SIEMENS

PROFINET PROFIdrive Controller Application

User Manual

11/2021

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
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

℗ 11/2021 Subject to change

Table of contents

Introduction 1

1.1 System Overview

Purpose of this documentation

This document aims to describe PROFIdrive controller example application. By means of PROFIdrive controller example application, users of PROFINET Driver can easily create a user application for motion controllers that supports PROFIdrive feature connected via PROFINET, because all standard functionalities for PROFINET and PROFIdrive are provided by easy to use interface functions. Creator of the user application part does not need to know all details of PROFINET and especially PROFIdrive profile to make connected drives work in desired way. In addition to that, it is independent from specific controller system and designed to support multiple operating systems.

Target group for the manual

These instructions are intended for software developers and should help them to create a motion controller application user program in a very short amount of time. This requires the following basic knowledge:

- Programming experience with C/C++
- User experience in Linux operating system
- Experience with PROFINET IO systems
- Experience with PROFIdrive technology
- Basic knowledge of the configuration software TIA Portal or PNConfigLib
- General knowledge of automation technology

What is PROFINET Driver?

PROFINET Driver is a PROFINET IO controller development kit. It is delivered as a complete source code with various example applications. PROFINET driver enables to develop a PROFINET IO controller in a little development effort. Additionally, PROFINET Driver is able to support PROFINET IRT (Isochronous Real-Time) communication in case of CP 1625 Development board usage in host mode or stand-alone mode. PROFINET IRT communication is highly important for motion control applications since it provides a communication cycle down to below 31.25 us and PROFINET Driver V2.3 is capable to provide 250 us for CP1625 Host variant and 500 us for CP1625 Stand-alone variant. PROFINET Driver is thus an attractive option particularly for machine builders who use their own control software, to simply and cost-effectively connect PROFINET field devices, such as I/Os or drives.

What is PROFIdrive?

PROFIdrive is the standard profile for drive technology in conjunction with the PROFIBUS and PROFINET communication systems. PROFIdrive is a vendor-neutral application profile from PROFIBUS and PROFINET International (PI) (https://www.profibus.com/pi-organization/) which is focused on drives, encoders, motors, and their applications, which range from simple to very demanding motion control tasks. It supports both PROFIBUS and PROFINET communication technologies.

What are the requirements?

In addition to the PROFINET Driver V2.3, you need a PC with a SIMATIC CP 1625 Development Board (https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6ES7648-2CF10-1AA0). PROFINET Driver Host and PROFINET Driver Stand alone variants. A hardware configuration can be created with the TIA Portal engineering system as well as with PNConfigLib. PNConfigLib is a library that allows you to create PROFINET projects, perform consistency checks ensure their validation, and to compile these projects. It provides an API to allow users to call PNConfigLib in their own code.

Supported PROFINET Driver variants

PROFIdrive Controller application supports CP1625 Stand-alone and CP1625 Host variants for PROFINET Driver V2.3.

Overview of the supplied documentation

a user may need additional documentation that are listed below to develop their system with PROFINET Driver and PNConfigLib.

Table 1.1: Supplied Documentation

Name of the document	Why you should read it, and where you can find it
PROFINET System Description System Manual	This document includes the basics of the following topics involved in PROFINET IO: Network components, data exchange and communication, PROFINET IO, application example for PROFINET IO controller. This document can be found over this web link.
Quick Start PROFINET Driver V2.3 Getting Started	Quick Start describes the steps required for commissioning PROFINET Driver on supported platforms. This document is included in PROFINET Driver CD delivery.
IO-Base User Programming Interface for PN Driver Programming Manual	This document describes the IO-Base API which represents the interface for creating your own user programs. This document is included in PROFINET Driver CD delivery.
PNConfigLib User Manual and Documentation Manual	The scope of this user documentation is to provide technical details about the technology, components, requirements and constraints of PNConfigLib as well as acting as a guide for using PNConfigLib. This document is included in PROFINET Driver CD delivery.
Technical Specification for PROFINET Manual	This document describes the PROFIdrive feature over PROFIBUS and PROFINET devices. This document can be found over this web link.
Isochronous Mode Guideline Guideline	This document describes PROFINET Isochronous Mode in detailed. This document can be found over this web link.
PROFIdrive Technical Specification for PROFIBUS and PROFINET Manual	This document describes PROFIdrive technology and its communication structure in detailed. This document can be found over this web link.

Additional support

Please send questions, comments and suggestions regarding this manual in writing to the specified e-mail addresses below. For useful product information about PROFINET Driver and PNConfigLib, please visit the following address (https://support.industry.siemens.com/cs/products/6es7195-3aa00-0ya0).

In addition, you can find general information on the internet (https://www.siemens.com/

profinet-development).

Technical contact information worldwide

Siemens Sanayi ve Ticaret A.Ş. - E-mail:(mailto:profinet.devkits.industry@siemens.com)
Office Address:
Yakacık Caddesi No 111
34870 Istanbul, Turkey

Technical contact information for the U.S.

The PROFI Interface Center - Phone: +1 (423) 262-2576
(https://www.profiinterfacecenter.com) - E-mail:(mailto:PIC.industry@siemens.com)
Office Address:
Siemens Industry, Inc.
C/O The PROFI Interface Center
One Internet Plaza
Johnson City, TN 37604

Technical contact information for China

The PROFI Interface Center China - Phone: +86 (10) 6476-4725
Office Address: - E-mail: (mailto:Profinet.cn@siemens.com)
7, Wangjing Zhonghuan Nanlu
100102 Beijing

1.2 Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, systems, machines and networks. In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement and continuously maintain a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept. Customers are responsible for preventing unauthorized access to their plants, systems, machines, networks. Such systems, machines and components should only be connected to an enterprise network or the internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place. For additional information on industrial security measures that may be implemented, please visit (https://www.siemens.com/industrialsecurity).

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customers' exposure to cyber threats. To stay informed about product updates, follow us on Twitter (@ProductCERT), register to our advisory mailing list or subscribe to the Siemens Industrial Security RSS Feed under (https://new.siemens.com/global/en/products/services/cert.html#Subscriptions).

1.3 Open Source Software

The product/system described in this document may use Open Source Software or any similar software of a third party (hereinafter referred to as “OSS”). The OSS is listed in the Readme_OSS-file of the product. The purchaser of the product/system described in this document (hereinafter referred to as “the Customer”) is responsible for the right to use OSS that is required for the product to operate safely and without any problems in accordance with the respective license conditions of the OSS.

Description of the example application

2

2.1 Overview

PROFIdrive motion controller example application aims to demonstrate how PROFINET Driver and PROFIdrive interfaces can be used in order to develop a motion controller for drives, encoders, industrial motor applications, which range from simple to very demanding motion control tasks.

Structure of PROFIdrive motion controller example application

In order to provide an actual demonstration, the components that are shown in Fig. 2.1 are used for this purpose.

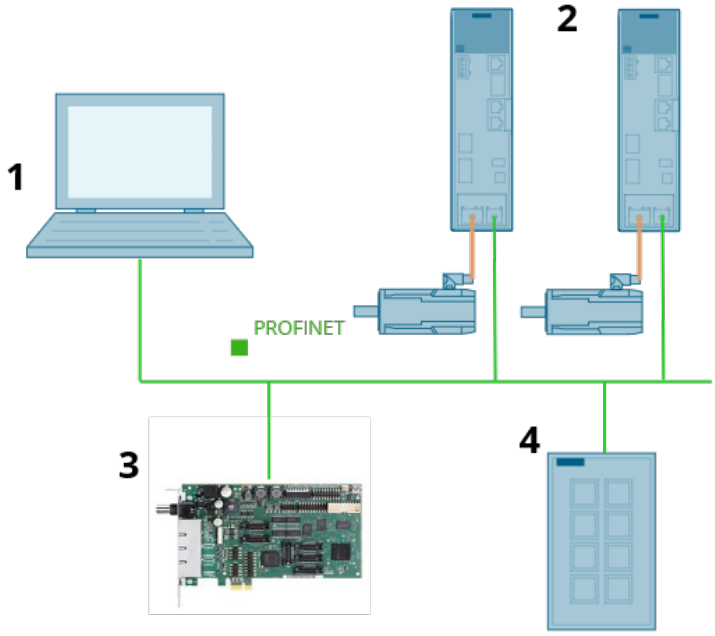

Fig. 2.1: PROFIdrive controller application

Note: The required components and their responsibilities in the example application are explained below. Article numbers for the related devices can be found at the table Table 2.1.

