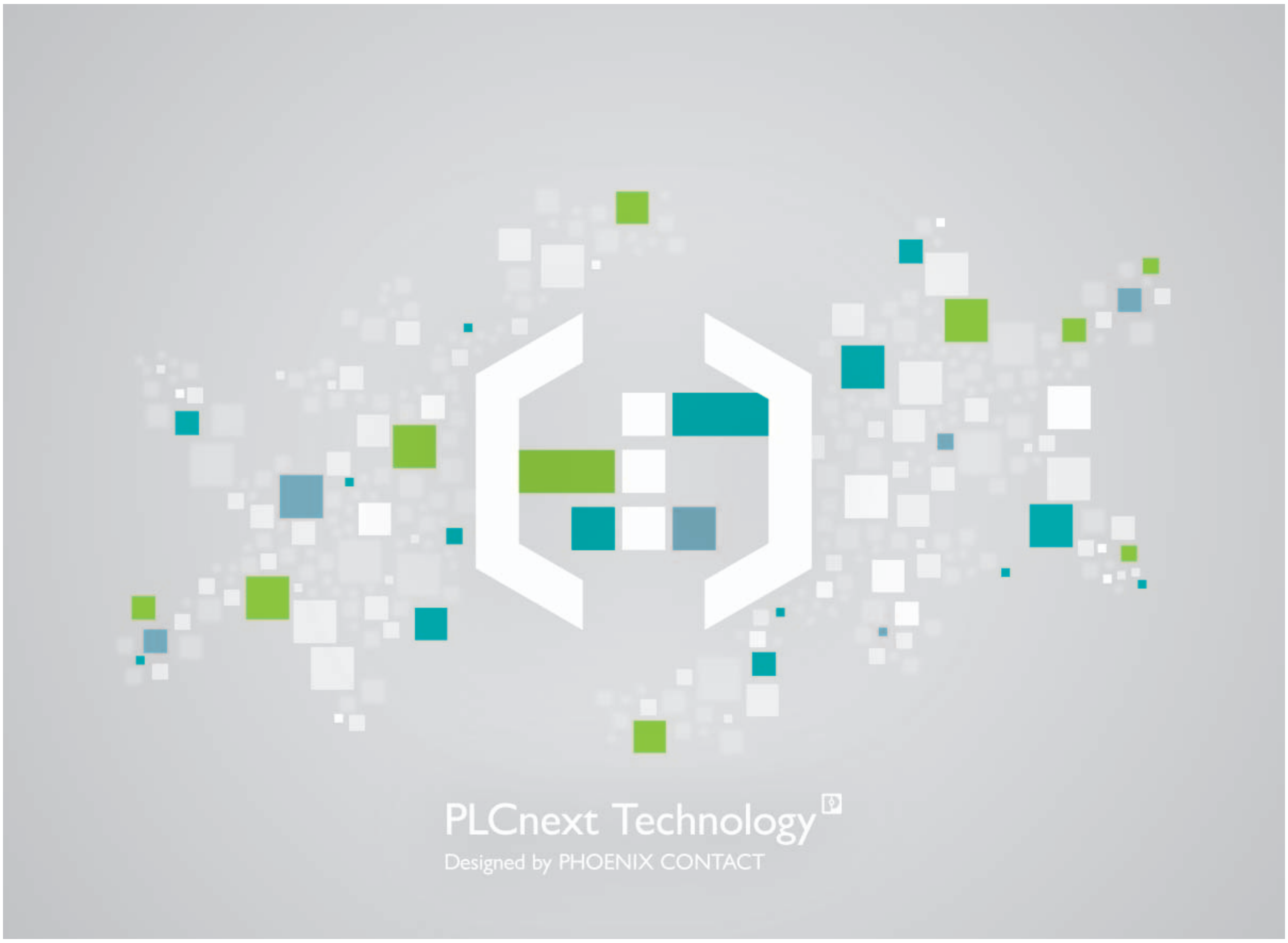


PLCnext Technology

User manual

UM EN PLCNEXT TECHNOLOGY 2019.0 LTS

PHOENIX CONTACT
INSPIRING INNOVATIONS

User manual

PLCnext Technology

UM EN PLCNEXT TECHNOLOGY 2019.0 LTS, Revision 03 2019-07-31

This user manual is valid for:

Designation	Version
PLCnext Technology firmware	2019.0 LTS

Designation	Order No.
AXC F 2152	2404267
AXC F 2152 STARTERKIT	1046568
RFC 4072S	1051328

108664_en_03

PHOENIX CONTACT GmbH & Co. KG • Flachsmarktstraße 8 • 32825 Blomberg • Germany
phoenixcontact.com

Table of contents

1 General information

Read this user manual carefully and keep it for future reference.

1.1 Identification of warning notes

⚠ This symbol indicates hazards that could lead to personal injury.

There are three signal words indicating the severity of a potential injury.

DANGER
Indicates a hazard with a high risk level. If this hazardous situation is not avoided, it will result in death or serious injury.

WARNING
Indicates a hazard with a medium risk level. If this hazardous situation is not avoided, it could result in death or serious injury.

CAUTION
Indicates a hazard with a low risk level. If this hazardous situation is not avoided, it could result in minor or moderate injury.

ⓘ This symbol together with the **NOTE** signal word warns the reader of actions that might cause property damage or a malfunction.

ℹ Here you will find additional information or detailed sources of information.

1.2 Qualification of users

The use of products described in this user manual is oriented exclusively to qualified application programmers and software engineers. The users must be familiar with the relevant safety concepts of automation technology as well as applicable standards and other regulations.

For programming of applications, familiarity with C/C++, IEC 61131-3 or MATLAB® Simulink® is assumed.

1.3 Introduction

PLCnext Technology

Demands on automation technology are increasing due to digitization in the industrial sector. Flexibility, networking, exchange of information, the “Internet of Things” are gaining ever more importance for modern, flexible and efficient production. Automation systems and their controllers must become more adaptable and must be able to react ever faster to new requirements.

To meet the changing demands of industry, Phoenix Contact developed PLCnext Technology as the basis for a new, open control platform. PLCnext Technology combines all of the communication properties and advantages of the traditional PLC world with the openness and flexibility of smart devices. Both the control platform and the cloud architecture are based on open-source components which are undergoing continuous, manufacturer-neutral development.

Previously, programming of PLC applications was only possible in the IEC 61131-3 programming languages. PLCnext Technology allows developers from various corporate areas, technology disciplines and generations to work in parallel with and yet independently of each other on one automation application. And they can do that in the programming environment to which they are accustomed. Whether with established and proven programming tools such as Microsoft® Visual Studio®, Eclipse®, MATLAB® Simulink® or PLCnext Engineer. The programming code can be created in the traditional way in accordance with IEC 61131-3, and in C/C++ or Simulink® code.

PLCnext Technology connects and combines traditional PLC programming and high-level language programming. Thus, different programming systems can be used on one platform.

PROFICLOUD, a professional cloud solution and part of PLCnext Technology, enables you to not only collect and process data directly on the controller, but to also have it transferred to the cloud. Therefore, data is more easily available and it can be retrieved and used regardless of the location and time. All performance and energy data of your system is available anywhere and at any time. Cloud-based services are also available, e.g., for data analysis, predictive maintenance, etc.

The open PLCnext Technology platform enables you to extend the benefits of conventional PLCs and provides you with a basis for cutting-edge automation capable of meeting all of the demands of the IoT world.

Advantages at a glance

- Accelerated startup in comparison to conventional control platforms because several developers can work on one program in parallel with and yet independently of each other in different programming languages
- Convenient engineering, thanks to the use of established programming tools such as PLCnext Engineer, MATLAB® Simulink®, Eclipse®, or Microsoft® Visual Studio®
- Reliable operation because determinism, real-time behavior and cycle-consistent data exchange are assured regardless of the programming language
- Cost-efficient and flexible, thanks to the use of freely available software from the open source community
- Future-proof and adaptable, because new functions and future technologies can be integrated quickly and easily

Creating programs with C++

With PLCnext Technology, you can create programs in C++ and import these into PLCnext Engineer. There, you can instantiate the programs and use them in the same way as conventional IEC 61131-3 programs in real time. Consistent data exchange with programs that were created with other high-level languages or IEC 61131-3 is ensured via the ports of the GDS. This way, the programs can run together in the same tasks.

Phoenix Contact provides an add-in for Eclipse, an SDK (Software Development Kit), as well as a LibraryBuilder. These tools support you in the use of your C++ program on a controller with PLCnext Technology. The Eclipse® development environment is particularly suitable because it can be used under both Windows® and Linux, and is commonly used in C++ programming (see Section “Creating programs with C++” on page 151).

Creating function blocks and functions with C#

Create function blocks and functions in C# with Microsoft® Visual Studio®. You can use an extension from Phoenix Contact to convert them into libraries, import them into PLCnext Engineer, and use them for your programming. The function blocks or functions can be called up in PLCnext Engineer, e.g., in the ST (Structured Text) or LD (Ladder Diagram) code worksheet, and used for your programming.

The IEC 61131 runtime system of the controller (eCLR) is a .Net-based runtime that supports both IEC 61131-3 and C# code. For this reason, the Visual Studio® development environment can be used for C# (see Section “Creating function blocks and functions with C#” on page 166).

1.4 Information about this document

This document describes the PLCnext Technology control platform. It describes the system and its components and provides instructions on creating, using and managing PLCnext Technology applications.

Detailed information on PLCnext Technology is available in the PLCnext Community at plcnext-community.net.

1.5 Trademarks/licensing information

Trademarks

All product names used are trademarks of the respective organizations.

– MATLAB® and Simulink® are registered trademarks of The MathWorks, Inc.
– Eclipse® is a trademark of the Eclipse Foundation.
– Microsoft® and Visual Studio® are trademarks of the Microsoft Corporation.

Licensing information on open source software

PLCnext controllers work with a Linux operating system.
All license information can be called using the “Legal Information” link on every page of web-based management (WBM) for controllers:

- Click on the “Legal Information” link on the bottom left of the WBM page.

Licenses for all of the open source software used are shown.

Further information on WBM of the respective controller is available in the corresponding user manual. The user manual can be downloaded from the product page at phoenixcontact.net/products.

1.6 Safety notes

NOTE: Risk of unauthorized network access

Connecting devices to a network via Ethernet always entails the risk of unauthorized access to the network.

Therefore, please check for the option of disabling active communication channels in your application (for instance SNMP, FTP, BootP, DCP, HTTP, HTTPS, etc.) or setting passwords to prevent third parties from accessing the controller without authorization and modifying the system.

Due to the communication interfaces of the controller, the controller should not be used in safety-critical applications unless additional security appliances are used.
Please take additional protective measures in accordance with the IT security requirements and the standards applicable to your application (e.g., virtual networks (VPN) for remote maintenance access, firewalls, etc.) for protection against unauthorized network access.

On first request, you shall release Phoenix Contact and the companies associated with Phoenix Contact GmbH & Co. KG, Flachsmarktstrasse 8, 32825 Blomberg in accordance with §§15ff.AktG (German Stock Corporation Act), hereinafter collectively referred to as “Phoenix Contact”, from all third-party claims made due to improper use.

For the protection of networks for remote maintenance via VPN, Phoenix Contact offers the mGuard product series security appliances; further information on this is available in the latest Phoenix Contact catalog (phoenixcontact.net/products).

Additional measures for protection against unauthorized network access are listed in the AH EN INDUSTRIAL SECURITY application note. The application note can be downloaded at phoenixcontact.net/products.

NOTE: Risk of unauthorized access to devices

Devices with PLCnext Technology do not feature mechanical access protection and are therefore at risk of manipulation. Unauthorized access can occur via the following device interfaces, for example:

- USB ports
- PCI Express interfaces
- Axioline bus
- SC card slot and the SD card contained therein
- Device HMI (touch panel as well as buttons, switches, etc.)
- Ethernet interfaces

To prevent damage due to authorized access, make sure that only authorized access is possible.

- Protect the interfaces by installing the devices in a control cabinet.
- Secure the control cabinet with a lock.
- Make sure that only authorized persons have access to the control cabinet key.
- Run cables in such a way that they are protected against unauthorized access.

1.7 PLCnext Technology product range

Modifications to hardware and firmware of the devices are not permitted.

Incorrect operation or modifications to the devices can endanger your safety or damage the devices. Do not repair the devices yourself. If the devices are defective, please contact Phoenix Contact.

Currently, the following products are available with PLCnext Technology:

Description	Type	Order No.
PLCnext Control for the direct control of Axioline F I/Os. With two Ethernet interfaces. Complete with connector and bus base module.	AXC F 2152	2404267
PLCnext Control with 4 x 10/100/1000 Ethernet, PROFINET controller with integrated PROFIsafe safety controller, PROFINET device, IP20 degree of protection, pluggable parameterization memory	RFC 4072S	1051328
AXC F 2152 starter kit including AXC F 2152 PLCnext Control, voltage switch, digital input and output module, analog input and output module, potentiometer, switch module, PROFICLOUD license, as well as a power supply unit, patch cable, country-specific adapter plugs, and documentation.	AXC F 2152 STARTERKIT	1046568
Engineering software platform for Phoenix Contact automation controllers. PLCnext Engineer is IEC 61131-3-compliant and its functions can be extended using add-ins.	PLCNEXT ENGINEER	1046008
Software add-on for the integration and execution of Matlab Simulink models on Remote Field and Axioline controllers	PC WORX TARGET FOR SIMULINK	2400041

Ensure that you always use the latest firmware and documentation. The latest firmware versions and documentation can be downloaded at phoenixcontact.net/products.

Software packages required for programming in C++ with Eclipse® or in C# with Visual Studio® can be downloaded from the download area for the AXC F 2152 controller.

Sample projects and the associated documentation can be downloaded at https://github.com/plcnext.

2 Structure of PLCnext Technology

Core components

PLCnext Technology is an open firmware platform executed on a Linux operating system with real-time patch. Different firmware components forming the core functions are called core components. These core components are a fixed part of the PLCnext Technology firmware. They cannot be modified. The core components are divided into the following groups:

– Middleware
– I/O components
– Service components
– System components

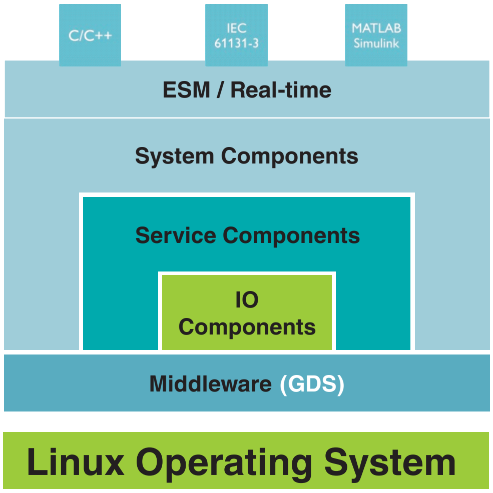


Figure 2-1 Core components of the PLCnext Technology firmware

Middleware

The middleware section decouples the PLCnext Technology firmware from the operating system. The GDS (Global Data Space) is part of the middleware and is responsible for a central function of PLCnext Technology. It enables consistent data exchange between different real-time components. For additional information on the GDS, please refer to Section “GDS (Global Data Space)” on page 27.

I/O components / fieldbus manager

The fieldbus manager connects the implemented fieldbus for the input and output of process data with PLCnext Technology. The following fieldbuses are supported (depending on the device):

– PROFINET controller
– PROFINET device
– Axioline F master (local bus)
– INTERBUS (with AXC F 2152 and AXC F IL Adapt)

Service components

The service components provide access to the ESM (Execution and Synchronization Manager), GDS (Global Data Space), and to the following system components:

- OPC UA server
- Proficloud gateway
- Web-based management
- PLCnext Engineer HMI (HTML5 web visualization)
- Accessible via OS
  - DCP
  - SFTP
  - VPN
  - SSH
  - NTP
  - Trace controller

System components

In the system components, all the basic functions of PLCnext Technology are implemented.

- System manager and PLC manager
  These components load all the other system components and monitor the stability of the system.
- ESM (Execution and Synchronization Manager)
  The ESM enables IEC 61131-3, C++ and Matlab® Simulink® programs to be executed in real time. When creating applications, programming languages can be combined in any way. For additional information on the ESM, please refer to Section "ESM (Execution and Synchronization Manager)" on page 21.
- User manager
  The user manager extends the standard Linux user management function. The different user roles are managed here. You can only execute operations in the PLCnext Technology firmware with a defined user role. You can select one or more user roles containing different permissions for each user.
- eCLR
  ProConOS embedded CLR is the open IEC 61131 control runtime system for different automation tasks.

Real-time user programs

User programs are not supplied as a standard with the PLCnext Technology firmware; they are created by the user. These user-defined programs can be created in conventional programming languages in accordance with IEC 61131-3, and also in C++ or Matlab® Simulink®. It is also possible to combine different programming languages.
The easiest way to use PLCnext Technology is to create user programs in programming languages in accordance with IEC 61131-3, in C++ or using Matlab® Simulink®, and to execute these in the real-time environment of the ESM. The user programs are downloaded to the controller and, after a cold restart of the device, are executed by the firmware.

The IN and OUT ports of the GDS guarantee data consistency and ensure seamless data exchange between tasks and programs.

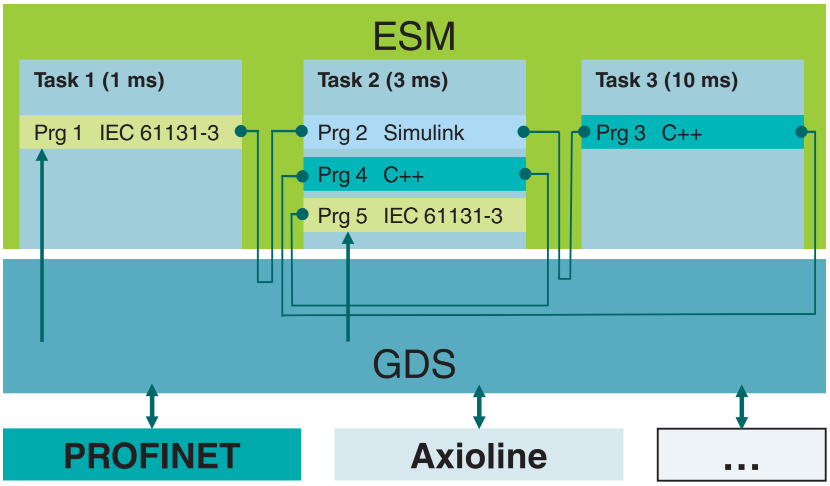


Figure 2-2 Task handling with ESM and GDS

2.1 Internal user components

Users can add their own components, so-called internal user components, to PLCnext Technology. The system manager loads and monitors these internal user components (see Section 2.3 “System manager”). All available PLCnext Technology APIs can be used:

- RSC (Remote Service Call): each component (system, service, I/O components) features a selection of individual services that can be called and used via RSC (see Section “RSC (Remote Service Calls)” on page 42).
- Component interface: the firmware calls the implemented functions of the component interface at certain times, e.g., during program startup.
- Data access: read and write access to the GDS (Global Data Space)

– Common classes: easy retrieval of system functions, e.g., file operations, socket services, threading (see Section "Common classes" on page 126).

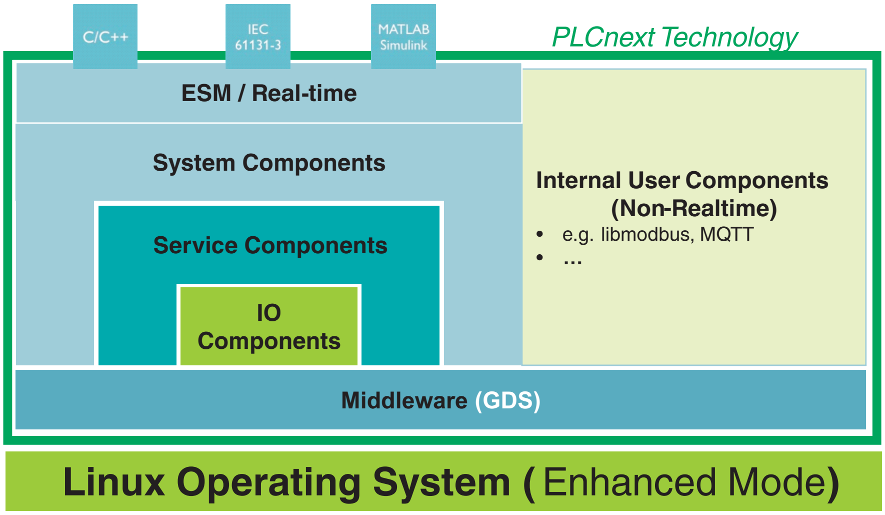


Figure 2-3 PLCnext Technology – internal user components

2.2 External user components

Internal user components are used to add new, internal functions. It can also be necessary to incorporate external processes. This may be the case if, for example, extensive software components such as Java® Frameworks or .Net Core® are used. These can be integrated as external user components. They can be started along with the PLCnext Technology framework.

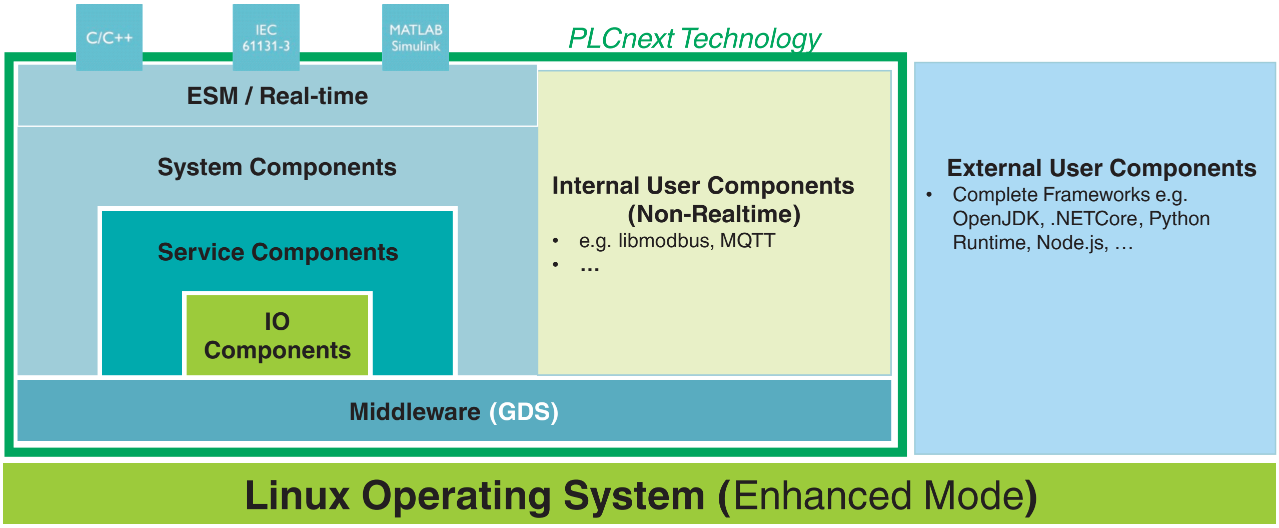


Figure 2-4 PLCnext Technology - external user components

2.3 System manager

PLCnext Technology is an open platform. This means that as a user you can integrate your own components and programs. During firmware startup, the system manager ensures that all integrated components and programs are configured and started in the right order. Here, all system processes are generated and internal user components are supplemented. A system component is always started, configured and shut down by the system manager.

2.4 PLC manager

The PLC manager is a firmware component that loads the necessary PLC program code into the memory and boots up or shuts down the programs. The program code can exist of an IEC 61131-3 program that was created and sent using PLCnext Engineer. The code can also be created in C++ or Matlab® Simulink®. C++ and Matlab® Simulink® programs are available on the controller as program code libraries (shared object, *.so). Configuration files on the controller are used to determine which libraries the PLC manager is to load and which programs in these libraries it is to instantiate. The PLC manager is therefore superordinate to the code. It controls boot up and shut down of the real-time system (ESM) as well as stopping and starting of data exchange via fieldbuses. If the controller is in the “stop” state, the real-time tasks monitored by the ESM are not executed. The signals of the sensors connected to the fieldbus are no longer read as inputs, and the output signals are no longer sent to the connected actuators.

The controller can be started in three different modes:

- **Cold restart:**
  During a cold restart, all data is reset.
- **Warm restart:**
  During a warm restart, all data is reset and all remanent data is restored (system start). Note: In firmware version 1.x, remanent data is only located in the eCLR/IEC programming environment.
- **Hot restart:**
  During a hot restart, the data is neither reset nor restored. All variable values are retained.

2.5 Managing of components

PLCnext Technology is based on components that are included via the “IComponentInterface” interface (see Section ““IComponent” and “ComponentBase”” on page 117).

Application Control Framework

The ACF (Application Control Framework) is a part of the system manager and manages the internal user components. The ACF is a framework that enables component-based platform development and the configurative composition of the firmware for the devices. It enables the dynamic and configurative integration of user functions into the system. The components managed by the ACF are thus firmware components and user components that are executed independently of the PLC program. The ACF generates the components when booting the firmware.
For additional information on the ACF, please refer to Section “ACF (Application Component Framework)” on page 123.

PLC manager / Program Library Manager

One firmware component that is managed by the ACF is the PLC manager (see "PLC manager" on page 16). It manages the PLC program (real-time user program). The associated components and libraries that make these user programs available are managed by the PLC manager via the PLM (Program Library Manager, see 5.4 on page 122). The configuration is executed with a file referenced by /opt/plcnext/projects/Default/Plc/Plm/Plm.config. The PLC manager generates the components when loading the program.
For additional information on the PLC manager and Program Library Manager, please refer to Sections "PLC manager" on page 16 and ""IComponent" and "ComponentBase"" on page 117.

Delimitation of ACF and PLM

- ACF and PLM use the same ILibrary and IComponent interface (see Sections ""ILibrary" and "LibraryBase"" on page 116 and ""IComponent" and "ComponentBase"" on page 117).
- ACF and PLM use the same format for configuration files (see Section "Configuration files" on page 17).
- Only components that are managed via PLM can make programs available that can be instantiated in ESM tasks (IProgramProvider, see Section 5.2.1 on page 119).
- Only components that are managed via the PLM can be stopped, modified and started up via download from PLCnext Engineer. This is also the case for ESM tasks and the programs instantiated therein.
- For components that are managed via the ACF, the firmware must be stopped, started or rebooted.
- Components that are managed via the ACF are retained, even if the PLC program is shut down, deleted or booted.

2.6 Configuration files

With PLCnext Technology, you can load programs and program instances onto the controller even without the PLCnext Engineer software. You can, for example, create a program in Eclipse® with C++ and transfer it directly to the controller. All of the important and necessary settings can be configured directly in the configuration files on the controller. The file system of the controller is accessed via the SFTP protocol. Use a suitable SFTP client software for this, e.g., WinSCP. The configuration files are XML files. You can edit them using an editor of your choice. The configuration files are imported following the boot procedure of the PLCnext Technology firmware. If there are no configuration errors in the configuration files, the components and program instances are subsequently executed by the ESM.

When PLCnext Engineer is used, all configuration files are created by PLCnext Engineer and downloaded to the controller.

2.6.1 acf, esm, gds configuration files

acf.config

The acf.config configuration file contains information on the library, component and program instances, as well as processes of shared objects (C++ programs).

The information is required by the ESM in order to load shared objects and to execute component instances or programs.

There are two ways to instantiate components:
- Included in Default.acf.config (projects/Default/Default.acf.config)
- Included in Plm.config (projects/Plc/Plm/Plm.config)

esm.config

The esm.config configuration file contains the task configuration, the instantiation of programs, and assignment to a processor core (one ESM per processor core). For additional information, please refer to Section "Task configuration via configuration files" on page 23.

gds.config

The gds.config configuration file contains the port definition as well as assignment of IN and OUT ports. For additional information, please refer to Section "GDS configuration using configuration files" on page 34.

2.6.2 .tic file

Phoenix Contact recommends creating the .tic files with the PLCnext Engineer software.

The .tic configuration file contains information on the bus configuration with the associated I/O process data of the IN and OUT ports.

2.6.3 Metafiles

Metadata describes classes and types of components and programs that were created in a PLCnext Technology application. They are required for using the program in PLCnext Engineer.

The metafile for the library (*.libmeta) contains links to the metafiles for the incorporated components (*.compmeta). These, in turn, link to the metafiles of the connected programs (*.progmeta). Configuration of the GDS data ports (IN or OUT ports) is defined in the *.progmeta file.
When the PLCnCLI (PLCnext Command Line Interface, see Section "PLCnCLI (PLCnext Command Line Interface)" on page 151) creates the C++ project, the *.compmeta, *.libmeta and *.progmeta metafiles are created automatically. The files are interlinked, as shown in Figure 2-5.

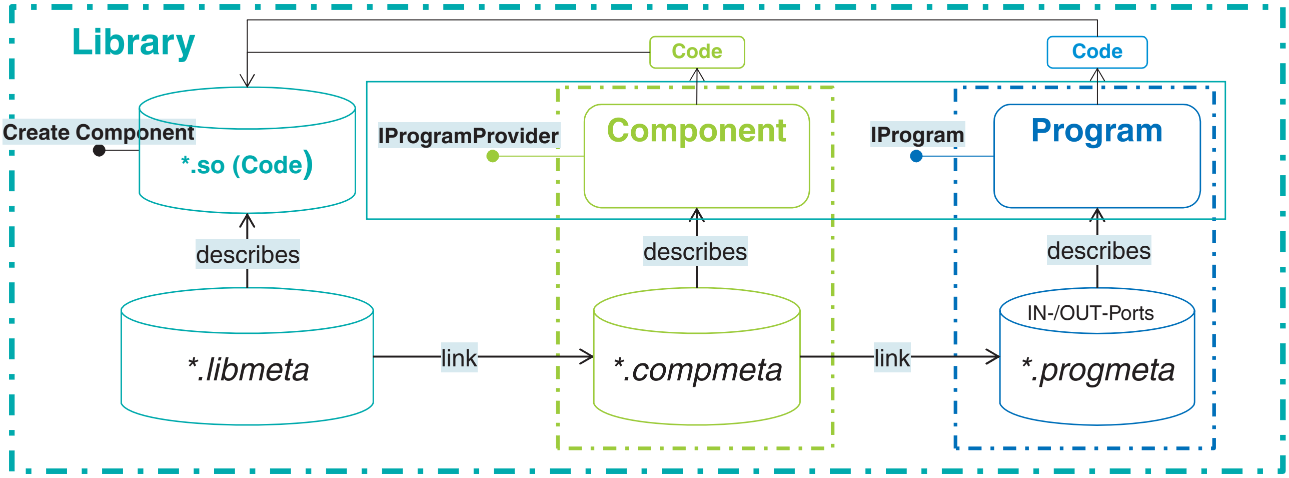


Figure 2-5 Overview of the metafiles

The LibraryBuilder combines the metadata and the code (*.so) into a library. This can be imported into PLCnext Engineer and used in the same way as an IEC 61131-3 program. Tasks and flow of data are configured directly in PLCnext Engineer. The PLCnCLI (see Section "Functions of the PLCnCLI" on page 153) generates the metafiles.

Example of a component list and a shared object (*.so) in the *.libmeta metafile:

```
<?xml version="1.0" encoding="utf-8"?>
<MetaConfigurationDocument
    xmlns="http://www.phoenixcontact.com/schema/metaconfig"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    schemaVersion="2.0">

    <Library name="CPP_Counter" applicationDomain="CPLUSPLUS"  >

        <File path="libCPP_Counter.so" checksum="_TODO_" />

        <!-- Included components in this library -->
        <ComponentIncludes>
           <Include path="CPP_Counter_C/CPP_Counter_C.compmeta" />
        </ComponentIncludes>

    </Library>

</MetaConfigurationDocument>
```

Example of a program list in the *.compmeta metafile:

```
<?xml version="1.0" encoding="utf-8"?>
<MetaConfigurationDocument
    xmlns="http://www.phoenixcontact.com/schema/metaconfig"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    schemaVersion="2.0">

    <Component type="CPP_Counter_C" >


        <!-- List of programs referenced in this component -->
        <ProgramIncludes>
           <Include path="CPP_Counter_P/CPP_Counter_P.progmeta" />
        </ProgramIncludes>

        <!-- List of component ports -->
        <Ports>
           <Port name="IP_CppEnable_bit" type="boolean" dimensions="1" attributes="Input|OpcUa" />
           <Port name="IP_CppSwitches_uint8" type="uint8" dimensions="1" attributes="Input|Ehmi"/>
           <Port name="OP_CppCounter_uint8" type="uint8" dimensions="1" attributes="Output"/>
        </Ports>
        -->

    </Component>

</MetaConfigurationDocument>
```

Example of the definition of IN and OUT ports in the *.progmeta metafile:

```
<?xml version="1.0" encoding="utf-8"?>
<MetaConfigurationDocument
    xmlns="http://www.phoenixcontact.com/schema/metaconfig"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    schemaVersion="2.0">

    <Program type="CPP_Counter_P" >

        <Ports>
            <Port name="IP_CppEnable_bit" type="boolean" dimensions="1" attributes="Input|OpcUa" />
            <Port name="IP_CppSwitches_uint8" type="uint8" dimensions="1" attributes="Input|Ehmi"/>
            <Port name="OP_CppCounter_uint8" type="uint8" dimensions="1" attributes="Output"/>
        </Ports>

    </Program>

</MetaConfigurationDocument>
```

2.6.4 Generating configuration files with PLCnext Engineer

The following description applies for all three *.config types (acf.config, esm.config, gds.config). The configuration files are automatically created when the PLCnext Engineer software is used. During the download, they are saved to the controller in the respective folder (e.g., /opt/plcnext/projects/PCWE/Esm for the PCWE.esm.config file) in the controller file system. This folder is intended for the download from PLCnext Engineer. The PCWE folder is automatically created and managed by PLCnext Engineer. **Do not store any other files here because PLCnext Engineer completely deletes this folder prior to the download**. When the configuration in PLCnext Engineer changes and the project is once again downloaded to the controller, the previous configuration files are overwritten. Manual changes are also lost.

2.6.5 Manual configuration

To make manual changes to the configuration files or to create new configuration files, you can copy the files generated by PLCnext Engineer and rename them. You can then manually configure these files. To prevent overwriting your own configuration files in the “PCWE” folder when downloading the project again from PLCnext Engineer, proceed as follows:

- Under /opt/plcnext/projects/ in the file system of the controller, create a new folder and name it accordingly. E.g., /opt/plcnext/projects/Example/
- Save the files to this folder.
- Include the configuration files via the `Include path` command (see example below).

The folder can be used for saving additional configuration files and for the manual, configurative extension of the components.

The `Include` mechanism is used to state which configuration files are to be used for the configuration. If files that are not in the folder used for inclusion are to be included, the respective path has to be specified. In the following example, all files that are in the same directory as the “Default.gds.config” file are included, i.e., /opt/plcnext/projects/Default/Plc/GDS (`<Include path=“*.gds.config” />`). If you want to include all the config files that are in the respective folder, replace the specific file name with an * (e.g., `*.gds.config`). This results in all files with the same file extension being included.

Example: `Include path`:

```
<Includes>
    <Include path=“*.gds.config“ />
    <Include path=“$ARP_PROJECTS_DIR$/PCWE/Plc/Gds/*.gds.config“ />
    <Include path=“$ARP_PROJECTS_DIR$/Example/*.gds.config“ />
</Includes>
```

In the example, all GDS configuration files from the following directories are included in the configuration. `„$ARP_PROJECTS_DIR$“` is a PLCnext Technology environment variable with the value `/opt/plcnext`:

– “Default”: /opt/plcnext/projects/Default/Plc/GDS (created by the firmware)
– “PCWE”: /opt/plcnext/projects/PCWE/Plc/GDS (created by PLCnext Engineer)
– “Example”: /opt/plcnext/projects/Example (created by the user)

When editing configuration files, observe the information in Sections “Task configuration via configuration files” on page 23 and “GDS configuration using configuration files” on page 34.

