INTERNAL
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_ALREADY_DONE
- PNIO_ERR_OS_RES
- PNIO_ERR_NO_RESOURCE

8.12 SERV_CP_set_name_and_ip()

Description

This function sets the device name of the CP 1626 and the IP addresses for the PROFINET interfaces "X1" or "X2". This applies only for operation in "STEP 7 Professional (TIA Portal)" as a series machine.

Syntax

```
PNIO_UINT32 SERV_CP_set_name_and_ip(
 PNIO_UINT32 CpIndex,
 SERV_SET_NAME_AND_IP_TYPE *const pData
);

typedef struct {
 SERV_INTERFACE_ID net_interface;
 SERV_IP_ADDRESS net_config;
 PNIO_UINT8 name_of_station[SERV_MAX_STATION_LENGTH];
} SERV_SET_NAME_AND_IP_TYPE;

typedef struct{
 PNIO_UINT8 ip_address[SERV_IPv4_LENGTH];
 PNIO_UINT8 ip_netmask[SERV_IPv4_LENGTH];
 PNIO_UINT8 ip_gateway[SERV_IPv4_LENGTH];
```

```
 PNIO_UINT8 mac_address[SERV_MAC_LENGTH];
 PNIO_UINT16 reserved;
} SERV_IP_ADDRESS;
```

Parameter

Name	Description		
CpIndex	CP index to identify the communications processor. Only a 1626 is supported in the PC, therefore always 1.		
const pData	Configuration data of a series machine		
	net_interface	Interface for: • SERV_INTERFACE_X1 • SERV_INTERFACE_X2	
	net_config	Network configuration of the relevant interface	
		ip_address	IP address
		ip_netmask	Network mask
		ip_gateway	Gateway address
		mac_address	MAC address of the PROFINET interface. Not taken into account when setting.
		reserved	reserved
	name_of_station	Device name for the associated interface Rules for the name: • The name consists of one or more parts separated by a period [.]. • Restriction to a total of 240 characters (lower case letters, numbers, hyphen or period) • A section within the device name, in other words, a string between two periods, must not exceed 63 characters. • A section within the name consists of the characters [a-z, 0-9]. • The device name must not begin or end with the "-" character, nor may this be the last character. • The device name must not begin with numbers. • The device name must not have the format n.n.n.n (n = 0 ... 999). • The device name must not begin with the string "port-xyz" or "port-xyz-abcde" (a, b, c, d, e, x, y, z = 0, ... 9).	

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_PRM_BUF

- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_PRM_INVALIDARG
- PNIO_ERR_WRONG_HND

8.13 SERV_CP_get_fw_info_extended()

Description

With this function the complete configuration (firmware version, IP addresses, MAC addresses etc.) of the CP 1626 is read out.

Syntax

```
PNIO_UINT32 SERV_CP_get_fw_info_extended (
 PNIO_UINT32 CpIndex,
 SERV_FW_INFO_DATA *pInfo
);

typedef struct{
 SERV_FW_VERS_TYPE fpga_version;
 SERV_FW_VERS_TYPE bootloader_version;
 SERV_FW_VERS_TYPE firmware_version;
 PNIO_UINT8 serial_number[SERV_MAX_SERIAL_NUMBER_LENGTH];
 PNIO_UINT8 mlfb_number[SERV_MLFB_LENGTH];
 PNIO_UINT16 hardware_version;
 PNIO_UINT8 name_of_station_x1[SERV_MAX_STATION_LENGTH];
 PNIO_UINT8 name_of_station_x2[SERV_MAX_STATION_LENGTH];
 PNIO_UINT8 type_of_station[SERV_MAX_TYPE_OF_STATION_LENGTH];
 SERV_IP_ADDRESS net_config_x1;
 SERV_IP_ADDRESS net_config_x2;
} SERV_FW_INFO_DATA;

typedef struct {
```

```
 PNIO_UINT16 v1;
 PNIO_UINT16 v2;
 PNIO_UINT16 v3;
 PNIO_UINT16 v4;
 PNIO_UINT16 v5;
 PNIO_UINT16 reserved;
 } SERV_FW_VERS_TYPE;

typedef struct{
 PNIO_UINT8 ip_address[SERV_IPv4_LENGTH];
 PNIO_UINT8 ip_netmask[SERV_IPv4_LENGTH];
 PNIO_UINT8 ip_gateway[SERV_IPv4_LENGTH];
 PNIO_UINT8 mac_address[SERV_MAC_LENGTH];
 PNIO_UINT8 reserved;
} SERV_IP_ADDRESS;
```

Parameter

Name	Description	
CpIndex	CP index to identify the communications processor. Only a 1626 is supported in the PC, therefore always 1.	
pInfo	Complete configuration and information of the communications processor	
	fpga_version	Version of the FPGA
	bootloader_version	Version of the bootloader
	firmware_version	Version of the firmware
	serial_number	Serial number of the communications processor
	hardware_version	Hardware version of the communications processor
	name_of_station_x1	Device name for PROFINET interface X1
	name_of_station_x2	Device name for PROFINET interface X2
	type_of_station	Device type of the CP 1626 (S7 PC)
	net_config_x1	see description in section "SERV_CP_set_name_and_ip() (Page 249)"
	net_config_x2	see description in section "SERV_CP_set_name_and_ip() (Page 249)"