Table 2.1: Component List

	Component Name	**Article Number**
1	SIMATIC CP 1625	6ES7648-2CF10-1BA0
	PROFINET Driver	6ES7195-3AA00-0YA0
2	SINAMICS S210 Servo Drive System	6SL3210-5HB10-1UF0
3	EB200P-2 Evaluation Board	6ES7195-3BE00-0YA0
4	SIMATIC HMI KP8	6AV3688-3AF37-0AX0

PROFINET Driver and SIMATIC CP 1625Dev development board

PROFINET Driver (https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6ES7195-3AA00-0YA0) is a development kit that allows the users to create their in-house PROFINET controller solutions. In this example application, PROFINET Driver controls and configures PROFINET IO Devices that are connected to the system regarding its user application algorithm.

The user application layer of this example is compiled with PROFINET Driver source code and PROFIdrive library that is delivered inside this example application. The user application calls IO-Base interface of PROFINET Driver to establish PROFINET communication with PROFINET IO Devices that are located on the network. To be able to send PROFINET IO data to IO Devices, the user application calls the related interface functions of IO-Base that is explained in the document *"IO-Base User Programming Interface for PN Driver"*. To be able to provide precise control on motion control devices and PROFINET IO Devices, the PROFINET Isochronous Realtime feature of PROFINET Driver has been used in this example. This example application can be considered as an example of PROFIdrive controller as well as Isochronous Realtime Controller since both features have been demonstrated.

Since the user application algorithm of this example application requires all PROFINET IO data to be handled within a restricted time, PROFINET Isochronous Realtime mode communication must be provided by PROFINET Driver. Due to only CP1625 Host and CP1625 Stand-alone variants of PROFINET Driver provide such a capability, one of these variants must be used as PROFINET IO Controller in the example application. In the next chapters, the usage of both variants are explained. The user who is a member of PROFINET organization PI (https://www.profibus.com) is able to find more detailed information about PROFINET Isochronous Realtime mode, the user can refer *"Isochronous Mode Guideline"** document.

SINAMICS S210 servo drive system

The SINAMICS S210 (https://mall.industry.siemens.com/mall/de/WW/Catalog/Product/6SL3210-5HB10-1UF0) is a single-axis servo motor drive, it is designed for connection

to SIMOTICS S-1FK2 synchronous servomotors. PROFINET RT/IRT is available for connection to a higher-level control system.

In order to establish communication between the PROFINET Driver and SINAMICS S210, PROFINET Driver configures SINAMICS S210 at the initialization step, and activate the topology-based initialization. SINAMICS S210 also imports the telegram settings from the PROFINET Driver.

PROFINET Driver communicates with SINAMICS S210 servo drive systems over PROFIdrive telegram messages. PROFIdrive telegrams are used to cyclically transfer IO data in a defined format between the PROFINET IO controller and IO devices that support PROFIdrive. By means of the telegram messages, this example application is able to receive the position of servos and adjust the velocity of servos.

For more detailed information about PROFIdrive technology and PROFIdrive telegram messages, the user who is a member of PROFINET organization PI (https://www.profibus.com) can refer to PROFIdrive specification *"PROFIdrive Technical Specification for PROFIBUS and PROFINET"*.

EB200P-2 evaluation board

EK-ERTEC 200P PN IO Evaluation kit for PROFINET IO Device development (https://mall.industry.siemens.com/mall/de/WW/Catalog/Product/6ES7195-3BE00-0YA0), this kit uses ERTEC 200P-2 Evaluation Board EB200P-2 as base development board. This product can be used in the PROFINET IO Device development process by the PROFINET IO device manufacturers.

Responsibility of EB200P-2 Evaluation Board in this example application is to set its output according to the command from PROFINET Driver. PROFINET Driver updates I/O data of the EB200P-2 Evaluation Board. By means of the output data of EB200P-2 Evaluation Board, external systems such as flashing led in this example application can be controlled for any purpose.

SIMATIC HMI KP8

SIMATIC HMI KP8 (https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6AV3688-3AF37-0AX0) is a Key Panel that is containing 8 short-stroke switches with multi-colored LEDs. It supports PROFINET communication. This HMI device shows you operating states of the system. In addition to that, it is used to get user inputs and transmit them to a super-ordinated system over PROFINET.

In this example application, PROFINET Driver receives HMI KP8's output messages and set the color of KP8's button regarding to program state. The received outputs from KP8 are transmitted to the user application over PROFINET Driver's user interface and these input messages are interpreted by the user application. According to these inputs, user application sets the system state.

2.1.1 Software layer structure overview

This section describes the general architecture of the PROFIdrive controller example application.

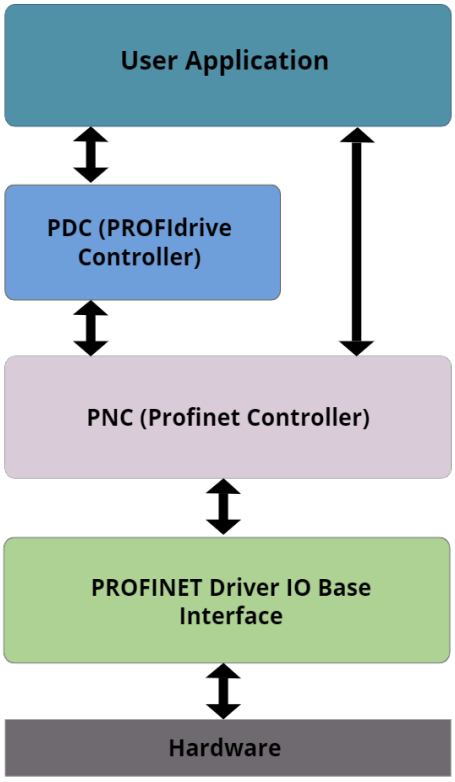

Fig. 2.2: Architecture of a PROFIdrive controller

For implementation of given PROFIdrive example application, it is necessary to use the PROFIdrive Controller layer and the Communication layer. Beside functionalities mentioned below, the PDC provides interface for motion control applications.

The PDC layer implements following PROFIdrive specification related functionalities:

- Handling the PROFIdrive state machine (e.g. switch an axis ON/OFF)
- Running an axis in velocity or position mode
- Reading and writing parameters
- Mapping telegrams
- Handling sign of life
- Handling faults and alarms

Note: All PROFIdrive related IO data are handled by PDC layer. When the PROFIdrive related IO data is ready, it is sent to PNC layer in order to transmit in PROFINET context.

The PNC layer implements PROFINET related functionalities:

- Cyclic data handling
- Acyclic data handling
- Alarm data handling
- Clock synchronization

Note: The PNC layer is a wrapper for PROFINET Driver IO Base interface, since this adaptation makes the usage easier.

2.2 Application layer structure

Inside PROFIdrive motion controller example application there are two interface layers that simplify the usage of PROFIdrive devices, which are connected to a PROFINET controller system.

The PNC (PROFINET Controller) API is used to provide PROFINET controller related functionalities like cyclic, acyclic and alarm data handling at application layer in a standardized way. So, it is possible to use the whole example application on different runtime systems by just implementing a proper wrapper for target system.

The PDC (PROFIdrive Controller) API is used to provide main PROFIdrive functionalities in a more abstract way. It provides functions to run connected axis with velocity or position set points with respect of chosen PROFIdrive application class and take over all necessary normalization, calculation and mapping in internal modules.

Both application interfaces provide standardized data types and functions to user application layer. That means public PNC function can be used in user application level as well as public PDC functions. PDC is using PNC functionalities as well. In addition to that, sources will include also internal functions and data types which are used by standard application layer (for basic controller application structure and elements).