2.7 ESM (Execution and Synchronization Manager)

PLCnext Technology also features task handling. The Execution and Synchronization Manager (ESM) performs task handling, monitoring, and chronological sequencing of programs from different programming languages. Each processor core of a controller is managed by one ESM. One ESM is therefore assigned to one processor core. If a controller has more than one processor core, then there are also several ESMs (example: controller AXC F 2152: 2 processor cores and 2 ESMs).

The ESM has the following advantages:

– Configuration and monitoring of cyclic tasks and idle tasks.
– The execution times of the tasks are available as system variables and can be used for diagnostics.
– System balancing.
– Multicore systems are supported.

The ESM can also be used to execute programs and program parts that were created in different programming environments in real time. These can include high-level languages such as (C++), IEC 61131-3 code, and model-based tools such as Matlab® Simulink®. Program parts that were created using different programming languages can also be combined and processed within a task. The EMS controls the processes and also executes the high-level language programs deterministically in the defined order. To ensure data consistency between the tasks at all times, all data is synchronized with the GDS whenever a task is called (see also Section "GDS (Global Data Space)" on page 27).

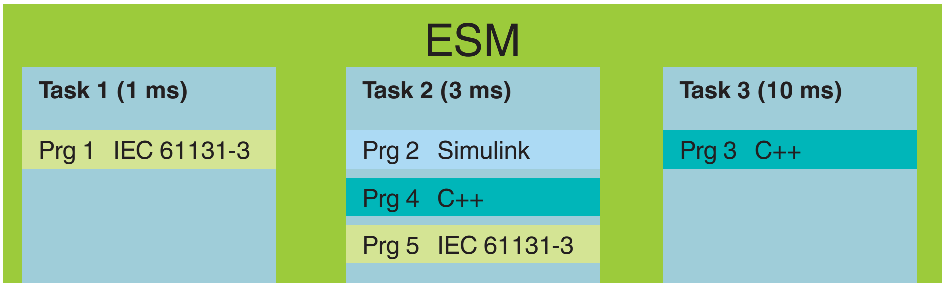
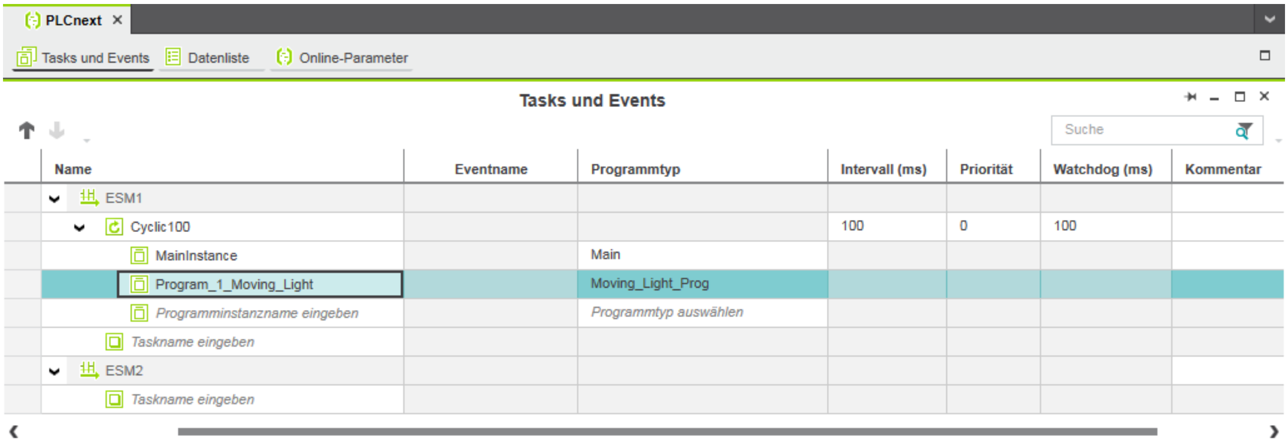


Figure 2-6 ESM (Execution and Synchronization Manager)

2.7.1 Task configuration with PLCnext Engineer

You can use the PLCnext Engineer software to easily create and configure tasks. Here, you can proceed as in a conventional IEC 61131-3 program. The IEC 61131-3 program, or the program created in a different programming environment and then imported into PLCnext Engineer, can be instantiated in a task. It does not matter whether the programs were created with C/C++, IEC 61131-3 or Matlab® Simulink®.

In the PLCnext Engineer "Tasks and Events" editor, you can instantiate a task and assign it to the desired program instances. A description of this procedure and further information on task handling with PLCnext Engineer is available in the online help, the PLCnext Engineer quick start guide (PLCNEXT ENGINEER, Order no. 1046008), and the AXC F 2152 controller user manual (Order no. 2404267). The documentation for the respective products can be downloaded at phoenixcontact.net/products. You can call the online help from within PLCnext Engineer.



Name	Eventname	Programmtyp	Intervall (ms)	Priorität	Watchdog (ms)	Kommentar
ESM1						
Cyclic100			100	0	100	
MainInstance		Main				
Program_1_Moving_Light		Moving_Light_Prog				
Programminstanzname eingeben		*Programmtyp auswählen*				
Taskname eingeben						
ESM2						
Taskname eingeben						

Figure 2-7 Example: tasks and program instances in PLCnext Engineer

2.7.2 Task configuration via configuration files

When a project is downloaded from PLCnext Engineer to the controller, the previous configuration files are overwritten with the new configuration files. Section "Manual configuration" on page 20 describes how you can safely store your manually modified configuration in the controller file system, thus preventing the loss of data.

You can also modify the task configuration, the instantiation of programs and the assignment to a processor core (one ESM per processor core) without PLCnext Engineer via configuration files in XML format. All of the important settings can be configured directly in the configuration file on the controller. To modify the configuration manually, the XML file can be edited using any editor. The ESM can load one or several configuration files and create a joint configuration.

Example of a configuration file:

```
<?xml version="1.0" encoding="utf-8"?>
<EsmConfigurationDocument xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" schemaVersion="1.0" xmlns="http://www.phoe-
nixcontact.com/schema/esmconfig">
    <Tasks>
        <PreDefinedEventTask name="Cold" stackSize="0" eventName="Arp.Plc.Esm.OnColdStart"
        confirmed="false" priority="0" watchdogTime="100000000" executionTimeThreshold="0" />
        <PreDefinedEventTask name="Warm" stackSize="0" eventName="Arp.Plc.Esm.OnWarmStart"
        confirmed="false" priority="0" watchdogTime="100000000" executionTimeThreshold="0" />
        <PreDefinedEventTask name="Except" stackSize="0" eventName="Arp.Plc.Esm.OnException"
        confirmed="false" priority="0" watchdogTime="100000000" executionTimeThreshold="0" />
        <IdleTask name="Idle" stackSize="0" watchdogTime="100000000" executionTimeThreshold="0" />
        <PreDefinedEventTask name="interbus" stackSize="0" eventName="Arp.Io.Interbus.OnCycleEnd"
        confirmed="true" priority="0" watchdogTime="100000000" executionTimeThreshold="0" />
    </Tasks>
    <EsmTaskRelations>
        <EsmTaskRelation esmName="ESM1" taskName="Cold" />
        <EsmTaskRelation esmName="ESM1" taskName="Warm" />
        <EsmTaskRelation esmName="ESM1" taskName="Execpt" />
        <EsmTaskRelation esmName="ESM1" taskName="Idle" />
        <EsmTaskRelation esmName="ESM1" taskName="interbus" />
    </EsmTaskRelations>
    <Programs>
        <Program name="Main1" programType="Main" componentName="Arp.Plc.Eclr" />
        <Program name="Main2" programType="Main" componentName="Arp.Plc.Eclr" />
        <Program name="Main3" programType="Main" componentName="Arp.Plc.Eclr" />
        <Program name="Main4" programType="Main" componentName="Arp.Plc.Eclr" />
        <Program name="Main5" programType="Main" componentName="Arp.Plc.Eclr" />
    </Programs>
    <TaskProgramRelations>
        <TaskProgramRelation taskName="Cold" programName="Arp.Plc.Eclr/Main1" order="0" />
        <TaskProgramRelation taskName="Warm" programName="Arp.Plc.Eclr/Main2" order="0" />
        <TaskProgramRelation taskName="Execpt" programName="Arp.Plc.Eclr/Main3" order="0" />
        <TaskProgramRelation taskName="Idle" programName="Arp.Plc.Eclr/Main4" order="0" />
        <TaskProgramRelation taskName="interbus" programName="Arp.Plc.Eclr/Main5" order="0" />
    </TaskProgramRelations>
<TaskEvents />
</EsmConfigurationDocument>
```

To configure tasks for execution in the ESM (Execution and Synchronization Manager) using the *.esm.config configuration file, proceed as follows:

Defining a task

A task is defined between tags `<Tasks>` and `</Tasks>`.

Example of defining a task:

```
<Tasks>
    <CyclicTask name="SquareWave_Cycle" stackSize="0" priority="0" cycleTime="100000000"
    watchdogTime="100000000" executionTimeThreshold="0" />
    <CyclicTask name="CPP_Cycle" stackSize="0" priority="0" cycleTime="100000000"
    watchdogTime="100000000" executionTimeThreshold="0" />
    <CyclicTask name="PCWE_Cycle" stackSize="0" priority="0" cycleTime="100000000"
    watchdogTime="100000000" executionTimeThreshold="0" />
</Tasks>
```

```
<Tasks>
    <PreDefinedEventTask name="Cold" stackSize="0" eventName="Arp.Plc.Esm.OnColdStart"
    confirmed="false" priority="0" watchdogTime="100000000" executionTimeThreshold="0" />
</Tasks>
```

Definition of the task starts with the task type.

- Here, enter the type:
  - `CyclicTask` = Cyclic task
  - `IdleTask` = Idle task
  - `PreDefinedEventTask` = System event task (cold restart, warm restart, hot restart, stop, exception)
- Define the tasks using the attributes from the following table:

Attribute	Description
`name`	The `name` attribute defines the task name. • Enter the task name. The name must be unique. It can only be used once per controller.
`priority`	The `priority` attribute defines the task priority. • Enter a value to specify the task priority. – 0: Highest priority – 31: Lowest priority – 32: Reserved for idle task (no `priority` is indicated for an idle task)
`cycleTime`	The `cycleTime` attribute defines the duration of the task (for cyclic tasks only). • Enter the value in nanoseconds. The minimum value of the AXC F 2152 controller is 500000 ns = 500 µs. Select a multiple of the minimum value. Depending on the application, the jitter time for a task may increase as the system load increases.

Attribute	Description
watchdogTime	The watchdog monitors whether the task was executed within the specified time. • Enter the value in nanoseconds. Value “0” means that there is no monitoring. The watchdogTime attribute can be used for cyclic and idle tasks.
EventName	The following values are valid for this attribute. – Arp.Plc.Esm.OnColdStart (cold restart) – Arp.Plc.Esm.OnWarmStart (warm restart) – Arp.Plc.Esm.OnHotStart (hot restart) – Arp.Plc.Esm.OnStop (stop) – Arp.Plc.Esm.OnException (exception)

Assigning a task

Once the task has been defined, it has to be assigned to the desired ESM. Assignment is defined between tags <EsmTaskRelations> and </EsmTaskRelations>.

Example: Assigning a task to an ESM

```
<EsmTaskRelations>
    <EsmTaskRelation esmName="ESM1" taskName="SquareWave_Cycle" />
    <EsmTaskRelation esmName="ESM1" taskName="CPP_Cycle" />
    <EsmTaskRelation esmName="ESM1" taskName="PCWE_Cycle" />
</EsmTaskRelations>
```

- Assign the task using the attributes from the following table:

Attribute	Description
esmName	The esmName attribute defines the name of the ESM the respective task is to be assigned to. • Enter the task name. Example: ESM1 or ESM2 for a controller with dual core processor.
taskname	The taskname attribute defines the task to be assigned. • Enter the name of the task to be assigned.

Instantiating of programs

Once the task is defined and assigned to an ESM, programs can be instantiated. Programs are defined between tags <Programs> and </Programs>. The definition of a program instance is introduced with the tag <Program>. Programs that have been programmed in IEC 61131-3 can only be sent and configured with PLCnext Engineer. The code example originates from a config file generated with PLCnext Engineer.

Instantiating of programs (example):

```
<Programs>
    <Program name="SquareWave" programType="PCWE_SquareWave_P" componentName="Arp.Plc.Eclr" />
    <Program name="CPP_Counter" programType="CPP_Counter_P"
    componentName="CPP_Counter.CPP_Counter_C-1" />
    <Program name="CPP_Counter_P1" programType="CPP_Counter_P"
    componentName="CPP_Counter.CPP_Counter_C-1" />
    <Program name="PCWE_Counter" programType="PCWE_Counter_P" componentName="Arp.Plc.Eclr" />
</Programs>
```

- Instantiate programs using the attributes from the following table:

Attribute	Description
`name`	The `name` attribute defines the name of the program instance. • Enter the name of the program instance. The name must be unique within the controller.
`programType`	The `programType` attribute defines the program type. • Enter the program type.
`componentName`	The `componentName` attribute defines the name of the component that contains the program. • Enter the name of the component as it is defined in the *.acf.config configuration file. The “Arp.Plc.Eclr” component is reserved for IEC 61131-3 programs that were instantiated with PLCnext Engineer. See Section 6.2.3 “Creating a new C++ project in Eclipse®”

Assigning program instances to a task

The program instances have to be assigned to a task. Assignment is made between tags `<TaskProgramRelations>` and `</TaskProgramRelations>`.

Assigning program instances to a task:

```
<TaskProgramRelations>
    <TaskProgramRelation taskName="SquareWave_Cycle" programName="Arp.Plc.Eclr/SquareWave"
    order="0" />
    <TaskProgramRelation taskName="CPP_Cycle" programName="CPP_Counter.CPP_Counter_C-
    1/CPP_Counter" order="0" />
    <TaskProgramRelation taskName="CPP_Cycle" programName="CPP_Counter.CPP_Counter_C-
    1/CPP_Counter_P1" order="1" />
    <TaskProgramRelation taskName="PCWE_Cycle" programName="Arp.Plc.Eclr/PCWE_Counter" order="0"
/>
</TaskProgramRelations>
```

- Assign the program instance to a task using the attributes from the following table.

Attribute	Explanation
`taskname`	The `taskname` attribute specifies the name of the task the program instance is to be assigned to. • Enter the name of the desired task the program instance is to be assigned to.
`programName`	The `programName` attribute defines the name of the program instance with the prefixed library and component names. The name is used to select the program instance to be processed in the task. • Enter the full name into the program instance to be processed.
`order`	The `order` attribute determines the order in which program instances are processed within a task, starting with 0. • Enter a number to determine the processing sequence. The program instance with the digit 0 is processed first. Each number can only be used once for each task.

2.8 GDS (Global Data Space)

The GDS enables communication relationships and data exchange between tasks and programs that were created in different programming languages. The GDS also ensures that inconsistencies as shown in the following example are prevented:

While tasks are being processed and data is being exchanged directly between tasks, read and write access can overlap due to different cycles and intervals. In the example in Figure 2-8, task 1 – with medium priority – begins processing and, once completed, outputs a value. This value is used by task 2 as an input value. Task 2 starts processing using this input value, and then the processing is interrupted by another start of task 1. Task 1 has a higher priority and is processed again, while task 2 is paused. At the end of processing task 1, a new value is output that is again sent to task 2 as an input value. Once task 1 is finished, processing of task 2 is continued. However, because the input value has changed in the meantime, there is data inconsistency which may lead to problems. The use of the GDS can prevent such inconsistencies.

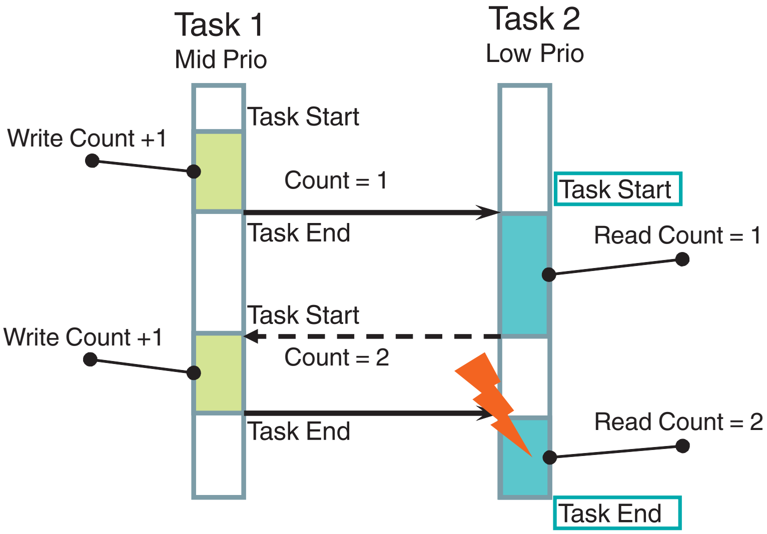


Figure 2-8 Example of data inconsistency

2.8.1 Port-based communication

With the GDS, it is easy to establish communication relationships between different programs. This communication between the PLCnext Technology programs is implemented via ports. A port represents an endpoint of a component within a PLCnext Technology application. Via a port, data can be exchanged with other components (e.g., program instances and I/O systems). To enable this type of data exchange, the program variables to be exchanged or process data of the I/O components are defined as IN or OUT ports. The data that is written from the tasks into the GDS is written via OUT ports. Tasks that receive input data read this data from the GDS via IN ports. Using the ports ensures data consistency and correctly functioning communication relationships at all times.

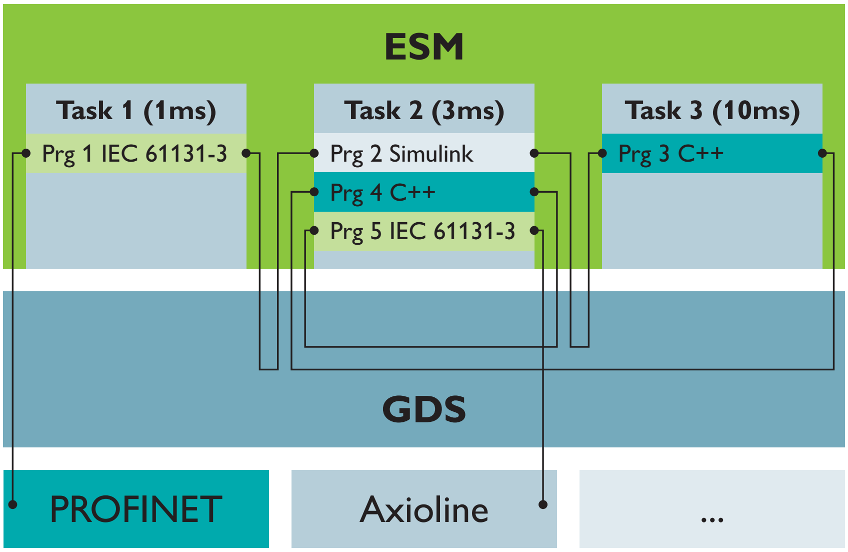


Figure 2-9 Example: Inter-task communication with ESM and GDS

The IN and OUT ports are coupled via buffer mechanisms and synchronized during execution.

- Start of a task cycle (OnTaskExecuting): The IN ports are read from the GDS buffer storage units by the program instances to be called in the task.
- End of a task cycle (OnTaskExecuted): The OUT ports are written to the GDS buffer storage units by the program instances to be called in the task.

The following figure shows the behavior of task 1:

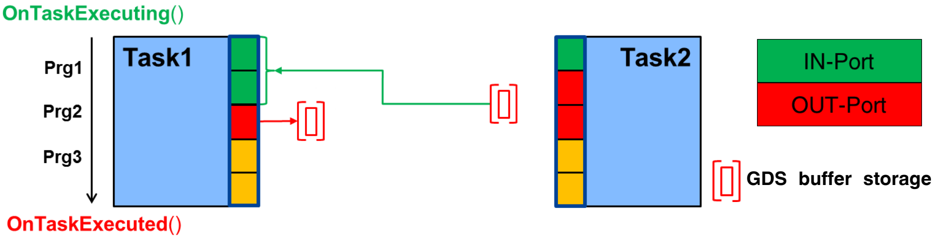


Figure 2-10 Example: Port communication with task 1 buffer storage unit

Data exchange for IN ports and OUT ports takes place at the beginning and at the end of each task execution. This only concerns data exchange of IN and OUT ports whose programs have been instantiated in different tasks. Ports within the same task are exchanged within the task. Furthermore, data exchange is only executed if the program instances are

linked to one another via IN and OUT ports. This ensures that the data is always stable and consistent during execution.
The following figure shows the behavior of task 1 and task 2:

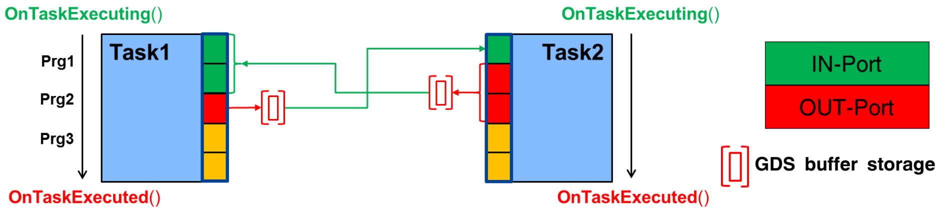


Figure 2-11 Example: Port communication with task 1 and task 2 buffer storage units

The following applies to the connection of IN and OUT ports:

- OUT ports can be shared within an application. One OUT port can be connected to several IN ports.
- An IN port can only be connected to one OUT port.

Further information on connecting ports is available in the PLCnext Engineer online help (see “Role mapping: Assigning PLCnext ports”).

2.8.2 Fieldbus connection

Connection via IN and OUT ports

In addition to the communication between different tasks, process data from and to fieldbuses can also be exchanged via IN and OUT ports. Fieldbus systems also have buffer storage units, although these are arranged logically within the fieldbus component itself. Values are written to and read from these buffer storage units by the GDS. The fieldbus performs further handling.

Figure 2-12 “Example: Fieldbus communication via IN and OUT ports” shows the behavior of fieldbus communication via ports.

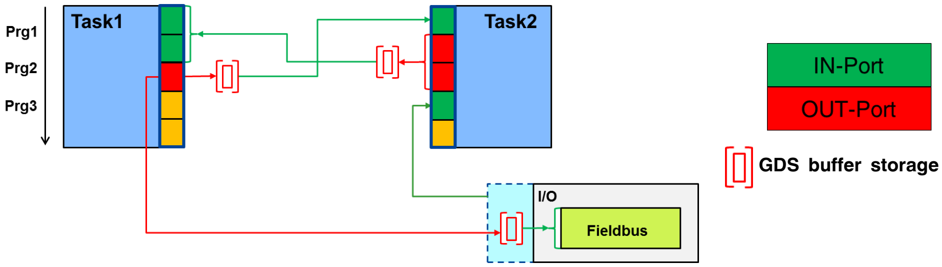


Figure 2-12 Example: Fieldbus communication via IN and OUT ports

All the components of a PLCnext Technology application as well as I/O components that are part of the data exchange process must be connected via IN and OUT ports. At the beginning of a task, the task retrieves the data from the buffer storage unit. At the end of the task, the calculated values are actively rewritten to the buffer storage unit. Therefore, the buffer storage unit is the data exchange area between an IN and an OUT port. The IN and OUT port connection between fieldbus components and tasks shows an analog behavior to the data exchange via ports between several tasks (see Section 2.8.1 “Port-based communication” on page 27).

By using the IN and OUT ports, the variable values are updated in the task context. The data is written to or read from the GDS buffer storage unit and is available to a task for the entire cycle. Using the IN and OUT ports secures task-internal data consistency. The fieldbus data is consistent as it comes from the same sampling cycle, and is also consistent within the task cycles. This also applies if programs are interrupted and a fieldbus is updated within this period of interruption. The variable value is buffered in the GDS and is available to the task for the period of time it requires to process all related programs, even with interruption.

When a fieldbus is connected to an ESM task, I/O data can be exchanged via the IN and OUT ports. For this, declare the variables in the data list of the “PLCnext” node of PLCnext Engineer as IN or OUT ports. The I/O data is updated in the update cycle of the ESM task.

Example
Port connection

A cyclic task has an update rate of 30 ms, for example. The user programs of the task consume the I/O data of the linked I/O variables. The I/O variables are updates at a cycle of 30 ms and are available to the programs. In this way, the programs in the task can also use the latest values. At the beginning of the task, the data is written from the fieldbus OUT port to the IN port of the task. At the end of the task, the output data is written to the IN port of the fieldbus via the OUT port of the task.

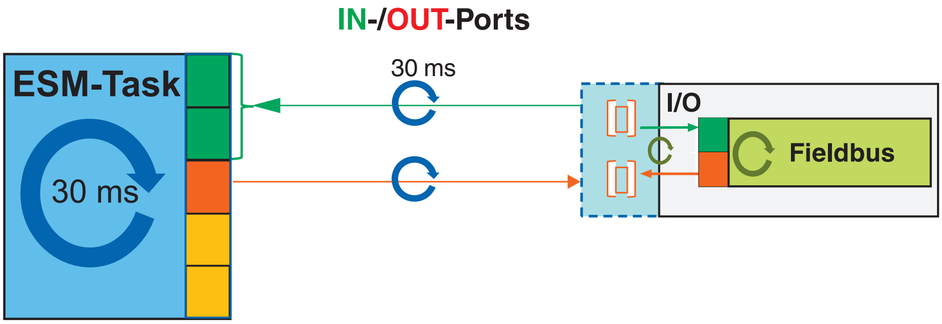


Figure 2-13 Fieldbus connection via ports

Connection via resource-global variables

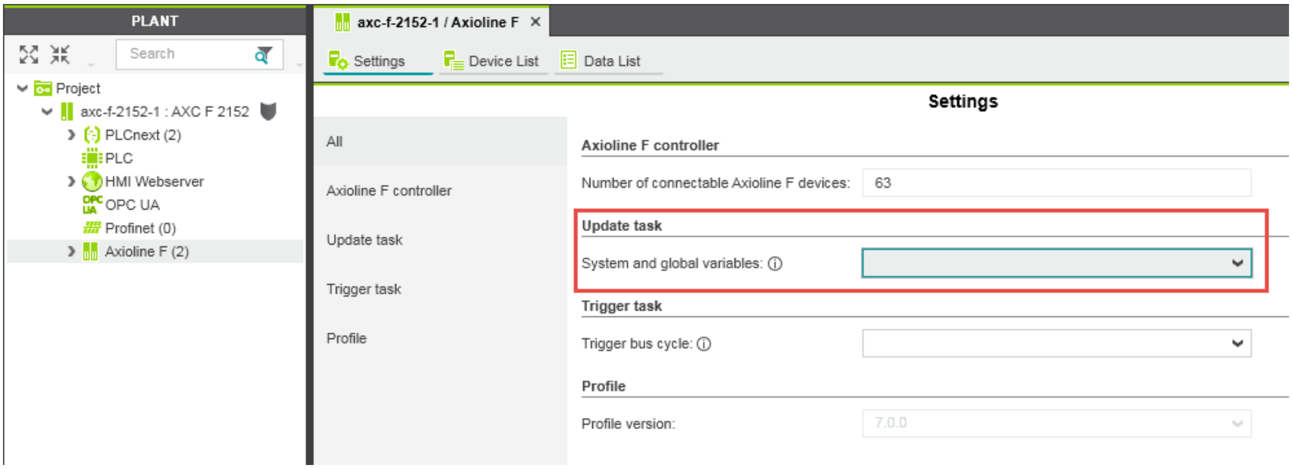
To avoid inconsistent I/O data and the resulting errors, Phoenix Contact recommends to implement the fieldbus connection via the IN and OUT ports.

Resource-global
variables as input

When creating your application, remember that there is no claim for data consistency when using resource-global variables. According to IEC 61131-3, resource-global variables can be used as an input in all Program Organization Units (POE) of the current resource. Several ESM tasks can access the fieldbus I/O data declared as resource-global variable. Declare the variables as “Program” in the local variable table or in the PLCnext Engineer data lists. This way, the I/O data can be used in the programs of different tasks. The update rate of the resource-global variable always depends on only one task. For the update behavior, an ESM task can be selected. You can make the setting in PLCnext Engineer:

- Open the respective node in the “PLANT” area for this.

- From the “Update task” drop-down list in the “Settings” editor, select the desired task.



Settings

Axioline F controller

Number of connectable Axioline F devices: 63

Update task

System and global variables:

Trigger task

Trigger bus cycle:

Profile

Profile version: 7.0.0

Figure 2-14 Selecting “Update task” (in this example: Axioline F)

NOTE: “GLOBALS” task update behavior

If no update task is selected that triggers the update cycle, the “GLOBALS” task is used (low priority and update rate of 50 ms). You can set one update task each for the groups of fieldbuses, ESM system variables and the system variables of the PLCnext Engineer HMIs.

Resource-global variables as output

Usually, resource-global variables are only used once as an ouput in order to avoid unintentional overwriting of values.

Note: If you use resource-global variables several times as an output, the last value overwrites the previous values.

Example of fieldbus connection via resource-global variables

In the example in Figure 2-15, the update rate of the global variable depends on the cycle time of ESM task 1. The update time of 30 ms for the global variables therefore also applies to ESM task 2 and other tasks, if applicable. The cycle time of ESM task 2 is 20 ms. Still, it must process the cycle with the values synchronized by task 1. For task 2, this means that the data does not exactly correspond to the current process data image of the fieldbus components, but that data can be up to 30 ms old.

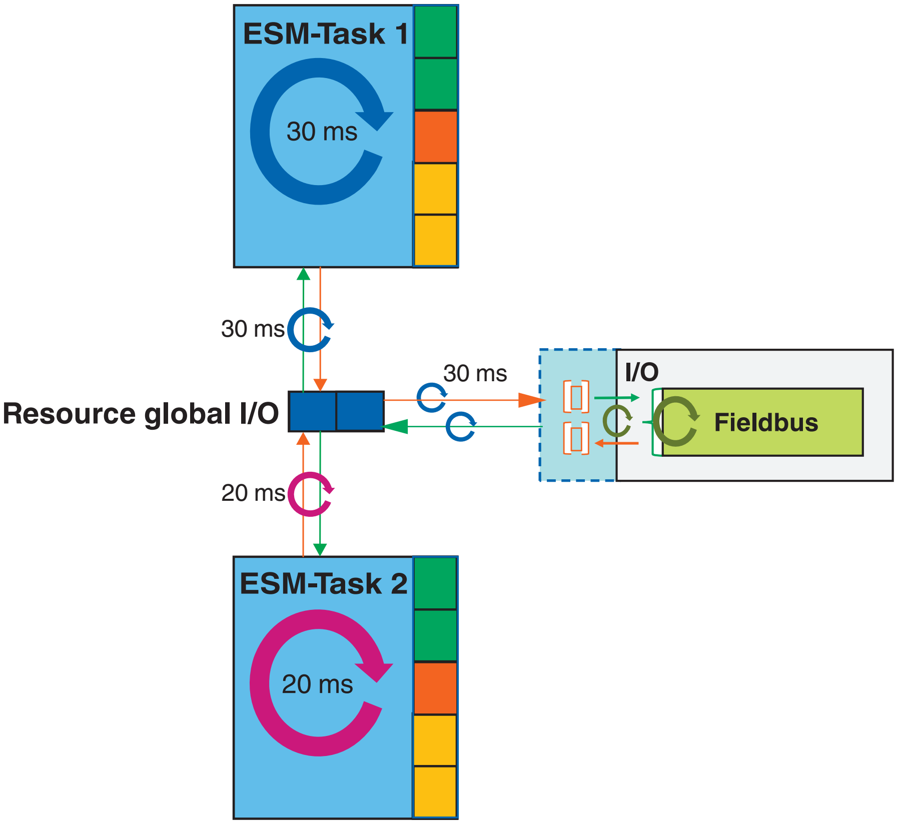


Figure 2-15 Example: Fieldbus connection via resource-global variables

Interruption of tasks

If task 2 from the example in Figure 2-15 has a higher priority than task 1, it might be the case that task 1 is interrupted by task 2 when updating fieldbus data. Then, task 2, with its higher priority, is processed. Once task 2 is processed, there already is an updated data image from the fieldbus due to the update rate. The interrupted task 1 is now continued and accesses the fieldbus data that is still missing. However, this data is not consistent with the data retrieved before interruption.

Take this behavior into consideration when creating your application. Processing of inconsistent data can result in errors in the process.

Comparison of connections via IN and OUT ports and resource-global variables

Connection via IN and OUT ports:

- Fieldbus data is consistent as it was retrieved and written to the buffer storage unit at the same time.
- Thanks to the GDS buffer storage units, fieldbus data is consistent during task processing.

- The user is not responsible for data consistency as it is ensured via the GDS buffer storage units.
- Connecting IN and OUT ports of the task and fieldbus components is complex and less flexible than using resource-global variables.

To avoid inconsistent I/O data and the resulting errors, Phoenix Contact recommends to implement the fieldbus connection via the IN and OUT ports.

Resource-global variables:

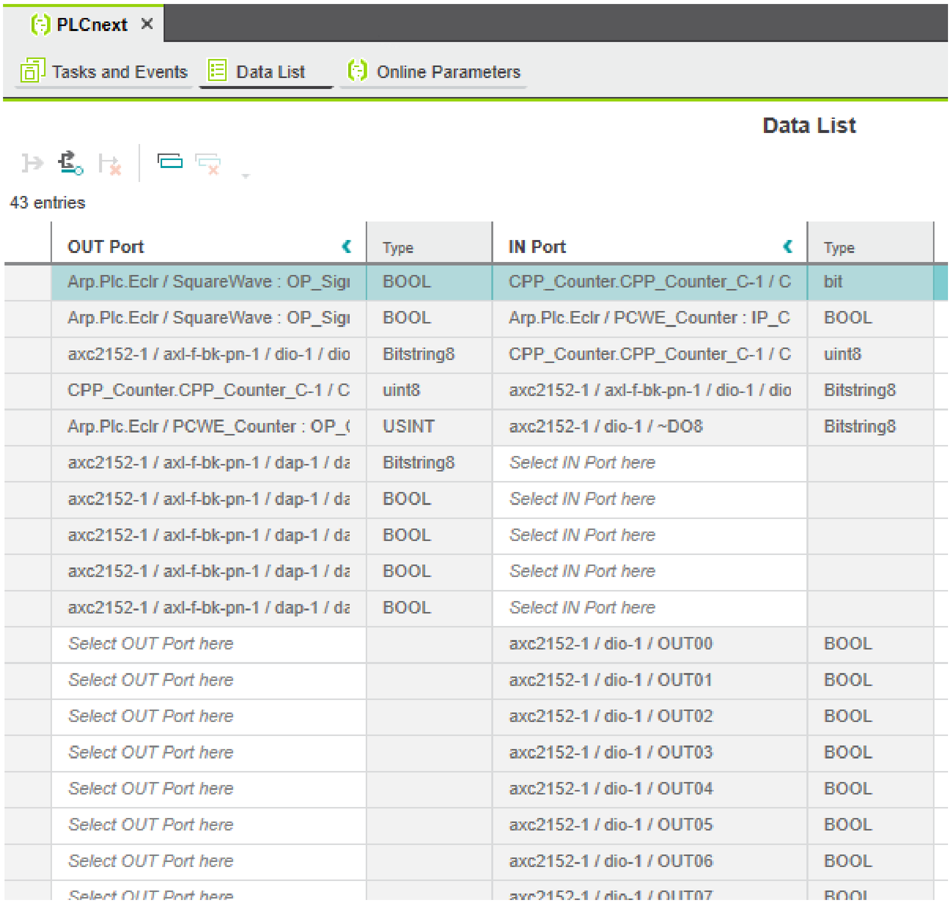
- At first sight, using resource-global variables seems to be quite easy.
- Update rates and interruptions of tasks due to priority can result in inconsistencies in the process data image.
- Inconsistent data can result in errors in the application.
- The user is responsible for data consistency, which is not ensured by the system.