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_PRM_CP_ID
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_PRM_BUF

8.14 SERV_CP_reset() (restart of the communications processor)

Description

This function restarts the CP 1616 or the CP 1626

Syntax

```
PNIO_UINT32 SERV_CP_reset(
 PNIO_UINT32 CpIndex,
 PNIO_CP_RESET_TYPE ResetType
);
```

Parameter

Name	Description	
CpIndex	CP index to identify the communications processor.	
ResetType	PNIO_CP_RESET_SAFE	Restart possible only, when no application is registered.
	PNIO_CP_RESET_FORCE	Restart also possible, when an application is 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_CREATE_INSTANCE
- PNIO_ERR_DRIVER_IOCTL_FAILED

- PNIO_ERR_INTERNAL
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_PRM_CP_ID

8.15 SERV_CP_info programming interface

Description

The SERV_CP_info programming interface contains general service functions supported by the CP 1604 / CP 1616 / CP 1626. Using this interface, a user application can, for example, obtain information on port events or the status of module LEDs.

8.16 SERV_CP_info_open( ) (register a user program with the SERV_CP_info programming interface)

Description

With this function, a user program registers with the SERV_CP_info programming interface.

Following this, Callback functions for the alarm and information events can be registered.

Syntax

```
PNIO_UINT32 SERV_CP_info_open(
 PNIO_UINT32 CpIndex,
 PNIO_UINT32 *Handle,
 PNIO_UINT32 Reserved
);
```

Parameter

Name	Description
CpIndex	CP index - Used to uniquely identify the communications module.
Handle	Handle that is returned to the user program if the registration was successful. This must be included in all further function calls.
Reserved	reserved

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_PRM_CP_ID
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_NO_FW_COMMUNICATION

8.17 SERV_CP_info_close( ) (deregister a user program from the SERV_CP_info programming interface)

Description

With this function, a user application deregisters from the SERV_CP_info programming interface.

Syntax

```
PNIO_UINT32 SERV_CP_info_close(
 PNIO_UINT32 Handle
);
```

Parameter

Name	Description
Handle	Handle from SERV_CP_info_open()

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_DRIVER_IOCTL_FAILED
- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_NO_RESOURCE

- PNIO_ERR_PRM_TYPE
- PNIO_ERR_WRONG_HND

8.18 SERV_CP_INFO_PRM (callback event parameter)

Description

The various callback events on the SERV_CP_info interface have a common data type SERV_CP_INFO_PRM, that groups the different parameters of the individual callback events using a "union".

Syntax

```
typedef struct
{
    PNIO_UINT32         Version;
    PNIO_UINT32         Handle;
    PNIO_UINT32         CpIndex;
    SERV_CP_INFO_TYPE   InfoType;
    union
    {
        ...          /*Union over the various
        ...            callback event parameters
                       (details in relevant sections)*/
    }u;
} ATTR_PACKED SERV_CP_INFO_PRM;
```

Parameter

Name	Description
Version	Version of the structure
Handle	Handle from SERV_CP_info_open( )
CpIndex	Module index - Used to uniquely identify the communications module.
InfoType	Callback event type

8.19 SERV_CP_info_register_cbf( ) (registration of callback functions)

Description

This function registers a callback function.

Syntax

```
PNIO_UINT32 SERV_CP_info_register_cbf(
 PNIO_UINT32 Handle,
 SERV_CP_INFO_TYPE InfoType,
 SERV_CP_INFO_CBF Cbf,
 PNIO_UINT32 Reserved
);
```

Parameter

Name	Description
Handle	Handle from SERV_CP_info_open( )
InfoType	Callback event type for which the callback function "Cbf" will be registered; see the section "SERV_CP_INFO_PRM (callback event parameter) (Page 256)".
Cbf	Address of the callback function to be started after arrival of the callback event "CbeType". If Cbf equals ZERO, the previously registered callback function is deregistered and no further events of the "InfoType" are forwarded to the user application.

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_ALREADY_DONE
- PNIO_ERR_DRIVER_IOCTL_FAILED
- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_WRONG_HND

8.20 SERV_CP_INFO_TYPE (callback event type)

Description

The SERV_CP_info user programming interface recognizes the following callback event types:

Callback event type	Callback event
SERV_CP_INFO_PDEV_DATA	Port alarm arrived.
SERV_CP_INFO_PDEV_INIT_DATA	Current port status arrived.*
SERV_CP_INFO_LED_STATE	LED status change arrived.
SERV_CP_INFO_STOP_REQ	Signal remote firmware download or reconfigura-tion request
SERV_CP_INFO_CBF_FATAL_ERROR	Firmware is no longer reacting

* This callback event type does not require explicit registration and is registered once automatically when the PNIO_CP_INFO_PDEV_DATA event is registered.

8.21 Callback event SERV_CP_INFO_STOP_REQ (register remote download request)

Description

The SERV_CP_INFO_STOP_REQ callback event reports to its user program that a remote download request (firmware update or reconfiguration) has been received. The callback event must already have been registered with the SERV_CP_info_register_cbf( ) function.