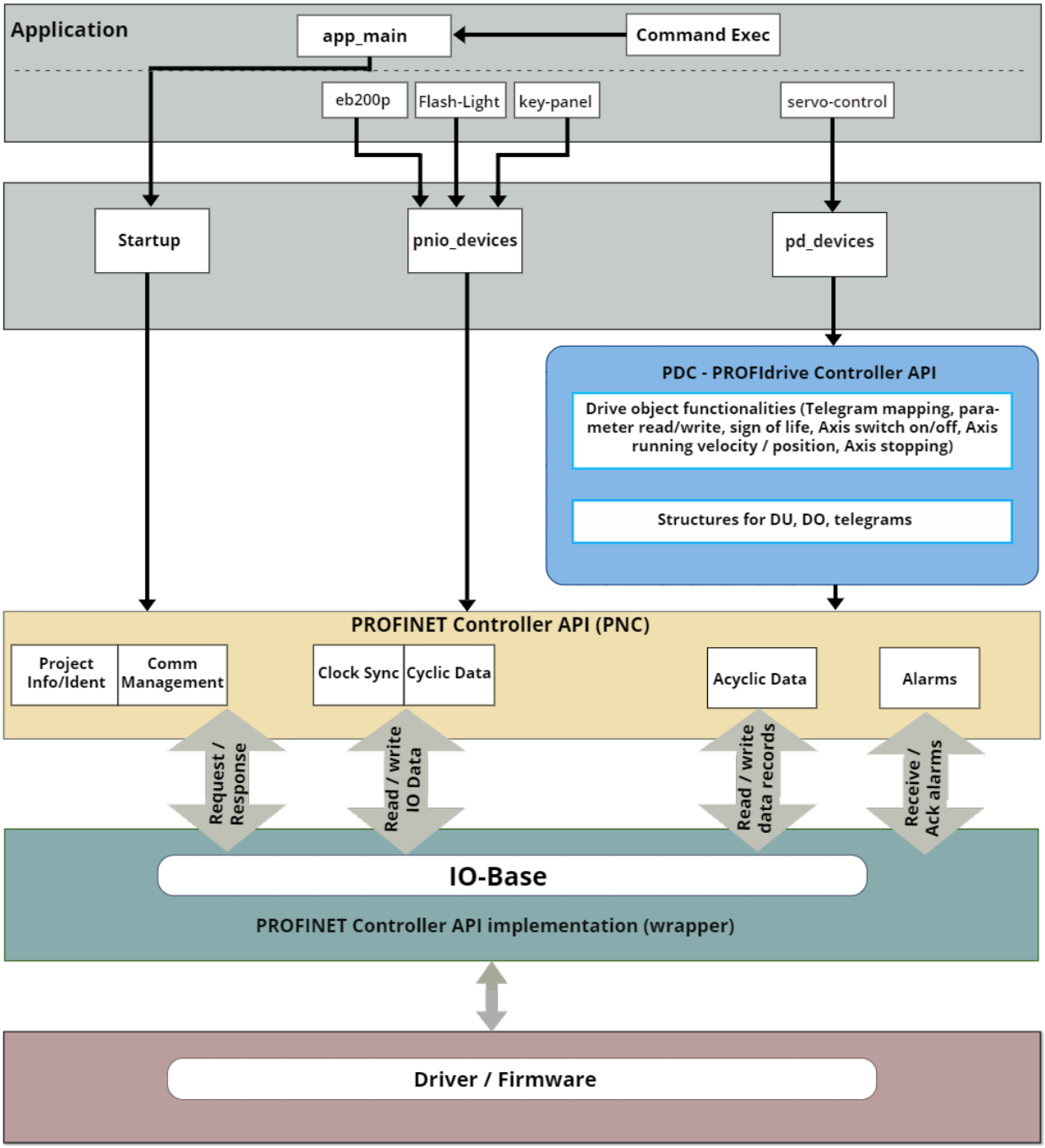

Fig. 2.3: Application Layer Structure

2.3 Runtime structure

After startup phase the example application runs in a main cycle divided into system and application parts. In system part all internal, not time critical tasks are executed. In application part the user can implement any background tasks by use of appropriate PNC and PDC functions. If there are configured and connected devices with cyclic data exchange, the cyclic part is separated into an ordered sequence of PROFINET and appropriate profile parts. The internal system part handles reading and writing of PROFINET input and output data (of submodules) and embeds the internal cyclic PROFIdrive related parts, like mapping raw data to used PROFIdrive telegrams. The user then will be able to implement cyclic related programs in Servo part.

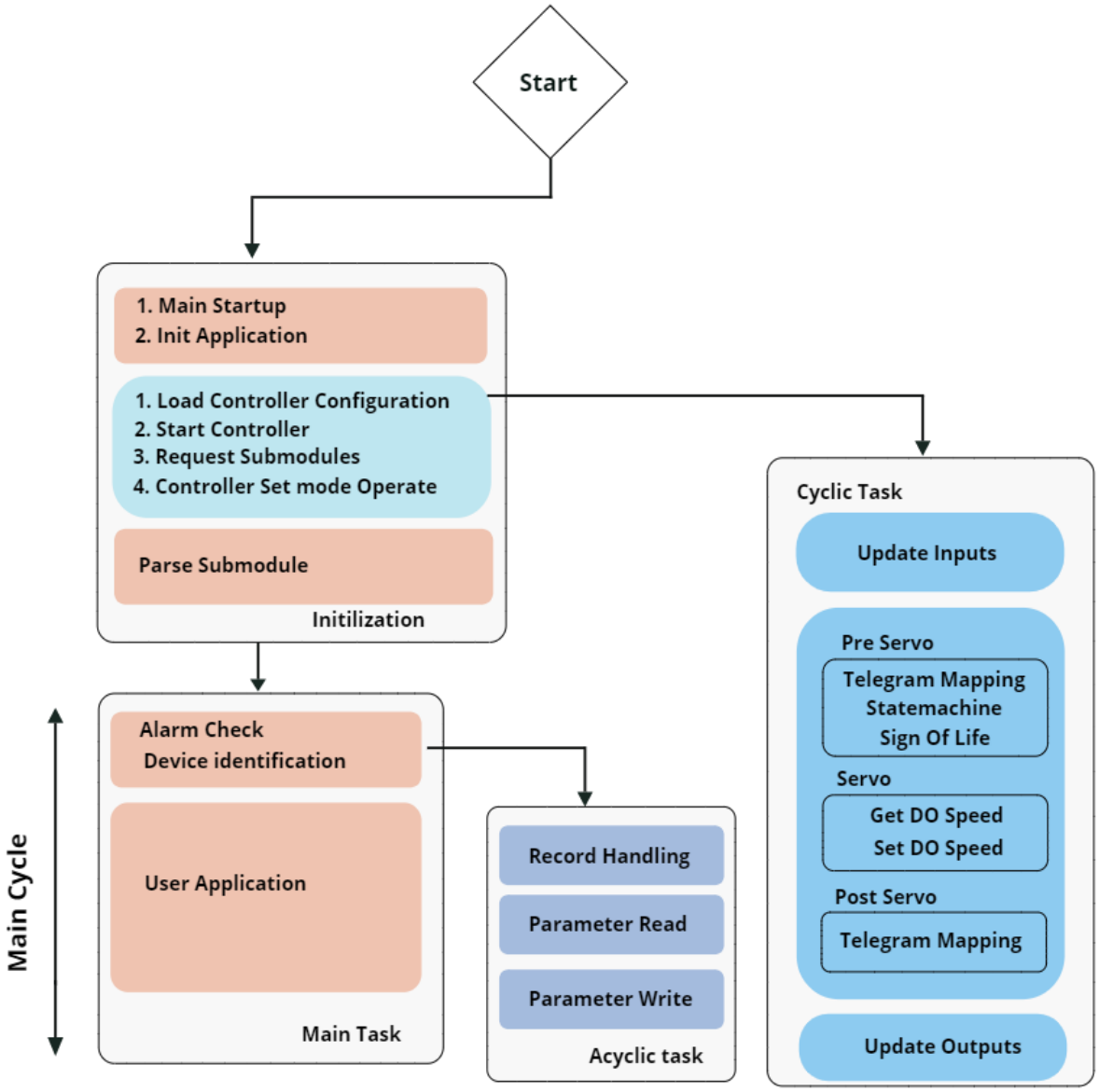

Fig. 2.4: PROFIdrive Controller Application Runtime Structure

2.3.1 Execution flow of the example application

The execution flow of the example application can be seen as a flow chart at Fig. 2.5.