2.8.3 GDS configuration with PLCnext Engineer

You can configure the GDS by connecting IN and OUT ports of programs to one another using the PLCnext Engineer software (see also 9 “Importing libraries into PLCnext Engineer”).

In PLCnext Engineer, select the desired use as an IN or OUT port in the “Data List” editor of the “PLCnext” node for the respective process data. You can see an overview of all IN and OUT ports available in the GDS in the “Data List“ editor. The programs communicate via the IN and OUT ports of the GDS. Communication is also possible between IEC 61131-3 programs, and programs that were created using other programming languages (e.g., C++). For this purpose, the IN and OUT ports must be assigned to one another.

IN and OUT ports are **only** displayed in the “Data List” editor of the “PLCnext” node.



Data List

43 entries

OUT Port	Type	IN Port	Type
Arp.Plc.Eclr / SquareWave : OP_Sigı	BOOL	CPP_Counter.CPP_Counter_C-1 / C	bit
Arp.Plc.Eclr / SquareWave : OP_Sigı	BOOL	Arp.Plc.Eclr / PCWE_Counter : IP_C	BOOL
axc2152-1 / axl-f-bk-pn-1 / dio-1 / dio	Bitstring8	CPP_Counter.CPP_Counter_C-1 / C	uint8
CPP_Counter.CPP_Counter_C-1 / C	uint8	axc2152-1 / axl-f-bk-pn-1 / dio-1 / dio	Bitstring8
Arp.Plc.Eclr / PCWE_Counter : OP_(	USINT	axc2152-1 / dio-1 / ~DO8	Bitstring8
axc2152-1 / axl-f-bk-pn-1 / dap-1 / da	Bitstring8	*Select IN Port here*	
axc2152-1 / axl-f-bk-pn-1 / dap-1 / da	BOOL	*Select IN Port here*	
axc2152-1 / axl-f-bk-pn-1 / dap-1 / da	BOOL	*Select IN Port here*	
axc2152-1 / axl-f-bk-pn-1 / dap-1 / da	BOOL	*Select IN Port here*	
axc2152-1 / axl-f-bk-pn-1 / dap-1 / da	BOOL	*Select IN Port here*	
Select OUT Port here		axc2152-1 / dio-1 / OUT00	BOOL
Select OUT Port here		axc2152-1 / dio-1 / OUT01	BOOL
Select OUT Port here		axc2152-1 / dio-1 / OUT02	BOOL
Select OUT Port here		axc2152-1 / dio-1 / OUT03	BOOL
Select OUT Port here		axc2152-1 / dio-1 / OUT04	BOOL
Select OUT Port here		axc2152-1 / dio-1 / OUT05	BOOL
Select OUT Port here		axc2152-1 / dio-1 / OUT06	BOOL
Select OUT Port here		axc2152-1 / dio-1 / OUT07	BOOL

Figure 2-16 Example: List of all available IN and OUT ports

A detailed description of the procedure and additional information on GDS configuration with PLCnext Engineer is available in the online help, the PLCnext Engineer quick start guide (PLCNEXT ENGINEER, Order no. 1046008) and the AXC F 2152 controller user manual (Order no. 2404267). The documentation for the respective products can be downloaded at phoenixcontact.net/products. You can call up the online help from within PLCnext Engineer.

2.8.4 GDS configuration using configuration files

When a project is downloaded from PLCnext Engineer to the controller, the previous PLCnext Engineer configuration files are overwritten with the new configuration files. Section "Manual configuration" on page 20 describes how you can safely store your manually modified configuration in the controller file system, thus preventing the loss of data.

You can also modify the GDS configuration without PLCnext Engineer but using configuration files. All of the important settings can be configured directly in the *.gds.config configuration file on the controller. All configuration files (except *.libmeta) are created by PLCnext Engineer. To modify the configuration, the XML file can be edited using any editor.

You will find the file in the “/opt/plcnext/projects/.../Plc” directory of your controller (see Section 2.14.1 “Directories of the firmware components in the file system”). The file system is accessed via the SFTP protocol. Use a suitable SFTP client software for this, e.g., WinSCP.

The individual connections with their respective IN and OUT ports are defined between the `<Connectors>` and `</Connectors>` tags. A connector is a directed connection from `startPort` to `endPort`. An OUT port is entered as a `startPort` attribute and an IN port as an `endPort` attribute. Within the *.gds.config, each port has its own unique name that has the following structure:

`ComponentInstanceName/GroupName.VariableName`

Attribute	Explanation
`ComponentInstanceName`	Components of a PLCnext Technology application can be – Programs – PROFINET controllers (e.g., “Arp.Io.PnC”/“Arp.Io.PnD”) – The IEC 61131-3 runtime (e.g., Arp.Plc.Eclr.EclrComponent) – Or a C++ application, (e.g., Counters.CppCounterComponent) Names that start with “Arp...” are specified by the PLCnext Technology firmware. The names of the C++ applications are specified by the user during instantiation. Exception: When instantiating C++ or Matlab® Simulink® programs in PLCnext Engineer, the name of the component instance is specified by PLCnext Engineer. There can be one or multiple instances of each component. In the case of a PROFINET device, it is possible to gain access by entering the node ID (for more information, please refer to the online help within PLCnext Engineer). • Enter the name of the instance here.
`GroupName`	With program instances, the name is specified. The name of a program instance must be unique within the controller. • Enter the name of the program instance. A local bus device is defined based on the position of the device in the local bus. The position numbering starts with digit 0. • Enter the position number instead of `GroupName`.
`VariableName`	• Enter the name of the IN and OUT ports or the process data here, depending on the component type.

Examples:

- Program port:
  `startPort/endPort=“Namespace.ComponentInstance/ProgramInstanceA.Freigabe“`
- Fieldbus port:
  `startPort/endPort=“Arp.Io.AxlC/0.~DO16“`

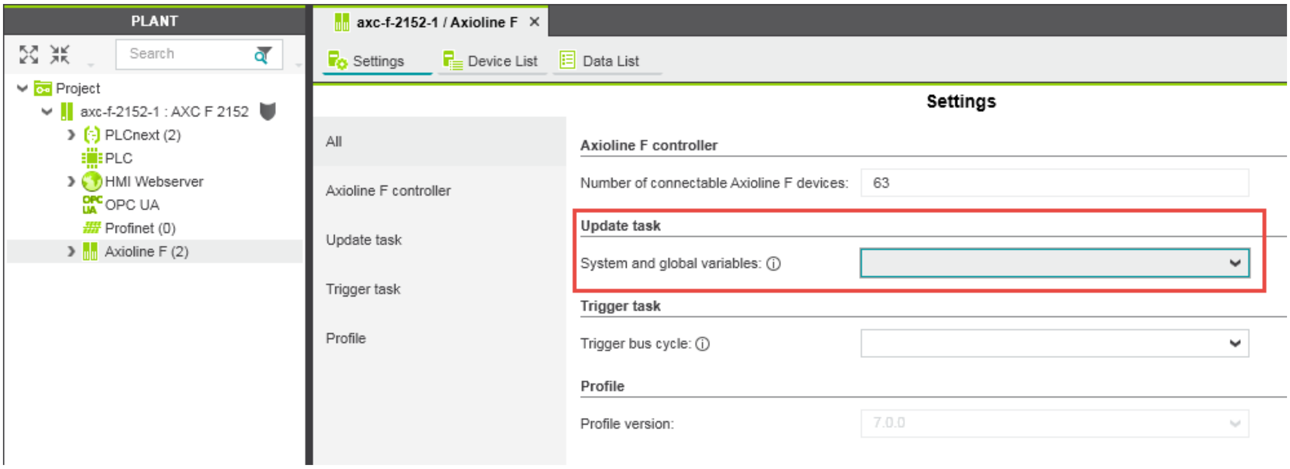
– Component port:
  `startPort/endPort="ComponentInstance/Freigabe"`

There are various types of ports that can be used for data exchange. The port concept can be used with the following process data:

Description	Code with example
Data of the runtime system of the controller	Enter the name of the program instance and the variable name as the port in the following format: `runtime system/program instance.variable name` (e.g.,: `Arp.Plc.Eclr/MainInstance.OUT_PORT_A`) You can use global variables as well as IN and OUT ports.
Data of a high-language project on the controller	Enter the name of the program instance and the variable name as the port in the following format (in accordance with the format in the *.acf.config file): `library name.component instance/program instance.variable name` (e.g.,: `CppCounterLibrary.CppCounterComponent-1/CppCounterProgram1.IP_CppEnable`)
PROFINET process data	Enter the name of the PROFINET controller and the process data name as the port in the following format: `PROFINET Controller/Node-ID.process data name` (e.g.,: `Arp.Io.PnC/46.~DI8`)
Local bus process data	Enter the name of the local bus controller and the process data name as the port in the following format: `Local bus controller/module position.process data name` (e.g.,: `Arp.Io.AxlC/0.~DO8`)

The cycle for updating system variables can also be configured. Resource-global variables (IEC 61131-3) can be connected to fieldbus variables or variables of other ARP components. As these are connections of component global variables, the events of a permanently defined task (update task) are used for the cyclic update of variable values. You can configure the update time in PLCnext Engineer:

- Open the respective node in the “PLANT” area for this.
- Select the desired task from the “Update task” drop-down list in the “Settings” editor.



Settings

Axioline F controller

Number of connectable Axioline F devices: 63

Update task

System and global variables:

Trigger task

Trigger bus cycle:

Profile

Profile version: 7.0.0

Figure 2-17 Selecting “Update task”

You can select an update task for the following variables:

Table 2-1 Update task

Component	Variables	`component` tag (config file)
Controller	ESM system variables	`Arp.Plc.Esm`
HMI web server	HMI system variables	`Arp.Services.Ehmi`
Axioline	– Axioline system variables – Global variables linked to I/O	`Arp.Plc.AxlC`
INTERBUS	– INTERBUS system variables – Global variables linked to I/O	`Arp.Io.IbM`
PROFINET	– PROFINET device system variables – PROFINET controller system variables – GlobVars linked to I/O	– `Arp.Io.PnD` – `Arp.Io.PnC`
SPNS	Safety and PROFIsafe system variables	`Arp.Services.SpnsProxy`

If you do not select a task for a component, the “Globals” task is used with a cycle time of 50 ms (default).

The settings made in PLCnext Engineer are saved to a *.gds.config file and downloaded to the controller.

The assignment of variables to an update task is made between tags `<ComponentTaskRelations>` and `</ComponentTaskRelations>`.

Example excerpt from a *.gds.config file. Here, the IN and OUT ports of an application are assigned to each other:

```
<?xml version="1.0" encoding="utf-8"?>
<GdsConfigurationDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xm-
lns:xsd="http://www.w3.org/2001/XMLSchema" schemaVersion="1.0" xmlns="http://www.phoenixcon-
tact.com/schema/gdsconfig">
<ComponentTaskRelations>
    <ComponentTaskRelation component="Arp.Plc.Esm" task="Cyclic100" />
    <ComponentTaskRelation component="Arp.Plc.AxlC" task ="Task100ms" />
    <ComponentTaskRelation component="Arp.Services.SpnsProxy" task ="Task100ms" />
</ComponentTaskRelations>
<Connectors>
    <Connector startPort="Arp.Plc.Eclr/SquareWave.OP_Signal1"
    endPort="CPP_Counter.CPP_Counter_C-1/CPP_Counter.IP_CppEnable_bit" />
    <Connector startPort="Arp.Plc.Eclr/SquareWave.OP_Signal2"
    endPort="Arp.Plc.Eclr/PCWE_Counter.IP_CounterEnable" />
    <Connector startPort="Arp.Plc.Eclr/SquareWave.OP_Signal2" endPort="Arp.Io.PnC/20.OUT00" />
    <Connector startPort="Arp.Io.PnD/PND_S1_PLC_RUN" endPort="Arp.Plc.Eclr/PND_S1_PLC_RUN" />
    <Connector startPort="Arp.Io.PnD/PND_S1_VALID_DATA_CYCLE"
    endPort="Arp.Plc.Eclr/PND_S1_VALID_DATA_CYCLE" />
    <Connector startPort="Arp.Io.PnD/PND_S1_OUTPUT_STATUS_GOOD"
    endPort="Arp.Plc.Eclr/PND_S1_OUTPUT_STATUS_GOOD" />
    <Connector startPort="Arp.Io.PnD/PND_S1_INPUT_STATUS_GOOD"
    endPort="Arp.Plc.Eclr/PND_S1_INPUT_STATUS_GOOD" />
    <Connector startPort="Arp.Io.PnD/PND_S1_DATA_LENGTH"
    endPort="Arp.Plc.Eclr/PND_S1_DATA_LENGTH" />
    <Connector startPort="Arp.Plc.Eclr/PND_S1_OUTPUTS" endPort="Arp.Io.PnD/PND_S1_OUTPUTS" />
    <Connector startPort="Arp.Io.PnD/PND_S1_INPUTS" endPort="Arp.Plc.Eclr/PND_S1_INPUTS" />
    <Connector startPort="Arp.Io.AxlC/AXIO_DIAG_STATUS_REG_HI"
    endPort="Arp.Plc.Eclr/AXIO_DIAG_STATUS_REG_HI" />
    <Connector startPort="Arp.Io.AxlC/AXIO_DIAG_STATUS_REG_LOW"
    endPort="Arp.Plc.Eclr/AXIO_DIAG_STATUS_REG_LOW" />
    <Connector startPort="Arp.Io.AxlC/AXIO_DIAG_PARAM_REG_HI"
    endPort="Arp.Plc.Eclr/AXIO_DIAG_PARAM_REG_HI" />
    <Connector startPort="Arp.Io.AxlC/AXIO_DIAG_PARAM_REG_LOW"
    endPort="Arp.Plc.Eclr/AXIO_DIAG_PARAM_REG_LOW" />
    <Connector startPort="Arp.Io.AxlC/AXIO_DIAG_PARAM_2_REG_HI"
    endPort="Arp.Plc.Eclr/AXIO_DIAG_PARAM_2_REG_HI" />
    <Connector startPort="Arp.Io.AxlC/AXIO_DIAG_PARAM_2_REG_LOW"
    endPort="Arp.Plc.Eclr/AXIO_DIAG_PARAM_2_REG_LOW" />
    <Connector startPort="Arp.Io.AxlC/AXIO_DIAG_STATUS_REG_PF"
    endPort="Arp.Plc.Eclr/AXIO_DIAG_STATUS_REG_PF" />
    <Connector startPort="Arp.Io.AxlC/AXIO_DIAG_STATUS_REG_BUS"
    endPort="Arp.Plc.Eclr/AXIO_DIAG_STATUS_REG_BUS" />
    <Connector startPort="Arp.Io.AxlC/AXIO_DIAG_STATUS_REG_RUN"
    endPort="Arp.Plc.Eclr/AXIO_DIAG_STATUS_REG_RUN" />
    <Connector startPort="Arp.Io.AxlC/AXIO_DIAG_STATUS_REG_ACT"
    endPort="Arp.Plc.Eclr/AXIO_DIAG_STATUS_REG_ACT" />
```

2.8.5 Supported data types

The programs of a PLCnext Technology application communicate via IN ports and OUT ports. The combination of the following data types is supported.

When setting the IN and OUT ports with PLCnext Engineer, you can only enter permitted combinations of data types.
If you implement the configuration without PLCnext Engineer but via the XML configuration file, you have to ensure that only the data type combinations listed in Table 2-2 are used. If an invalid combination is configured, the startup process of the PLCnext Technology firmware is interrupted.
Information on the startup behavior of the firmware is available in the Output.log diagnostic file. The file contains status and error messages as well as warning notes that help you find the source of error. The Output.log file is stored in the controller file system in the “/opt/plcnext/logs” directory. The file system is accessed via the SFTP protocol. Use a suitable SFTP client software for this, e.g., WinSCP (see Section ““Template Loggable”” on page 129).

2.8.5.1 Elementary data types

Table 2-2 Supported data type combinations between C++, Simulink®, and PLCnext Engineer programs

C++	Simulink®	PLCnext Engineer	Use in array data type	Use in struct data type
Boolean	Boolean	BOOL	x	x
int8	int8	SINT	x	x
uint8	uint8	USINT	x	x
int16	int16	INT	x	x
uint16	uint16	UINT	x	x
int32	int32	DINT	x	x
uint32	uint32	UDINT	x	x
int64		LINT	x	x
uint64		ULINT	x	x
uint8	uint8	BYTE	x	x

Table 2-2 Supported data type combinations between C++, Simulink®, and PLCnext Engineer programs

C++	Simulink®	PLCnext Engineer	Use in array data type	Use in struct data type
uint16	uint16	WORD	x	x
uint32	uint32	DWORD	x	x
uint64		LWORD	x	x
float32	single	REAL	x	x
float64	double	LREAL	x	x
		Array of primitive data types		x

The following data type combinations between C++ programs are supported:

Table 2-3 Supported combinations of IN and OUT ports between C++ programs

StartPort	EndPort
Boolean	**Boolean**
	uint8
	uint16
	uint32
	uint64
	char8
	int8
	int16
	int32
	int64
	float32
	float64
char8	**char8**
	uint16
	uint32
	uint64
	int16
	int32
	int64
	float32
	float64
uint8	**uint8**
	uint16
	uint32
	uint64
	int16
	int32
	int64
	float32
	float64
uint16	**uint16**
	uint32
	uint64
	int32
	int64
	float32
	float64
uint32	**uint32**
	uint64
	int64
	float64
uint64	**uint64**
int8	**int8**
	int16
	int32
	int64
	float32
	float64
int16	**int16**
	int32
	int64
	float32
	float64
int32	**int32**
	int64
	float64
int64	**int64**
float32	**float32**
	float64
float64	**float64**

2.8.5.2 Connection of structures

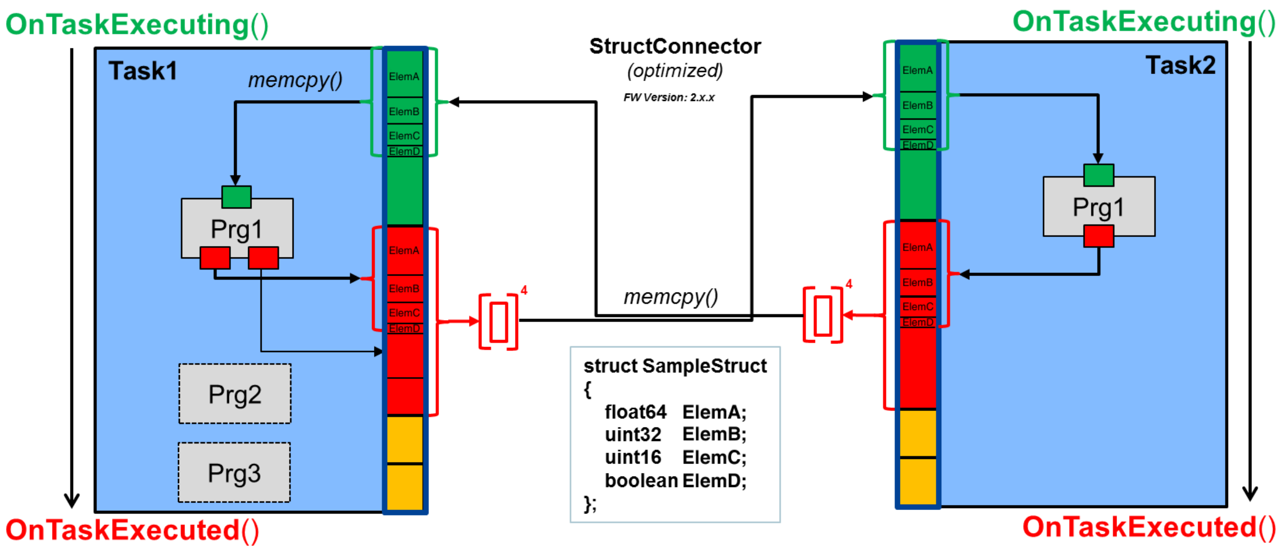
In addition to connecting elementary data types and arrays, the GDS also enables the connection of entire structures. Structures are data types that can exist of elements of different data types. The design of structures can be freely defined.

A StructConnector is used for connecting two struct ports. A StructConnector is a management object that is used to implement the connection of complete structures and the actual data transport. The connection logic of two struct ports therefore contains a comparison based on type definitions. It is checked whether the size of the structures, the data type, the offset, the dimensions, and the number of elements match. It is also checked if the alignment of the individual struct members is consistent. This is important for determining the binary compatibility of the structures. The check is implemented without the names of the individual struct members.

Complete copying

The firmware completely copies the structures via `memcpy()`. Binary compatibility of the type definitions is required for this method.
To compare the size of the structure types and subsequently, the data types and offsets of the individual structure fields, a type check is run.

Below figure shows the connection of structures between tasks. Here, the communication between two programs is implemented via a type “SampleStruct” port



```
struct SampleStruct
{
    float64  ElemA;
    uint32   ElemB;
    uint16   ElemC;
    boolean  ElemD;
};
```

Figure 2-18 Connection of structures

2.9 RSC (Remote Service Calls)

Internal user components can communicate with the PLCnext Technology core components via the RSC interface. You can access various functions and data items via the interfaces. For example, you can gain read and write access to GDS data using the “IDataAccessService” RSC service.

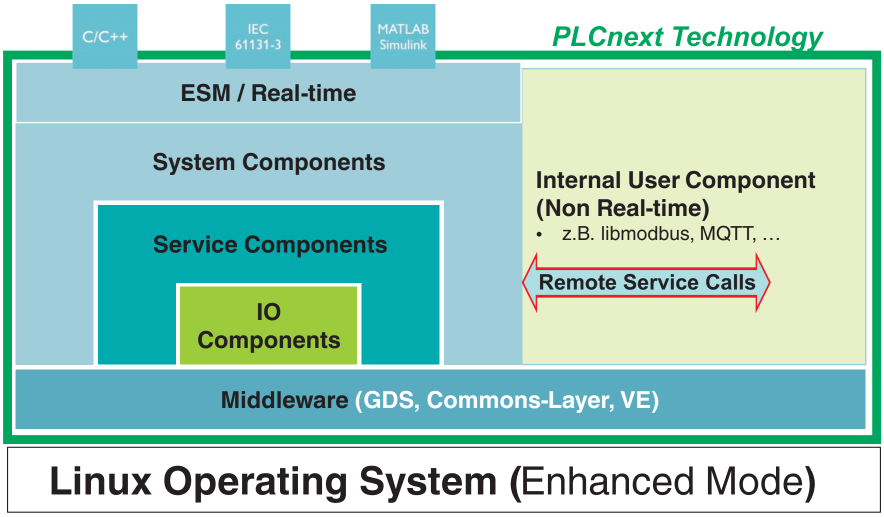

C/C++
IEC 61131-3
MATLAB Simulink
PLCnext Technology
ESM / Real-time
System Components
Service Components
IO Components
Internal User Component (Non Real-time)
• z.B. libmodbus, MQTT, ...
Remote Service Calls
Middleware (GDS, Commons-Layer, VE)
Linux Operating System (Enhanced Mode)

Figure 2-19 PLCnext Technology – RSC services

ServiceManager

You have the option of using the already registered RSC services of the SDK (Software Development Kit) via the ServiceManager. The ServiceManager acts as the RSC API and is used to request services.

For additional information on using RSC services, please refer to Section 5.8, “Using RSC services”.

2.10 PLCnext embedded OPC UA server (eUA)

2.10.1 OPC UA

OPC Unified Architecture (UA) is a standardized protocol for the industrial IT and OT communication. On request, an OPC UA server provides an OPC UA client with process data and variable values from a running application.

The PLCnext controllers contain an embedded OPC UA server (eUA). It is integrated in the controller in addition to the runtime. It enables access to components, programs, function blocks, structures, and variables of PLCnext Technology.

2.10.2 Configuration

If a controller features an integrated OPC UA server, the server is displayed in the PLCnext Engineer software in the “PLANT” area (“OPC UA”). Here, you can configure the OPC UA server. The configuration is loaded to the controller as part of a PLCnext Engineer project and in form of a configuration file. It contains all parameters for setting the OPC UA server. All users of an OPC UA client must authenticate themselves to the OPC UA server with a user name and a password. You can create a user via WBM of the controller (see Section "Web-based management (WBM)" on page 73). In the course of this, you have to assign the necessary roles:

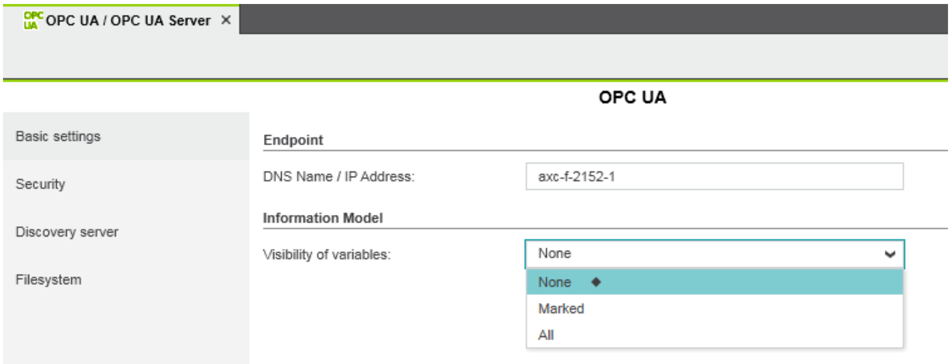
- DataViewer
- DataChanger
- FileReader
- FileWriter

For additional information on user roles, please refer to Section ""User Authentication" page" on page 82.

For further information on configuring OPC UA in PLCnext Engineer, please refer to the PLCnext Engineer online help.

With PLCnext Engineer, the following configurations are possible:

- **Defining the server endpoint URI**
  Define the name of the network node the eUA server is to use in the server URI and in the endpoint URL.
- **Defining a Global Discovery Server:**
  Define which certificate the server is to use.
- **Visibility of variables and alarms for the OPC UA clients:**
  Due to security reasons, the variables and ports of a program in PLCnext Engineer are set to not visible by default. Visibility can be set in your PLCnext Engineer project. In the "PLANT" area, open the "OPC UA" node and then the "Basic settings". Via the "Visibility of variables" drop-down list, you can set the visibility of variables and alarms for the OPC UA clients.



DNS Name / IP Address: axc-f-2152-1

Visibility of variables: None

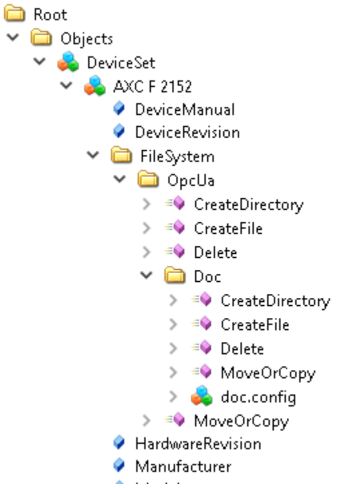
Figure 2-20 Setting the visibility of variables

- **Privilege settings for data access:**
  You can configure access of clients to the file system of the server. Read and write access for clients to selected folders and files in the file system of the server as well as creating additional directories and files are possible. Once this option is active, only PLCnext Technology users with a "FileReader" or "FileWriter" role can read or write files. The required roles are assigned to the user in WBM of the controller (see Section ""User Authentication" page" on page 82).

2.10.3 OPC UA file access

All directories and files are displayed in the http://phoenixcontact.com/OpcUA/Files/ namespace. Via the namespace, the corresponding “NamespaceIndex” can be determined from the “NamespaceArray”. The “NamespaceIndex” is a numeric value for identification of the namespace. It is saved in the “Namespace Array”. Via the “NamespaceIndex” and the “NodeIdentifier”, the released data and directories can be retrieved.

The initial entry point for released directories and files is the “FileSystem” folder of type “FolderType”. In this folder, released files and directories are displayed, starting with the symbolic folder indicated in the config file.



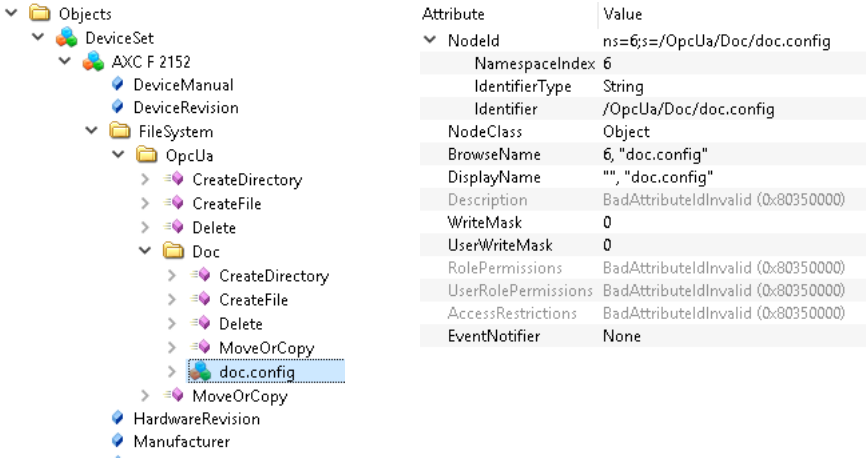
```
Root
  Objects
    DeviceSet
      AXC F 2152
        DeviceManual
        DeviceRevision
        FileSystem
          OpcUa
            CreateDirectory
            CreateFile
            Delete
            Doc
              CreateDirectory
              CreateFile
              Delete
              MoveOrCopy
              doc.config
            MoveOrCopy
        HardwareRevision
        Manufacturer
```

Figure 2-21 “FileSystem” folder

The “NodeIdentifier” is structured as follows:

[Symbolic name]|[directory name]/[file name]

Example: OpcUa|OpcUa/Doc/doc.config



```
Objects
  DeviceSet
    AXC F 2152
      DeviceManual
      DeviceRevision
      FileSystem
        OpcUa
          CreateDirectory
          CreateFile
          Delete
          Doc
            CreateDirectory
            CreateFile
            Delete
            MoveOrCopy
            doc.config
          MoveOrCopy
      HardwareRevision
      Manufacturer
```

Attribute	Value
NodeId	ns=6;s=/OpcUa/Doc/doc.config
NamespaceIndex	6
IdentifierType	String
Identifier	/OpcUa/Doc/doc.config
NodeClass	Object
BrowseName	6, "doc.config"
DisplayName	"", "doc.config"
Description	BadAttributeIdInvalid (0x80350000)
WriteMask	0
UserWriteMask	0
RolePermissions	BadAttributeIdInvalid (0x80350000)
UserRolePermissions	BadAttributeIdInvalid (0x80350000)
AccessRestrictions	BadAttributeIdInvalid (0x80350000)
EventNotifier	None

Figure 2-22 “NodeIdentifier” structure

2.10.4 Alarms

You can exchange alarm and status messages between a program and the eUA server. For additional information, please refer to Section “Alarms” on page 48.

2.10.5 Subscriptions

An OPC UA client can subscribe to a list of variables that are then monitored for changes by the GDS. The variables are checked for changes at a defined interval. If a changed variable is detected during the cyclic check, the GDS informs the OPC UA server, which forwards the message to the client. When subscribing to a variable, the client can define a sample rate which is then served by the server. You can choose from predefined sample rates. Sample rates are device-specific. The sample rates for the AXF F 2152 controller are defined as follows, for example:

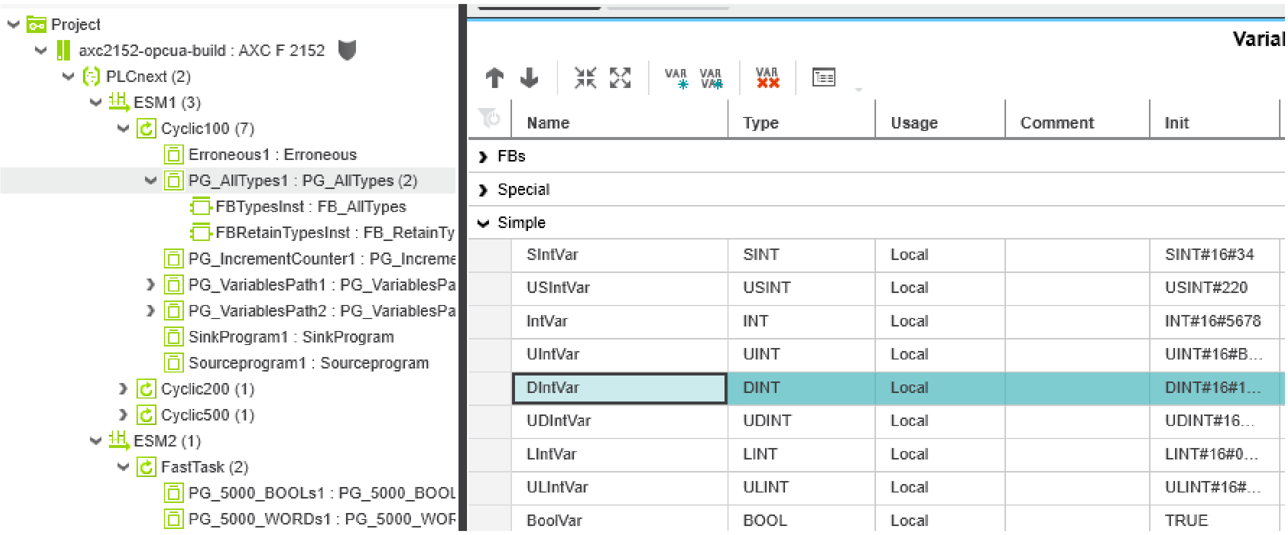
- 100 ms
- 250 ms
- 500 ms
- 1000 ms
- 2000 ms
- 5000 ms

If an OPC UA client requests a variable with the desired rate of 600 ms, for example, it will be assigned to the “500 ms” group. The server assigns the subscription to the group that is closest to the desired value and returns this value to the client as “RevisedSamplingInterval”.