On receiving this callback, the user program must send a SERV_CP_info_close( ). Following this, the user program can once again call SERV_CP_info_open( ).

The functions SERV_CP_info_close( ) and SERV_CP_info _open( ), however, must not execute within the function belonging to the callback event SERV_CP_INFO_STOP_REQ.

During the download, the SERV_CP_info_open( ) function returns with the error code "PNIO_ERR_CONFIG_IN_UPDATE" or "PNIO_ERR_NO_FW_COMMUNICATION".

Following a successful download, the SERV_CP_info_open( ) function returns with the "PNIO_OK" error code.

Note

If a user program does not register this callback, there can be no remote download request or reconfiguration as long as the user program is active.

Syntax

This event type has no other specific parameters as part of the "union" from the SERV_CP_INFO_PRM structure; see the section "SERV_CP_INFO_PRM (callback event parameter) (Page 256)".

8.22 Callback event SERV_CP_INFO_PDEV_DATA (port alarm arrived)

Description

The SERV_CP_INFO_PDEV_DATA callback event reports the arrival of a port alarm to its user program.

A port alarm informs the user program of a change in port status parameters, such as link down, frame dropped and redundancy information. The callback event must already have been registered with the SERV_CP_info_register_cbf( ) function.

The syntax below shows the specific parameters for this event as part of the "union" from the SERV_CP_INFO_PRM structure; see the section "SERV_CP_INFO_PRM (callback event parameter) (Page 256)".

Syntax of PNIO_ALARM_TINFO

```
typedef struct
{
    PNIO_UINT16        CompatDevGeoaddr;
    PNIO_UINT8         ProfileType;
    PNIO_UINT8         AinfoType;
    PNIO_UINT8         ControllerFlags;
    PNIO_UINT8         DeviceFlag;
    PNIO_UINT16        PnioVendorIdent;
    PNIO_UINT16        PnioDevIdent;
    PNIO_UINT16        PnioDevInstance;
} ATTR_PACKED PNIO_ALARM_TINFO;
```

Elements of PNIO_ALARM_TINFO

Name	Description
CompatDevGeoaddr	bit 0 - 10 = device_no bit 11 - 14 = io-subsys-nr bit 15 = 1 (pnio identifier)
ProfileType	0x08 => bit 0 - 3 for PNIO bit 4 - 7 for DP
AinfoType	bit 0-3 alinfotyp = 0x00 (transparent)
ControllerFlags	bit 0 = 1 (ext. dp-interface) bit 1 - 7 reserved
DeviceFlag	bit 0 for PNIO: apdu-stat-failure bit 1 - 7 reserved

Name	Description
PnioVendorIdent	for PNIO: vendor identification
PnioDevIdent	for PNIO: device identification
PnioDevInstance	for PNIO: instance of device

Syntax of SERV_CIB_PDEV_EXTCHNL_DIAG

```
typedef struct            /* for details refer to IEC 61158-6 */
{
    PNIO_UINT16      Channel;
    PNIO_UINT16      Properties;
    PNIO_UINT16      ErrType;
    PNIO_UINT16      ExtErrType;
    PNIO_UINT16      ExtAdvAlLo;
    PNIO_UINT16      ExtAdvAlHi;
} ATTR_PACKED SERV_CIB_PDEV_EXTCHNL_DIAG;
                /* coding of the ExtChannelDiagnosisData */
```

Elements of SERV_CIB_PDEV_EXTCHNL_DIAG

Name	Description	
Channel	chapter "Coding of the field ChannelNumber" 0x0000-0x7FFF manufacturer specific 0x8000 Submodule: Submodule 0x8001 - 0xFFFF: Reserved	
Properties	chapter "Coding of the field ChannelProperties"	
	Bit 0 - 7: Bit 8: Bit 9 - 10: Bit 11 - 12: Bit 13 - 15:	ChannelProperties.Type ChannelProperties.Accumulative ChannelProperties.Maintenance ChannelProperties.Specifier ChannelProperties.Direction
ErrType	Chapter "Coding of the field ChannelErrorType"	
ExtErrType	chapter "Coding of the field ExtChannelErrorType"	
ExtAdvAlLo	chapter "Coding of the field ExtChannelAddValue"	
ExtAdvAlHi	chapter "Coding of the field ExtChannelAddValue"	

Syntax of SERV_CIB_PDEV_DATA

```
typedef struct
{
    PNIO_UINT32                 Version;
    PNIO_IO_TYPE                IODataType;        /* in, out */
    PNIO_UINT32                 LogAddr;
    PNIO_UINT32                 Slot;
    PNIO_UINT32                 Subslot;
    PNIO_ALARM_TYPE             AlarmType;
    PNIO_APRIO_TYPE             AlarmPriority;
    PNIO_UINT32                 AlarmSequence;
    PNIO_UINT32                 StationNr;
    PNIO_UINT8                  Reserved1[4];
    PNIO_UINT8                  Reserved2[48];
    PNIO_ALARM_TINFO            AlarmTinfo;
    SERV_CIB_PDEV_EXTCHNL_DIAG  Diagnostics;
    PNIO_UINT8                  DiagDs[4];
    PNIO_ALARM_INFO             AlarmAinfo;
} ATTR_PACKED SERV_CIB_PDEV_DATA;
```