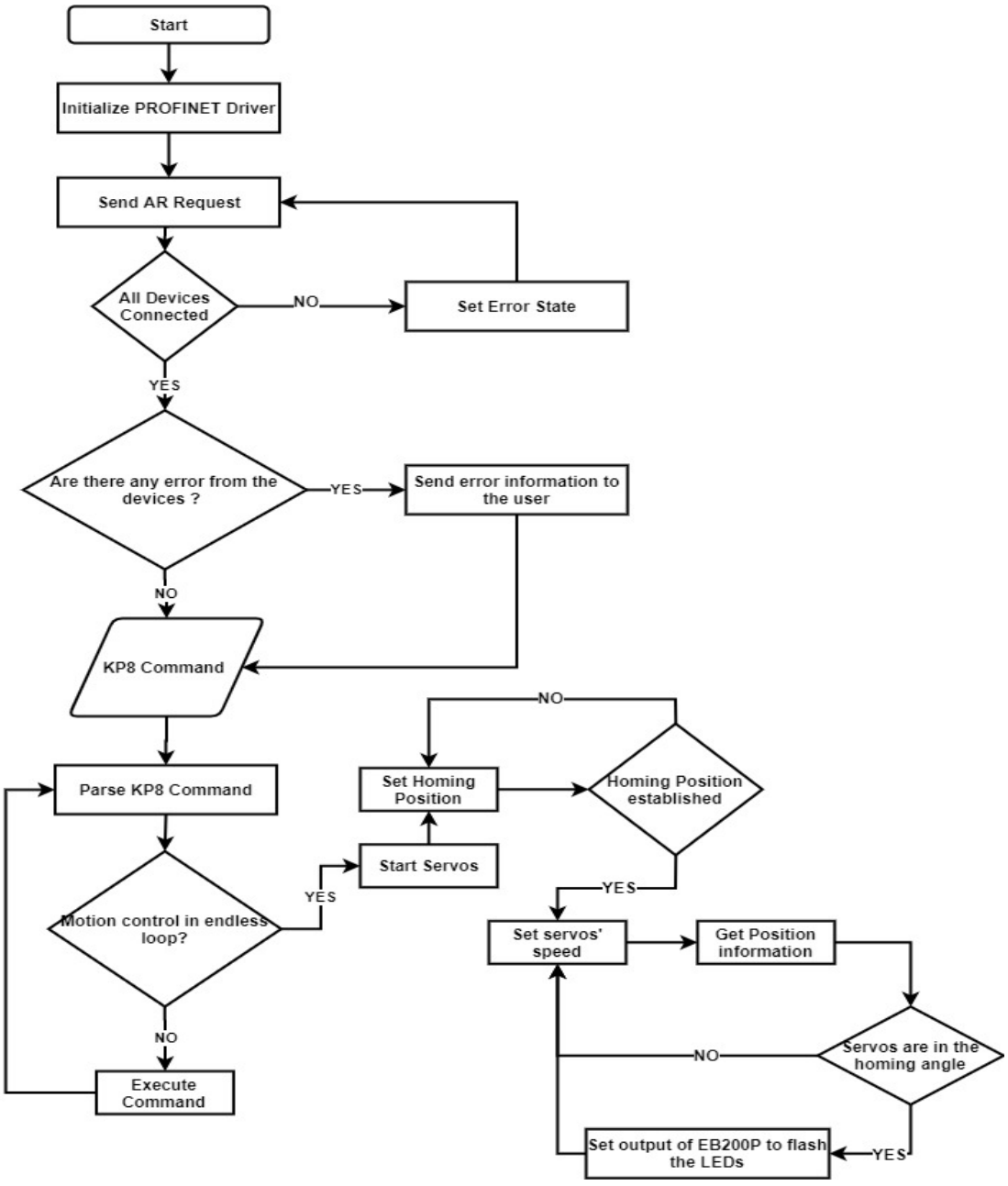

Fig. 2.5: Execution flow

The execution flow of the example application and critical steps are explained below.

System Startup

Initialization of the system is initiated by calling `ctrl_Startup()` inside the main entry. The initialization phase consists of the following steps:

Table 2.2: Initialization Steps

Step	Function	Description
1	ctrl_Startup()	This function starts all PROFINET Driver's tasks and calls all necessary functions in order to establish PROFINET communication.
2	PNC_ctrl_load_config()	Configuration XML which is produced by engineering tools is loaded to PROFINET Driver over IO-Base interface.
3	PNC_ctrl_start()	Services that are required to start PROFINET Driver are initiated in this function. Inside this function, callback functions that are necessary in order to communicate with IO-Base interface are registered.
4	PNC_submodList_read_req()	Submodule parameters of each device are transported to the user application level over the related IO-Base function. During communication with PN IO Devices, these parameters are used to control the related parameters of PN IO Devices that are located on the PROFINET network.
5	PNC_ctrl_set_mode()	The mode of PROFINET Driver is set to OPERATE in order to initiate PROFINET Driver's services.
6	APP_Common_Init_Us_Param()	Initialization of global user data related to the this example application.

Note: For more detailed information about callback mechanism, please refer to *"IO-Base User Programming Interface for PN Driver"* document.

Registered callback functions

Callback functions that are used in this example applications are explained below.

Table 2.3: Callback Functions

Callback Function	Description
l_cbf_ctrl_diag_resp()	This function is called by IO-Base interface when a response for the diagnostic request arrives from the PN IO device.
l_cbf_mode_ind()	This function is called by IO-Base interface when the local mode of PROFINET Driver is changed.
l_cbf_ds_read_conf()	This function is called by IO-Base interface when read data record arrives from the PN IO device.
l_cbf_ds_write_conf()	This function is called by IO-Base interface when write data record arrives from the PN IO device.
l_cbf_alarm_ind()	This function is called by IO-Base interface when an alarm is received from the PN IO device. All alarms are handled inside this callback function.
l_cbf_dev_act()	This function is called by IO-Base interface when the result of activation/deactivation message arrives.
l_cbf_opfault_ind()	This function is called by IO-Base interface when a violation of isochronous real-time mode occurs.
l_cbf_startop_ind()	This function is called by IO-Base interface to inform the start of isochronous real-time data processing.

Establishment of PROFINET communication

The establishment process of PROFINET communication is started inside the `ctrl_Startup()` function by calling `IOB_ctrl_start()` function. PROFINET Driver tries to establish an Application Relation (AR) with the PN IO devices that are provided in engineering XML input.

Table 2.4: Device return

Function	Description
alrm_DeviceReturn()	For every successful Application Relation establishment with each PN IO device, the user application layer is informed over `l_cbf_alarm_ind` callback function with alarm type `PNIO_ALARM_DEV_RETURN`.

Device failure control

In case of a failure in PN IO devices which are located on the PROFINET network, alarm information is noticed to the user application over `l_cbf_alarm_ind` callback function with the `PNC_PNIO_ALARM_DEV_FAILURE` flag.

By means of `alrm_DeviceFailure` function, the user application layer is able to determine the PN IO device that has alarm information.

Table 2.5: Alarm device return

Function	Description
alrm_DeviceFailure()	The user application is informed in case of alarm information is available in one of the PN IO devices.

Command execution

In every communication cycle time the user application is informed by `l_cbf_startop_ind()` callback function. The IO data of all devices must be processed within a restricted time inside this function handler. IO data that are coming from IO devices are updated and stored in the global buffer. Inside the function `ctrl_cycUpdateData()`, data is processed regarding whether it is coming from PROFIdrive devices or other PN IO devices. If the data belongs to one of the other PN IO devices expect PROFIdrive, it is handled by `usr_IoDevs()` function.

Table 2.6: Command execution functions

Function	Description
KeyPanel_Handler()	This function handles the IO data of HMI KP8.
EB200P_Device _Handler()	This function handles the IO data of EB200P-2 Evaluation Board.

If `devNo` is equal to the value of `L_DEV_ID_KEYB` that is defined in the header file *"pn_device.h"* which is located under *"ProfidriveAppl/src/pnio_common"* folder, `KeyPanel_Handler()` is executed. Inside this function, the output data from the HMI KP8 is processed regarding the definitions in *"ProfidriveAppl/src/pnio-devices/KP8/kp8.h"*.

There are 8 types of commands that can be executed over HMI KP8. Each command is assigned to a different button of HMI KP8. The definition of the commands can be found in *"ProfidriveAppl/src/command-exec/cmd_exec.cpp"*.

Control of the PROFIdrive devices

After the successful establishment of AR with the PROFIdrive devices which are 2 SINAMICS S210 in this example application, the user application will be informed in every IRT clock cycle over `l_cbf_startop_ind()` callback function that is registered at the startup sequence. The handling of the PROFIdrive state machine, the sign of life, and the absolute degree update are handled in this function.

Table 2.7: Control function of PROFIdrive devices

Step	Function	Description
1	pd_state_machine()	This function handles PROFIdrive state machine mechanism regarding to PROFIdrive technical specification.
2	pd_SOL()	This function handles of Sign Of Life.
3	pf_read_XIST2_cyc()	This function updates absolute degree value of encoders.

IRT send clock is selected 500 us for CP1625 Host variant and 1 ms for CP1625 Stand-alone variant for this example application, the user can adjust these values over the engineering interface.

Motion control in endless loop

After the successful initialization of PROFINET Driver and AR establishment with the PN IO devices as described above, in case the user sends the “Motion control in endless loop” command over HMI KP8 device, the system starts to run in endless loop as described the Table 2.8.

Motion control in an endless loop is a state machine that can be enabled over HMI KP8 panel by the user. In this state machine, the system is running forever until the user stops. Execution steps in this state machine:

Table 2.8: Steps of motion control in endless loop

Step	Function	Description
1	Servo_Init()	Initializes of servos to be able start to move.
2	Servo_Home_Axes()	Sets the position angle of servos to the homing position that is defined in *"servo-control.hh"* which is located under *"ProfidriveAppl/src/servo-control/"*. When the homing position established both servos are stopped.
3	DemoWallStateMachine()	Starts to run both servos from the homing position. The velocity of servos are determined regarding the defined values in *"demowall-statemachine.cpp"* which is located under *"ProfidriveAppl/src/"* The speed value of servos is increased until it reaches `MAX_SPEED_1000RPM` value. If the velocity is equal to `MAX_SPEED_1000RPM` value, then the velocity of servos is decreased slowly until it reaches to 0.
4	Servo_Get_Position()	The actual positions of both servos are updated in every cycle inside `l_cbf_startop_ind()`.
5	Flash_LED_ON()	If the position of servos matches the homing position, the related output of the EB200P-2 Evaluation board is set to high in order to flash the LEDs that are connected to the EB200P-2 Evaluation board.