2.10.6 “GlobalDataSpace” namespace

All programs and variables from the IEC 61131 context are displayed in the http://phoenixcontact.com/OpcUA/PLCnext/GlobalDataSpace/ namespace. Via the namespace, the corresponding “NamespaceIndex” can be determined from the “NamespaceArray”. The “NamespaceIndex” is a numeric value for identification of the namespace. It is saved in the “Namespace Array”. Via the “NamespaceIndex” and the “NodeIdentifier”, a corresponding process variable, structure, or function block can be retrieved. The “NodeIdentifier” is always structured as follows:
[component name]/[program name].[variable name]

Example: Arp.Plc.Eclr/PG_AllTypes1.DIntVar

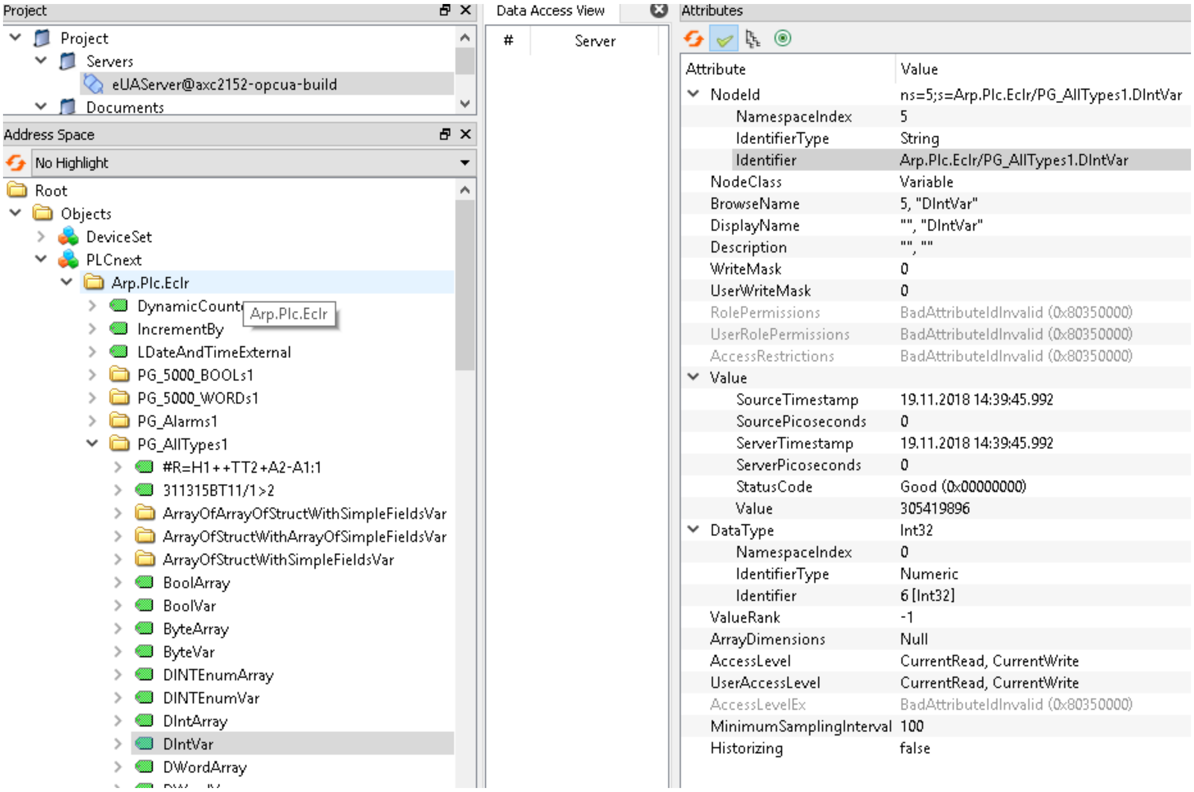


- Project
  - axc2152-opcua-build : AXC F 2152
    - PLCnext (2)
      - ESM1 (3)
        - Cyclic100 (7)
          - Erroneous1 : Erroneous
          - PG_AllTypes1 : PG_AllTypes (2)
            - FBTypesInst : FB_AllTypes
            - FBRetainTypesInst : FB_RetainTy
          - PG_IncrementCounter1 : PG_Increme
          - PG_VariablesPath1 : PG_VariablesPa
          - PG_VariablesPath2 : PG_VariablesPa
          - SinkProgram1 : SinkProgram
          - Sourceprogram1 : Sourceprogram
        - Cyclic200 (1)
        - Cyclic500 (1)
      - ESM2 (1)
        - FastTask (2)
          - PG_5000_BOOLs1 : PG_5000_BOOL
          - PG_5000_WORDs1 : PG_5000_WOF

Varial

Name	Type	Usage	Comment	Init
FBs				
Special				
Simple				
SIntVar	SINT	Local		SINT#16#34
USIntVar	USINT	Local		USINT#220
IntVar	INT	Local		INT#16#5678
UIntVar	UINT	Local		UINT#16#B...
DIntVar	DINT	Local		DINT#16#1...
UDIntVar	UDINT	Local		UDINT#16...
LIntVar	LINT	Local		LINT#16#0...
ULIntVar	ULINT	Local		ULINT#16#...
BoolVar	BOOL	Local		TRUE

Figure 2-23 “Arp.Plc.Eclr/PG_AllTypes1.DIntVar”



Project

- Project
  - Servers
    - eUAServer@axc2152-opcua-build
  - Documents

Address Space

No Highlight

- Root
- Objects
  - DeviceSet
  - PLCnext
    - Arp.Plc.Eclr
      - DynamicCount
      - IncrementBy
      - LDateAndTimeExternal
      - PG_5000_BOOLs1
      - PG_5000_WORDs1
      - PG_Alarms1
      - PG_AllTypes1
        - #R=H1++TT2+A2-A1:1
        - 311315BT11/1>2
        - ArrayOfArrayOfStructWithSimpleFieldsVar
        - ArrayOfStructWithArrayOfSimpleFieldsVar
        - ArrayOfStructWithSimpleFieldsVar
        - BoolArray
        - BoolVar
        - ByteArray
        - ByteVar
        - DINTEnumArray
        - DINTEnumVar
        - DIntArray
        - DIntVar
        - DWordArray

Arp.Plc.Eclr

Data Access View

#	Server

Attributes

Attribute	Value
NodeId	ns=5;s=Arp.Plc.Eclr/PG_AllTypes1.DIntVar
NamespaceIndex	5
IdentifierType	String
Identifier	Arp.Plc.Eclr/PG_AllTypes1.DIntVar
NodeClass	Variable
BrowseName	5, "DIntVar"
DisplayName	"", "DIntVar"
Description	"", ""
WriteMask	0
UserWriteMask	0
RolePermissions	BadAttributeIdInvalid (0x80350000)
UserRolePermissions	BadAttributeIdInvalid (0x80350000)
AccessRestrictions	BadAttributeIdInvalid (0x80350000)
Value	
SourceTimestamp	19.11.2018 14:39:45.992
SourcePicoseconds	0
ServerTimestamp	19.11.2018 14:39:45.992
ServerPicoseconds	0
StatusCode	Good (0x00000000)
Value	305419896
DataType	Int32
NamespaceIndex	0
IdentifierType	Numeric
Identifier	6 [Int32]
ValueRank	-1
ArrayDimensions	Null
AccessLevel	CurrentRead, CurrentWrite
UserAccessLevel	CurrentRead, CurrentWrite
AccessLevelEx	BadAttributeIdInvalid (0x80350000)
MinimumSamplingInterval	100
Historizing	false

Figure 2-24 “Arp.Plc.Eclr/PG_AllTypes1.DIntVar” (Software: UA Expert)

The component name for C++ components is defined in the respective C++ project. It therefore differs from the component name from the IEC 61131 context (“Arp.Plc.Eclr”).

2.10.7 Device Integration (DI) namespace

In compliance with the “OPC UA for Devices” OPC UA Companion Specification, the eUA server provides the following nodes:

Table 2-4 Device Integration node

Node	Meaning
DeviceManual	Path or URL to the user manual. Note: The eUA server does not support this node. Default value: Empty string
DeviceRevision	General revision status of the device Note: The eUA server does not support this node. Default value: Empty string
HardwareRevision	Hardware revision status, e.g., 02
Manufacturer	Device manufacturer, e.g., “en”, “Phoenix Contact”
Model	Name of the model, e.g., “en”, “AXC F 2152”
RevisionCounter	Revision status of the static device data, e.g., 2
SerialNumber	Device serial number, e.g., 13254768
SoftwareRevision	Software version number e.g., 2019.0 LTS (19.0.0.15906)

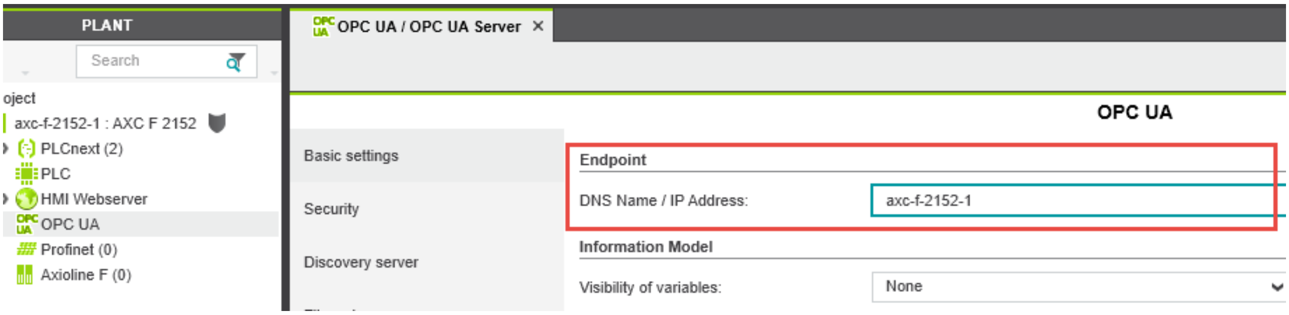
2.10.8 Data types

For a mapping table containing the PLCnext Technology data types and the corresponding OPC UA data type, see Section “Available data types” on page 191.

2.10.9 UA server endpoints

The eUA server offers an endpoint to which the clients can connect. In its URL, you can either configure the IP address of the controller or the DNS name. Implement the configuration in the PLCnext Engineer software:

- In PLCnext Engineer, open the “OPC UA” node in the “PLANT” area.
- In the “Basic settings”, either enter the DNS name or the IP address in the input field.



Endpoint

DNS Name / IP Address: axc-f-2152-1

Information Model

Visibility of variables: None

Figure 2-25 Endpoint configuration

2.10.10 Encryption algorithms

You can offer the following encryption algorithms for the endpoints to the OPC UA clients:

- Basic 128 RSA15
- Basic 256
- Basic 256 SHA256

By default, the “Basic128Rsa15" encryption algorithm is not active as this algorithm is no longer regarded as secure. However, you can activate this algorithm to be able to connect the eUA server to older OPC UA clients that support this algorithm at the maximum.

2.10.11 Ethernet ports at the controller

Currently, there is no mapping to a certain Ethernet port at the controller. Therefore, connection is possible via all connections.

2.10.12 Disabling user authentication

You can also work without security settings. To this end, deactivate the user authentication via web-based management of your controller. For a description as well as safety notes, see Section 3.9.1 on page 82. When the user authentication is disabled, the OPC UA client must not authenticate itself to the OPC UA server. This way, unrestricted access to the OPC UA server is possible.

2.11 Alarms

In PLCnext Technology, alarms are mapped as messages informing about status changes in the controller.

The alarms can be transmitted between different components. Possible alarm sources are the IEC 61131 alarm blocks, but also programs written in C++ or Matlab® Simulink®. A typical alarm recipient is the OPC UA server, which forwards the messages as OPC UA alarms to interested OPC UA clients. However, alarm clients that react to alarm messages can also be written in C++.

Alarms are defined by a complex status. All status changes are sent as a message by the alarm source. As the status changes are forwarded as OPC UA alarms, the statuses arise from the definition in the OPC UA specification.

Essential features of the alarm status are:

- Active – The alarm condition is active.
- Acknowledged – The user has seen the alarm condition.
- Confirmed – The user has solved the problem that led to the alarm.
- Severity – Severity of the alarm (from 1 = information to 1000 = severe error).
- Retain – The alarm is to be shown to the user (is evaluated by the client).
- Message – Message to be shown to the user.
- AlarmType – Alarm type (can be used for filtering in the client).
- AlarmId – Unique name of the alarm on the device.
- Time stamps for a variety of substatuses can be set as an option.

To introduce an alarm to the system, it first has to be registered. This has to be done before the first use. As a result, you in the OPC UA server which alarms can occur.

It is not possible to cancel alarms. However, they are deleted during each cold and warm restart. This is why alarms have to be registered after each cold or warm restart. Registering an alarm twice does not result in an error message.

Some alarms must be acknowledged and sometimes even confirmed. To do so, the “Acknowledge” and “Confirm” methods can be used. These are also messages in PLCnext Technology. However, they are sent from a client to the alarm source. The alarm source must process this message. Usually, this results in a new alarm status which, as usual, is sent to all clients.

Often there is the requirement to add additional information to an alarm, which is available in the client and can be displayed in the message. For this, there are alternative alarm blocks that can take on a structure with additional parameters. These parameters must be entered during registration so that they are known to the client. In the message, the parameters can be referenced using placeholders. The following placeholders are supported:

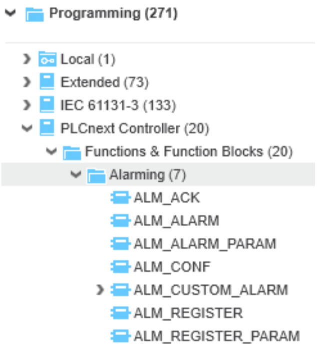
Table 2-5 Placeholders for additional parameters

Placeholder	Meaning
{0}	Alarm name The name must be unique within the controller.
{1}	Alarm type
{2}	First user parameter
{3}	Second user parameter

The “ALM_ALARM” and “ALM_ALARM_PARAM” function blocks implement a defined semantics of the alarms. Via “Requires Acknowledge” and “Requires Confirm” only, you can specify if the alarm is to be acknowledged or confirmed. For more complex scenarios, you can write your own alarm blocks. This way you can, for example, check additional conditions, implement a different time behavior, or implement several alarms with one block. For these tasks, the “ALM_CUSTOM_ALARM” block is available. It provides the “Low Level” methods that were used for implementing the other blocks. The same functions are also available in C++.

2.11.1 IEC 61131 alarm function blocks

The alarm function blocks are part of the “PLCnext Controller” library.



- Programming (271)
  - Local (1)
  - Extended (73)
  - IEC 61131-3 (133)
  - PLCnext Controller (20)
    - Functions & Function Blocks (20)
      - Alarming (7)
        - ALM_ACK
        - ALM_ALARM
        - ALM_ALARM_PARAM
        - ALM_CONF
        - ALM_CUSTOM_ALARM
        - ALM_REGISTER
        - ALM_REGISTER_PARAM

Figure 2-26 “Alarming” function blocks

There are three types of alarm function blocks:

1. Standard alarm blocks (“ALM_REGISTER”, “ALM_ALARM")
2. Blocks with user parameters (“ALM_REGISTER_PARAM”, “ALM_ALARM_PARAM”)
3. User-defined alarm blocks (“ALM_CUSTOM_ALARM”)

Alarm blocks use the “ALM_ALARM” structure for internal data storage. It is used by the blocks and does not have to be described outside the block.

Standard alarm blocks

To register an alarm, you can use the “ALM_REGISTER” function block. A rising edge or constant TRUE signal at the input introduces the alarm to the system. This function block must be called until the “DONE” output delivers a TRUE signal. If many alarms are registered at the same time, a FALSE signal might occur.

Several alarms can be registered in the same PLC cycle. However, if the “DONE” output outputs a FALSE signal, the registration can only be continued during the next cycle.

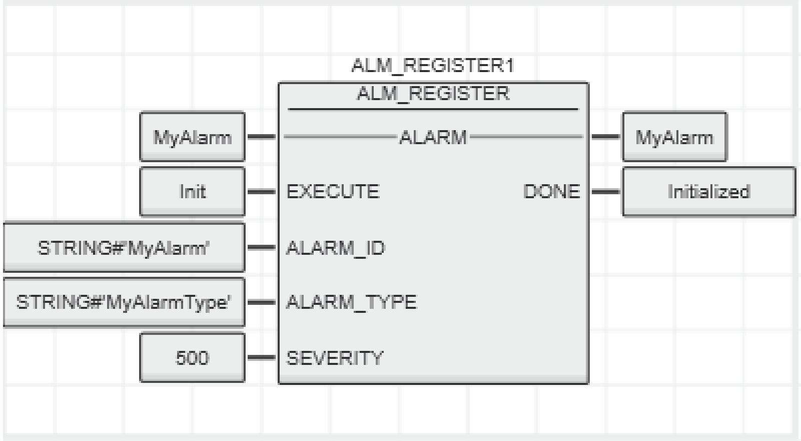


Figure 2-27 “"ALM_REGISTER” function block

To send alarm statuses, you can use the “ALM-ALARM” function block. With each change of the “ACTIVE” input, the new status is sent to the OPC UA server. This function block must be called until the “DONE” output outputs a TRUE signal. If many alarms are transmitted at the same time, a FALSE signal might occur. In this case, the call must be tried again in the next cycle.

The “AUTO_ACK” and “AUTO_CONF” inputs can be used to control if alarms require acknowledgement or confirmation. If both inputs are set to TRUE, the alarm will disappear as soon as the condition at the “ACTIVE” input changes to FALSE again.

If acknowledgement or confirmation is desired, the function block has to process these requirements. It is therefore required that the function block is called continuously so that is can internally process these messages.

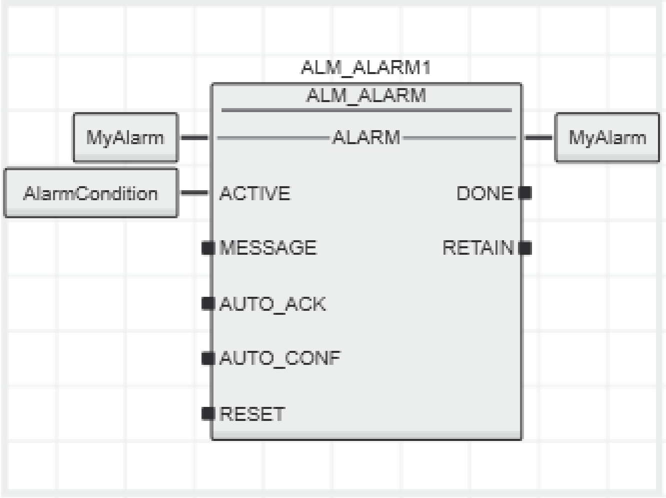


Figure 2-28 ALM_ALARM function block

Blocks with user parameters

To send user-defined parameters together with an alarm, you can use extended function blocks:

The user-defined parameters have to be introduced to the server via a structure. You can define and use a structure of up to ten parameters. The parameters must each have an elementary data type. For “ALM_REGISTER_PARAM", the names and types of the structure components are taken from the data type definition (worksheet) as names and types for the parameters. For the “ALM_ALARM_PARAM” function block, the values from the structure are transmitted with the alarm status.

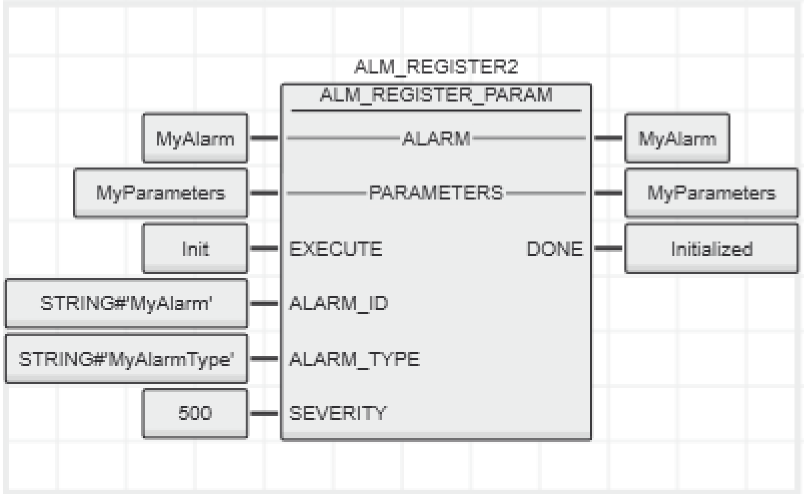


Figure 2-29 “ALM_REGISTER_PARAM” function block

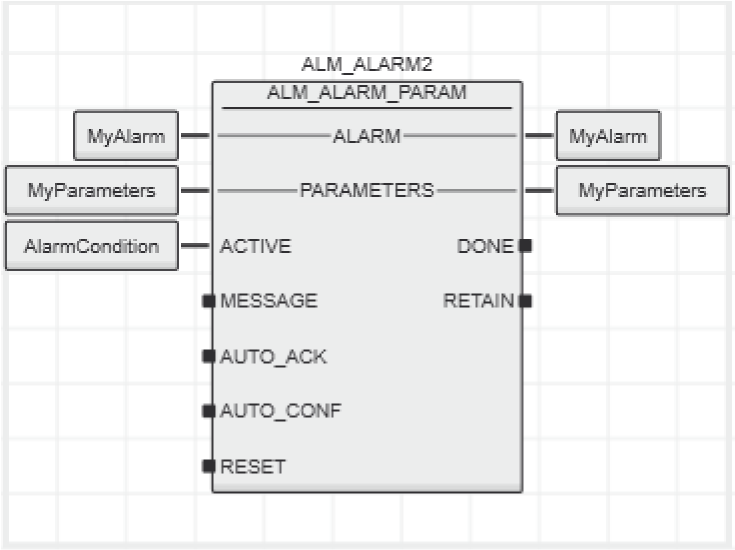
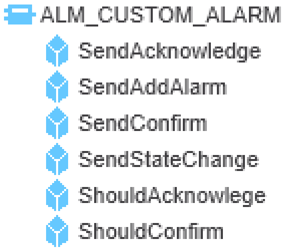


Figure 2-30 “ALM_ALARM_PARAM” function block

User-defined alarm blocks

Use the “ALM_CUSTOM_ALARM” function block to create your own alarm blocks. This might be useful if special timing or filter is required, for example. In addition, you can use the user-defined alarm block to set another semantics of “ACK”, “CONF” or “RETAIN”, or implement several alarms in a combined function block.

The “ALM_CUSTOM_ALARM” function block allows for access to methods of the subordinate alarm system. All methods are implemented as methods of the block.



ALM_CUSTOM_ALARM
- SendAcknowledge
- SendAddAlarm
- SendConfirm
- SendStateChange
- ShouldAcknowlege
- ShouldConfirm

Figure 2-31 Methods of the “ALM_CUSTOM_ALARM” function block

The following methods send the corresponding messages to the OPC-UA server:

- “SendAcknowledge”
- “SendAddAlarm”
- “SendConfirm”
- “SendStateChange”

Use the following methods to determine if the function block has received the “Acknowledge” or “Confirm” wish:

- “ShouldAcknowledge”
- “ShouldConfirm”

Acknowledging and confirming alarms

Instead of the OPC UA server, you can also use the “ALM_ACK” and “ALM_CONF” function blocks to acknowledge and confirm any alarm. The alarms are clearly identified by the “ALARM_ID”.

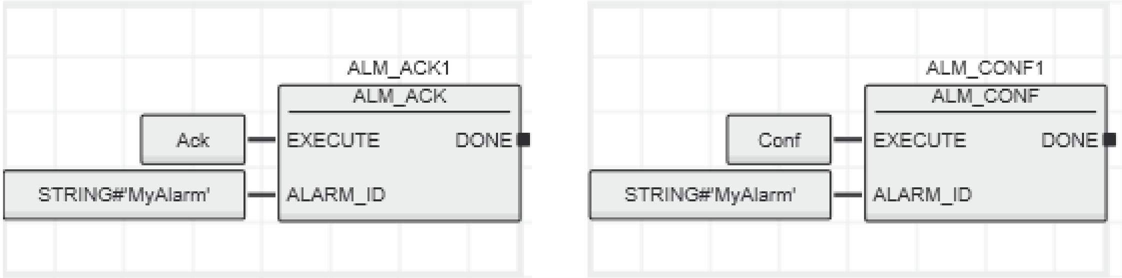


Figure 2-32 “ALM_ACK” and “ALM_CONF” function blocks

2.11.2 Alarms in C++ programs

For programming in C++, the PLCnext Technology SDK provides the “AlarmAccess” class. It can be used for alarm sources and alarm sinks.

During initialization of PLCnext Technology, the “AlarmAccess” class should be instantiated and initialized (e.g., in “SetupConfig” of a PLCnext Technology component). C++ programs can use the instance to send alarms. Using one instance for all alarms makes sense because many “AlarmAccess” instances would result in a correspondingly large number of messages in PLCnext Technology, which would strain the system.

The classes can be included using the following headers:

```
#include "Arp/System/NmPayload/Alarms/AlarmAccess.hpp"
#include "Arp/System/NmPayload/Alarms/AlarmState.hpp"
#include "Arp/System/NmPayload/Alarms/AlarmUserParameter.hpp"
```

The alarm classes are included in the Arp::System::NmPayLoad::Alarms namespace. To use a class, you have to create an instance and call the `Init` method.

```
using namespace Arp::System::NmPayload::Alarms;
AlarmAccess alarmAccess;
alarmAccess.Init("PHOENIX.CppClient");//unique name for this AlarmAccess instance
```

The following methods are used for sending messages:

```
/// Announce a new alarm to the server (e.g. for browsing in OPC UA).
/// the following fields are used:
/// alarmId is a unique name of the alarm instance on the server
/// alarmType is a name of the subtype of DiscreteAlarmType that is generated as a result of
/// this method call
/// severity is a number between 1 and 1000 where 1 is informative and 1000 is a fatal error
/// the user parameters with name and type are mapped to variables of the new AlarmType
/// returns 0 if successful, -1 if not initialized, -2 if empty string parameters
int32 AddAlarm(AlarmState& alarmState);

/// Inform the AlarmServer about a new alarm state
/// (coming alarms, going alarms / changes to sub states).
/// An alarm source can inform about state changes of its alarms.
/// Only the source should call this methods for its alarms.
/// returns 0 if successful, -1 if not initialized, -2 if empty alarmId
int32 NewAlarmState(AlarmState& alarmState);
```

```
/// A client can ask for acknowledgement of an alarm instance (if supported by an alarm).
/// returns 0 if successful, -1 if not initialized, -2 if empty alarmId
int32 Acknowledge(const String& alarmId);

/// A client can ask for confirmation of an alarm instance (if supported by an alarm).
/// returns 0 if successful, -1 if not initialized, -2 if empty alarmId
int32 Confirm(const String& alarmId);
```

You can use these methods to receive messages:

```
/// An alarm source the supports acknowledgable alarms should subscribe to
/// acknowledge request by a client.
/// As a result to a call of the handler function the alarm source might change
/// the state of the alarm and call NewAlarmState again.
/// Note: The provided handler function is called by a different thread!
/// returns 0 if successful, -1 if not initialized, -2 no handler function
int32 SubscribeAcknowledge(std::function<void(const String& alarmId, const String& comment, const
String& language, const String& user)> handler);

/// An alarm source the supports confirmable alarms should subscribe to confirm request by a client.
/// As a result to a call of the handler function the alarm source might change
/// the state of the alarm and call NewAlarmState agein.
/// Note: The provided handler function is called by a different thread!
/// returns 0 if successful, -1 if not initialized, -2 no handler function
int32 SubscribeConfirm(std::function<void(const String& alarmId, const String& comment, const
String& language, const String& user)> handler);

/// A client can subscribe to AddAlarm calls
/// (used e.g. by the OPC-UA Server to show the alarms in the address space).
/// Note: The provided handler function is called by a different thread!
/// returns 0 if successful, -1 if not initialized, -2 no handler function
int32 SubscribeAddAlarm(std::function<void(const AlarmState& alarmState)> handler);

/// A client can subscribe to NewAlarmState calls to track the state of alarms.
/// Note: The provided handler function is called by a different thread!
/// returns 0 if successful, -1 if not initialized, -2 no handler function
int32 SubscribeNewAlarmState(std::function<void(const AlarmState& alarmState)> handler);
```

The alarm status is held in the “AlarmState” structure:

```
// State with sub states of an alarm.
class AlarmState
{
public:
 // unique Id of the alarm instance within the PLC
 String AlarmId;
 // type of the alarm. Use as key to lookup a localized text for the alarm message.
 String AlarmType;
 // optional message (if Message is empty the AlarmType is the key retrieval of a translated message)
 String Message;
 // indication if the alarm condition is true
 bool ActiveState;
 // timestamp for the last change of ActiveState
 DateTime ActiveStateChanged;
 // acknowledged state of the alarm (the user has seen the alarm)
 bool AckedState;
 // timestamp for the last change of AckedStateChanged
 DateTime AckedStateChanged;
 // confirmed state of the alarm (the user has resolved the problem)
 bool ConfirmedState;
 // timestamp for the last change of ConfirmedStateChanged
 DateTime ConfirmedStateChanged;
 // the server wants to hide the alarm
 bool SuppressedState;
 // timestamp for the last change of SuppressedState
 DateTime SuppressedStateChanged;
 // severity between 1 and 1000 (1 = information; 1000 = fatal error)
 int16 Severity;
 // true to indicate that the alarm should be displayed to the client
 bool Retain;
 // timestamp for the last change of RetainChanged
 DateTime RetainChanged;
 // timestamp for the last change of AlarmChanged
 DateTime AlarmChanged;
 // User parameters
 std::vector<AlarmUserParameter> UserParameters;
};
```

2.11.3 OPC UA server

The OPC UA server is the typical interface for accessing alarms. Here, alarm messages are mapped to OPC UA alarms.

Registered alarms

In the OPC UA Adress Space, you will find the alarms in the “Root/Objects/Server/Alarms”. Here, all registered alarms are displayed with their alarm ID. Below, you will see the alarm status, with “ActiveState”, “AckedState” or “Retain” as OPC UA properties.

Alarm types

The alarm types are written as subtypes of the “DiscreteAlarmType”. You will find them in the OPC UA Address Space under “Root/Types/EventTypes/BaseEventType/ConditionType/AcknowledgeableConditionType/AlarmConditionType/DiscreteAlarmType”.

Subscribing to events

If you wish to be notified about alarm status changes, you can create an OPC UA subscription and subscribe to the “Root/Types/EventTypes/BaseEventType/ConditionType/AcknowledgeableConditionType/AlarmConditionType/DiscreteAlarmType” object. The subscription will then send events about all alarm status changes.

Acknowledging and confirming alarms

Alarms are acknowledged and confirmed via the standard OPC UA alarm function. Note that the status change takes place in the alarm source (e.g., in the PLC program). Only after the alarm source has implemented the request for acknowledgement/confirmation, the OPC UA server will send the new event with the updated state (“AckedState” = TRUE, or “ConfirmedState” = TRUE).

Filtering alarms

During the OPC UA subscription, you can specify which alarm properties are to be sent with the event. Here, you can also include the user parameters, so that these values are transmitted consistently with the alarm from the source to the client.

2.12 Notification manager

The Notification manager enables sending and receiving of notifications between components on a controller. Notifications are messages referring to special events. They are identified via a name and can transport user data (payload). User data is information data without control or protocol information (no process data, no real-time requirements). Both the PLCnext Technology firmware and user-defined components can send notifications.

A sender can register a notification with a notification name with the Notification manager and thus make it known. Then, the sender can send notifications. One or several recipients subscribe under one notification name and receive all notifications that are published under this or a subordinate name. Notification names are structured hierarchically. When a part of the name is indicated, all subordinate notifications can be received. Communication can be from one component to one or several recipients. As a basis, the Notification manager uses RSC services to transmit messages between the components.

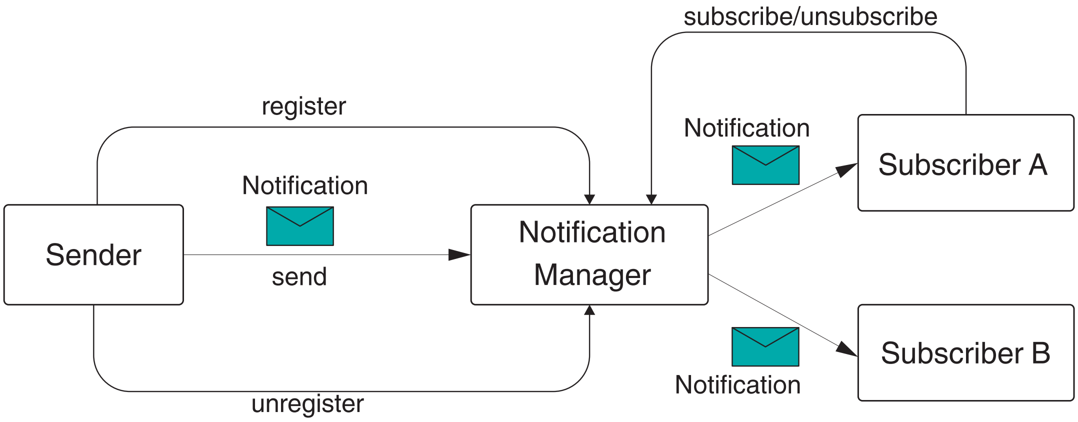


Figure 2-33 Notification manager

The Notification manager is suitable, e.g., for the following scenarios:

- Network settings have been changed
- Network interface up/down (link up/link down: connection established/interrupted)
- New component added
- Detected error, exception in one component

For additional information on using the notification manager in a C++ component, please refer to Section “Notifications” on page 147.