Elements of SERV_CIB_PDEV_DATA

Name	Description
Version	Version of the structure - Always 0.
IODataType	Type of IO data
LogAddr	Logical address assigned during configuration.
Slot	Slot
Subslot	Subslot
AlarmType	Alarm type
AlarmPriority	Alarm priority
AlarmSequence	Number that is incremented with each new alarm.
StationNr	Station number assigned during configuration.
RESERVED1 [4]	reserved
Reserved2[48]	reserved
AlarmTinfo	Expanded configuration parameters
Diagnostics	See section "Syntax of SERV_CIB_PDEV_EXTCHNL_DIAG"
DiagDs	Additional data corresponding to the data in the diagnostics interrupt OB (OB82), see STEP 7 documentation.
AlarmAinfo	Alarm data

8.23 Callback event SERV_CP_INFO_PDEV_INIT_DATA (current port status data arrived)

Description

The SERV_CP_INFO_PDEV_INIT_DATA callback event reports the arrival of current port statuses to its user program.

In PROFINET IO, every Ethernet interface consists of an interface submodule (submodule number = 0x8i00) and n port submodules (submodule number = 0x8ipp, pp = port number, i = assigned interface number).

The syntax below shows the specific parameters for this event as part of the "union" from the SERV_CP_INFO_PRM structure; see the section "SERV_CP_INFO_PRM (callback event parameter) (Page 256)".

Syntax

```
typedef struct
{
    PNIO_UINT8         Reserved1;
    PNIO_UINT8         Reserved2;
    PNIO_UINT16        Reserved3;

    PNIO_IO_TYPE       IODataType;     /* input, output */
    PNIO_UINT32        LogAddr;        /* logical address,
                                          configured by
                                          engineering tool */
    PNIO_UINT32        Slot;           /* slot number */
    PNIO_UINT32        Subslot;        /* subslot number */

    PNIO_UINT16        PortState;      /* bit field,
                                          SERV_CIB_PS_... */

    PNIO_UINT16        Reserved4;
} ATTR_PACKED SERV_CIB_PDEV_INIT_STATE;

typedef struct
{
    PNIO_UINT32       ItemNumber;
    SERV_CIB_PDEV_INIT_STATE
              InitState[SERV_CIB_PDEV_MAX_PORT_ITEMS];
} ATTR_PACKED SERV_CIB_PDEV_INIT_DATA;
```

Parameter

Name	Description
Reserved1	Reserved
Reserved2	Reserved

Name	Description
Reserved3	Reserved
IODataType	Type of IO data (input/output data)
LogAddr	Logical address assigned during configuration.
Slot	Slot
Subslot	Subslot
PortState	Port state
Reserved4	reserved

Note

If there is not yet a configuration for the module ports, the "ItemNumber" parameter of the SERV_CP_INFO_PDEV_INIT_DATA callback event has the value 0.

8.24 Callback event SERV_CP_INFO_LED_STATE (LED status change arrived)

Description

The SERV_CP_INFO_LED_STATE callback event reports the arrival of an LED status change to its user program.

The user program can, for example, convert this information to a screen display. The callback event must already have been registered with the SERV_CP_info_register_cbf( ) function.

The syntax below shows the specific parameters for this event as part of the "union" from the SERV_CP_INFO_PRM structure; see the section "SERV_CP_INFO_PRM (callback event parameter) (Page 256)".

Syntax

```
typedef enum
{
  SERV_CIB_LED_TYPE_BF = 1,        /* bus fault led */
  SERV_CIB_LED_TYPE_SF = 2         /* group fault led */
} SERV_CIB_LED_TYPE;

typedef enum
{
  SERV_CIB_LED_STATE_OFF = 1,     /* off */
  SERV_CIB_LED_STATE_ON  = 2      /* on */
} SERV_CIB_LED_STATE;

typedef struct
{
    SERV_CIB_LED_TYPE     LedType;
    SERV_CIB_LED_STATE    LedState;
    PNIO_UINT8            Reserved1[4];
} ATTR_PACKED SERV_CIB_LED;
```

Parameter

Name	Description
LedType	LED type
LedState	LED status

8.25 Callback event SERV_CP_INFO_FATAL_ERROR (firmware is no longer reacting)

Description

The callback event "SERV_CP_INFO_FATAL_ERROR" signals the user program the absence of messages from the firmware.

The syntax does not include any further parameters.

Product-specific functions for the PN driver 9

The following additional functions are available to the PN driver as an IO controller:

- SERV_CP_init
- SERV_CP_undo_init
- SERV_CP_get_network_adapters
- SERV_CP_startup
- SERV_CP_shutdown
- SERV_CP_set_trace_level

9.1 SERV_CP_init

Description

This function has several tasks and needs to be called in the start sequence of the PN Driver. It performs the following tasks:

- Initialization of the trace subsystem (if enabled)
- Initialization of the PNIO stack
- Initialization of the mailbox subsystem
- Initialization of the threads
- Starting the threads

Note

The function "SERV_CP_init" must be called as the first function in the start sequence. Only afterwards can "SERV_CP_startup" and "PNIO_controller_open" or "PNIO_interface_open" be called.