2.4 Source code structure

The following table describes all source structure elements and the corresponding content referring to the functionality inside the PROFIdrive Controller Application. On the table below, the "mandatory" statement refers that these components are compulsory, and they are required by PROFINET Driver library, on the contrary to the "mandatory" statement, the "optional" statement refers that there is no obligation to implement these APIs in case the user prefers to adapt this source code to his application. Such as "math.h", if application does not require this library, the user can remove it from the "common.h" header file. However, files that are stated as "mandatory" must be used in the user's custom application.

Table 2.9: Source Structure

Folder	File	Description
Common	common.h	Common includes and definitions (mandatory). Additional common includes and definitions (optional) (for example <math.h>).
	trace.cpp trace.h	Involves trace modules (mandatory). Header includes macro definitions for trace endpoint (for example “printf” on a console).
	utils.cpp utils.h	Header includes common macros (mandatory). Implementation of some macro content is optional for PNC wrapper layer specific utilities (optional). The user is free to use some of these macros according to his application
pnc	pnc_api.cpp pnc_api.h	PROFINET controller application interface Implementation of PNC functions and data types (calls system specific functions of PNC wrapper).
	pnc_wrapper.cpp pnc_wrapper.h	Implementation of wrapper functions (mandatory). Content depends completely on used system and is connected to defined functions in PNC.
pnio_devices	kp8.cpp kp8.h	KP8 general button input control structure and color management, KP8 as an IOD is declared in header file.
	tm-didq-10-24v.cpp kp8.h tm-didq-10-24v.h	Timer DIDQ-10-24v module internal data structure definitions, input / output control management.

Table 2.10: Source Structure

Folder	File	Description
pdc	pdc_acyc.cpp pdc_acyc.h	PROFIdrive controller application interface Implementation of PDF functions and data types Handling of PROFIdrive Parameter functionalities (calls system specific functions of PDC wrapper).
	pdc_api.cpp pdc_api.h	Handling of cyclic PROFIdrive related data implementation of state machine, fault handling, sign of life, telegram data mapping, interpolator and position Controller.
	pdc_int.h	Type and structure definitions for PROFIdrive related content.
	pdc_main.cpp	Main implementations of PROFIdrive related content.
ctrl	ctrl_acyc_data.cpp ctrl_acyc_data.h	Processing of acyclic data updates and data records.
	ctrl_alrm_data.cpp ctrl_alrm_data.h	Handling of alarm data.
	ctrl_cyc_data.cpp ctrl_cyc_data.h	Handling of cyclic update of data (called / triggered by PNC API). Handling of change of data status Implementation for cyclic thread (for RT only configurations).
	ctrl_cycle.cpp	Cyclic operation of the controller and includes functionality for create reference numbers and device identification.
	ctrl_main.cpp	Main function of application triggers startup and cyclic operation phase Enables stop functionality for controller (PNC API).
	ctrl_startup.cpp	Handling of startup phase Initialization of Application Loading of controller configuration. • Start controller. • Readout submodule configuration. • Change controller mode (PNC API). • Creation of cyclic (RT only configuration) and interpolator thread.
	ctrl.h	Common header for the controller (basic structures for controller and device(s)).

Table 2.11: Source Structure

Folder	File	Description
demo-app	cmd_exec.cpp cmd_exec.h	Execute incoming commands from main application or Demo-app state machine. All system upper level commands are proceeded.
	demoapp-Statemach.cpp	Demo-app state machine is controlled. It controls the word to be shown to the user according to system's speed.
	eb200p.cpp	Control of EB200P IO data as IOD in cyclic communication.
	flash-light.cpp flash-light.h	EB200P IOD control from user application regarding to demo-app application's special application requirements.
	key-panel.cpp	KP8 input / output control from user application.
	servo-control.cpp servo-control.h	Controlling drive unit according to application requirements. Control of velocity, homing angle and position control.
	app_main.cpp	Check all command sources, acknowledge all devices errors, user parameter initialization. Control of application specific application.
	demoapp-common.h	Common data types and definitions for Demo-app
pd_devices	pd-servo-DU.cpp pd-servo-DU.h	Velocity setting, Axis tipping, Axis start / stop management.
	pdc_acyc.cpp pdc_acyc.h	Handling of PROFIdrive Parameter functionalities.
	pdc_api.cpp pdc_api.h	PROFIdrive controller application interface. Implementation of PDC functions and data types.
	pdc_main.cpp	Main implementations of PROFIdrive related content.
pn-common	pn_device.h	Common data structure definition for PROFINET devices.

2.5 PDC - PROFIdrive controller

In PDC, user related part includes just necessary and easy to use functions and data types for handling a PROFIdrive device in the controller application.

The internal handling of PROFIdrive controller in connection with PROFINET controller interface is implemented by the standard application and internal PDC parts. The interface also handles all functions and data from user related part. The sources for the internal part are also visible / present but they shouldn't be known or modified by the user.

2.5.1 PDC Data types

2.5.1.1 Standard data types

"PROFIdrive Technical Specification for PROFIBUS and PROFINET" is referenced for data types of PDC interface. The type definitions for used runtime system are placed in the header file of PDC API function interface.

Note: The complete list of standard type definitions can be found in file "pdc_api.h".

Table 2.12: Standard Data Types

Data type	**PROFIdrive Data Type (numeric identifier)**	**PDC Data type**
Boolean	1	PDC_BOOL
Integer8	2	PDC_INT8
Unsigned16	6	PDC_UINT16
Unsigned32	7	PDC_UINT32

2.5.1.2 Standard function return values

Besides function related return values "status" is describing the actual execution state of function. All possible returned values are grouped in separate ranges to classify the status more quickly.

Table 2.13: Standard Function Return Values

Return values	**Description**
0x00000000	Function executed successfully (without any error)
0x0001 – 0x0FFF	Function returns specific status
0x1000 – 0x10FF	User defined error codes
0x8000 – 0x8FFF	Error because of an internal execution
0xFFFFFFFF	Global error (not specified in detail)

2.5.1.3 PDC - configuration

Drive object reference

Reference to identify a drive object in list of all available drive objects.

Table 2.14: Drive Object Reference

PDC_DO_REF	**Description**
ID	Identification of drive object

Drive object properties

This structure represents all available properties as set by PROFIdrive parameters in the DO (Drive Object). For instance, parameter p922 is used to display the actual setting of the PROFIdrive telegram configuration in the DO. Please refer *“PROFIdrive Technical Specification for PROFIBUS and PROFINET”*.

Table 2.15: Drive Object Properties

PDC_DO_PROP	**Description**
ID	Identification of drive object (PDC_DO_REF)
DOtype	(p975.1) Manufacturer specific
DOtypeClass	(p975.5) CU, AXIS, INFEED
HWversion	Hardware version(s) (p975)
SWversion	Software version(s) (p975)
SpeedNormVal	Speed normalization value
SpeedMax	Maximum speed
SpeedRes	Speed resolution
G1MeasType	Measurement type of G1 (p979)
G1Res	Resolution of G1 (p979)
G1RefStratType	Reference strategy type of G1 (p979)
SolTolErr	Sign of life tolerated errors (p925)
FaultBufSize	Fault buffer size (p944-p952)

Note: It may be used to set / write (some) properties of the DOs in future versions.

Drive object status

Table 2.16: Drive Object Status

PDC_DO_STAT	**Description**
InOp	Device is in operation
InPositioning	Device is in Positioning mode
InVelocity	Device is in Velocity mode
InError	Device is in Error state

Telegram content

The telegram structure consists of all possible telegram elements, which can appear in implemented telegrams. The elements are mapped to input / output data by internal functions.

Table 2.17: Telegram Content

PDC_DO_TELEGRAM	**Description**
TelNo	Number of telegram
STW1	Control word 1
N_SOLL_A	Set point value (16 Bit)
N_SOLL_B	Set point value (32Bit)
STW1	Control word 2
. . .	. . .
ZSW1	Status word 1

Note: There is a lot of PDC_DO_TELEGRAM data type that is defined in PROFIdrive standard. In order to provide a less complex and understandable table in this document, only a few of them are mentioned at the table above, to be able to find all data types please refer to *"PROFIdrive Technical Specification for PROFIBUS and PROFINET"*.

2.5.1.4 Parameter handling

For each element of a Parameter, a corresponding structure is defined.