2.12.1 Notifications of the PLCnext Technology firmware

Table 2-6 Notifications triggered by the firmware

NotificationName	SenderName	Severity	PayloadTypeName	PayloadString
Arp.Device.Interface.EthernetLinkStateChanged	Device interface	Info	Arp::System::NmPayload::Device::EthernetLinkStatePayload	Link state changed: interface {number}, port {number}, status: {"Up"|"Done"}
Arp.Device.Interface.ExtensionModulesState	Device interface	Info	Arp::System::NmPayload::Device::PciDeviceStatePayload	Extension device status: {"OK"|"Diagnosis"|"Error"}
Arp.Device.Interface.NetworkConfigurationChanged	Device interface	Info	Arp::System::NmPayload::Device::NetworkConfigurationChangedPayload	Configuration of network interface {number} changed: {Parameter} = {Value}

Table 2-6 Notifications triggered by the firmware

NotificationName	SenderName	Severity	PayloadTypeName	PayloadString
Arp.Device.Interface.NetworkConfigurationFailed	Device interface	Error	Arp::System::NmPayload::Device::NetworkConfigurationChangedPayload	Configuration of network interface {number} failed: {Parameter} = {Value}
Arp.Device.Interface.SdCardStateChanged	Device interface	Info	Arp::System::NmPayload::Device::SdCardStateChangedPayload	sd card state changed: sdCardId {"1"}, state {"true"|"false"}
Arp.Io.PnC.ArAbort	Arp.Io.PnC	Info	Arp::System::NmPayload::Io::ProfinetStack::MessageWithStationNamePayload	"Connection to device established: {StationName}" | "Connection to device with differences established: {StationName}"
Arp.Io.PnC.ArReady	Arp.Io.PnC	Info	Arp::System::NmPayload::Io::ProfinetStack::MessageWithStationNamePayload	Connection to device aborted: {stationName}
Arp.Io.PnC.PnStationStateChanged	Arp.Io.PnC	Internal	Arp::System::NmPayload::Io::ProfinetStack::PnStationStatePayload	Led state changed: Arp.Io.PnC (Controller), BF={"On"|"Off"}, SF={"On"|"Off"}
Arp.Io.PnC.ResetToFactoryDefaults	Arp.Io.PnC	Info	Arp::System::Nm::StringPayload	This station is reset to factory defaults.
Arp.Io.PnC.SetInterfaceAddress	Arp.Io.PnC	Info	Arp::System::NmPayload::Io::ProfinetStack::InterfaceAddressPayload	Interface address changed to: IP=x.x.x.x Netmask=x.x.x.x Gateway=x.x.x.x IsVolatile={"true"|"false"}
Arp.Io.PnC.SetStationName	Arp.Io.PnC	Info	Arp::System::NmPayload::Io::ProfinetStack::MessageWithStationNamePayload	The station name is set to: {StationName}
Arp.Io.PnD.ArAbort	Arp.Io.PnD	Info	Arp::System::Nm::StringPayload	"Connection to device established: {StationName}" | "Connection to device with differences established: {StationName}"
Arp.Io.PnD.ArReady	Arp.Io.PnD	Info	Arp::System::NmPayload::Io::ProfinetStack::MessageWithStationNamePayload	Connection to device aborted: {stationName}
Arp.Io.PnD.PnStationStateChanged	Arp.Io.PnD	Internal	Arp::System::NmPayload::Io::ProfinetStack::PnStationStatePayload	Led state changed: Arp.Io.PnD (Device), BF={"On"|"Off"}, SF={"On"|"Off"}
Arp.Io.PnD.ResetToFactoryDefaults	Arp.Io.PnD	Info	Arp::System::Nm::StringPayload	This station is reset to factory defaults.
Arp.Io.PnD.SetInterfaceAddress	Arp.Io.PnD	Info	Arp::System::NmPayload::Io::ProfinetStack::InterfaceAddressPayload	Interface address changed to: IP=x.x.x.x Netmask=x.x.x.x Gateway=x.x.x.x IsVolatile={"true"|"false"}
Arp.Io.PnD.SetStationName	Arp.Io.PnD	Info	Arp::System::NmPayload::Io::ProfinetStack::MessageWithStationNamePayload	The station name is set to: {StationName}
Arp.Plc.Domain.PlcManager.StateChanged	PLC	Info	Arp::System::NmPayload::Plc::PlcStateChangedPayload	Plc state changed: {"None"|"Ready"|"Stop"|"Running"|"Halt"|"Changing","Warning"|"Error"|"SuspendedBySwitch"|"DcgNotPossible"|"DcgRealTimeViolation"} ==> {"None"|"Ready"|"Stop"|"Running"|"Halt"|"Changing","Warning"|"Error"|"SuspendedBySwitch"|"DcgNotPossible"|"DcgRealTimeViolation"}
Arp.Plc.Esm.Exception.Arp.Plc.Esm	Arp.Plc.Esm	Error	Arp::System::NmPayload::Plc::ExceptionInformationPayload	Exception Information typeId={} subTypeId={} subType={} taskName={} programName={} information={} extendedInformation={}
Arp.Service.NotificationLogger.ClosingArchive	NotificationLogger	Info	Arp::System::NmPayload::NotificationLogger::MessageWithArchiveNamePayload	Closing archive '{ArchiveName}'. This entry is made during download of the firmware. For the time being, ArchiveName is always "Default". Might not be made during voltage drops (depending on the device).
Arp.Services.Alarms.Log.AcknowledgeRequest.eUAServer	eUAServer	Internal	Arp::System::Nm::StringPayload	See alarms

Table 2-6 Notifications triggered by the firmware

NotificationName	SenderName	Severity	PayloadTypeName	PayloadString
Arp.Services.Alarms.Log.AddAlarm.eUAServer	eUAServer	Internal	Arp::System::NmPayload::Alarms::Internal::AlarmPayload	See alarms
Arp.Services.Alarms.Log.ConfirmRequest.eUAServer	eUAServer	Internal	Arp::System::Nm::StringPayload	See alarms
Arp.Services.Alarms.Log.NewState.eUAServer	eUAServer	Internal	Arp::System::NmPayload::Alarms::Internal::AlarmPayload	See alarms
Arp.System.Acf.SystemManager.Startup	SystemManager	Info	Arp::System::NmPayload::Acf::SystemStartupPayload	System starting up. Firmware version {"2019.0 LTS"}
Arp.System.Acf.SystemManager.StateChanged	SystemManager	Info	Arp::System::NmPayload::Acf::SystemManagerStatePayload	SystemManager state changed: {"None"|"Ready"|"Stop"|"Running"}, error={"true"|"false"}, warning={"true"|"false"}
Arp.System.Nm.ExceptionDuringNotify.LocalIoProcess	NotificationManager	Error	Arp::System::Nm::StringPayload	Only if controller has local I/Os (e.g., AXC F 2152) Caught an exception during dispatching notification '{notificationName}': {Exception Message incl. Call-Stack}
Arp.System.Nm.ExceptionDuringNotify.MainProcess	NotificationManager	Error	Arp::System::Nm::StringPayload	Caught an exception during dispatching notification '{notificationName}': {Exception Message incl. Call-Stack}
Arp.System.Nm.ExceptionDuringNotify.ProfinetProcess	NotificationManager	Error	Arp::System::Nm::StringPayload	Caught an exception during dispatching notification '{notificationName}': {Exception Message incl. Call-Stack}
Arp.System.Nm.SubscribeToNotRegisteredNotification	NotificationManager	Warning	Arp::System::Nm::NmSubscribeFailedPayload<Arp::System::Nm::NmSubscribeToNotRegisteredPayload>	A subscriber subscribed to not registered notification name: {notificationName}
Arp.System.Nm.SubscribeToUnregisteredNotification	NotificationManager	Error	Arp::System::Nm::NmSubscribeFailedPayload<Arp::System::Nm::NmSubscribeToUnregisteredPayload>	A subscriber subscribed to unregistered notification name: {notificationName}

2.13 Notification logger

The Notification logger enables the saving of notifications in a database. The saved notifications can be displayed and evaluated using external tools. The Notification logger registers with the Notification manager for all configured notifications and thus receives all applicable messages that are sent. The Notification logger can be used for analyzing firmware and applications.

The PLCnext Technology SDK contains helpful classes for the Notification logger. If you want to use a class, integrate it into your program via an `#include` command, (e.g., `#include Arp/Services/NotificationLogger/Services/INotificationLoggerService.hpp`). Further information on the classes and their applications is available directly in the code commentary.

2.13.1 Displaying notifications in the PLCnext Engineer cockpit

The notifications of a controller are displayed in PLCnext Engineer in the “Cockpit” editor of your controller.

- In the “PLANT” area, click on the controller (e.g., AXC F 2152).
- Select the “Cockpit” editor.
- Select “Notifications”.

For more detailed information, please refer to the online help for PLCnext Engineer.

2.13.2 Receiving notifications

An archive subscribes to all notifications to be received. Additionally, in an archive, filters can be used for the notifications subscribed to, which refer to the metadata. This way you can select which notifications are actually saved. The notification logger contains several archives that enable notifications to be saved for different issues and purposes.

2.13.3 Configuring the notification logger

If necessary, you can configure the Notification logger via a config file in XML format. To modify the configuration manually, the XML file can be edited using any editor. If you do not create your own configuration, the standard settings from the settings file will be used. Changes to the configuration are automatically applied after a restart of the controller.

You will find the configuration files in the /opt/plcnext/projects/Default/Services/NotificationLogger/*.config directory. The configuration files are imported during the start of the firmware.

A configuration file for the notification logger is structured as shown in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<NotificationLoggerConfigurationDocument
xmlns="http://www.phoenixcontact.com/schema/notificationloggerconfiguration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.phoenixcontact.com/schema/notificationloggerconfiguration"
schemaVersion="1.0">
<Includes>
    <Include path="$ARP_PROJECTS_DIR$/Default/" />
</Includes>
```

```
<Archives>
    <Archive name="Default"
        <Notifications>
            <Notification name="Arp" />
        </Notifications>
        <ExclusionFilter>
           <Severity Operator="LE" Threshold="Info" />
        </ExclusionFilter>
        <Storage path="$ARP_PROJECTS_DIR$/Default/Services/NotificationLogger/archives">
           <SizeLimitation>
                <FileSizeLimitation MaxFileSize="64MB" />
           </SizeLimitation>
           <SizeReduction>
                <DeleteOldestEntires NumberOfEntriesToDelete="16" />
           </SizeReduction>
        </Storage>
    </Archive>
</Archives>
</NotificationLoggerConfigurationDocument >
```

- Configure the Notification logger using the following attributes:

Integrating configuration files

Configuration files are integrated between tags `<Includes>` and `</Includes>`.

- Use the `Include path` attribute to reference further files to be integrated by indicating their path. The element can occur any number of times.

Defining an archive

An archive is defined between tags `<Archives>` and `</Archives>`.

- Define the archive using the following attributes:

Attribute	Description
`Archive name`	Name of the archive. The name is used as the basis for the file names.
`Notification name`	Name of the notification to be saved to the archive. This element can occur any number of times.
`ExclusionFilter`	Specification of the input filter for notifications. This can refer to any part of the namespace. All subordinate notifications are recorded. All notifications matching the filter are discarded and not saved.
– `And`	– AND-link of several filter elements. Contains any number of other filter elements.
– `Or`	– OR-link of several filter elements. Contains any number of other filter elements.
– `Not`	– Negation of a filter element. Contains exactly one other filter element.
– `NotificationName`	– `Regex` attribute: Regular expression for the notification name. Regular expressions are character strings for describing search patterns using a syntactic standard.
– `SenderName`	– `Regex` attribute: Regular expression for the sender name of the notification
– `Severity`	– `Operator` attribute: Comparison function: GT, GE, LT, LE, EQ; `Threshold` attribute: Comparison value: Internal, Info, Warning, Error, Critical, Fatal

Attribute	Description
Storage	Specification of the persistent storage of the archive
– path	– Path for saving the archive files
– SizeLimitation	– Limitation of file size
– FilesizeLimitation	– MaxFilesize attribute: Maximum storage space in bytes; positive integer required, permitted suffixes: kB (*1024), MB (*1048576)
– SizeReduction	– Action for reducing the file size. The action is executed as soon as SizeLimitation is violated.
– DeleteOldestEntries	– NumberOfEntriesToDelete attribute: Number of elements to be deleted, positive integer

2.13.4 Saving notifications

Saving the notifications

The Notification logger uses one or several archives for registering, saving and querying notifications. It provides a uniform interface to these archives and enables the configuration of the archives via the configuration files. The notifications are permanently stored in an SQLite database. This way, the file is not system-oriented and, after copying it to another system, can be opened and processed with the appropriate tools. The pre-defined database is available on the file system of your controller at /etc/plcnext/logs/default.sqlite. The archives offer functions for acquiring and filtering incoming notifications as well as for querying and deleting. Using different archives makes it possible to save notifications for different issues and purposes. For example:

- **Firmware diagnostics**
  Short-term saving of firmware events for diagnostics by the service.
- **Network diagnostics**
  Saving of diagnostic messages from network components over a short period for diagnosing disconnections or sporadic delays.
- **Application diagnostics**
  Saving notifications of the application program, e.g., malfunctions of machines, refilling of consumables, error messages by integrated aggregates.

2.13.5 Querying notifications

Saved notifications can be queried via RSC interfaces. You can use filter criteria for the query in order to specify the query. You can query all archives or the notifications of one specific archive. To do so, use the `Arp/Services/NotificationLogger/Services/INotificationLoggerService.hpp` interface. Further information on the classes and their applications is available directly in the code commentary.

Use the `QueryStoredNotifications` method to query saved notifications. You can limit the query using the following parameters:

Table 2-7 Parameters

Parameter	Data type	Description
archives	String[]	List of names of archives to be queried. If this list is empty, all archives with sufficient access rights are queried.
filter	NotificationFilter	Specification of the filter for the query

Table 2-7 Parameters

Parameter	Data type	Description
limit	int32	Maximum number of notifications to be returned
sortOrder	SortOrder	Sorting criterion of the query
language	String	Desired language of user data (de, en_GB, etc.). Currently not supported.

If using RSC, the filters are described by a structure with reference values or regular expressions. The limits are part of the specified area. The individual criteria are ANDed.

Table 2-8 INotificationLoggerService - Struct: NotificationFilter

Field	Data type	Description
StoredIdLowerLimit	uint64	Lower limit of the StoredId (>=1), is ignored if = 0 In the notification logger, a notification is clearly identified by a StoredId (uint64). The StoredId is assigned by the notification logger when adding the notification to the input buffer.
StoredIdUpperLimit	uint64	Upper limit of the StoredId (>=1, <=18446744073709551615, max. uint64), is ignored if = 0
NotificationNameRegExp	String	Regular expression for the notification name. Is ignored if field is empty.
SenderNameRegExp	String	Regular expression for the sender name. Is ignored if field is empty.
TimestampBefore	String	Oldest applicable notification. Is ignored if field is empty. The format complies with ISO 8601 plus indication of microseconds. YYYY-MM-DDThh:mm:ss.SSSSSS
TimestampAfter	String	Latest applicable notification. Is ignored if field is empty. The format complies with ISO 8601 plus indication of microseconds. YYYY-MM-DDThh:mm:ss.SSSSSS
SeverityLowerLimit	String	Lowest applicable severity. Is ignored if field is empty.
SeverityUpperLimit	String	Highest applicable severity. Is ignored if field is empty.

The values are returned via the StoredNotification struct object. The object includes the following fields:

Field	Data type	Description
Id	uint64	ID of saved notifications
Archives	String	Name of the archive the notification was loaded from. If the notification is present in several archives, this field contains a list of archives separated by commas.
NotificationName	String	Name of the notification
SenderName	String	Name of the sender
TimeStamp	String	Time stamp of when the notification was sent. The format complies with ISO 8601 plus additional indication of microseconds. YYYY-MM-DDThh:mm:ss.SSSSSS
Severity	String	Severity of the notification
Payload	String[]	Translated and formatted representation of user data
PayloadXml	String[]	User data as XML

2.13.6 Permissions

Table 2-9 shows which user role is authorized to what extent to call the methods of the INotificationLoggerService. User roles not contained in this table do not have access permission. For additional information on user roles, please refer to Section 3.9.1 on page 82.

x = Authorization available

- = No autorization

Table 2-9 Authorizations for calling of methods of the INotificationLoggerService

Role	QueryStoredNotifications QueryNotifications	DeleteNotifications	ListArchives	Other
Admin	x	x	x	x
Engineer	x	x	x	-
Commissioner	x	-	x	-
Service	x	-	x	-
DataViewer	x	-	x	-
DataChanger	x	-	x	-
Viewer	x	-	x	-

2.14 Operating system

The PLCnext Technology control platform is based on a Linux operating system with real-time patch. Linux is a highly reliable open source operating system suitable for applications that require a high stability. A wide range of open source software is available for the Linux operating system, which is supported by a large community of users and developers. You can also use this open software, software blocks, and technologies for your PLC application (e.g., SQL server). PLCnext Technology uses the Linux operating system and extends it by the functions of a PLC such as the cyclic processing of tasks and cycle-consistent data exchange. Core changes or extensions are not possible. To add functions to the system, the user must compile and, if necessary, execute installations with “root” rights.

The operating system features the following components and services:

- Firewall (nftables is used for the firewall. The firewall can be configured via WBM, see Section 3.9.3 on page 95.)
- OpenVPN
- strongSwan
- SSH/SFTP
- NTP
- DNS

2.14.1 Directories of the firmware components in the file system

PLCnext Controllers work with a Linux operating system. You can access the controller via SFTP or via SSH, view the directories and files in the file system (on the internal parameterization memory and on the SD card), and modify them if necessary.

Directories and files that are provided by Phoenix Contact (also through firmware updates) are stored on the internal parameterization memory of the controller.

If you make changes to the directories or files, the Linux operating system generates an overlay filesystem. The directory structure depends on whether you operate the controller with or without an SD card:

Operation without an SD card

If you make changes to the directories or files on the internal parameterization memory, the Linux operating system generates an overlay file system here.

Some controllers have to be operated with an SD card.

- Check this in the user manual of our controller.

Operation with an SD card

If you operate the controller with an SD card, the overlay file system is generated on the SD card.
Settings that you have configured yourself (e.g., network configuration, project bus configuration, PLCnext Engineer project, etc.) are also saved to the SD card.

Table 2-10 Directory structure on the internal parameterization memory and the SD card

Directory in the root file system	Content
/usr/local/lib	Directory for storing additional open source libraries that are used by customized C++ programs. For additional information on programming the controller with C++, please refer to Section 6, “Creating programs with C++”.
/usr/share/common-licenses	License information on the individual Linux packages of the controller
/opt/plcnext	Home directory of the “admin” Linux user and working directory of the device firmware Files written by the application program are stored in this directory if the specified file name does not contain a memory path.
/opt/plcnext/logs	Directory for storing the log files of the diagnostic logger as well as the database of the notification logger Here, you will find the Output.log file. It contains information on the startup behavior of the firmware, status and error messages as well as warning notes that help you find the source of error.
/opt/plcnext/projects	Directory for storing project folders and files
/opt/plcnext/projects/PCWE	Directory for storing PLCnext Engineer projects All files and subdirectories in this directory are managed exclusively by PLCnext Engineer. • Do not make any changes to this directory.
/opt/plcnext/Security	Directory for storing certificates (IdentityStores and TrustStores)
/opt/plcnext/Security/Certificates/https	Directory for storing HTTPS certificates

Table 2-10 Directory structure on the internal parameterization memory and the SD card

Directory in the root file system	Content
/opt/plcnext/apps	All active apps are mounted to this directory. The directory is part of the PLCnext Store.
/opt/plcnext/installed_apps	Directory for storing all installed app containers. The directory is part of the PLCnext Store.
/opt/plcnext/lttng	Directory for storing the default configuration files for tracing via LTTng
/opt/plcnext/backup	Directory for download change processes The directory is used to create a backup of the project folder. In case of an error, the content of the backup folder is restored.
/opt/plcnext/retaining	Directory for storing remanent data
/opt/plcnext/shadowing	Directory for the internal storage of copies of C++ user libraries that were configured and downloaded in PLCnext Engineer.
/opt/plcnext/profinet	Directory for storing temporary PROFINET files

Using SFTP to access the file system

The file system is accessed via the SFTP protocol. SFTP client software is required for this (e.g., WinSCP).

Access to the file system via SFTP requires authentication with a user name and password. The following access data is set by default with administrator rights:

User name: admin
Password: printed on the controller

2.14.2 System time

2.14.2.1 NTP (Network Time Protocol)

Server

An ntpd (Network Time Protocol daemon) is included in the operating system for time synchronization. It is possible to connect to other NTP servers or to start your own server.

Client

The NTP service from ntp.org (Network Time Protocol Project) is integrated into the operating system. This service can be configured via the /etc/ntp.conf configuration file. As an admin user, you have sufficient rights to modify the data. The changes are adopted after restarting the ntp daemon.

- Execute the `sudo /etc/init.d/ntpd` script.

The changes made in the configuration file will be active after the next controller restart.

In the standard configuration, the operating system time is synchronized with the Real-Time Clock (RTC) installed in the hardware.
You have the option of specifying IP addresses and names in the configuration.

Additional information

– You will find a general introduction to the Network Time Protocol daemon (ntpd) at https://www.eecis.udel.edu/~mills/ntp/html/ntpd.html.

– You will find a detailed description of the configuration options at https://www.eecis.udel.edu/~mills/ntp/html/confopt.html.

2.14.2.2 Changing the system time via the shell

As an alternative to synchronization with an NTP server, you can also change the system time manually via the shell. Authentication with the user name and password is necessary for SSH access to the shell. The following access data is set by default with administrator rights:

User name: admin
Password: printed on the controller.

Requesting the system time

- Open the shell.
- Request the system time via the `date` command.

Setting the system time

- Enter shell command `sudo date -s "YYYY-MM-DD hh:mm:ss"`.
  - `YYYY`: Year
  - `MM`: Month
  - `TT`: Day
  - `hh`: Hours
  - `mm`: Minutes
  - `ss`: Seconds

2.14.2.3 Setting the system time in PLCnext Engineer

You can also set the system time using the PLCnext Engineer software. Access via PLCnext Engineer requires authentication with the user name and password.

- In the “PLANT” area, click on “PLCnext”.
- Select the “Online Parameters” editor.
- Enter the desired values for the date and time in the corresponding input fields.

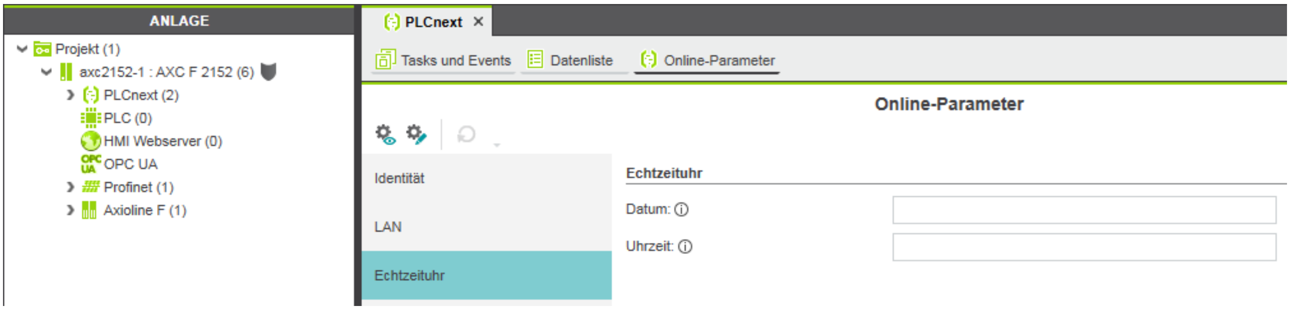


Figure 2-34 Setting the system time in PLCnext Engineer

2.14.3 OpenVPN™ client

With the OpenVPN™ software, you have the option of establishing a virtual private network (VPN) and therefore a secure connection via an unsecure network. The data is encrypted using suitable protocols.

A description on how to set up a VPN tunnel using openVPN™ is available in the PLCnext Community at plcnext-community.net.

2.14.4 IPsec (strongSwan)

IPsec is an encryption and authentication protocol with which VPN connections (Virtual Private Networks) can be established. StrongSwan is an implementation of the IKE (Internet Key Exchange) protocol and can be used for VPN connections via IPsec.

For details, please refer to http://www.strongswan.org.

AXC F 2152 configuration notes

You can edit the /etc/ipsec.conf configuration file with admin user rights.

- Use the following commands to start, stop and restart the daemon:
  – Start: `sudo ipsec start`
  – Stop: `sudo ipsec stop`
  – Restart: `sudo ipsec restart`
- Use the following command to call the status: `sudo ipsec status`

Configuration examples

Configuration example are available at https://wiki.strongswan.org.

2.14.5 Text editors

“Nano” and “Vim” are installed on the controller as text editors. When you are connected to the controller via the SSH console, you can call the desired editor via the command line.

- To open a file with the desired editor, enter `nano <file name>` or `vim <file name>`.

Nano

The “Nano” text editor is easy to use and is therefore recommended for inexperienced users.

Vim

The “Vim” text editor has an extended range of functions and is a popular editor in the Linux environment.

2.14.6 User rights

When a PLCnext user logs into the SSH console with the “admin” user role, the user is also recognized with the same name and password by the Linux system. The user is therefore assigned to the “plcnext” Linux group. Files that this user may read, write and/or execute are assigned to the “plcnext” group in the file system. In addition to the Linux user rights, the PLCnext Technology firmware also has its own user management of which the configuration is described in Section ““User Authentication” page” on page 82.

Executing Linux commands that require higher rights is made possible for the users via `sudo`. Which Linux commands the PLCnext users are allowed to execute via `sudo` is configured in the Linux system.

The following rights are available:

Table 2-11 User rights

Rights	“plcnext” group	admin	sudo required
Setting and inspecting IP settings (including ifconfig, ping, netstat, etc.)	x	x	x (ifconfig)
Configuring the firewall	x	x	
Starting/stopping the firewall (init script)	x	x	x
Inspecting the firewall with nft	x	x	x

Table 2-11 User rights

Rights	“plcnext” group	admin	sudo required
Configuring VPN (IPsec and OpenVPN™)		x	
Starting/stopping VPN services (IPsec and OpenVPN™)		x	x
Editing the “Default” PLCnext folder for individual ACF, ESM, GDS configurations, and *.so	x	x	
Starting/stopping the PLCnext Technology firmware processes `sudo /etc/init.d/plcnext start|stop|restart`		x	x
Reading PLCnext log files	x	x	
Calling and configuring TOP/HTOP	x	x	
Firmware update via RAUC		x	
Firmware update via update script `sudo update-plcnext`		x	x
Configuring the NTP server	x	x	
Setting the root password with `passwd`		x	
Requesting the system time with `date`	x	x	
Setting the system time with `sudo date -s`	x	x	x
Restarting/shutting down the controller with `reboot` or `shutdown`		x	x
Write access to /opt/plcnext and /opt/plcnext/projects	x	x	
Recording network traces with `tcpdump`	x	x	x
Starting the gdbserver with root rights (see also PLCnext Community)		x	x
Resetting to factory defaults `sudo recover-plcnext 1` See also “Factory reset” on page 72		x	x

2.14.7 Root rights

PLCnext Controllers are supplied with a preset admin user. This enables access to the most important functions. For some settings, you require advanced rights, e.g., to configure settings for remote debugging of C# code. To this end, create a user with root rights.

NOTE: Risk of damage to equipment

If safety functions are switched off, the controller must not be used for live operation.

- Ensure that there is no risk of damage to the equipment or personal injury.

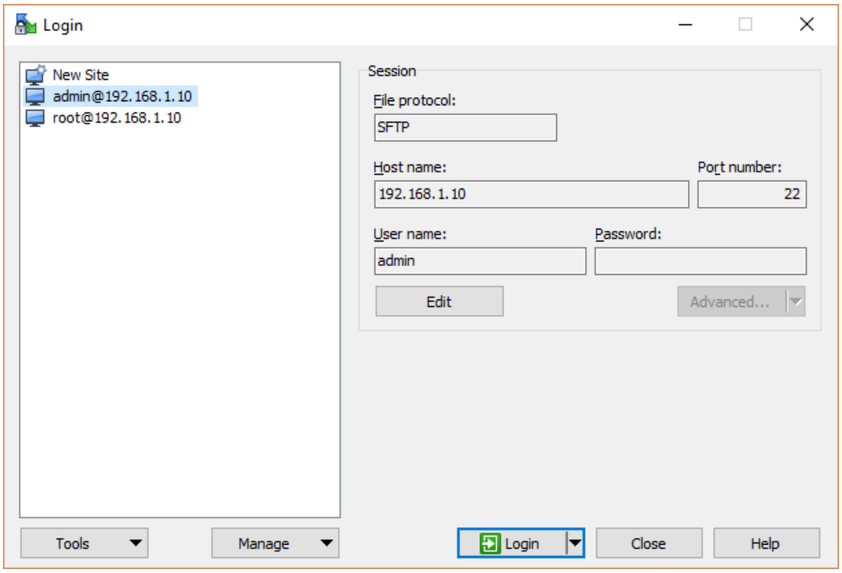
NOTE: Changes to the firmware

As a “root” user, you have the right to make any change on the controller. Root rights are therefore only suitable for qualified application programmers and software engineers with relevant experience.

- Make sure you do not make changes to the PLCnext Technology firmware.

Creating a root user

- Configure the connection settings in WinSCP.
- Connect to the controller by entering the IP address of the controller and the password for the admin user.
- Click “Login”.



New Site
admin@192.168.1.10
root@192.168.1.10

Session

File protocol: SFTP

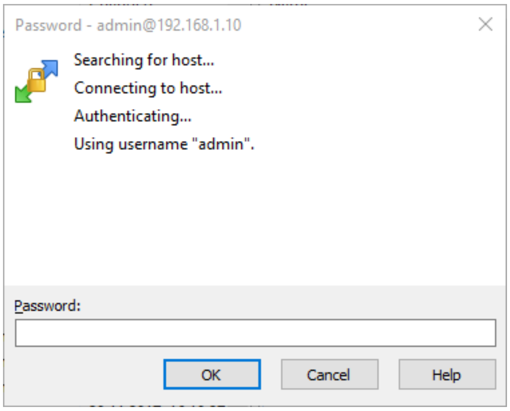
Host name: 192.168.1.10

Port number: 22

User name: admin

Password:

Figure 2-35 Connecting WinSCP to the controller (1)



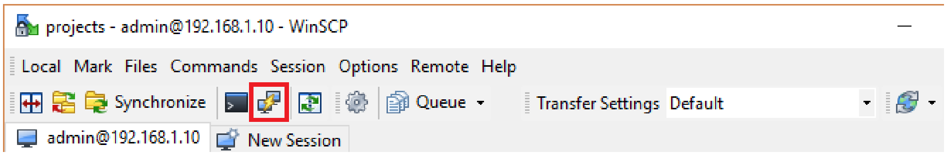
Password - admin@192.168.1.10

Searching for host...
Connecting to host...
Authenticating...
Using username "admin".

Password:

Figure 2-36 Connecting WinSCP to the controller (2)

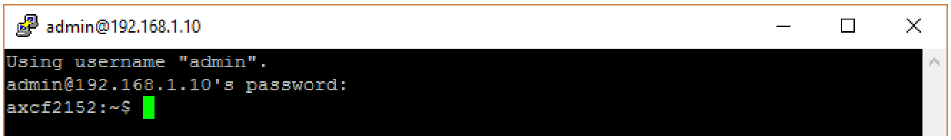
- Subsequently, open the console by clicking the “Open in PuTTY” button.



projects - admin@192.168.1.10 - WinSCP
Local Mark Files Commands Session Options Remote Help
Synchronize
Queue
Transfer Settings Default
admin@192.168.1.10
New Session

Figure 2-37 Opening PuTTY

- Log in with the “admin” user name and the controller password. The default password of the controller is printed on the housing.

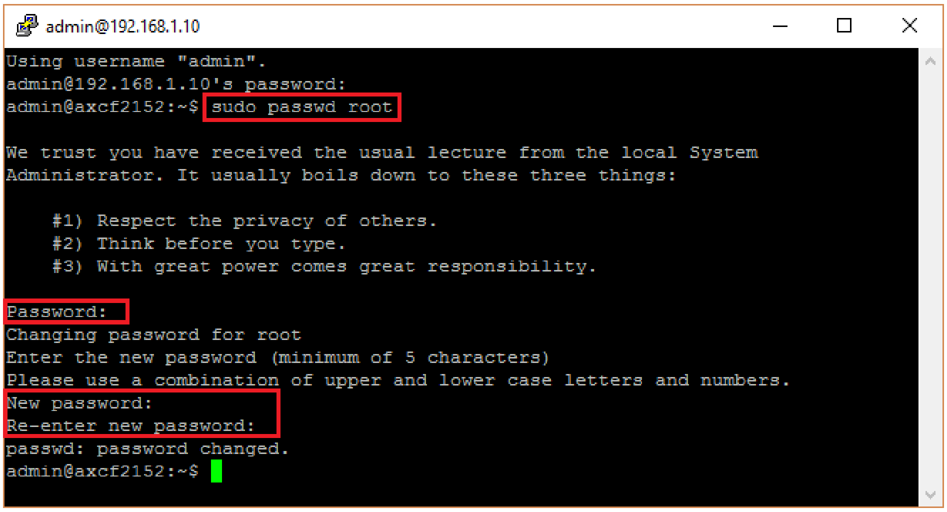


admin@192.168.1.10

```
Using username "admin".
admin@192.168.1.10's password:
axcf2152:~$
```

Figure 2-38 Logging in as admin user

- Enter the `sudo passwd root` command.
- Enter the admin password.
- Enter a new password for the root user.
- Confirm the new password by entering it again.



```
Using username "admin".
admin@192.168.1.10's password:
admin@axcf2152:~$ sudo passwd root