Note

The Trace function is useful for troubleshooting if there are problems in the configuration or system adaptation.

Contact Siemens support to analyze the trace data and to obtain advice on setting the correct trace level.

Syntax

```
PNIO_UINT32 SERV_CP_init (PNIO_DEBUG_SETTINGS_PTR_TYPE DebugSetting);
```

Parameter

Name	Description
DebugSetting	This structure allows the user to make a callack function available that is called as soon as the trace buffer is full. The trace levels can also be set. If the parameter has the value "NULL", the internal trace is not used.

Return values

if successful, the function returns "PNIO_OK" otherwise "PNIO_ERR_SEQUENCE".

9.2 SERV_CP_undo_init

Description

This function completes the shutdown sequence.

Note

The function "SERV_CP_undo_init" needs to be called as the last function in the shutdown sequence. Prior to this, the functions "PNIO_controller_close" or "PNIO_interface_close" and "SERV_CP_shutdown" need to be called.

Syntax

```
PNIO_UINT32 SERV_CP_undo_init ();
```

Parameter

This function has no parameters.

Return values

This function has no return values.

9.3 SERV_CP_get_network_adapters

Description

This function returns a list of all detected network adapters.

Note

The function "SERV_CP_init" must have been called previously.

Syntax

```
PNIO_UINT32 SERV_CP_get_network_adapters (
 PNIO_CP_ID_PTR_TYPE CpList,
 PNIO_UINT8 *NrOfCp
);
```

Parameter

Name	Description
CpList	List of network adapters
NrOfCp	Number of list elements in CpList.

Return values

If successful, the function returns "PNIO_OK" otherwise "PNIO_ERR_NO_ADAPTER_FOUND".

9.4 SERV_CP_startup

Description

This function starts all internal tasks and configures the PNIO stack according to the configuration.

The following parameters are set by the user program:

- Selection of the Ethernet interfaces based on the MAC address when using the WinPcap driver or based on the PCI location with all other variants. PN Driver V1.0 only supports operation of a single interface. This means that "CpList" may only contain a single element and that "NrOfCp" must always be "1".
- Preparation of the PROFINET interface.

Note

The "SERV_CP_startup" function must be called before the "PNIO_controller_open()" function.

Syntax

```
PNIO_UINT32 SERV_CP_startup (
 PNIO_CP_ID_PTR_TYPE CpList,
 PNIO_UINT32 NrOfCp,
 PNIO_UINT8 *pConfigData,
 PNIO_UINT32 ConfigDataLen,
 PNIO_UINT8 *pRemaData,
 PNIO_UINT32 RemaDataLen,
 PNIO_SYSTEM_DESCR *pSysDescr
);
```

Parameter

Name	Description
CpList	Array of the designation that selects the network adapter to be used.
NrOfCp	Number of elements in "CpList". Must always be set to "1".
pConfigData	Pointer to the memory containing the XML PROFINET configuration in the form of a character string. The end of the string must be indicated by a "0" in the last byte (terminating NULL).
ConfigDataLen	Length of "pConfigData" in bytes without the terminating NULL.
pRemaData	Pointer to the retentive data, can be NULL.
RemaDataLen	Length of the memory area the "pRemaData" parameter references.
pSysDescr	Pointer to the structure "PNIO_SYSTEM_DESCR".

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_SEQUENCE
- PNIO_ERR_PRM_POINTER
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_INTERNAL
- PNIO_ERR_CORRUPTED_DATA

9.5 SERV_CP_shutdown

Description

This must be called as the last function when exiting the user program. After the return of the function, all internal threads have been exited and the entire local memory has been released again.

Note

The "SERV_CP_shutdown" function may only be called after the "PNIO_controller_close( )" function.

Syntax

```
PNIO_UINT32 SERV_CP_shutdown ();
```

Parameter

No parameters.

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_SEQUENCE

9.6 SERV_CP_set_trace_level

Description

With this function, the trace level of individual internal components can be changed during operation. The function can be called at any time after the "SERV_CP_startup( )" function. After being called, the function returns immediately and acknowledges the successful change of the trace level with the "CbfSetTraceLevelDone" callback function.

Syntax

```
PNIO_UINT32 SERV_CP_set_trace_level (

PNIO_UINT32 Component,
PNIO_UINT32 TraceLevel,
PNIO_PNTRC_SET_TRACE_LEVEL_DONE CbfSetTraceLevelDone);
```

Parameter

Name	Description
Component	Selection of the PNIO stack component (enum pnio_stack_comp in servusrx.h).
TraceLevel	Selection of the PNIO stack trace level (enum pnio_trace_level in seruvsrx.h).
CbfSetTraceLevelDone	Callback function for acknowledging the change.

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_SEQUENCE
- PNIO_ERR_INTERNAL
- PNIO_ERR_PRM_INVALIDARG

9.7 PNIO_IOS_RECONFIG_ MODE

Description

The function "PNIO_IOS_RECONFIG_ MODE()" lists the possible variant for the "Mode" parameter of the function "PNIO_interface_reconfig()".