Table 2.18: Paramater Values

PDC_PAR_VALUE_DATA	**Description**
ParamNo	Number of parameter
ParamSubIdx	Sub index of parameter
NoOfElements	Count of parameter (single or sub index)
NoOfValues	Count of values (sub index)
nDataLen	Length of data
Format	Data format
nFormatLen	Length of data format
pi8Value. . . pf32Value	Reference to parameter values

Table 2.19: Paramater Description

PDC_PAR_DESC_DATA	**Description**
ParamDesc	Parameter data description

Table 2.20: Paramater Text

PDC_PAR_TEXT_DATA	**Description**
aParamText	Character array with parameter text

2.5.1.5 Fault handling

Table 2.21: Fault Handling

PDC_ERROR	**Description**
FaultEntryNo	Fault error entry number
FaultNo	Fault error code (p944)

2.6 PNC - PROFINET controller

In PNC the PROFINET functionalities of the controller are represented by functions and data types for the user application part, which are not related to a specific driver or platform solution. These functions and data types are oriented to PROFINET standards for cyclic, acyclic and alarm channel ARs. The implementation of these standardized functions then represents the usage of the target PROFINET controller platform with their specific functionalities. Therefore, for PNC it will be necessary to write a proper wrapper file for used controller type to adapt it to used platform. All application content is independent of used controller platform.

2.6.1 Data types

Basic data types from the C standard library are used by the PNC component.

2.6.1.1 Reference

Table 2.22: Submodule

PNC_GEO_ADDR_SUBMOD	Data type	Description
devNo	uint16_t	device / station number
API	uint32_t	application identifier
slot	uint32_t	slot number
subSlot	uint32_t	subslot number

Table 2.23: Submodule Reference

PNC_SUBMOD_REF	Data type	Description
GeoAddr	PNC_GEO_ADDR_SUBMOD	Geographical address (submodule) according to PROFINET standard
handleID	uint32_t	Handle for identification of communication object

Table 2.24: Device

PNC_GEO_ADDR_DEV	Data type	Description
devNo	uint16_t	device / station number

Table 2.25: Device Reference

PNC_DEV_REF	Data type	Description
GeoAddr	PNC_GEO_ADDR_SUBMOD	Geographical address (submodule) according to PROFINET standard
handleID	uint32_t	Handle for identification of communication object

2.6.1.2 Function return values

Table 2.26: PNC Function return values

Name	Data type	Description
PNC_RESULT	uint32_t	Geographical address (submodule) according to PROFINET standard
PNC_OK	uint32_t	Function executed successfully (without any error)
PNC_ERR	uint32_t	Error at internal Execution
PNC_xyz	uint32_t	Specific error description depending on returning function

2.6.1.3 Project ident

Table 2.27: PN IO Types

PNC_IO_TYPE (enum)	Value	Description
PNC_IO_IN	0	Input data type
PNC_IO_OUT	1	Output data type

Table 2.28: PN data types

PNC_PN_DATA_TYPE (enum)	Value	Description
PNC_DATA_RT	0	Realtime data
PNC_DATA_IRT	1	Isochronous Realtime data

Table 2.29: Submodule list

PNC_SUBMOD_LIST	**Data type**	**Description**
SmRef	PNC_SUBMOD_REF	Submodule reference
lenIn	uint32_t	Length of input data
dataType	PNC_IO_TYPE	Type of data (input/output)
lenOut	uint32_t	Length of output data
dataType	PNC_PN_DATA_TYPE	RT/IRT data
cycleTime	uint32_t	Cycle time in us
cacf	uint32_t	CACF controller application cycle factor
redFactor	uint32_t	Reduction Factor
phase	uint32_t	Phase
ti	uint32_t	Ti time in us
to	uint32_t	To time in us
modId	uint32_t	Module ID
submodId	uint32_t	Submodule ID

Table 2.30: Controller Information

PNC_CTRL_INFO	**Data type**	**Description**
CtrlName	Array of uint8_t [PNC_IP_LEN]	Controller name (a.k.a. PN name)
CtrlIp	Array of uint8_t [PNC_IP_LEN]	Controller IP address
SubnetMask	Array of uint8_t [PNC_IP_LEN]	Controller Subnet Mask
GatewayIp	Array of uint8_t [PNC_IP_LEN]	Controller Gateway IP address (a.k.a. Router IP)

Table 2.31: Controller Cycle Information

PNC_CTRL_CYCLE_INFO	**Data type**	**Description**
tcaStart	uint32_t	Time controller application start value (referring to cycle start) in us
tcaEnd	uint32_t	Time controller application end value (referring to cycle start) in us

Table 2.32: PN Controller Configuration

PNC_CONFIG	**Data type**	**Description**
pConfig	uint8_t *	Pointer to configuration data
configLen	uint32_t	Length of configuration data
pRema	uint8_t *	Pointer to remanent data
remaLen	uint32_t	Length of remanent data

Table 2.33: Device Information

PNC_DEV_LIST	**Data type**	**Description**
DevRef	PNC_DEV_REF	Pointer to configuration data
struct deviceIfProp	Array of uint8_t [PNC_PN_NAME_LEN]	Length of configuration data
DevName	Array of uint8_t [PNC_PN_NAME_LEN]	Device name (a.k.a. PN name)
DevIP	Array of uint8_t [PNC_IP_LEN]	Device IP address
SubnetMask	Array of uint8_t [PNC_IP_LEN]	Device Subnet Mask
GatewayIP	Array of uint8_t [PNC_IP_LEN]	Device Gateway IP ad-dress (a.k.a. Router IP)

2.6.1.4 Communication management

Table 2.34: Controller Mode

PNC_PN_DATA_TYPE (enum)	**Value**	**Description**
PNC_PNIO_GOOD	0	Offline
PNC_PNIO_BAD	1	Operate

2.6.1.5 Cyclic communication

Table 2.35: PROFINET IO data status

PNC_PN_DATA_TYPE (enum)	**Value**	**Description**
PNC_PNIO_GOOD	0	Status ok
PNC_PNIO_BAD	1	Status not ok

2.6.1.6 Acyclic communication

Table 2.36: PN Error Status

PNC_PN_DATA_TYPE (enum)	**Value**	**Description**
ErrCode	uint8_t	ErrorCode: Most significant word, most significant byte of PNIO Status
ErrDecode	uint8_t	ErrorDecode: Most significant word, least significant byte of PNIO Status
ErrCode1	uint8_t	ErrorDecode: Least significant word, most significant byte of PNIO Status
ErrCode2	uint8_t	ErrorCode2: Least significant word, least significant byte of PNIO Status
AddValue1	uint16_t	additional information 1
AddValue2	uint16_t	additional information 2

2.6.1.7 Alarms

Table 2.37: PN Alarm priorities Types

PNC_PN_ALARM_PRIO_TYPE (enum)	**Value**	**Description**
PNC_PNIO_APRIO_LOW	0	Alarm priority low
PNC_PNIO_APRIO_HIGH	1	Alarm priority high

Table 2.38: PN Alarm Types

PNC_PN_ALARM_TYPE (enum)	**Value**	**Description**
PNC_PNIO_ALARM_DIAGNOSTIC	0x01	Diagnostic alarm PN IOD
PNC_PNIO_ALARM_PROCESS	0x02	Process alarm from PN IOD
PNC_PNIO_ALARM_PULL	0x03	Pull alarm from PN IOD
PNC_PNIO_ALARM_PLUG	0x04	Plug alarm from PN IOD
PNC_PNIO_ALARM_STATUS	0x05	Status alarm from PN IOD
PNC_PNIO_ALARM_UPDATE	0x06	Update alarm from PN IOD
PNC_PNIO_ALARM_REDUNDANCY	0x07	Redundancy alarm from PN IOD
PNC_PNIO_ALARM_CONTROLLED_BY-_SUPERVISOR	0x08	Controlled by Supervisor alarm from PN IOD
PNC_PNIO_ALARM_RELEASED_BY- _SUPERVISOR	0x09	Released by Supervisor alarm from PN IOD
PNC_PNIO_ALARM_PLUG_WRONG	0x0A	Wrong Plug alarm from PN IOD
PNC_PNIO_ALARM_RETURN_OF- _SUBMODULE	0x0B	Return of submodule alarm from PN IOD
PNC_PNIO_ALARM_DIAGNOSTIC- _DISAPPEARS	0x0C	Diagnostic disappeared alarm from PN IOD
PNC_PNIO_ALARM_MCR_MISMATCH	0x0D	MCR mismatch alarm from PN IOD
PNC_PNIO_ALARM_PORT_DATA-_CHANGED	0x0E	Port data changed alarm from PN IOD
PNC_PNIO_ALARM_SYNC_DATA-_CHANGED	0x0F	Sync data changed alarm from PN IOD
PNC_PNIO_ALARM_ISOCHRONE-_MODE_PROBLEM	0x10	ISO mode problem alarm from PN IOD
PNC_PNIO_ALARM-_TIME_DATA_CHANGED	0x12	Time data changed alarm from PN IOD
PNC_PNIO_ALARM_UPLOAD_AND-_STORAGE	0x1E	Upload and storage alarm from PN IOD
PNC_PNIO_ALARM_PULL_MODULE	0x1F	Pull module alarm from PN IOD
PDC_PNIO_ALARM_DEV_FAILURE	0x00010000	Device failure alarm from PN IOD
PDC_PNIO_ALARM_DEV_RETURN	0x00010001	Device return alarm from PN IOD

Table 2.39: PN Alarm data

PNC_CTRL_ALARM_DATA (enum)	**Value**	**Description**
alarmType	PNC_PN_ALARM_TYPE	Original alarm-identifier from alarm-data-block
alarmPrio	PNC_PN_ALARM_PRIO	Priority of alarm
devNum	uint32_t	Device number that sends alarm data
slotNum	uint32_t	Slot number
subslotNum	uint32_t	Sub slot number

How to use the example application

3

There are two possible use cases of the example application. First use case is PROFINET Driver CP1625 Stand-alone variant with CP 1625Dev board. The other use case is PROFINET Driver CP1625 Host variant with CP 1625Dev board. The user needs to use the Linux operating system to create the necessary firmware and applications for both PROFINET Driver variants.