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

Password:
Changing password for root
Enter the new password (minimum of 5 characters)
Please use a combination of upper and lower case letters and numbers.
New password:
Re-enter new password:
passwd: password changed.
admin@axcf2152:~$
```

Figure 2-39 Creating a new root user

- To end the connection, enter the `exit` command.

SSH login as root

The SSH login as root user is prevented for security reasons. To enable direct login via SSH for the root user, you have to configure this option in the sshd_config file.

- Open the /etc/ssh/sshd_config file using a suitable editor.
- Under `# Authentication:`, enable the commented out `PermitRootLogin yes` entry.
- Restart the SSH service with `/etc/init.d/sshd restart`.

2.14.8 Linux scripts of the PLCnext Technology firmware

Firmware update

As of firmware version 2019.0 LTS, you can update the firmware conveniently via web-based management of your controller. For older firmware versions, you can start the firmware update via the `sudo update-plcnext` shell script, which you will find in the file system of the controller.

- Download the *.zip firmware file from the download area of your controller.
- Unpack the *.zip firmware file.
- Run the *.exe setup file.
- Follow the instructions of the installation wizard.
- Open your SFTP client software (e.g., WinSCP).
- Log in as an administrator.

The following access data is set by default:

User name: admin
Password: printed on the controller

- Copy the *.raucb update file to the /opt/plcnext directory (home directory of the “admin” Linux user).
- Open the shell using a command line tool (e.g., PuTTY or Tera Term).
- Log in as an administrator.

The name of the update script is the same for every controller: `sudo update-plcnext`. The script is available in the file directory under `/usr/sbin/`. Under `/usr/sbin/`, you will also find symbolic links with the respective product designation in the name, e.g., `sudo update-axcf2152`.

The script executes the following operations:

1. Stopping the PLCnext Technology process.
2. Performing the firmware update.
3. Rebooting the system and deleting the firmware container.

Factory reset

You can reset the controller to the default settings using a shell script. Here, a distinction is made between two types of default settings:

- Type 1: All user-specific data is deleted (settings, programs, users, etc.). The current PLCnext Technology firmware remains unchanged.
- Type 2: In addition to the user-specific data (type 1), the firmware of the controller is reset to default state.

The script is available in the controller file system. The name of the factory reset script is the same for every controller: `sudo recover-plcnext`. You will find the script under `/usr/sbin/`. Under `/usr/sbin/`, you will also find symbolic links with the respective product designation in the name, e.g., `recover-axcf2152 1` for type 1 default settings.

> You can also reset your controller to type 1 and type 2 default settings via the device-specific operating elements (e.g., reset button or operating display). For additional information, refer to the corresponding user manual.
>
> The type 1 default settings can be restored via the “Cockpit” editor in PLCnext Engineer.

3 Web-based management (WBM)

Each PLCnext Technology controller features a web-based management (WBM). In the WBM, you can access static and dynamic controller information and modify certain controller settings. You can call up WBM via every Ethernet interface of the controller.

The WBM systems of controllers with PLCnext Technology all have the same structure and are described generally in the following.

3.1 Establishing a connection to WBM

To establish a connection to WBM, proceed as follows:

- Open the web browser on your PC.
- In the address field, enter the URL “https://IP address of the controller” (example: “https://192.168.1.10”).

Please note:
WBM can only be called if the controller has a valid IP address. Upon delivery, the controller has IP address 192.168.1.10 (Ethernet interface LAN1 in case of several Ethernet interfaces).

If there is a PLCnext Engineer HMI application on the controller, entering URL “https://IP address of the controller” calls the PLCnext Engineer application.

- To call WBM, enter URL “https://IP address of the controller/wbm” in this case.

Initial access: TLS certificate

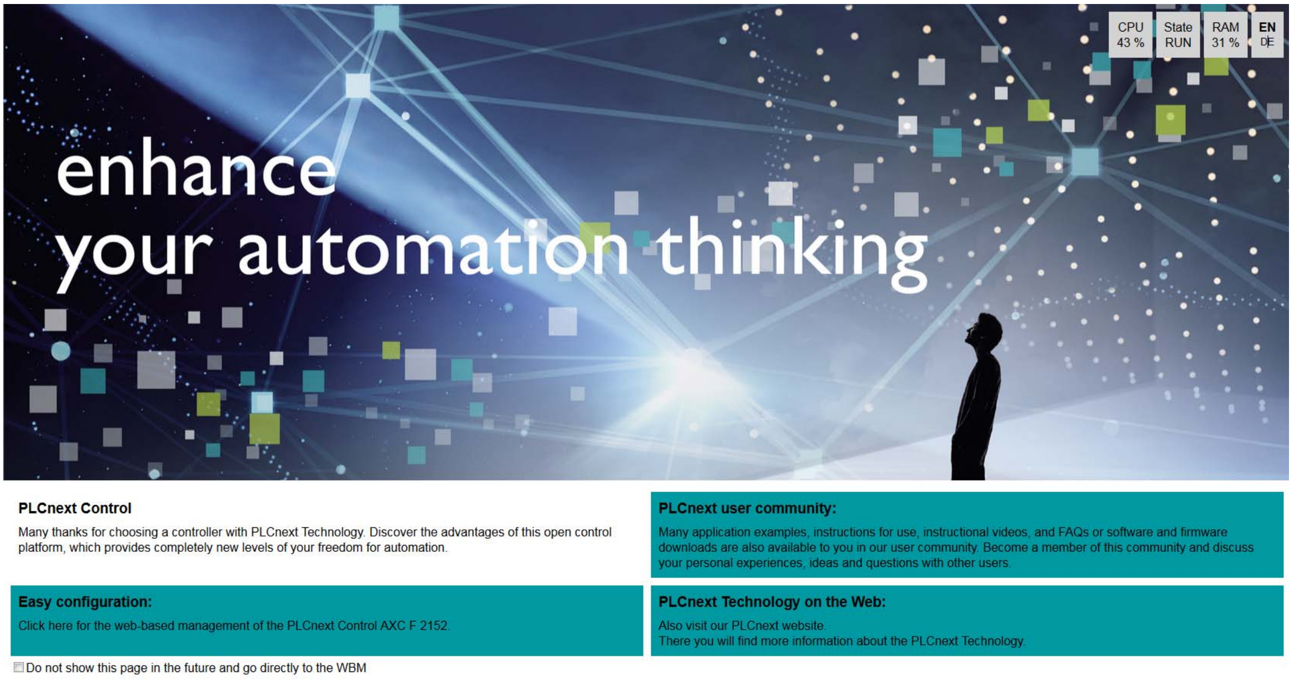
For secure communication, the web server of the controller uses a self-signed TLS certificate automatically generated by the controller. Before the web server of the controller can be accessed, you must authorize the TLS certificate in your web browser.

Please note:

- The controller generates the TLS certificate during the boot phase.
- The certificate uses the IP address of the Ethernet interface with PROFINET controller function.
- The certificate is used for all Ethernet interfaces of the controller.
- Each IP address of the controller must be authorized in the web browser before a PLCnext Engineer HMI application can be accessed via this address and therefore via the corresponding Ethernet interface.
- If the controller is reset to the default settings, the certificate is generated again.
- The certificate and a corresponding private key are available in the following directories of the controller file system:
  - /opt/plcnext/Security/Certificates/https/https_cert.pem
  - /opt/plcnext/Security/Certificates/https/https_key.pem

Initial access: Welcome page

The PLCnext Technology controller welcome page is shown when the controller web server is accessed for the first time.



PLCnext Control

Many thanks for choosing a controller with PLCnext Technology. Discover the advantages of this open control platform, which provides completely new levels of your freedom for automation.

PLCnext user community:

Many application examples, instructions for use, instructional videos, and FAQs or software and firmware downloads are also available to you in our user community. Become a member of this community and discuss your personal experiences, ideas and questions with other users.

Easy configuration:

Click here for the web-based management of the PLCnext Control AXC F 2152.

PLCnext Technology on the Web:

Also visit our PLCnext website.
There you will find more information about the PLCnext Technology.

□ Do not show this page in the future and go directly to the WBM

Figure 3-1 WBM: Welcome page of the PLCnext Technology controller

The welcome page contains links to the following web content:

- WBM of the controller
- PLCnext Community
- PLCnext website

If you do not want the welcome page to be displayed each time the controller web server is accessed:

- Enable the “Do not show this page in the future and go directly to the WBM” check box.

The next time you access the controller web server, the login page of the WBM opens, see Section 3.4.

Alternatively, you can enter URL “https://IP address of the controller/wbm” (example: “https://192.168.1.10/wbm”) in your browser address field.
In this case, WBM is displayed immediately.

- The welcome page remains accessible via URL “https://IP address of the controller/welcome”.

3.2 Licensing information on open source software

PLCnext Technology controllers work with a Linux operating system.
All license information can be called via the “Legal Information” link on every page of WBM:

- Click on the “Legal Information” link on the bottom left of the WBM page.

Licenses for all of the open source software used are shown.

3.3 Changing the language

You can change the language for the WBM user interface in the top left of the web browser window.

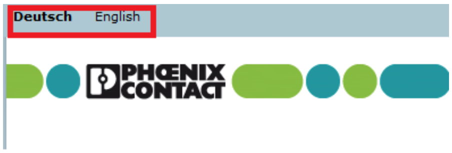


Figure 3-2 WBM: Changing the language

- Click on the “Deutsch” or “English” link to change the language.

WBM then immediately switches to the desired language.

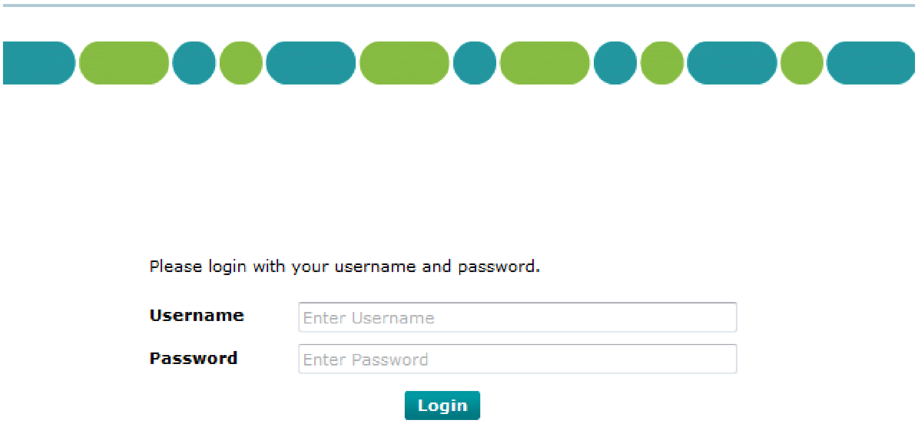
3.4 Login

The WBM login page opens when

– You access WBM for the first time.

– You have enabled the WBM User Authentication function, see Section 3.9.1.

If you disable user authentication, logging in is not necessary to access WBM. In this case, the start page of WBM opens when WBM is accessed, see Section 3.5.



Please login with your username and password.

Username Enter Username

Password Enter Password

Figure 3-3 WBM: Login page

Initial access as an administrator

When you access WBM for the first time, log in as the administrator.

- Enter the user name “admin” in the “Username” input field.
- Enter the administrator password in the “Password” input field.
  The administrator password is printed on the controller. For more detailed information, please refer to the user manual for your controller.
- To open WBM, click on the “Login” button.

The WBM start page opens (see Section 3.5).

Recommended:

- Only use the administrator password for the initial login.
- Once you have logged in successfully, change the administrator password to prevent unauthorized administrator access (see Section 3.9.1).

Please note:

After changing the access data for the administrator, it is no longer possible to log in with the user name “admin” and the administrator password printed on the controller.

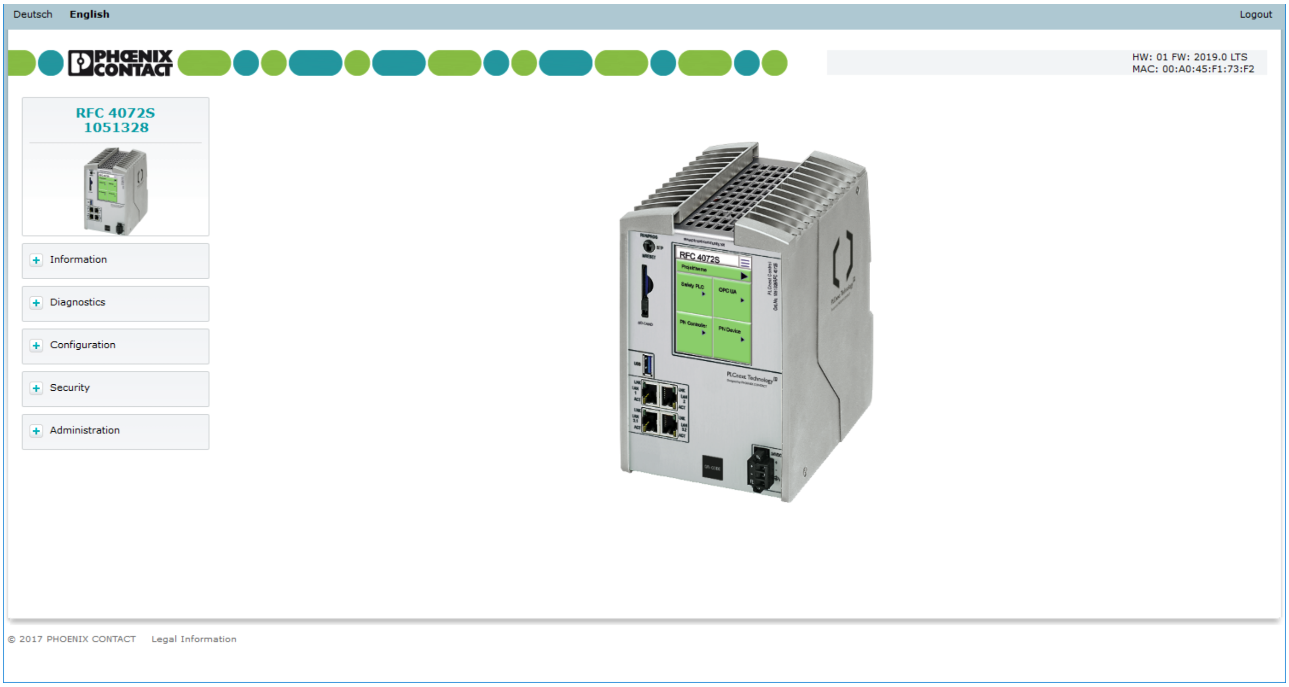
Logging in as a user

If WBM user authentication is enabled, log in using your user details.

- Enter your user name in the “Username” input field.
- Enter your password in the “Password” input field.
- To open WBM, click on the “Login” button.

The WBM start page opens (see Section 3.5).

3.5 Start page – areas and functions



HW: 01 FW: 2019.0 LTS
MAC: 00:A0:45:F1:73:F2

© 2017 PHOENIX CONTACT Legal Information

Figure 3-4 WBM: Start page

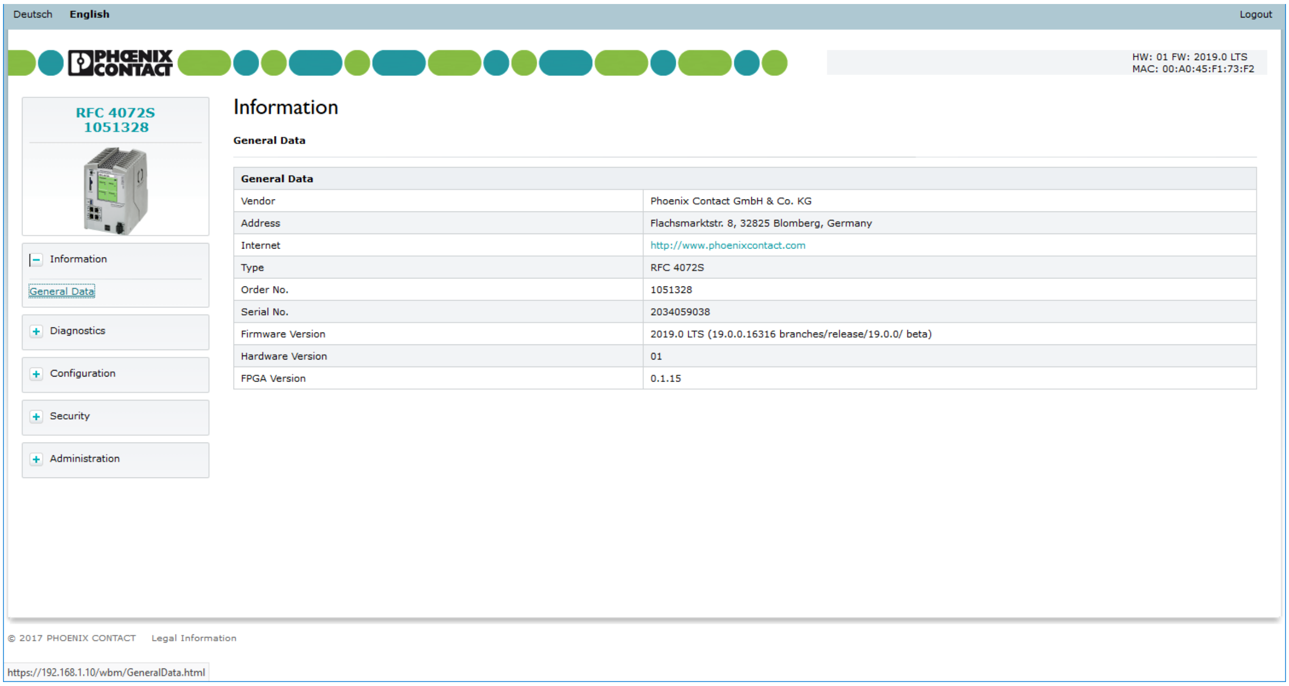
The WBM is split into the following areas:

- Information: General device information
- Diagnostics: PROFINET
- Configuration: PROFICLOUD
- Security: User authentication, certificate authentication and firewall
- Management: Firmware update of the non-safety-related device firmware

3.6 “Information” area

3.6.1 “General Data” page

On the “General Data” page, you will find general details on the device, e.g., hardware and firmware versions, the order number, as well as manufacturer details.



Information

General Data

General Data	
Vendor	Phoenix Contact GmbH & Co. KG
Address	Flachsmarktstr. 8, 32825 Blomberg, Germany
Internet	http://www.phoenixcontact.com
Type	RFC 4072S
Order No.	1051328
Serial No.	2034059038
Firmware Version	2019.0 LTS (19.0.0.16316 branches/release/19.0.0/ beta)
Hardware Version	01
FPGA Version	0.1.15

Figure 3-5 WBM: “General Data” page

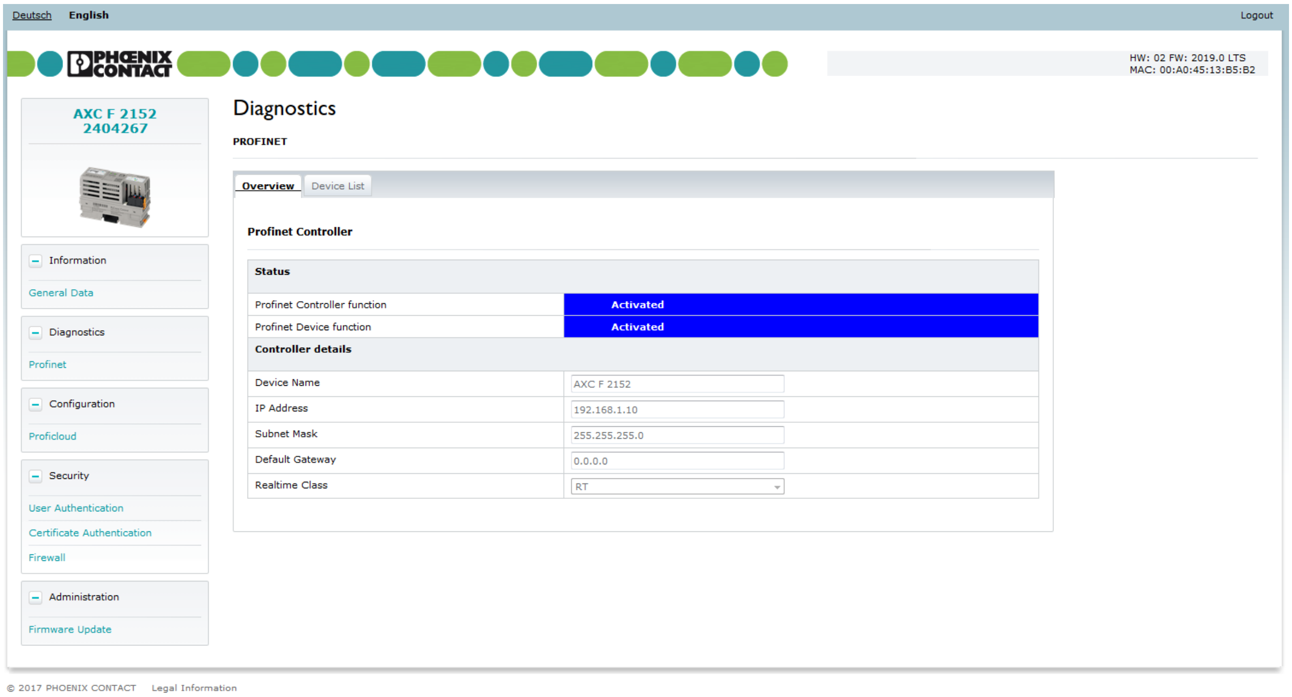
3.7 “Diagnostics” area

3.7.1 “PROFINET” page

On the “PROFINET” page, you can view information on the controller and the connected PROFINET devices.

“Overview” tab

On the “Overview” tab, you will find information on the current PROFINET function of the controller and its IP settings.



Profinet Controller

Status	
Profinet Controller function	Activated
Profinet Device function	Activated

Controller details	
Device Name	AXC F 2152
IP Address	192.168.1.10
Subnet Mask	255.255.255.0
Default Gateway	0.0.0.0
Realtime Class	RT

Figure 3-6 WBM: “PROFINET” page, “Overview” tab

“Status” area

In the “Status” area, you can see if the controller is currently used as PROFINET controller and/or as PROFINET device.

“Controller details” area

In the “Controller details” area, the current IP settings of the controller are displayed.

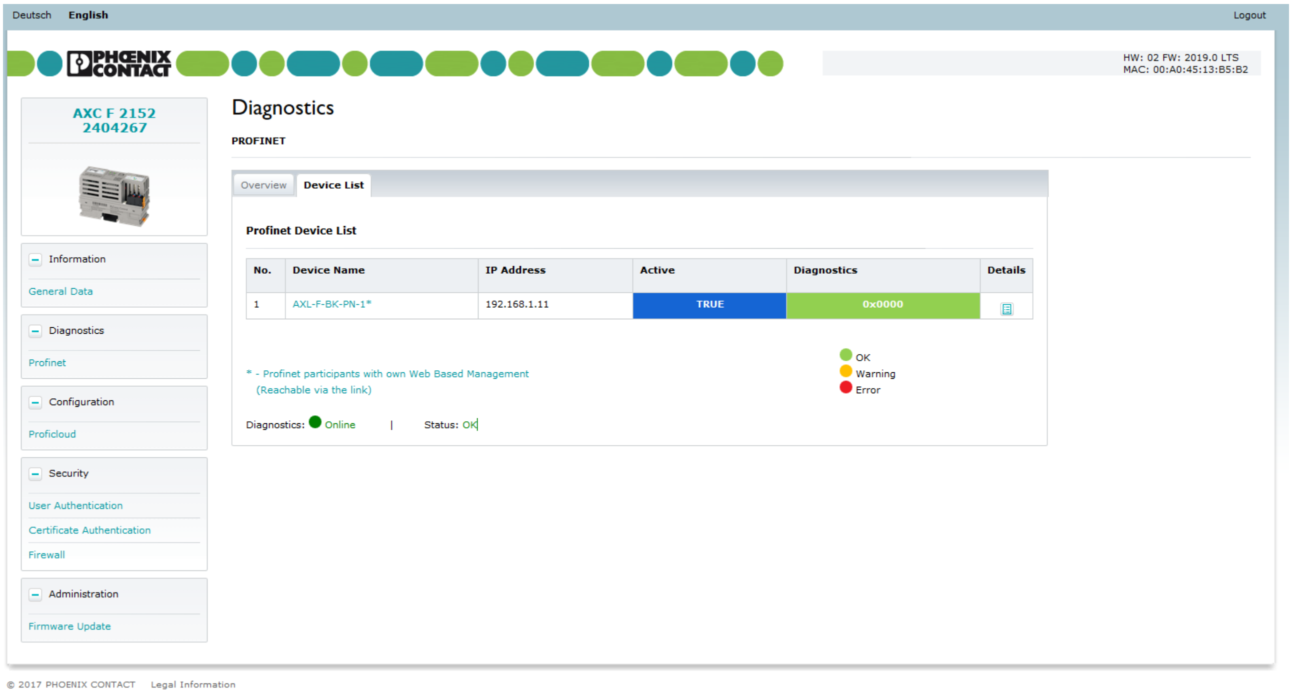
Diagnostic information on the connected PROFINET devices are available on the “Device List” tab.

“Device list” tab

The “Device List” tab provides an overview of the configured PROFINET devices. The overview contains the device names of the PROFINET devices, the current IP settings, the activation status (TRUE = active, FALSE = inactive) as well as the diagnostic state and code.

Table 3-1 PROFINET diagnostics: possible diagnostic states

Diagnostic state	Description
OK	No error
Warning	A warning occurred, e.g., “Maintenance Required”, “Maintenance Demanded”, and “Diagnosis available”
Error	A PROFINET bus error occurred.



Profinet Device List

No.	Device Name	IP Address	Active	Diagnostics	Details
1	AXL-F-BK-PN-1*	192.168.1.11	TRUE	0x0000	▤

* - Profinet participants with own Web Based Management (Reachable via the link)

● OK
● Warning
● Error

Diagnostics: ● Online | Status: OK

Figure 3-7 WBM: “PROFINET” page, “Device List” tab

Opening WBM of a PROFINET device

Some PROFINET devices feature their own web-based management (WBM). You can open WBM of a connected PROFINET device via a link to the “Device List” tab.

- In the column “Device Name”, click on the PROFINET device whose WBM you want to open.

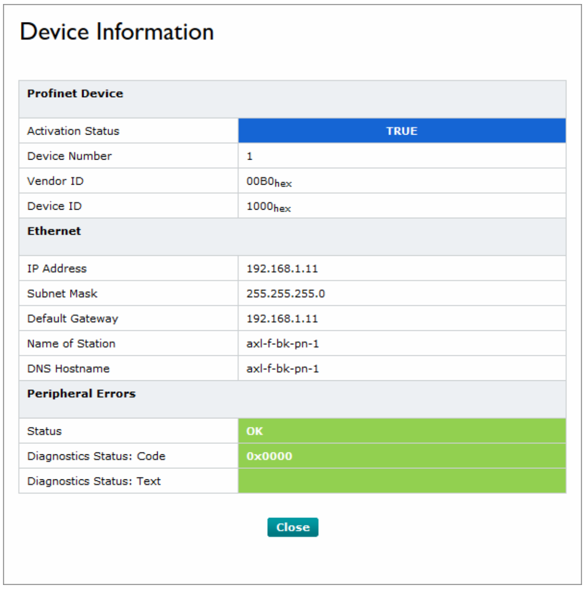
The WBM of the PROFINET device is opened in the web browser in a new tab.

Opening the “Device Information” view

In the “Device Information” view, you will find certain general information, the current IP settings as well as diagnostic information for each connected PROFINET device.

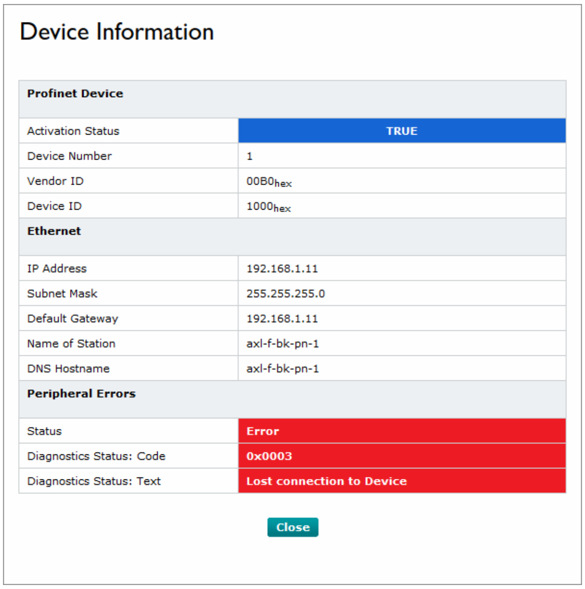
- In the “Details” column, click on the ▤ button.

The “Device Information” view opens (see Figure 3-8).



Device Information

Profinet Device	
Activation Status	TRUE
Device Number	1
Vendor ID	00B0$_{hex}$
Device ID	1000$_{hex}$
Ethernet	
IP Address	192.168.1.11
Subnet Mask	255.255.255.0
Default Gateway	192.168.1.11
Name of Station	axl-f-bk-pn-1
DNS Hostname	axl-f-bk-pn-1
Peripheral Errors	
Status	OK
Diagnostics Status: Code	0x0000
Diagnostics Status: Text	



Device Information

Profinet Device	
Activation Status	TRUE
Device Number	1
Vendor ID	00B0$_{hex}$
Device ID	1000$_{hex}$
Ethernet	
IP Address	192.168.1.11
Subnet Mask	255.255.255.0
Default Gateway	192.168.1.11
Name of Station	axl-f-bk-pn-1
DNS Hostname	axl-f-bk-pn-1
Peripheral Errors	
Status	Error
Diagnostics Status: Code	0x0003
Diagnostics Status: Text	Lost connection to Device

Figure 3-8 WBM: “PROFINET” page, “Device List” tab, “Device information” view

Update frequency for diagnostic data

Diagnostic data is updated with an update frequency of 1/s. The following information is updated:

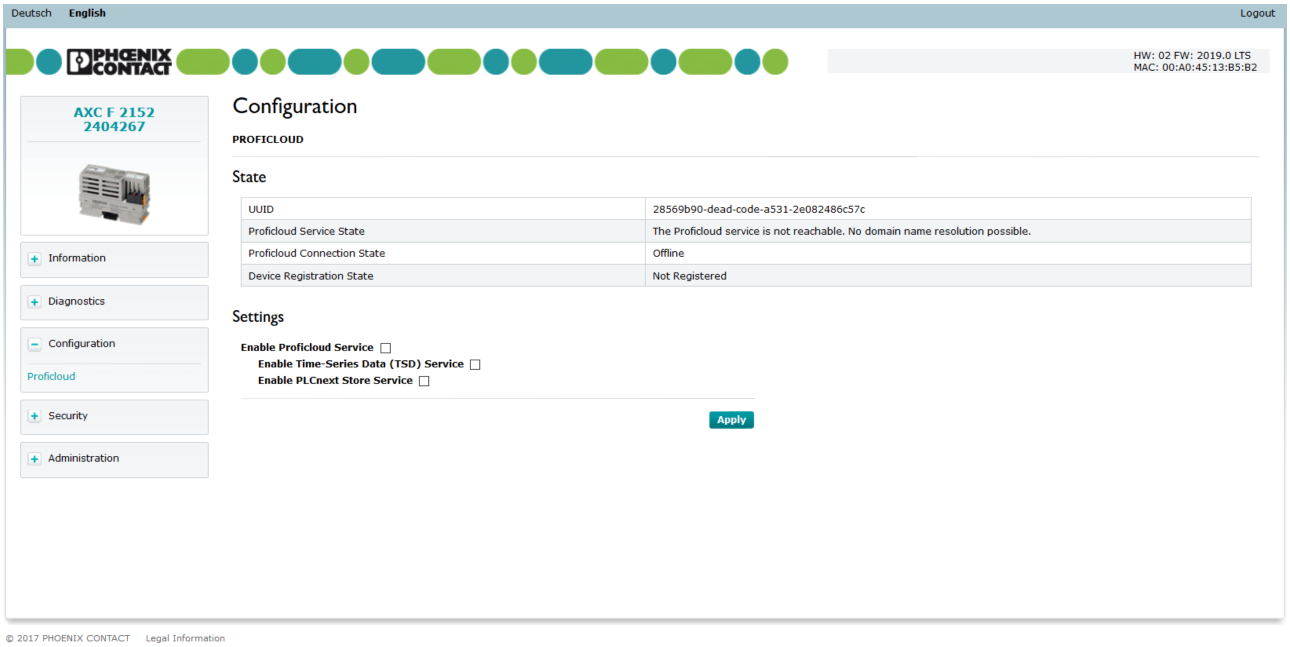
- Device list: The device list can be recomposed, e.g., after changing and updating the PLCnext Engineer project.
- Diagnostic data:
  - Change of connection status of the PROFINET devices.
  - Diagnostic state of the PROFINET devices.

For additional information on the diagnostic code, please refer to the Appendix A 4 on page 200.

3.8 “Configuration” area

3.8.1 “PROFICLOUD” page

The “PROFICLOUD” page provides status information on the connection of the controller to PROFICLOUD.
You can also specify if you want to operate the controller with or without a PROFICLOUD connection and which PROFICLOUD services you want to use.



Configuration

PROFICLOUD

State

UUID	28569b90-dead-code-a531-2e082486c57c
Proficloud Service State	The Proficloud service is not reachable. No domain name resolution possible.
Proficloud Connection State	Offline
Device Registration State	Not Registered

Settings

Enable Proficloud Service □
Enable Time-Series Data (TSD) Service □
Enable PLCnext Store Service □

Figure 3-9 WBM: “PROFICLOUD” page

Activating the PROFICLOUD connection

If you want to operate the controller with a PROFICLOUD connection, you have to enable the Proficloud service of the controller. If the Proficloud service of the controller is enabled, the controller tries to establish a connection to PROFICLOUD.

- Enable the “Enable Proficloud Service” check box.
- Click on the “Apply” button.

The PROFICLOUD connection of the controller is enabled.

When the PROFICLOUD connection is enabled, you can specify which service you want to use.

To protect the Proficloud configuration against unauthorized access, see Section “Blocking the PROFICLOUD access” on page 101.

Enabling the TSD service

- To transmit process data from a PLCnext Engineer project to the TSD PROFICLOUD service, enable the “Enable Time-Series Data (TSD) Service” check box.

Enabling the PLCnext Store service

To download apps from the PLCnext Store to the controller, the PLCnext Store service must be enabled on the device.

- To enable the PLCnext Store service on the device, enable the “Enable PLCnext Store Service” check box.

Disabling the PROFICLOUD connection

If you want to operate the controller without a PROFICLOUD connection, you have to disable the Proficloud service of the controller. In this case, establishing a connection between the controller and PROFICLOUD is not possible due to technical reasons.

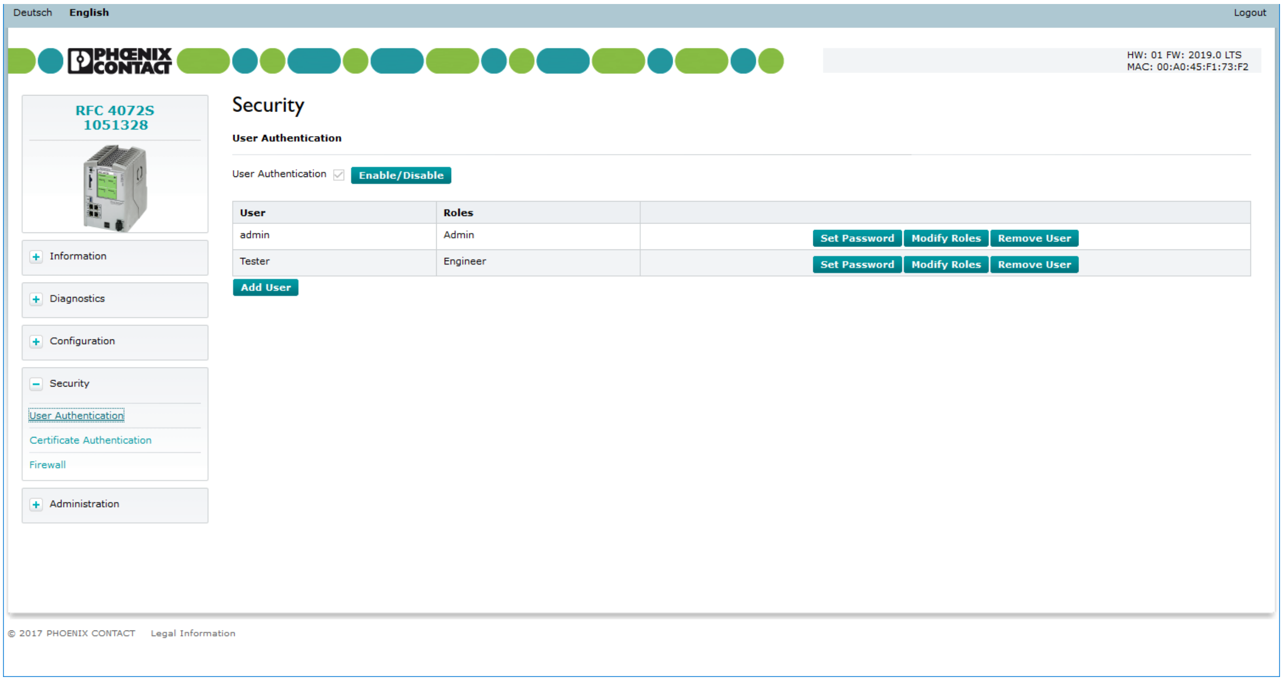
- Disable the “Enable Proficloud Service” check box.
- Click on the “Apply” button.

The PROFICLOUD connection of the controller is disabled.

3.9 “Security” area

In the “Security” area, the security-relevant settings for the controller are configured.

3.9.1 “User Authentication” page



HW: 01 FW: 2019.0 LTS
MAC: 00:A0:45:F1:73:F2

Security

User Authentication

User Authentication ☑ Enable/Disable

User	Roles	
admin	Admin	Set Password Modify Roles Remove User
Tester	Engineer	Set Password Modify Roles Remove User

Figure 3-10 WBM: “User Authentication” page

User authentication

Enable or disable user authentication on the “User Authentication” page. When user authentication is enabled, authentication with a user name and password is required for access to certain components of the controller and certain functions in PLCnext Engineer. When user authentication is disabled, authentication is not necessary to access WBM, the OPC UA server of the controller, or to access the controller using PLCnext Engineer. Access to the file system via SFTP and access to the shell via SSH requires authentication (with administrator rights) even if user authentication is disabled.

User authentication is enabled by default. Upon delivery, the “admin” user is already created with administrator rights.

Recommended:

- Only use the administrator password printed on the controller for logging into WBM for the first time.
- Once you have logged in successfully, change the administrator password to prevent unauthorized administrator access.

The modified administrator access data is stored in the overlay file system on the internal parameterization memory. If you operate the controller with an SD card, the overlay file system is saved to the SD card.

Please note:

Enabled user authentication only provides a limited degree of protection against unauthorized network access.
Due to the communication interfaces of the controller, the controller should not be used in safety-critical applications unless additional security appliances are used.

- Ensure that you always operate the controller with the latest firmware version.

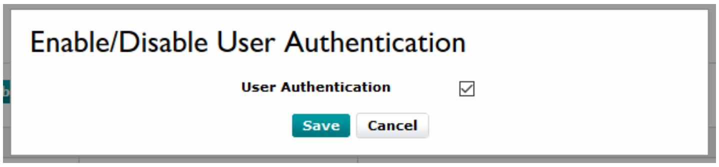
Follow the security advice on unauthorized network access in Section 1.6.

Enabling/disabling user authentication

To enable/disable user authentication, proceed as follows:

- Click on the “Enable/Disable” button next to the “User Authentication” check box.

The “Enable/Disable User Authentication” dialog opens.



Enable/Disable User Authentication

User Authentication ☑

Figure 3-11 WBM: “User Authentication” page, “Enable/Disable User Authentication” dialog

- To enable user authentication, enable the “User Authentication” check box.
- To disable user authentication, disable the “User Authentication” check box.
- Click the “Save” button to adopt the settings.

User management

Via user authentication, the access data of all users authorized to access the controller is managed and the required access permissions are assigned to each user.

The access data of all newly created users is stored on the internal parameterization memory. If you operate the controller with an SD card, the access data is saved to the SD card. If the SD card is inserted into another controller of the same type, the access data stored on the SD card is used for access to the controller.

When inserting the SD card into another controller please note:

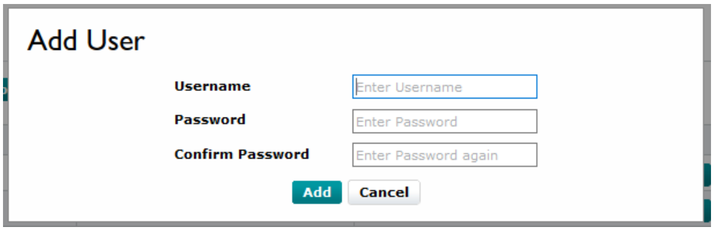
If you have changed the administrator access data after logging into WBM for the first time, the modified access data stored on the SD card is used for accessing the controller. In this case, it is no longer possible to log in with the “admin” user name and the administrator password printed on the device.

Adding a user

Proceed as follows to add a user:

- On the “User Authentication” page, click on the “Add User” button.

The “Add User” dialog opens.



Add User

Figure 3-12 WBM: “User Authentication” page, “Add User” dialog

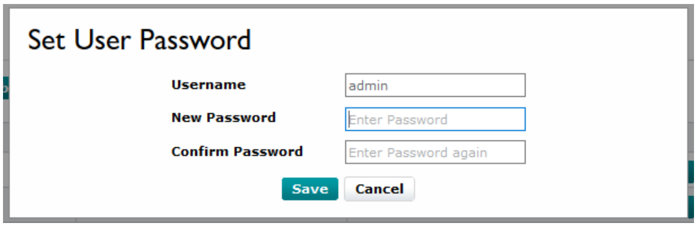
- Enter the user name and password into the respective input field.
- To add the user in the user manager, click on the “Add” button.

Setting a password

Proceed as follows to change a user password:

- On the “User Authentication” page, click on the “Set Password” button in the line for the desired user.

The “Set User Password” dialog opens.



Set User Password

Username: admin

Figure 3-13 WBM: “User Authentication” page, “Set User Password” dialog

- Enter the new password into the “New Password” and “Confirm Password” input fields.
- To save the new password, click on the “Save” button.

Modifying user roles

You can select one or more user roles containing different permissions for each user. These permissions control access to

– The controller SD card

– The controller using PLCnext Engineer

– The PLCnext Engineer HMI

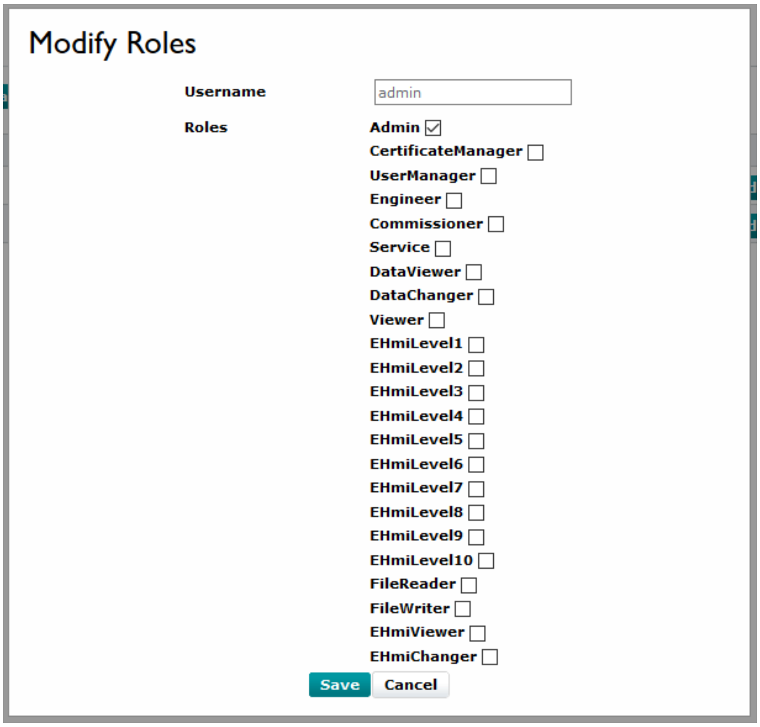
– WBM

– The OPC UA server of the controller

To assign one or more user role(s) to a user, proceed as follows:

- On the “User Authentication” page, click on the “Modify Roles” button in the line for the desired user (see Figure 3-10 on page 82).

The “Modify Roles” dialog opens.



Modify Roles

Username: admin

Roles:
- Admin ☑
- CertificateManager □
- UserManager □
- Engineer □
- Commissioner □
- Service □
- DataViewer □
- DataChanger □
- Viewer □
- EHmiLevel1 □
- EHmiLevel2 □
- EHmiLevel3 □
- EHmiLevel4 □
- EHmiLevel5 □
- EHmiLevel6 □
- EHmiLevel7 □
- EHmiLevel8 □
- EHmiLevel9 □
- EHmiLevel10 □
- FileReader □
- FileWriter □
- EHmiViewer □
- EHmiChanger □

Figure 3-14 WBM: “User Authentication” page, “Modify Roles” dialog

- Enable the check box of the user role(s) that you would like to assign to the user.

> You can manage access permission to the PLCnext Engineer HMI application via the HmiLevel1...10, EHmiViewer and EHmiChanger user roles. The assigned user roles specify if and to what extend a user can read and write from/to the HMI application.
>
> For detailed information on the security functions in a PLCnext Engineer HMI application as well as on handling HMI user roles, please refer to the PLCnext Engineer online help.

- Click on the “Save” button to save the selected user role(s) for the user.

Table 3-2 shows the possible user roles and access permission.

x = Access permission available
- = No access permission

Table 3-2 User roles and assigned access permissions in the various applications

Application or component of the controller	Access permission	User role													
		Admin	**CertificateManager**	**UserManager**	**Engineer**	**Commissioner**	**Service**	**DataViewer**	**DataChanger**	**Viewer**	**EHmiLevelX**	**File Reader**	**FileWriter**	**EHmiViewer**	**EHmiChanger**
SD card/parameterization memory	SFTP access to the file system with an SFTP client **Please note:** Authentication with a user name and password is **always** required for SFTP access, even when user authentication is disabled.	x	-	-	-	-	-	-	-	-	-	-	-	-	-
Shell	SSH access to the shell **Please note:** Authentication with a user name and password is **always** required for SSH access, even when user authentication is disabled.	x	-	-	-	-	-	-	-	-	-	-	-	-	-
PLCnext Engineer	View values in the cockpit (e.g., utilization, etc.)	x	-	-	x	x	x	x	x	x	-	-	-	-	-
PLCnext Engineer	Transfer a project to the controller	x	-	-	x	-	-	-	-	-	-	-	-	-	-
PLCnext Engineer	Start (cold/warm restart) or stop the controller	x	-	-	x	x	x	-	-	-	-	-	-	-	-
PLCnext Engineer	Restart (reboot) the controller	x	-	-	-	-	-	-	-	-	-	-	-	-	-
PLCnext Engineer	Reset the controller to default setting type 1	x	-	-	-	-	-	-	-	-	-	-	-	-	-
PLCnext Engineer	View online variable values	x	-	-	x	-	x	x	x	x	-	-	-	-	-
PLCnext Engineer	Overwrite variables	x	-	-	x	-	x	-	x	-	-	-	-	-	-
PLCnext Engineer	Set and delete breakpoints	x	-	-	x	-	x	-	-	-	-	-	-	-	-
WBM	View “General Data” page	x	-	x	x	-	-	-	-	-	-	-	-	-	-
WBM	Manage users	x	-	x	-	-	-	-	-	-	-	-	-	-	-
WBM	Edit TrustStores and IdentityStores	x	x	-	-	-	-	-	-	-	-	-	-	-	-
WBM	Configure the firewall	x	-	-	-	-	-	-	-	-	-	-	-	-	-
WBM	Update the firmware	x	-	-	-	-	-	-	-	-	-	-	-	-	-
PLCnext Engineer HMI application	View online variable values	x	-	-	-	-	-	-	-	-	-	-	-	x	-

Table 3-2 User roles and assigned access permissions in the various applications

Application or component of the controller	Access permission	User role													
		Admin	CertificateManager	UserManager	Engineer	Commissioner	Service	DataViewer	DataChanger	Viewer	EHmiLevelX	File Reader	FileWriter	EHmiViewer	EHmiChanger
PLCnext Engineer HMI application	Overwrite variables	x	-	-	-	-	-	-	-	-	-	-	-	-	x
OPC UA client	View online variable values	x	-	-	x	-	x	x	x	x	-	-	-	-	-
OPC UA client	Overwrite variables	x	-	-	x	-	x	-	x	-	-	-	-	-	-
OPC UA client	Read files	x	-	-	-	-	-	-	-	-	-	x[1]	-	-	-
OPC UA client	Write files	x	-	-	-	-	-	-	-	-	-	-	x[2]	-	-

[1] FileReaders can only read files via an OPC UA client if the OPC UA file transfer is activated in PLCnext Engineer (for additional information, please refer to the PLCnext Engineer online help).

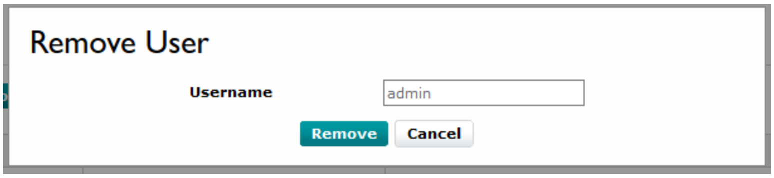
[2] FileWriters can only write files via an OPC UA client if the OPC UA file transfer is activated in PLCnext Engineer (for additional information, please refer to the PLCnext Engineer online help).

Removing a user

Proceed as follows to remove a user:

- On the “User Authentication” page, click on the “Remove User” button in the line of the user to be removed.

The “Remove User” dialog opens.



Remove User

Username: admin

Figure 3-15 WBM: “User Authentication” page, “Remove User” dialog

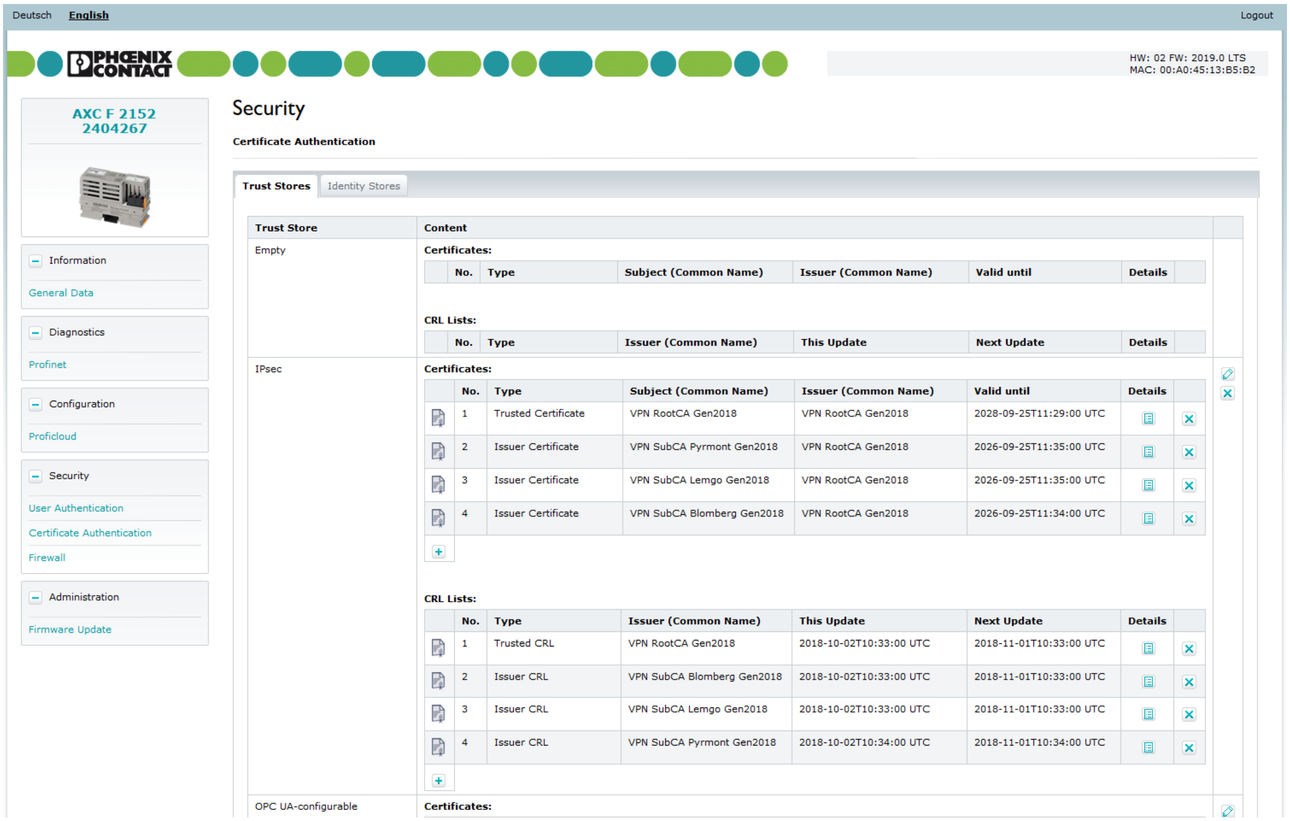
- Click on the “Remove” button to delete the user.

3.9.2 “Certificate Authentication” page

The “Certificate Authentication” page is used to manage certificates for secure communication of the controller. For this, the “TrustStores” and “IdentityStores” tabs are available.
On the “TrustStores” tab, you can store trusted and untrusted certificates of possible communication partners.
On the “IdentityStores” tab, you can store your own certificates.

3.9.2.1 “TrustStores” tab

On the “TrustStores” tab, you can create different TrustStores, name them and add certificates and revocation lists.



Security

Certificate Authentication

Trust Store	Content
Empty	Certificates:
	CRL Lists:
IPsec	Certificates:
	CRL Lists:
OPC UA-configurable	Certificates:

Empty – Certificates:

	No.	Type	Subject (Common Name)	Issuer (Common Name)	Valid until	Details	

Empty – CRL Lists:

	No.	Type	Issuer (Common Name)	This Update	Next Update	Details	

IPsec – Certificates:

	No.	Type	Subject (Common Name)	Issuer (Common Name)	Valid until	Details	
	1	Trusted Certificate	VPN RootCA Gen2018	VPN RootCA Gen2018	2028-09-25T11:29:00 UTC	▤	✕
	2	Issuer Certificate	VPN SubCA Pyrmont Gen2018	VPN RootCA Gen2018	2026-09-25T11:35:00 UTC	▤	✕
	3	Issuer Certificate	VPN SubCA Lemgo Gen2018	VPN RootCA Gen2018	2026-09-25T11:35:00 UTC	▤	✕
	4	Issuer Certificate	VPN SubCA Blomberg Gen2018	VPN RootCA Gen2018	2026-09-25T11:34:00 UTC	▤	✕
+							

IPsec – CRL Lists:

	No.	Type	Issuer (Common Name)	This Update	Next Update	Details	
	1	Trusted CRL	VPN RootCA Gen2018	2018-10-02T10:33:00 UTC	2018-11-01T10:33:00 UTC	▤	✕
	2	Issuer CRL	VPN SubCA Blomberg Gen2018	2018-10-02T10:33:00 UTC	2018-11-01T10:33:00 UTC	▤	✕
	3	Issuer CRL	VPN SubCA Lemgo Gen2018	2018-10-02T10:33:00 UTC	2018-11-01T10:33:00 UTC	▤	✕
	4	Issuer CRL	VPN SubCA Pyrmont Gen2018	2018-10-02T10:34:00 UTC	2018-11-01T10:34:00 UTC	▤	✕
+							

Figure 3-16 WBM: “Certificate Authentication” page, “TrustStores” tab

Adding a TrustStore

- To add a TrustStore, click on the + button at the end of the TrustStore table.
- In the dialog that opens, enter a name in the “Name” input field.
- Then click on the “Add” button.

Deleting a TrustStore

- To delete a TrustStore, click on the ✕ button to the right of the TrustStore table.
- In the window that opens, click on the “Remove” button.

Renaming a TrustStore

- To rename a TrustStore, click on the ✎ button to the right of the TrustStore table.
- In the dialog that opens, enter a new name in the “Name” input field.
- Click on the “Rename” button.

Creating a TrustStore

Each TrustStore is represented by two tables in WBM:

– “Certificates” table:
In this table, you can manage trusted certificates and issuer certificates.

– “CRL Lists” table:
In this table, you can manage the revocation lists (CRL - Certificate Revocation Lists) for the corresponding TrustStore. For this, you store untrusted certificates and issuer certificates.

Adding certificates and CRLs

You can add certificates and CRLs to a TrustStore:

- To add a certificate, click on the + button below the “Certificates” table of the corresponding TrustStore.

⇒ The “Add Certificate” dialog opens (Figure 3-17).

- To add a CRL, click on the + button below the “CRL Lists” table of the corresponding TrustStore.

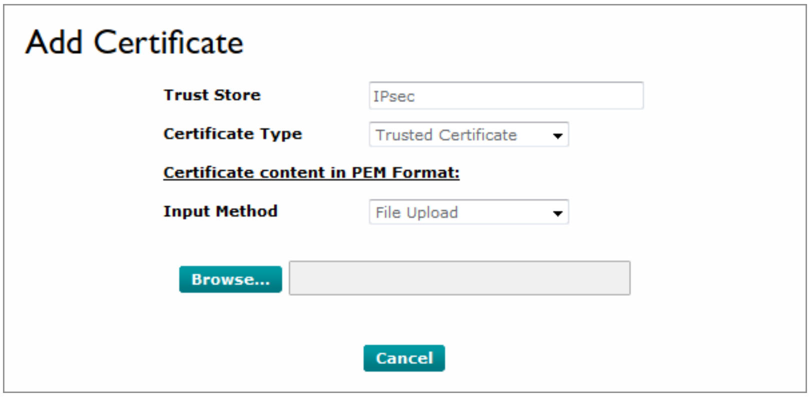
⇒ The “Add CRL List” dialog opens (Figure 3-18).

In the “Certificates” TrustStore table, a distinction is made between the “Issuer Certificate” and “Trusted Certificate” certificate types.

- Select the desired certificate type from the “Certificate Type” drop-down list.

Possible settings:

– Issuer Certificate
– Trusted Certificate



Add Certificate

Trust Store: IPsec

Certificate Type: Trusted Certificate

Certificate content in PEM Format:

Input Method: File Upload

Figure 3-17 WBM: “Certificate Authentication” page, “TrustStores” tab, “Add Certificate” dialog, File Upload

Input method

- From the “Input Method” drop-down list, select the way a certificate or CRL is to be added to the TrustStore.

Possible settings:

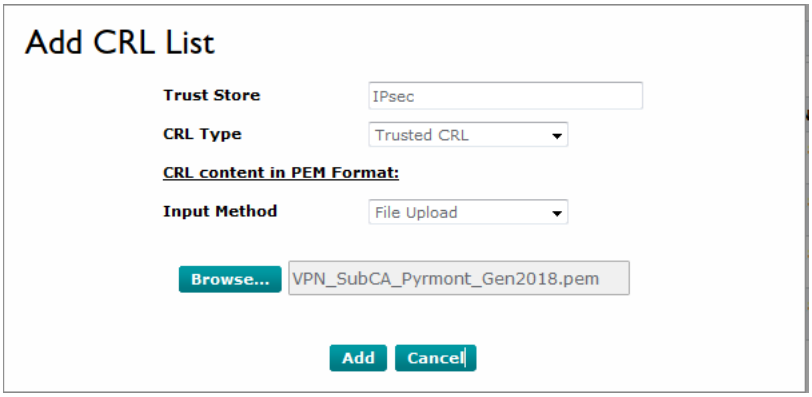
– File Upload
– Text Content

“File Upload”:

You can upload a certificate or CRL.

- To upload a certificate in .pem format, select “File Upload”.
- Click on “Browse”.
- In the file explorer that opens, select the desired certificate.
- Then click on the “Add” button.

⇒ The certificate or CRL is added to the TrustStore.



Add CRL List

Trust Store: IPsec

CRL Type: Trusted CRL

CRL content in PEM Format:

Input Method: File Upload

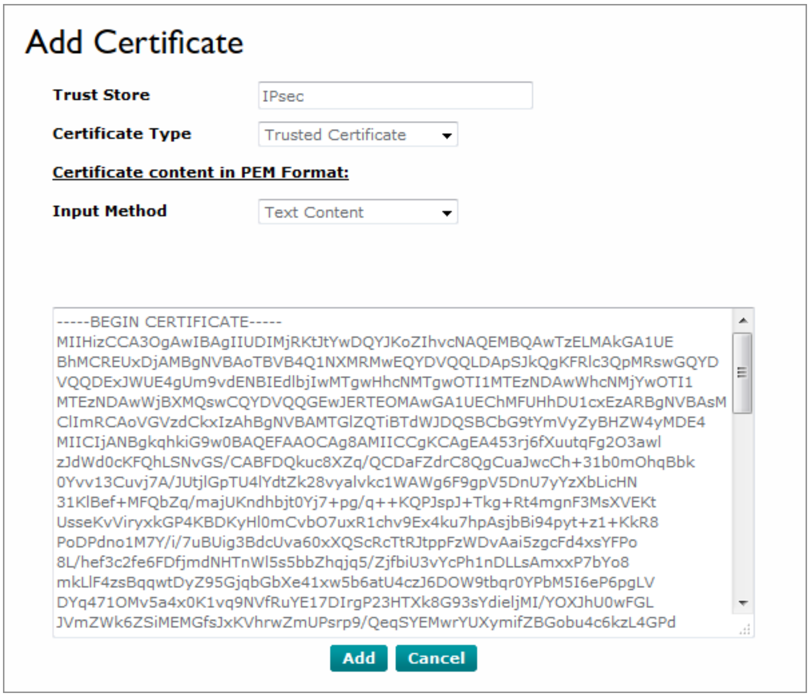
Browse... VPN_SubCA_Pyrmont_Gen2018.pem

Figure 3-18 WBM: “Certificate Authentication” page, “TrustStores” tab, “Add CRL List” dialog, File Upload

“Text Content”:

- To add a certificate or CRL in text format, select “Text Content”.
- Enter the text in the input field.
- Then click on the “Add” button.

⇒ The certificate or CRL is added to the TrustStore.



Add Certificate

Trust Store: IPsec

Certificate Type: Trusted Certificate

Certificate content in PEM Format:

Input Method: Text Content