Syntax

```
typedef enum {
 PNIO_IOS_RECONFIG_MODE_DEACT = 1,
 PNIO_IOS_RECONFIG_MODE_TAILOR = 2
}PNIO_IOS_RECONFIG_MODE;
```

Elements

Name	Description
PNIO_IOS_RECONFIG_MODE_DEACT	Deactivates all IO devices.
PNIO_IOS_RECONFIG_MODE_TAILOR	Starts the reconfiguration of the IO system with the configuration set by the user and then activates the IO devices according to the configuration.

9.8 Data types for the PN driver

9.8.1 PNIO_CP_SELECT_TYPE

Description

The "PNIO_CP_SELECT_TYPE " data type selects the addressing mode for the network adapter.

Syntax

```
typedef enum
{
 PNIO_CP_SELECT_WITH_PCI_LOCATION = 1,
 PNIO_CP_SELECT_WITH_MAC_ADDRESS = 2
} PNIO_CP_SELECT_TYPE;
```

Elements

Name	Description
PNIO_CP_SELECT_WITH_PCI_LOCATION	Selection using PCI location
PNIO_CP_SELECT_WITH_MAC_ADDRESS	Selection based on MAC address (relevant only with WinPcap)

9.8.2 PNIO_MAC_ADDR_TYPE

Description

The "PNIO_MAC_ADDR_TYPE" data type shows an array for transferring the MAC address in conjunction with WinPCAP.

Syntax

```
#define PNIO_MAC_ADDR_SIZE 6

typedef PNIO_UINT8 PNIO_MAC_ADDR_TYPE[PNIO_MAC_ADDR_SIZE];
```

9.8.3 PNIO_PCI_LOCATION_TYPE

Description

The "PNIO_PCI_LOCATION_TYPE" shows the parameter for transferring the PCI location of the network adapter.

Syntax

```
typedef struct pnio_pci_location_s
{
 PNIO_UINT16 BusNr;
 PNIO_UINT16 DeviceNr;
 PNIO_UINT16 FunctionNr;
} PNIO_PCI_LOCATION_TYPE;
```

Elements

Name	Description
BusNr	PCI bus number
DeviceNr	PCI device number
FunctionNr	PCI function number

9.8.4 PNIO_DEBUG_SETTINGS_TYPE / PNIO_DEBUG_SETTINGS_PTR_TYPE

Description

This structure contains the callback function sand the initial trace levels of the trace subsystems. The user specifies this structure as a parameter for "SERV_CP_init". If the parameter "CbfPntrcBufferFull" is set to "ZERO", the internal trace system is not used. Otherwise, if the trace buffer overflows, this callback function is called and the user can back up the transferred trace data.

Syntax

```
typedef struct pnio_debug_settings_s
{
 PNIO_PNTRC_BUFFER_FULL CbfPntrcBufferFull;
 PNIO_UINT8 TraceLevels[PNIO_TRACE_COMP_NUM];
} PNIO_DEBUG_SETTINGS_TYPE, *PNIO_DEBUG_SETTINGS_PTR_TYPE;
```

Element

Name	Description
CbfPntrcBufferFull	Is used to transfer trace data.
TraceLevels	Array of all internal modules for setting the initial trace level.

9.8.5 PNIO_PNTRC_BUFFER_FULL()

Description

This callback function is used to receive trace data. This gives the user the option of saving the trace data in the file system or similar.

Syntax

```
typedef void (*PNIO_PNTRC_BUFFER_FULL)
 ( PNIO_UINT8 * pBuffer,
 PNIO_UINT32 BufferSize );
```

Elements

Name	Description
pBuffer	Pointer to the trace data buffer
BufferSize	Length of the memory area the "pBuffer" parameter references.

9.8.6 PNIO_PNTRC_SET_TRACE_LEVEL_DONE()

Description

This callback function is called as soon as the trace levels have been set. This callback is registered with the SERV_CP_set_trace_level call.

Syntax

```
typedef void (*PNIO_PNTRC_SET_TRACE_LEVEL_DONE)(void);
```

9.8.7 PNIO_CP_ID_TYPE / PNIO_CP_ID_PTR_TYPE

Description

The data type "PNIO_CP_ID_TYPE" contains information on a network adapter.

Syntax

```
typedef struct pnio_cp_id_s {
 PNIO_CP_SELECT_TYPE CpSelection;
 PNIO_MAC_ADDR_TYPE CpMacAddr;
 PNIO_PCI_LOCATION_TYPE CpPciLocation;
 PNIO_UINT8 Description[300];
} PNIO_CP_ID_TYPE, *PNIO_CP_ID_PTR_TYPE;
```

Elements

Name	Description
CpSelection	Address mode for selecting the network adapter. With "WINPcap" you need to set the value to "PNIO_CP_SELECT_WITH_MAC_ADDRESS".
CpMacAddr	MAC address of the network adapter (WinPcap).
CpPciLocation	PCI location of the network adapter.
Description[300]	Contains a description of the network adapter.

9.8.8 PNIO_SET_IP_NOS_MODE_TYPE

Description

The data type "PNIO_SET_IP_NOS_MODE_TYPE" lists the possible variants for the parameter "Mode" of the function "PNIO_interface_set_ip_and_nos().