3.1 CP1625 Stand-alone use case

Overview

This section describes the steps for commissioning PROFINET Driver under a custom Linux operating system, built with Buildroot.

Customers are responsible for building PROFINET Driver source code and preparing target operating system that is Buildroot image for SIMATIC CP 1625Dev board. The required steps are explained below.

Requirements

The example application requires a Buildroot image for the SIMATIC CP 1625Dev Board and PROFINET Driver application. You must configure your board for stand-alone usage. For more details, please refer to CP 1625 operating instructions in (https://support.industry.siemens.com/cs/ww/en/view/109756564). You also need a cable for serial connection. We recommend using TTL-232R-RPI cable from FTDI.

3.1.1 Preparing Buildroot Image

CP 1625 Dev board requires a buildroot image to boot up. You must prepare buildroot image for stand-alone usage. Please refer to document *"Quick Start PROFINET Driver V2.3"* and chapter *"5.2 Installing Buildroot Image"*. To prepare Buildroot image, the user must follow the instructions which are explained under this chapter.

3.1.2 Building the application for stand-alone usage

To be able to build the example application, firstly the user must build PROFINET Driver as static library and copy it under the *"../PROFIdrive_Controller_Application/"*, run following commands:

```
$ cd {Work-Space}/PROFINET Driver/src/examples/lib/cp1625_
↪standalone/build
$ make
```

libpndriver.a will be created under *"../PROFINET-Driver/src/examples/lib/cp1625_standalone/lib/"* after the build process has been successfully completed. The user must copy *libpndriver.a* under *"../PROFIdrive_Controller_Application/CP1625_Standalone/pnlib-standalone/"*.

To build the demo application, the user must edit two areas of the makefile which is located under *"PROFIdrive_Controller_Application/CP1625_Standalone/build/"* directory;

1. Example application is running on CP 1625Dev board environment. Due to this reason, build process must be handled via cross-compiler. The user must give compiler location- The user must give compiler location information to the makefile. `BUILD_ROOT_DIR` is a makefile variable which holds cross-compiler location information. The user must assign buildroot location to this variable. The makefile will use the cross-compiler which is located in the buildroot package.

   For example:

   ```
   BUILD_ROOT_DIR=/home/siemens-pc/buildroot-2021.02.1/
   ```

2. Example application is using PROFINET Driver internal functions. Due to this reason, the makefile must know the location of PROFINET Driver. The user must provide the source code directory of PROFINET Driver by editing `PN_STACK_DIR` variable which is declared in makefile.

   For example:

   ```
   PN_STACK_DIR= /home/siemens-pc/PNDriver/src/source
   ```

After editing make file the user can build the application, run following commands:

```
$ cd  PROFIdrive_Controller_Application/CP1625_Standalone/build
$ make
```

Binary file *profiDriveControllerApp.bin* should be crated under *"../PROFIdrive_Controller_Application/CP1625_Standalone/"* directory.

3.1.3 Running the application on the target

To run PROFINET Driver application on SIMATIC CP 1625Dev board, you need to transfer the application binary and the configuration file to the board. Buildroot' s overlay structure is used for integrating files in the Linux image. To do so, follow the procedure below.

Procedure

1. Copy the files *profiDriveControllerApp.bin* which is produced by the build process of the makefile, *homing* and *"PROFIdrive_Controller_Application/Configuration/TIA/standalone/demo-app-sa.xml"* which is given to the user inside example application package to the directory *"/buildroot-2021.02.1/board/cp1625/rootfs_overlay/root"*.
2. To start firmware application automatically on CP 1625Dev board, edit the file *"/buildroot-2021.02.1/board/cp1625/rootfs_overlay/root/.profile"* as follows:

- Comment the last line like the figure below;

```
#!/bin/sh

# You may uncomment the following lines if you want `ls' to be colorized
export LS_OPTIONS='--color=auto'
alias ls='ls $LS_OPTIONS'
alias ll='ls $LS_OPTIONS -l'
alias l='ls $LS_OPTIONS -lA'

# Some more alias to avoid making mistakes:
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

test ! -c /dev/linux_irte_drv && modprobe linux_irte_drv
cd /root
#/root/test_app
```

Fig. 3.1: .profile file with commented line

- Write */root/profiDriveControllerApp* to last line of *".profile"* file like the figure below.

```
#!/bin/sh

# You may uncomment the following lines if you want `ls' to be colorized:
export LS_OPTIONS='--color=auto'
alias ls='ls $LS_OPTIONS'
alias ll='ls $LS_OPTIONS -l'
alias l='ls $LS_OPTIONS -lA'

# Some more alias to avoid making mistakes:
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

test ! -c /dev/linux_irte_drv && modprobe linux_irte_drv
cd /root
#/root/test_app
/root/profiDriveControllerApp
```

Fig. 3.2: .profile file with the execution command line

1. Run the following command in "~/buildroot-2021.02.1"

```
$ make all
```

4. The image called *"uboot+linux.bin"* will be created in *"/buildroot-2021.02.1/output/images"* directory. Use this image to run the application as described in document *"Quick Start PROFINET Driver V2.3"* chapter *"5-Quick start for CP1625Stand-alone"*.

3.2 CP1625 Host use case

Overview

This section describes the steps required for commissioning PROFINET Driver under a custom Linux operating system, built with Buildroot (https://buildroot.org/). In CP1625 Host variant, it is possible to run PROFINET Driver V2.3 distributed on Linux host PC and on CP 1625 PCIe card. With the help of the processing power of the host PC, you are able to profit both from the features of CP 1625 board and from the variety of applications on the host PC at the same time.

Customers are responsible for building PROFINET Driver source code and preparing target operating system that is Buildroot image for CP 1625Dev board. The required steps are explained below.

Requirements

For this example application , the building steps which are given below are explained on Debian 10.10 with Linux kernel V4.19 including RT patch; therefore, the user needs a Linux host development PC running on Debian 10.10 with kernel V4.19 including RT patch. Please refer to document *"Quick Start PROFINET Driver V2.3"* and section *"6-Quick Start for CP 1625 Host"* for installation. You also need a CP 1625Dev board inserted into a free PCIe slot on the host machine. You must configure it for PCIe usage. For more details, please refer to CP 1625Dev operating instructions in (https://support.industry.siemens.com/cs/ww/en/view/109756564).

3.2.1 Preparing Buildroot image

Please refer to the document *"Quick Start PROFINET Driver V2.3"* and section *"6.2 Installing Buildroot Image"* for installation of Buildroot image.

Adding CP 1625 firmware to the buildroot image

You need the firmware application to run on CP 1625 board. Therefore, you need to transfer the application's binary to the board. For details about how to create the firmware application binary, please refer to the document *"Quick Start PROFINET Driver V2.3"* and section *"6.4 Building PROFINET Driver application"*. Buildroot's overlay structure is used for integrating files in the Linux image.

After the following steps from quick start document, *uboot+linux.bin* will be created under *"/buildroot-2021.02.1/output/images"* directory after the build process

has been successfully completed. The user must copy *uboot+linux.bin* under *"../PROFIdrive_Controller_Application/CP1625_Host/"*. PROFINET Driver application will search for this image under the same path where it is located.

3.2.2 Installing PN Device Driver

To use CP 1625Dev board under Linux, PROFINET Driver requires the PN Device Driver (PnDevDriver). For details on how to use PnDevDriver under Linux Debian 10.10 with kernel V4.19, see section *"3.6 Using the PN Device Driver"*.