```
-----BEGIN CERTIFICATE-----
MIIHizCCA3OgAwIBAgIIUDIMjRKtJtYwDQYJKoZIhvcNAQEMBQAwTzELMAkGA1UE
BhMCREUxDjAMBgNVBAoTBVB4Q1NXMRMwEQYDVQQLDApSJkQgKFRlc3QpMRswGQYD
VQQDExJWUE4gUm9vdENBIEdlbjIwMTgwHhcNMTgwOTI1MTEzNDAwWhcNMjYwOTI1
MTEzNDAwWjBXMQswCQYDVQQGEwJERTEOMAwGA1UEChMFUHhDU1cxEzARBgNVBAsM
ClImRCAoVGVzdCkxIzAhBgNVBAMTGlZQTiBTdWJDQSBCbG9tYmVyZyBHZW4yMDE4
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA453rj6fXuutqFg2O3awl
zJdWd0cKFQhLSNvGS/CABFDQkuc8XZq/QCDaFZdrC8QgCuaJwcCh+31b0mOhqBbk
0Yvv13Cuvj7A/JUtjlGpTU4lYdtZk28vyalvkc1WAWg6F9gpV5DnU7yYzXbLicHN
31KlBef+MFQbZq/majUKndhbjt0Yj7+pg/q++KQPJspJ+Tkg+Rt4mgnF3MsXVEKt
UsseKvViryxkGP4KBDKyHl0mCvbO7uxR1chv9Ex4ku7hpAsjbBi94pyt+z1+KkR8
PoDPdno1M7Y/i/7uBUig3BdcUva60xXQScRcTtRJtppFzWDvAai5zgcFd4xsYFPo
8L/hef3c2fe6FDfjmdNHTnWl5s5bbZhqjq5/ZjfbiU3vYcPh1nDLLsAmxxP7bYo8
mkLlF4zsBqqwtDyZ95GjqbGbXe41xw5b6atU4czJ6DOW9tbqr0YPbM5I6eP6pgLV
DYq471OMv5a4x0K1vq9NVfRuYE17DIrgP23HTXk8G93sYdieljMI/YOXJhU0wFGL
JVmZWk6ZSiMEMGfsJxKVhrwZmUPsrp9/QeqSYEMwrYUXymifZBGobu4c6kzL4GPd
```

Figure 3-19 WBM: “Certificate Authentication” page, “TrustStores” tab, “Add Certificate” dialog, Text Content

Deleting certificates and CRLs

- To delete a certificate or CRL from a TrustStore, click on the ✕ button of the respective certificate or CRL.

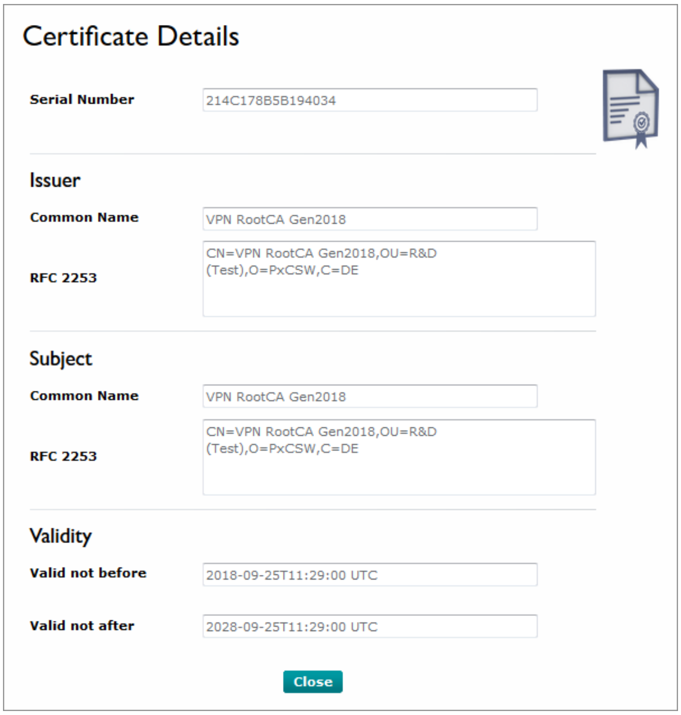
- In the window that opens, click on the “Remove” button.

Detail view

The detail views provide detailed information on every certificate and CRL:

- To open the detail view, click on the ☰ button of a certificate or CRL.

⇒ The detail view opens (see Figure 3-20 and Figure 3-21).



Certificate Details

Serial Number: 214C178B5B194034

Issuer

Common Name: VPN RootCA Gen2018

RFC 2253: CN=VPN RootCA Gen2018,OU=R&D (Test),O=PxCSW,C=DE

Subject

Common Name: VPN RootCA Gen2018

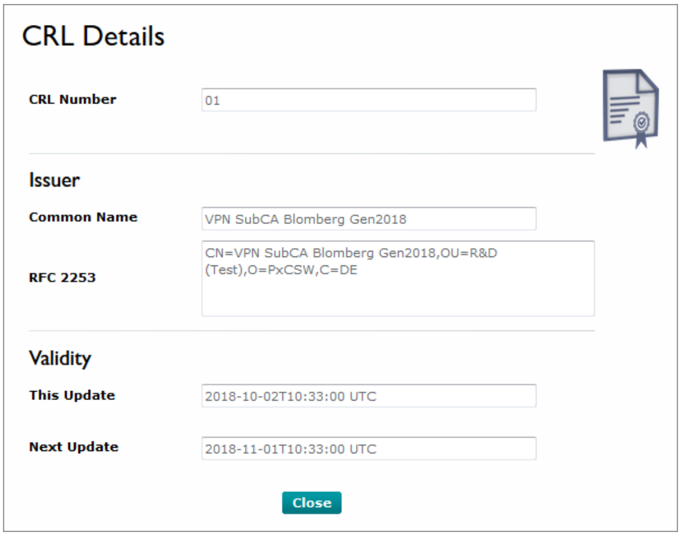
RFC 2253: CN=VPN RootCA Gen2018,OU=R&D (Test),O=PxCSW,C=DE

Validity

Valid not before: 2018-09-25T11:29:00 UTC

Valid not after: 2028-09-25T11:29:00 UTC

Figure 3-20 WBM: “Certificate Authentication” page, “TrustStores” tab, Certificate Details



CRL Details

CRL Number: 01

Issuer

Common Name: VPN SubCA Blomberg Gen2018

RFC 2253: CN=VPN SubCA Blomberg Gen2018,OU=R&D (Test),O=PxCSW,C=DE

Validity

This Update: 2018-10-02T10:33:00 UTC

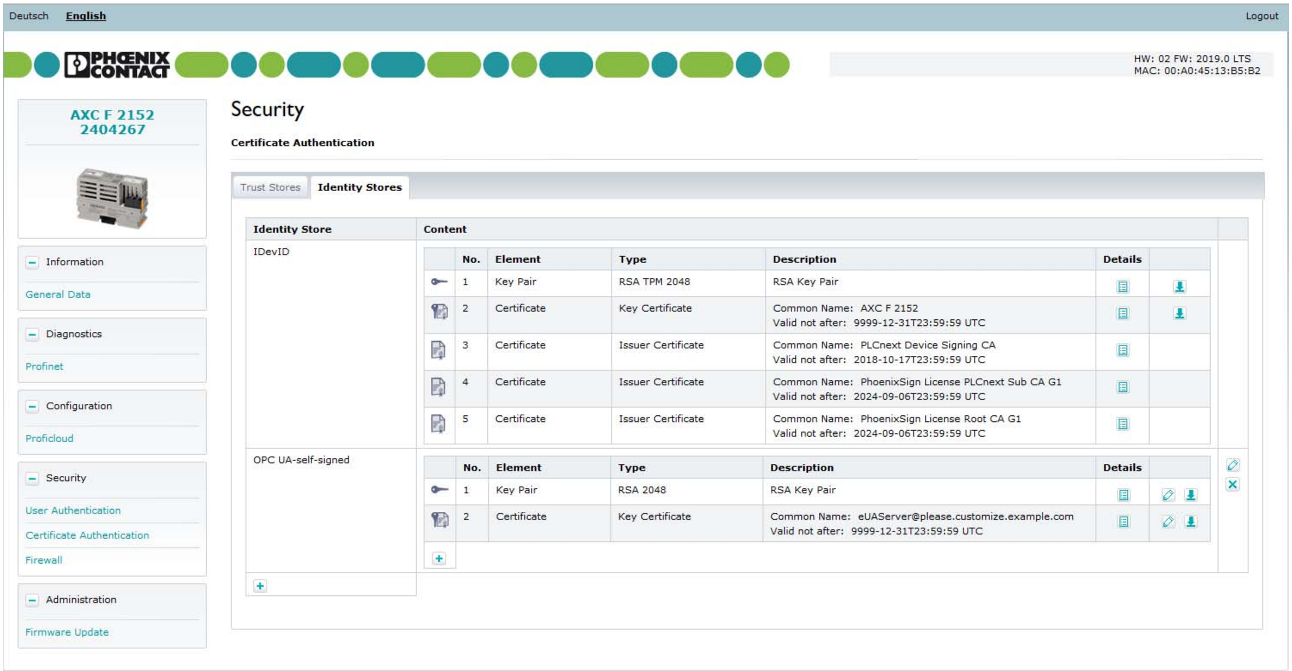
Next Update: 2018-11-01T10:33:00 UTC

Figure 3-21 WBM: “Certificate Authentication” page, “TrustStores” tab, Certificate Details

- To close the detail view, click on the “Close” button.

3.9.2.2 “IdentityStores” tab

On the “IdentityStores” tab, you can create and manage different IdentityStores.
Each IdentityStore usually contains an RSA key pair and the corresponding key certificate.
As an option, you can add further issuer certificates to an IdentityStore. The “IDevID”and “OpcUA-SelfSigned” IdentityStores are part of the system and supplied with the controller.



HW: 02 FW: 2019.0 LTS
MAC: 00:A0:45:13:B5:B2

Security

Certificate Authentication

Identity Store	Content
IDevID	

	No.	Element	Type	Description	Details
	1	Key Pair	RSA TPM 2048	RSA Key Pair	
	2	Certificate	Key Certificate	Common Name: AXC F 2152 Valid not after: 9999-12-31T23:59:59 UTC	
	3	Certificate	Issuer Certificate	Common Name: PLCnext Device Signing CA Valid not after: 2018-10-17T23:59:59 UTC	
	4	Certificate	Issuer Certificate	Common Name: PhoenixSign License PLCnext Sub CA G1 Valid not after: 2024-09-06T23:59:59 UTC	
	5	Certificate	Issuer Certificate	Common Name: PhoenixSign License Root CA G1 Valid not after: 2024-09-06T23:59:59 UTC	

Identity Store	Content
OPC UA-self-signed	

	No.	Element	Type	Description	Details
	1	Key Pair	RSA 2048	RSA Key Pair	
	2	Certificate	Key Certificate	Common Name: eUAServer@please.customize.example.com Valid not after: 9999-12-31T23:59:59 UTC	

Figure 3-22 WBM: “Certificate Authentication” page, “IdentityStores” tab

Adding an IdentityStore

- To add an IdentityStore, click on the + button at the end of the IdentityStore table.
- In the dialog that opens, enter a name in the “Name” input field.
- From the “Key Pair” drop-down list, select the way the key pair is to be added.

Possible settings:

– Enter

– Generate

“Enter“

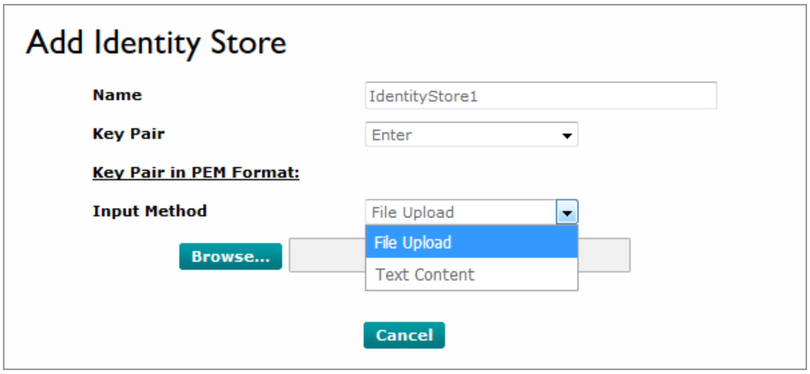
- From the “Input Method” drop-down list, select the way the key pair is to be added to the IdentityStore.

Possible settings:

– File Upload

– Text Content

For additional information on the input method, please refer to Section “Input method” on page 89.



Add Identity Store

Name: IdentityStore1

Key Pair: Enter

Key Pair in PEM Format:

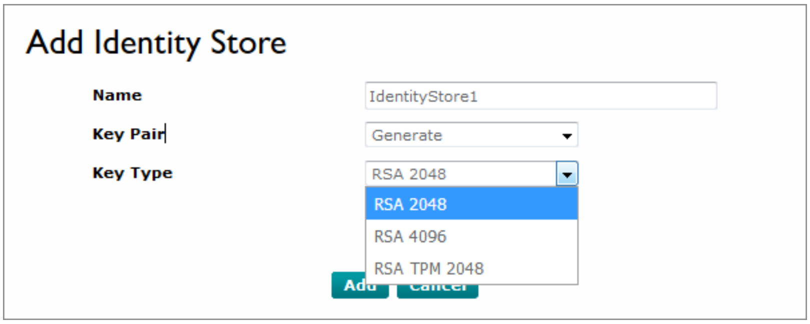
Input Method: File Upload

Figure 3-23 WBM: “Certificate Authentication” page, “IdentityStores” tab, “Add IdentityStore” dialog, entering a key pair

“Generate”

The controller automatically generates a key pair.

- From the “Key Type” drop-down list, select the encryption method.



Add Identity Store

Name: IdentityStore1

Key Pair: Generate

Key Type: RSA 2048

Figure 3-24 WBM: “Certificate Authentication” page, “IdentityStores” tab, “Add IdentityStore” dialog, generating a key pair

- To add the IdentityStore, click on the “Add” button.

Deleting an IdentityStore

- To delete an IdentityStore, click on the ✕ button to the right of the IdentityStore table.
- In the window that opens, click on the “Remove” button.

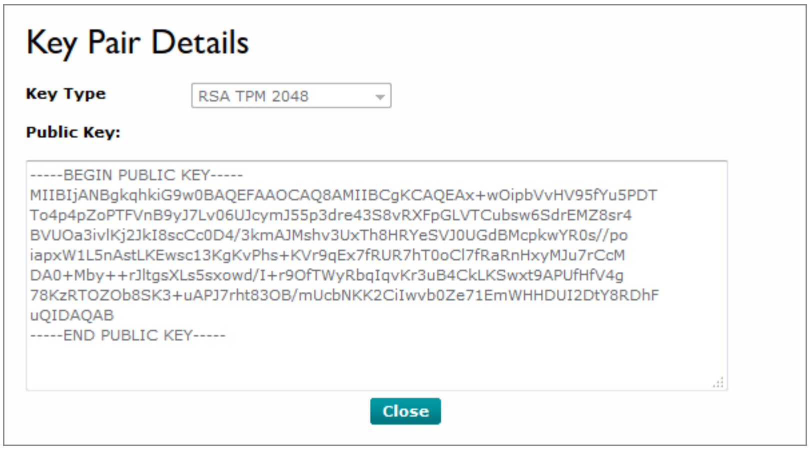
Renaming an IdentityStore

- To rename an IdentityStore, click on the ✎ button to the right of the IdentityStore table.
- In the dialog that opens, enter a new name in the “New Name” input field.
- Click on the “Rename” button.

Detail view

The detail views provide detailed information on every key pair, key certificate or issuer certificate:

- To open the detail view, click on the ☰ button of a key pair or certificate (see also Figure 3-20 “WBM: “Certificate Authentication” page, “TrustStores” tab, Certificate Details”).
- To close the detail view, click on the “Close” button.



Key Pair Details

Key Type: RSA TPM 2048

Public Key:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAx+wOipbVvHV95fYu5PDT
To4p4pZoPTFVnB9yJ7Lv06UJcymJ55p3dre43S8vRXFpGLVTCubsw6SdrEMZ8sr4
BVUOa3ivlKj2JkI8scCc0D4/3kmAJMshv3UxTh8HRYeSVJ0UGdBMcpkwYR0s//po
iapxW1L5nAstLKEwsc13KgKvPhs+KVr9qEx7fRUR7hT0oCl7fRaRnHxyMJu7rCcM
DA0+Mby++rJltgsXLs5sxowd/I+r9OfTWyRbqIqvKr3uB4CkLKSwxt9APUfHfV4g
78KzRTOZOb8SK3+uAPJ7rht83OB/mUcbNKK2CiIwvb0Ze71EmWHHDUI2DtY8RDhF
uQIDAQAB
-----END PUBLIC KEY-----
```

Figure 3-25 WBM: “Certificate Authentication” page, “IdentityStores” tab, Key Pair Details

Downloading public keys or key certificates

You can download the content of the public key of a key pair as a .pem file.
If a key certificate is available, you can download it as a .crt file.