Syntax

```
typedef enum {
 PNIO_SET_IP_MODE = 0x0001,
 PNIO_SET_NOS_MODE = 0x0002
}PNIO_SET_IP_NOS_MODE_TYPE;
```

Elements

Name	Description
PNIO_SET_IP_MODE	The "IP Suite" can be changed.
PNIO_SET_NOS_MODE	The station name can be changed

9.8.9 PNIO_IPv4

Description

The data type "PNIO_IPv4" sets the "IP Suite" of the function "PNIO_interface_set_ip_and_nos".

Syntax

```
typedef struct {
 PNIO_UINT8 IpAddress[4];
 PNIO_UINT8 NetMask[4];
 PNIO_UINT8 Gateway[4];
 PNIO_BOOL Remanent;
} PNIO_IPv4;
```

Elements

Name	Description
IpAddress[4]	IP address
NetMask[4]	Subnet mask
Gateway[4]	Gateway
Retentive	If the parameter "PNIO_TRUE" is set, you can use the callback function to store the parameters retentively.

9.8.10 PNIO_NOS

Description

The data type "PNIO_NOS" sets the station name of the function "PNIO_interface_set_ip_and_nos".

Syntax

```
typedef struct {
 PNIO_UINT8 Nos[256];
 PNIO_UINT16 Length;
 PNIO_BOOL Remanent
;} PNIO_NOS;
```

Elements

Name	Description
Nos[256]	Station name
Length	Length of the station name
Retentive	If the parameter "PNIO_TRUE" is set, you can use the callback function to store the parameters retentively.

Descriptions of the functions and data types of the Ethernet interface

10

Description

Note

The functions and callbacks described in this section are only to be used for the product PN Driver".

Apart from the IO controller functionality, you can use the functions of the local Ethernet interface to

- read data records from the local Ethernet interface.
- receive alarms generated by the local Ethernet interface.
- adapt parameters such as "IP Suite" or the station name of the local Ethernet interface.

Note

You need to call the function Funktion "PNIO_interface_open()" to be able to use the functionality.

Overview

The table below shows an overview of the interface functionality provided by the SIMATIC NET products.

Function group and name	Products		
	SOFTNET PN IO (RT)	CP 1616 / CP 1604 (RT + IRT) with DK-16xx ab V2.0	PN driver
Management functions			
PNIO_interface_open()	–	–	x
PNIO_interface_register_cbf()	–	–	x
PNIO_interface_close()	–	–	x
PNIO_interface_set_ip_and_nos()	–	–	x
PNIO_CBE_IFC_SET_IP_AND_NOS callback event	–	–	x
Data interface			
PNIO_interface_rec_read_req()	–	–	x
PNIO_CBE_IFC_REC_READ_CONF callback event	–	–	x

Function group and name	Products		
	SOFTNET PN IO (RT)	CP 1616 / CP 1604 (RT + IRT) with DK-16xx ab V2.0	PN driver
Alarm interface			
PNIO_CBE_IFC_ALARM_IND callback event	–	–	x

10.1 Management functions

10.1.1 PNIO_interface_open()

Description

With this function you can receive alarms, read data records and change the "IP Suite" and also assign the station name of the local Ethernet interface. The function "PNIO_interface_open()" must have been called to use functions (setting the IP or "NameOfStation", reading data records, receiving alarms ...) of the local Ethernet interface.

Syntax

```
PNIO_UINT32 PNIO_interface_open(
 PNIO_UINT32 CpIndex
 PNIO_CBF cbf_rec_read_conf
 PNIO_CBF cbf_alarm_ind
 PNIO_UINT32 *Handle
);
```

Parameter

Name	Description
CpIndex	Module index
cbf_rec_read_conf	Address of the callback function that is called as soon as the read data record job is completed. The callback event type is "PNIO_CBE_IFC_REC_READ_CONF". "ZERO" can also be transferred.
cbf_alarm_ind	Address of the callback function that is called as soon as an alarm is signaled. The callback event type is "PNIO_CBE_IFC_ALARM_IND". "ZERO" can also be transferred.
Handle	Handle that is returned to the user if successful and references a local Ethernet interface. This handle must be used with all following calls of the local Ethernet interfface.

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_RESOURCE
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_PRM_HND
- PNIO_ERR_SEQUENCE

10.1.2 PNIO_interface_register_cbf()

Description

With this function, you register callback functions with the Ethernet interface after you have called the "PNIO_interface_open()" function.

Syntax

```
PNIO_UINT32 PNIO_interface_register_cbf(
 PNIO_UINT32 Handle
 PNIO_CBE_TYPE CbeType
 PNIO_CBF Cbf
);
```

Parameter

Name	Description
Handle	Handle from "PNIO_interface_open()"
CbeType	Callback event type for which the callback function will be registered.
Cbf	Address of the callback function that is to be called. **Note** The function pointer must not be "NULL". if a function pointer is already registered, no further one can 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_ALLREADY_DONE
- PNIO_ERR_INTERNAL
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_WRONG_HND

10.1.3 PNIO_interface_close()

Description

With this function you close the local Ethernet interface. No registered callback function will be called afterwards.

Note

Until execution of "PNIO_interface_close()" is completed, callback functions can still be called.