To use the network adapters under Linux, PROFINET Driver requires the PN Device Driver (PnDevDriver). You must compile the modules yourself as described section *"3.6 Using the PN Device Driver"*. After the building PnDevDriver successfully, *PnDev_DriverU32.so* must be copied to *"PROFIdrive_Controller_Application/CP1625_Host/"* directory.

3.2.3 Loading the PN Device Driver

Procedure

To load the PN Device Driver temporarily (until the next restart), use the following command in the same folder in which the *"PnDevDrv_32.ko"* or *"PnDevDrv_64.ko"* module is located:

- **for 32-bit systems:**

```
$ sudo insmod PnDevDrv_32.ko
```

- **for 64-bit systems:**

```
$ sudo insmod PnDevDrv_64.ko
```

After the loading device driver, the user must bind device driver to appropriate PCI Card. Further information to bind PCI card please refer to *"Quick Start PROFINET Driver V2.3"* and section *"3.6 Using the PN Device Driver"*.

3.2.4 Building the example application

Procedure

To build the example application, follow the steps below:

1. Build PROFINET Driver as a library in PROFINET Driver source directory on your local workspace *"src/examples/shared/cp1625_host/linux_host/build/"*. Make sure that *libpndriver.a* has been created in *"src/examples/shared/cp1625_host/linux_host/build/lib"* folder. Copy *"libpndriver.a"* to *"PROFIdrive_Controller_Application/CP1625_Host/pnlib-host/"*.

```
$ make all
```

2. Since this example application is using PROFINET Driver common functions, the user must edit makefile which is located under the path *"PROFIdrive_Controller_Application/CP1625_Host/build/"*. In makefile `PN_STACK_DIR` variable is used to define location of PROFINET Driver stack. The user must assign PROFINET Driver' s local path to `PN_STACK_DIR`.

For example:

```
PN_STACK_DIR=/home/siemens-pc/PNDriver/src/source
```

1. Build example application in *"PROFIdrive_Controller_Application/CP1625_Host/build/"*.

```
$ make all
```

3.2.5 Running the example application

Requirements

Before you start PROFIdrive example application, be sure that firmware image *uboot+linux.bin*, shared object file *PnDev_DriverU32.so* and configuration file *homing* are on the same directory with executable of application.

Procedure

To start PROFIdrive example application, follow steps below;

1. Change directory to where application binary is located, as default *"PROFIdrive_Controller_Application/CP1625_Host/"*:

```
$ cd ~/PROFIdrive_Controller_Application/CP1625_Host/
```

2. Run the application:

```
$ sudo ./ProfidriveAppl ../Configuration/TIA/CP1625 Host/
↪ProfidriveAppl-host.xml
```

Hardware configuration in engineering system 4

Overview

Engineering XML configuration files prepared for the examples for the PROFINET variants that are CP1625 Host and CP1625 Stand-alone are available in the directory *"../PROFIdrive_Controller_Application/Configuration/"*. If you prefer to modify the engineering configuration and create your own XML configuration files, you can either use TIA Portal or PNConfigLib.

4.1 Hardware configuration in the TIA Portal

Overview

This section describes how the user can create required xml configuration files with the TIA portal using existing TIA Portal projects.

4.1.1 Importing TIA projects

Requirements

Archived TIA Portal projects created for the example application. CP1625 Host and CP1625 Stand-alone variants have their own corresponding TIA Portal project. These projects exist under,

- *"../PROFIdrive_Controller_Application/Configuration/TIA/CP1625 Host/CP1625Host.zap17"*.
- *"../PROFIdrive_Controller_Application/Configuration/CP1625 Stand-alone/CP1625_Stand-alone.zap17"*.

Procedure

1. Click **Retrieve** under **Project**
2. Navigate the path according to the use case,
   - "../PROFIdrive_Controller_Application/Configuration/TIA/CP1625 Host/"
   - "../ProfidriveApp-Demowall/Configuration/TIA/CP1625 Stand-alone"
3. Select the .zap17 extension file and click open.

4.1.2 Generating the configuration XML

Requirements

One of the example application of TIA Portal project must be opened in TIA portal.

Procedure

1. Click **Project View**.
2. **Compile** the entire project.
3. When your configuration is consistent, the TIA Portal automatically generates the XML configuration file for the PROFINET Driver. The storage path of the XML configuration file and other messages are displayed in the Inspector window under **Info**.

4.2 Hardware configuration in PNConfigLib

Overview

PNConfigLib is a library that allows you to create hardware configuration for PROFINET projects. You can perform consistency checks to ensure created project validation.

4.2.1 Generating an XML configuration file

From the user point of view, PNConfigLib accepts three files (Configuration, ListOfNodes and optional Topology) as inputs and generate XML configuration file as an output. see Fig. 4.1.

Note: GSDML files of PN IO Devices that are used in the system must be provided by the user by referencing the local path address in ListOfNodes.xml file.

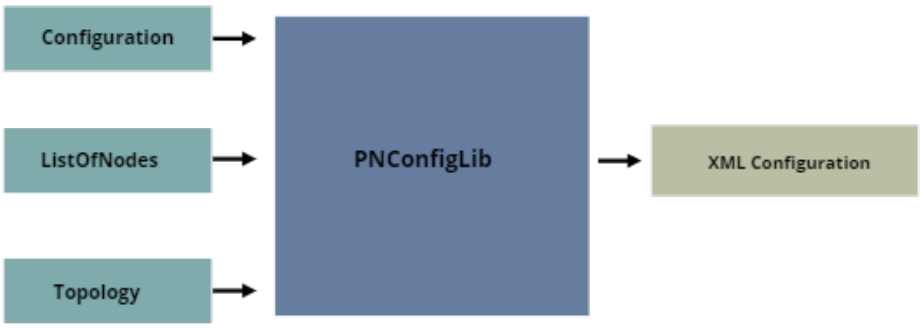

Fig. 4.1: PNConfigLib Files

Requirements

The input files of PNConfigLib consist of two required and optional XML files:

- ListOfNodes.xml(mandatory)
- Configuration.xml(mandatory)
- Topology.xml (optional)

ListOfNodes.xml. In PNConfigLib concept, a “node” represents an IO device or an IO controller, which is configured in the “Configuration” file. The user should present here all the nodes with their identifiers.

Configuration.xml: The nodes defined in the ListOfNodes file are configured in the Configuration file. The IO controller and IO device settings, their modules and submodules, subnets, sync domain etc. are defined here.

Topology.xml: The port interconnections are defined in the optional Topology file. In this file, the user may provide information on how devices are connected to each other.

You can find examples for ListOfNodes.xml, Configuration.xml and Topology.xml in the PNConfigLib manual.

Procedure

1. Use PNConfigLib to read the provided input files.
2. Use PNConfigLib to compile the project. If there is no error, an XML based configuration is created for each PN IO Controller within the project.
3. Integrate the generated XML configuration file in your system.

Additional Notes

In order to generate an XML configuration file, you can use PnConfigLib.dll to create your own application or you can use **PNConfigLibRunner.exe**. With PNConfigLibRunner.exe you can call PnConfigLib.dll from command prompt with the following parameters:

```
PNConfigLibRunner.exe -c “Configuration.xml” -l “ListOfNodes.xml” -t␣
↪“Topology.xml” -o “OutputFolderPath”
```

- -c, –configuration Required. Path to Configuration XML file
- -l, –listofnodes Required. Path to ListOfNodes XML file
- -t, –topology Optional. Path to Topology XML file
- -o, –output Optional. Output folder
- –help Displays help screen
- –version Displays version information

For more details, refer to the PNConfigLib manual.

PNConfigLib input files for this application

The user can find the required PNConfiglib input files in order to produce input XML file for PROFINET Driver under the locations below.

CP1625 Host use case,

- *"../PROFIdrive_Controller_Application/Configuration/PNConfigLib/CP1625 Host/Configuration.xml*
- *"../PROFIdrive_Controller_Application/Configuration/PNConfigLib/CP1625 Host/ListOfNodes.xml*
- *"../PROFIdrive_Controller_Application/Configuration/PNConfigLib/CP1625 Host/Topology.xml*

CP1625 Stand-alone use case,

- *"../PROFIdrive_Controller_Application/Configuration/PNConfigLib/CP1625 Stand-alone/Configuration.xml*
- *"../PROFIdrive_Controller_Application/Configuration/PNConfigLib/CP1625 Stand-alone/ListOfNodes.xml*
- *"../PROFIdrive_Controller_Application/Configuration/PNConfigLib/CP1625 Stand-alone/Topology.xml*

Required GSDML files

The user can download the GSDML files of PN IO Devices that are used in this application example by referring Table 4.1

Table 4.1: GSDML Files

PN IO Device	**GSDML Web link**
SINAMICS S210	Web link.
EB200P-2 evaluation board	Web link.
SIMATIC HMI KP8	Web link.