Note

After closing the local Ethernet interface, the handle supplied with "PNIO_interface_open()" becomes invalid and can no longer be used.

Syntax

```
PNIO_UINT32 PNIO_interface_close(
 PNIO_UINT32 Handle
);
```

Parameter

Name	Description
Handle	Handle from "PNIO_interface_open()"

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_WRONG_HND

10.1.4 PNIO_interface_set_ip_and_nos()

Description

With this function, you can change the "IP Suite" and / or the station name of the local Ethernet interface.

Note

The requirement for using this function is that the hardware configuration has been specified, that the IP address or the station name may be adapted locally.

Syntax

```
PNIO_UINT32 PNIO_interface_set_ip_and_nos(
 PNIO_UINT32 Handle,
 PNIO_SET_IP_NOS_MODE_TYPE Mode,
 PNIO_IPv4 IpV4,
 PNIO_NOS NoS,
);
```

Parameter

Name	Description
Handle	Handle from "PNIO_interface_open()"
Mode	• PNIO_SET_IP_MODE (0x0001): The IpV4 is to be changed. • PNIO_SET_NOS_MODE (0x0002): The station name is to be changed.
IPv4	"IP Suite" to be changed.
NoS	Station name 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_INTERNAL
- PNIO_ERR_WRONG_HND
- PNIO_ERR_MODE_VALUE
- PNIO_ERR_NOT_REENTERABLE
- PNIO_ERR_PRM_NOS_LEN
- PNIO_ERR_SET_IP_NOS_NOT_ALLOWED
- PNIO_ERR_PRM_IP PNIO_ERR_PRM_NOS

10.2 Data record interface

10.2.1 PNIO_interface_rec_read_req()

Description

With this function you can read data records from the local Ethernet interface.

Syntax

```
PNIO_UINT32 PNIO_interface_rec_read_req(
 PNIO_UINT32 Handle,
 PNIO_ADDR *pAddr,
 PNIO_UINT32 RecordIndex,
 PNIO_REF ReqRef,
 PNIO_UINT32 Length
);
```

Parameter

Name	Description
Handle	Handle from "PNIO_interface_open()"
pAddr	Address of a submodule of the local Ethernet interface on which the read data record job will be executed.
RecordIndex	Data record index

Name	Description
ReqRef	Reference specified by you. This reference allows you to distinguish between several jobs. You can set any value for this parameter and it is returned with the callback event "PNIO_CBE_IFC_REC_READ_CONF".
Length	Specifies the maximum record length of 4096 bytes. The pointer to the data and the real length are returned by the callback event "PNIO_CBE_IFC_REC_READ_CONF"

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_PRM_ADD
- PNIO_ERR_PRM_REC_INDEX
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND
- PNIO_ERR_NO_RESOURCE

10.2.2 PNIO_CBE_IFC_REC_READ_CONF

Description

This callback event signals that a previous read data record job was completed.

Syntax

```
typedef struct{
 PNIO_UINT32 Length;
 const PNIO_UINT8 *pBuffer;
 PNIO_ADDR *pAddr;
 PNIO_UINT32 RecordIndex;
 PNIO_REF ReqRef;
 PNIO_ERR_STAT Err;
} ATTR_PACKED PNIO_CBE_PRM_REC_READ_CONF;
```

Parameter

Name	Description
Length	Length of the data record in bytes to which the "pBuffer" parameter points.
pBuffer	Pointer to the data record containing the requested data record. The data buffer is only valid while the callback function is executing, and it becomes invalid as soon as the function is completed.
pAddr	Address of a module of the local Ethernet interface, that responded to the read data record job.
RecordIndex	Data record index
ReqRef	Reference specified by you. This reference allows you to distinguish between several jobs. You transferred this parameter with by calling "PNIO_interface_rec_read_req()".
Err	Error code on the execution of the job.

Return values

No return values.

10.3 Alarm interface

10.3.1 PNIO_CBE_IFC_ALARM_IND

Description

With this callback event, alarms of the local Ethernet interface are signaled.

Note

As long as this callback event has not been acknowledged, no further alarms are signaled.

Syntax

```
typedef struct{
 PNIO_ADDR *pAddr;
 PNIO_REF ReqRef;
 const PNIO_CTRL_ALARM_DATA *pAlarmData;
} ATTR_PACKED PNIO_CBE_PRM_ALARM_IND;
```

Parameter

Name	Description
pAddr	Address of the module from which the alarm originates.
ReqRef	Reference specified by you. This reference allows you to distinguish between several jobs. You transferred this parameter with by calling "PNIO_interface_rec_read_req()".
pAlarmData	Structure that contains the alarm information. "pAlarmData" points to data of the type "PNIO_CTRL_ALARM_DATA". The data buffer is only valid while the callback function is executing, and it becomes invalid as soon as the function is completed. The PNIO_CTRL_ALARM_DATA structure is defined in the pniousrx.h header file and contains, among other things, the parameters "Alarm type" and "Alarm specifier". For more detailed information on alarms, refer to the following documentation: • STEP 7 documentation, for example in the STEP 7 online help on SFB 54. • Documentation of the IO device

Return values

No return values.

Index

C

D

E

G

H

I

M

P

S

T