- Click on the ⭳ button in the last column of the respective table entry.
- Save the file to a directory of your choice or directly open the file with a suitable tool.

Setting a key pair

- To set a key pair, click on the ✎ button in the last column of the table entry.

The options for setting a key pair correspond to the options in Section “Adding an IdentityStore” on page 93.

Setting a key certificate

- To set a key certificate, click on the ✎ button in the last column of the table entry.

The options for setting a key certificate correspond to the options in Section “Adding an IdentityStore” on page 93.

Adding issuer certificates

- To add an issuer certificate, click on the + button below the table of the corresponding IdentityStore.
- Select an input method. See Section “Adding certificates and CRLs” on page 89.

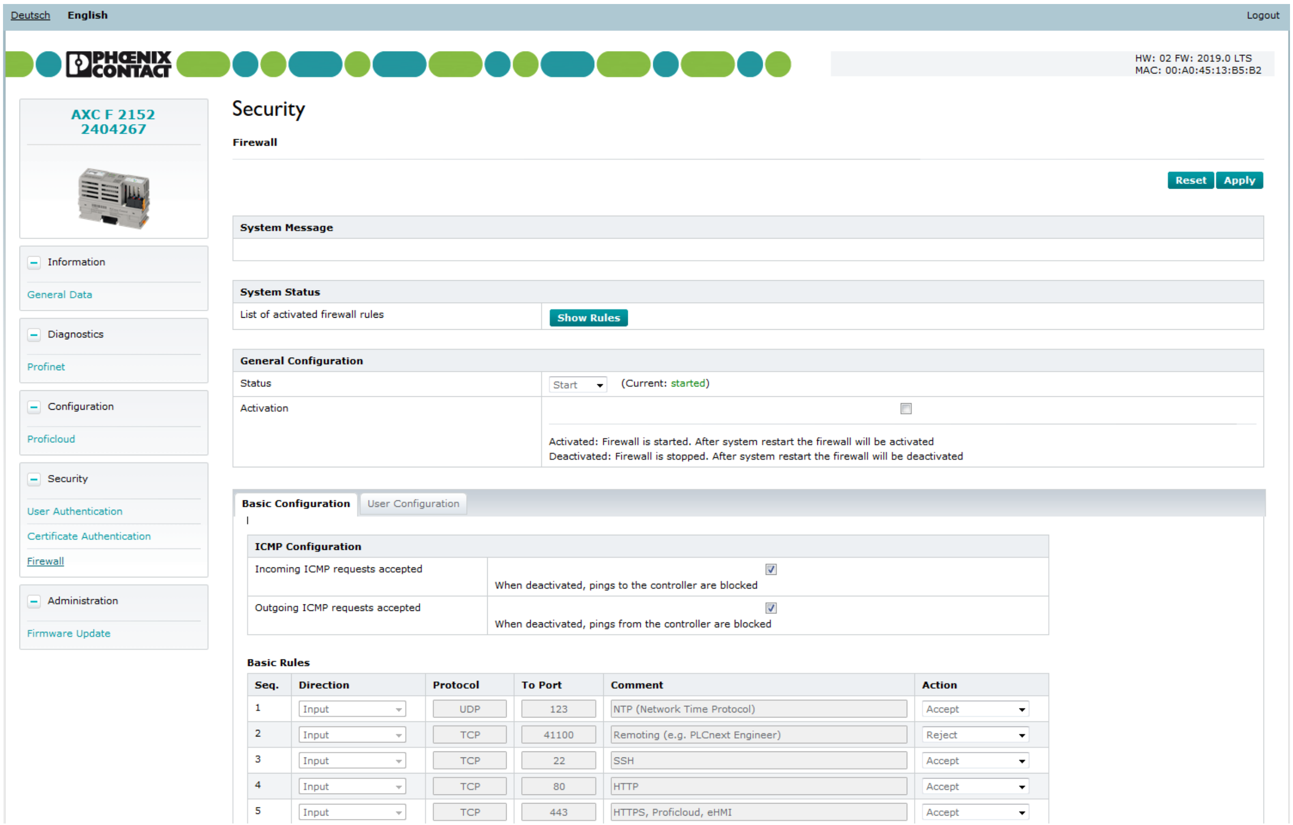
Deleting issuer certificates

- To delete an issuer certificate, click on the ✕ button of the certificate.
- In the window that opens, click on the “Remove” button.

3.9.3 “Firewall” page

On the “Firewall” page, you can configure the firewall of your controller.

> You can only open the “Firewall” page if you are logged into WBM as an administrator. For information on user roles, please refer to Section 3.9.1 on page 82.



HW: 02 FW: 2019.0 LTS
MAC: 00:A0:45:13:B5:B2

Security

Firewall

System Message

System Status

List of activated firewall rules

General Configuration

Status: Start (Current: started)

Activation: □

Activated: Firewall is started. After system restart the firewall will be activated
Deactivated: Firewall is stopped. After system restart the firewall will be deactivated

ICMP Configuration

Incoming ICMP requests accepted ☑
When deactivated, pings to the controller are blocked

Outgoing ICMP requests accepted ☑
When deactivated, pings from the controller are blocked

Basic Rules

Seq.	Direction	Protocol	To Port	Comment	Action
1	Input	UDP	123	NTP (Network Time Protocol)	Accept
2	Input	TCP	41100	Remoting (e.g. PLCnext Engineer)	Reject
3	Input	TCP	22	SSH	Accept
4	Input	TCP	80	HTTP	Accept
5	Input	TCP	443	HTTPS, Proficloud, eHMI	Accept

Figure 3-26 WBM: “Firewall” page

“Reset” and “Apply” buttons

– **“Reset”**: Click on the “Reset” button to reset the firewall to default settings.

– **“Apply”**: Click on the “Apply” button to transfer all configured firewall settings to the controller.

“System Message”area

In the “System Message” area, responses and warnings are displayed. The following system messages are possible:

Table 3-3 Possible system messages

System message	Description
Status=Ok	The configured firewall settings were successfully transferred to the controller.
Warning	A warning of the controller is issued, e.g., if one or several additional filter configurations are available in the system. The warning contains the designations of all additionally loaded filter tables.
Error	At least one firewall configuration is faulty.

“System Status” area

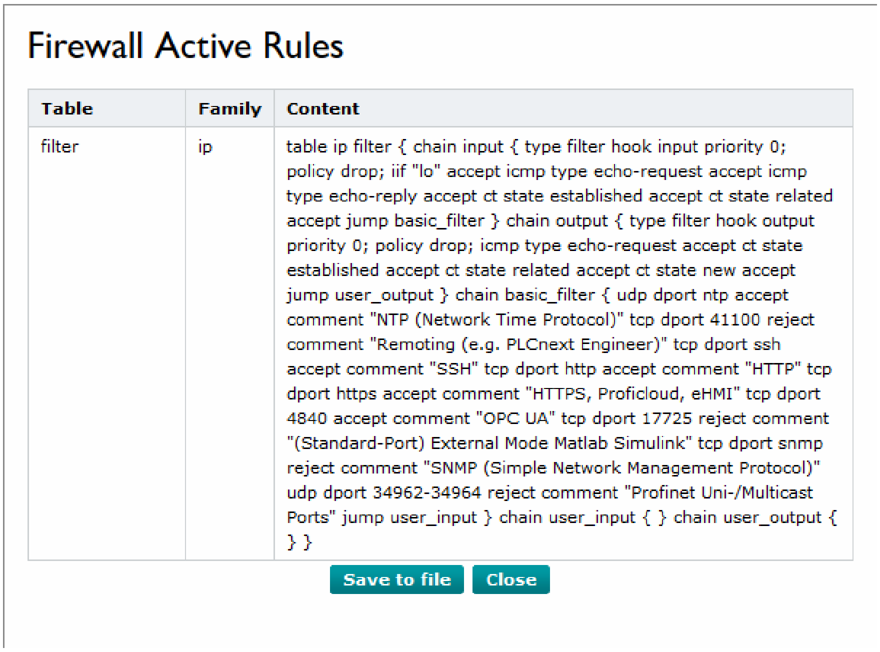
If the firewall is active, you can generate an overview of all enabled firewall rules in a TXT file in the “System Status” area.

- Click on the “Show Rules” button.

The TXT file with the activated firewall rules is generated and opened.

- To save the TXT file, click on the “Save to file” button in the open dialog.

The TXT file with the activated firewall rules is saved to the selected directory.



Firewall Active Rules

Table	Family	Content
filter	ip	table ip filter { chain input { type filter hook input priority 0; policy drop; iif "lo" accept icmp type echo-request accept icmp type echo-reply accept ct state established accept ct state related accept jump basic_filter } chain output { type filter hook output priority 0; policy drop; icmp type echo-request accept ct state established accept ct state related accept ct state new accept jump user_output } chain basic_filter { udp dport ntp accept comment "NTP (Network Time Protocol)" tcp dport 41100 reject comment "Remoting (e.g. PLCnext Engineer)" tcp dport ssh accept comment "SSH" tcp dport http accept comment "HTTP" tcp dport https accept comment "HTTPS, Proficloud, eHMI" tcp dport 4840 accept comment "OPC UA" tcp dport 17725 reject comment "(Standard-Port) External Mode Matlab Simulink" tcp dport snmp reject comment "SNMP (Simple Network Management Protocol)" udp dport 34962-34964 reject comment "Profinet Uni-/Multicast Ports" jump user_input } chain user_input { } chain user_output { } }

Figure 3-27 WBM: “Firewall” page, “Firewall Active Rules” dialog

“General Configuration” area

In the “General Configuration” area, you can view the current firewall status (e.g., Current: stopped), temporarily activate or deactivate the firewall, or permanently activate or deactivate the firewall.

Temporarily activating or deactivating the firewall

- To temporarily activate the firewall, select the “Start” or “Restart” entry from the drop-down list in the “Status” line.
- To activate the configuration, click on the “Apply” button.

The firewall is activated. This setting is no longer active after a restart of the controller.

- To temporarily deactivate the firewall, select the “Stop” entry from the drop-down list in the “Status” line.
- To activate the configuration, click on the “Apply” button.

The firewall is deactivated. This setting is no longer active after a restart of the controller.

Permanently activating or deactivating the firewall:

- To permanently activate the firewall, enable the check box in the “Activation” line.

The firewall is activated. The firewall remains activated even after a restart of the controller.

- To permanently deactivate the firewall, disable the check box in the “Activation” line.

The firewall is deactivated. The firewall remains deactivated even after a restart of the controller.

3.9.3.1 Configuring the firewall

Configuration of the firewall rules is divided into “Basic Configuration” and “User Configuration”. The “Basic Configuration” tab provides pre-defined firewall rules while you can create your own firewall rules on the “User Configuration” tab.

“Action” column

The options for activating and deactivating the filter rules are available in the “Action” column on the “Basic Configuration” tab as well as on the “User Configuration” tab.

- Select a setting from the drop-down list in the “Action” column for each firewall rule:

Table 3-4 Settings in the “Action” column

Option	Description
Accept	Connections are accepted. The connection request is accepted. The connection can be established.
Drop	Connections are dropped. There is no response to the request. The packet is dropped.
Reject	Connections are rejected. The sender receives a response via the rejected connection.
Continue	The rule is not executed. Choose this option to skip the basic rule and instead use a user-specific rule for the port. User-specific rules are configured in the “User Configuration” area.

- To activate the configuration, click on the “Apply” button.

“Basic Configuration” tab

On the “Basic Configuration” tab, the rules that are stored for the firewall upon delivery are displayed. You can select how the respective connections are to be treated for each rule.

“ICMP Configurations”

In the “ICMP Configurations”, you can specify how incoming and outgoing ICMP echo requests are to be treated. Possible settings:

- **“Incoming ICMP requests accepted”**
  Check box enabled: Incoming ICMP echo requests are accepted.
  Check box disabled: Incoming ICMP echo requests are blocked.
  The controller cannot be reached using a ping command.
- **“Outgoing ICMP requests accepted”**

Check box enabled: Outgoing ICMP echo requests are accepted.

Check box disabled: Outgoing ICMP echo requests are blocked. Ping commands cannot be issued by the controller.

“Basic Rules”

The “Basic Rules” area provides pre-defined firewall rules for different incoming connections, which you can enable or disable in the “Action” column. The configuration baseline is stored in the /etc/nftables/plcnext-filter file in the controller file system.

The configuration baseline contains the following rules for incoming connections (“Direction”: “Input“).

- Select an action for each rule (see Section ““Action” column” on page 97):

Table 3-5 Basic rules

Description	Protocol	Port
NTP (Network Time Protocol)	UDP	Port 123
Common Remoting, e.g., using PLCnext Engineer	TCP	Port 41100
SSH connections (e.g., for SSH shell connection or SFTP connection)	TCP	Port 22
HTTP	TCP	Port 80
HTTPS, Proficloud, eHMI (web server for eHMI and WBM)	TCP	Port 443
OPC UA	TCP	Port 4840
External Mode Matlab® Simulink®	TCP	Port 17725
SNMP (Simple Network Management Protocol)	TCP	Port 161
PROFINET uni/multicast ports	UDP	Ports 34962-34964

The settings are valid for all Ethernet interfaces. A limitation to certain Ethernet interfaces is specified via a user-specific rule in the “User Configuration” area (see Section “User configurations” on page 99).

With some activated firewall rules, there is a risk that accessing the controller becomes difficult due to blocked ports. Restoring access permissions can result in the loss of user data. Therefore, please consider the following notes when configuring basic rules:

Blocking the WBM access:

If you select the “Reject” or “Drop” action for basic rule no. 5 (TCP Port 443 - HTTPS, Proficloud, eHMI), you can no longer access the WBM of the controller after activating the rule (“Apply”). Therefore, you can no longer change the firewall rules via WBM.

- In case of a permanently started firewall (enabled “Activation” check box):
  To stop the firewall in this case, you have to reset the controller to the default settings. For more detailed information, please refer to the user manual for your controller. Note that during a reset to the default settings, user-specific data (applications, configuration, etc.) are deleted.
  Once the firewall is deactivated, you can again access WBM.
- In case of a permanently stopped firewall (disabled “Activation” check box):
  The firewall is stopped after restarting the controller. You can again access WBM.

Observe the following when using a PROFINET controller:

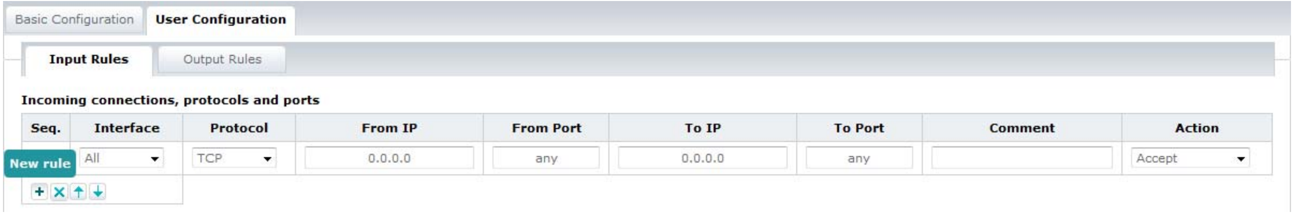
If you use the controller as a PROFINET controller, you have to ensure that, with an activated firewall, “Accept” is selected for basic rule no. 9 (UDP ports 34962-34964 - PROFINET uni/multicast ports). Otherwise, establishing a connection to certain PROFINET devices is not possible.

User configurations

In addition or as an alternative to the basic rules, you can define and activate your own, user-specific firewall rules for different filter categories in the “User Configuration” area. You can create new rules, delete rules or change the order of rules using the buttons at the end of the table.

Adding a new rule

Button	Function
+	“New Rule”: Add a new filter rule
✕	“Delete Rule”: Delete the selected filter rule
↑ ↓	“Move rule up/down” Move the filter rule upwards/downwards. The order determines the priority of the rules.



Basic Configuration | User Configuration

Input Rules | Output Rules

Incoming connections, protocols and ports

Seq.	Interface	Protocol	From IP	From Port	To IP	To Port	Comment	Action
New rule	All	TCP	0.0.0.0	any	0.0.0.0	any		Accept

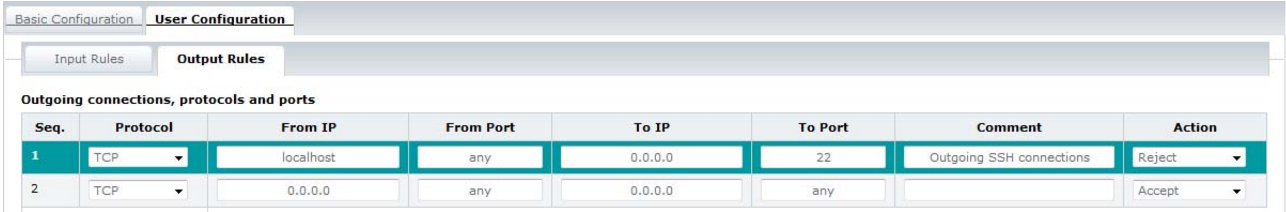
Figure 3-28 WBM: “Firewall” page, “User Configuration” tab, “Input Rules”, adding a new rule

You can define user-specific filter rules for specific ports, protocols and IP addresses for incoming (“Input Rules“) and outgoing (“Output Rules“) connections.

- For a user-specific filter rule, define the following parameters:

Table 3-6 User configurations

Column	Description
Interface (“Input Rules” only)	You can configure “Input Rules” specifically for an interface. • From the drop-down list, select the desired Ethernet interface to which the filter rule is to be applied. The “Output Rules” apply to all interfaces.
Protocol	• From the drop-down list, select the TCP, UDP, or UDPLITE protocol or all of them.
From IP From Port	• In the “From IP” field, enter an IP address, if applicable. In the “From Port” field, enter the corresponding ports, if applicable. The rule applies to connections coming in from this address. You can specify all ports, selected ports, or a value range.
To IP To Port	In the “To IP” field, enter an IP address, if applicable. In the “To Port” field, enter the corresponding ports, if applicable. The rule applies to connections going out to this address. You can specify all ports, selected ports, or a value range.
Comment	Here, enter a description of the filter rule.
Action	The options described in Section ““Action” column” on page 97 can be used as the actions for the filter rules.



Outgoing connections, protocols and ports

Seq.	Protocol	From IP	From Port	To IP	To Port	Comment	Action
1	TCP	localhost	any	0.0.0.0	22	Outgoing SSH connections	Reject
2	TCP	0.0.0.0	any	0.0.0.0	any		Accept

Figure 3-29 WBM: “Firewall” page, “User Configuration” tab, “Output Rules”, creating user-specific firewall rules

- To activate the configured settings and transmit them to the system, click on the “Apply” button.

If a configuration is already available in the system, it is overwritten during this process.

- To drop the current configuration and call the basic settings, click on the “Reset” button.

Changing a basic rule

To change a basic rule, proceed as follows:

- In the “Basic Configuration” area, set the basic rule to “Continue” in the “Action” column. This way, this rule is skipped.
- Now, create a new rule in the “User Configuration”, “Input Rules” area.
- Configure the rule for the protocol and the port of the basis rule from the “Basic Configuration” area.
  Example: You can specify incoming SSH connection requests via TCP port 22 in more detail by excluding certain IP addresses or exclusively establishing the access of some IP addresses.

Blocking the PROFICLOUD access

Access to the “PROFICLOUD” WBM page is not controlled via user authentication (see Section ““User Authentication” page” on page 82). Each user with access to WBM can also access the “PROFICLOUD” page and make settings.

To protect the PROFICLOUD configuration against unauthorized access, you can create a user without access permission to WBM.

However, if WBM access is required, you can also block the connection to PROFICLOUD via a firewall configuration.

- For this, create a new rule under “User Configuration” using the following parameters:
  - Direction: Output
  - Protocol: TCP
  - Port: 443
  - Action: Drop/Reject
- To activate the configuration, click on the “Apply” button.

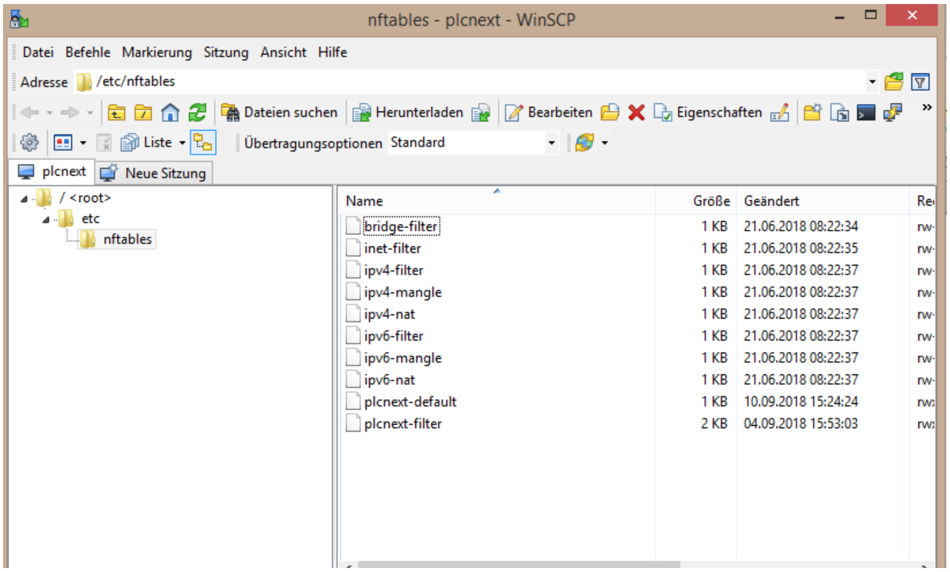
Take into consideration that due to this firewall rule, HTTPS and HMI connections are also blocked.

To permanently block the PROFICLOUD access for a user, you have to configure this user without security permissions (WBM page “User Authentication”). This way, a user cannot access the “Firewall” WBM page. The user can therefore not activate the firewall rule which blocks access to PROFICLOUD. If the user changes the configuration on the “PROFICLOUD” WBM page, this does not have any consequences as the corresponding ports are blocked by the firewall. The device cannot establish a connection to PROFICLOUD.

⇒ If you want to generally inhibit the communication with PROFICLOUD for one user, you have to configure this accordingly via the firewall and protect the firewall configuration against unauthorized access.

3.9.3.2 Activating another firewall file

In addition to the PLCnext Technology filter table, you can activate other filter tables. This might be necessary if you require certain functions that are not supported by the WBM configuration. This additional configuration is implemented via independent filter tables. You have to create the required functions via nftables commands. For this, you can edit a rule set in Linux using a text editor or load the file to the PC and change it.



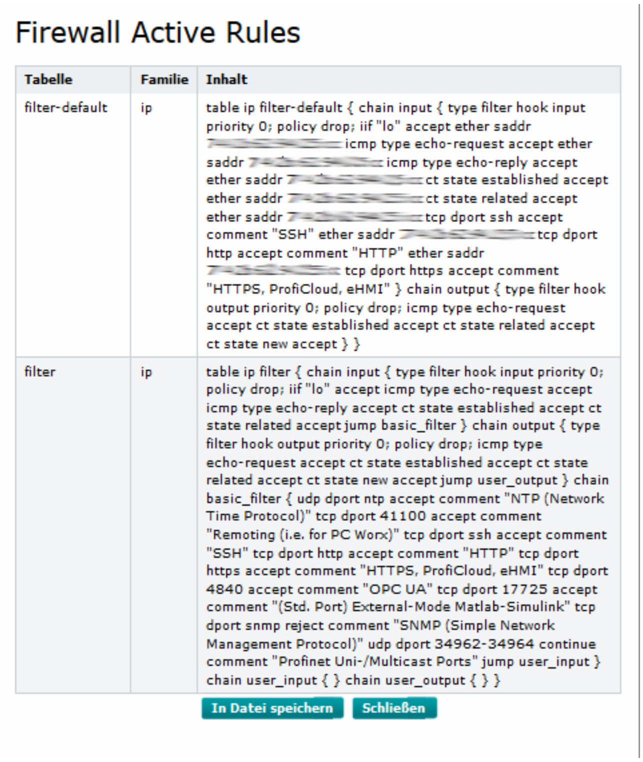
Adresse /etc/nftables

Übertragungsoptionen Standard

Name	Größe	Geändert	Re
bridge-filter	1 KB	21.06.2018 08:22:34	rw-
inet-filter	1 KB	21.06.2018 08:22:35	rw-
ipv4-filter	1 KB	21.06.2018 08:22:37	rw-
ipv4-mangle	1 KB	21.06.2018 08:22:37	rw-
ipv4-nat	1 KB	21.06.2018 08:22:37	rw-
ipv6-filter	1 KB	21.06.2018 08:22:37	rw-
ipv6-mangle	1 KB	21.06.2018 08:22:37	rw-
ipv6-nat	1 KB	21.06.2018 08:22:37	rw-
plcnext-default	1 KB	10.09.2018 15:24:24	rwx
plcnext-filter	2 KB	04.09.2018 15:53:03	rwx

Figure 3-30 Example: nftables filter files

In WBM, clicking on the “Show Rules” button in the “System Status” area, all activated filter tables are displayed.



Firewall Active Rules

Tabelle	Familie	Inhalt
filter-default	ip	table ip filter-default { chain input { type filter hook input priority 0; policy drop; iif "lo" accept ether saddr [redacted] icmp type echo-request accept ether saddr [redacted] icmp type echo-reply accept ether saddr [redacted] ct state established accept ether saddr [redacted] ct state related accept ether saddr [redacted] tcp dport ssh accept comment "SSH" ether saddr [redacted] tcp dport http accept comment "HTTP" ether saddr [redacted] tcp dport https accept comment "HTTPS, ProfiCloud, eHMI" } chain output { type filter hook output priority 0; policy drop; icmp type echo-request accept ct state established accept ct state related accept ct state new accept } }
filter	ip	table ip filter { chain input { type filter hook input priority 0; policy drop; iif "lo" accept icmp type echo-request accept icmp type echo-reply accept ct state established accept ct state related accept jump basic_filter } chain output { type filter hook output priority 0; policy drop; icmp type echo-request accept ct state established accept ct state related accept ct state new accept jump user_output } chain basic_filter { udp dport ntp accept comment "NTP (Network Time Protocol)" tcp dport 41100 accept comment "Remoting (i.e. for PC Worx)" tcp dport ssh accept comment "SSH" tcp dport http accept comment "HTTP" tcp dport https accept comment "HTTPS, ProfiCloud, eHMI" tcp dport 4840 accept comment "OPC UA" tcp dport 17725 accept comment "(Std. Port) External-Mode Matlab-Simulink" tcp dport snmp reject comment "SNMP (Simple Network Management Protocol)" udp dport 34962-34964 continue comment "Profinet Uni-/Multicast Ports" jump user_input } chain user_input { } chain user_output { } }

Figure 3-31 WBM: “Firewall” page, “Show Rules”, several active filter tables

If an additional filter table is active, this is displayed as a warning message in the “System Message” area. The warning contains the designations of all additionally loaded filter tables.

Configuring the firewall with an additional filter table

- First, empty the active firmware configuration. To do so, enter the following command in the shell:
  `root# nft flush ruleset`
- Create another independent filter table using command `nft add table <family> <tablename>`.
  Example: `root# nft add table ip loadfilter`.
- Add an “Input Chain” of type “filter” and “hook input” to the created table. Use the following command:
  `nft add chain [<family>] <table> <name> { type <type> hook <hook> [device <device>]  priority <priority> \; }`
  Example: `root# nft add chain ip loadfilter input_limiter { type filter hook input priority 0 \; }`
- Limit the network load:
  - Limit the number of packets and indicate the parameters (icmp, tcp, udp, udplite, ip) using the following command.
    Example: `root# nft add rule loadfilter input_limiter icmp type echo-request limit rate 10/second accept`

  - Limit the data rate (bytes/second, mbytes/second, mbytes/minute).
    Example: `root# nft add rule loadfilter input_limiter limit rate 10 mbytes/second accept` or
    `root# nft add rule loadfilter input_limiter limit rate over 10 mbytes/second drop`
- When adding a rule, select the Ethernet interface to which the rule is to be applied (`iif <network interface>`).
  Example: `root# nft add rule loadfilter input_limiter iif eth0 icmp type echo-request limit rate over 100bytes/minute drop`
- To count packets or display the throughput of bytes, use the following commands:
  - For all incoming packets:
    `nft add rule <table> <chain> counter`
    Example: `root# nft add rule loadfilter input_limiter counter`
    Note:
    For “Accept” action: Accepted packets are counted.
    For “Drop” action: Blocked (dropped) packets are counted.
  - For a specific protocol:
    `nft add rule <table> <chain> counter ip protocol <protocol>`
    Example: `root# nft add rule loadfilter`
- To drop or accept the data traffic for a specific protocol, use the following commands:
  `nft add rule <table> <chain> ip protocol <protocol> accept/drop`
  Example: `root# nft add rule loadfilter input_limiter ip protocol udp accept` or
  `root# nft add rule loadfilter input_limiter ip protocol udplite drop`

Example of a simple filter file:

```
table ip loadfilter {
    chain input_limiter {
        type filter hook input priority 0; policy drop
        icmp type echo-request accept
        tcp dport ssh accept  comment "ssh wegen Fernzugriff zulassen"
    }
    chain output_limiter {
        type filter hook output priority 0; policy drop;
        icmp type echo-request accept
    }
}
```

General nftables commands

Table 3-7 General nftables commands

Command	Description
`root# nft list tables`	List all active filter tables
`nft delete/flush/list table <table>`	Delete/empty/list a filter table Example: `root# nft flush table loadfilter`
`root# nft list table <table> --handle`	Delete a rule You can delete a rule by means of its handle number. First, use this command to list the handle numbers of the individual rules. Example: `root# nft list table loadfilter -handle`
`nft delete rule [<family>] <table> <chain> [handle <handle>]`	Then, you can delete the desired rule by means of its handle number. Example: `root# nft delete rule filter input handle 90`

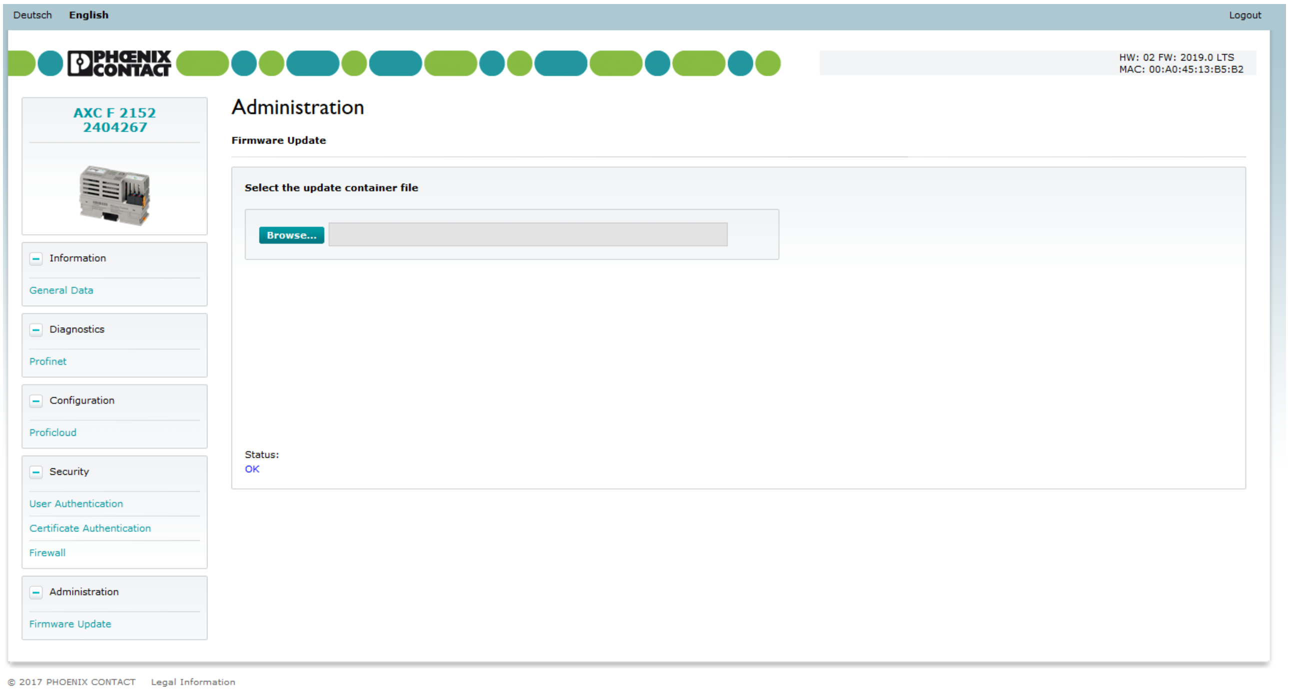
Table 3-7 General nftables commands

Command	Description
`nft -f <filter-file>`	Load the content of a filter table from a file Example: `root# nft -f loadfilter.rules`
`nft list table <table> > <file>`	Save the content of a filter table to a file Example: `root# nft list table loadfilter > loadfilter.rules`

3.10 “Administration” area

3.10.1 “Firmware Update” page

On this page, you can implement a firmware update of the controller in the admin user role.



HW: 02 FW: 2019.0 LTS
MAC: 00:A0:45:13:B5:B2

Administration

Firmware Update

Select the update container file

Status:
OK

© 2017 PHOENIX CONTACT

Figure 3-32 WBM: “Firmware Update” page

To update the controller firmware, proceed as follows:

- Download the *.zip firmware file at phoenixcontact.net/product/1051328.
- Unpack the *.zip firmware file.
- Run the *.exe setup file.
- Follow the instructions of the installation wizard.

During installation, the update file (*.raucb) and the files with device-specific information will be copied to the selected destination directory.

Noting the installed firmware version

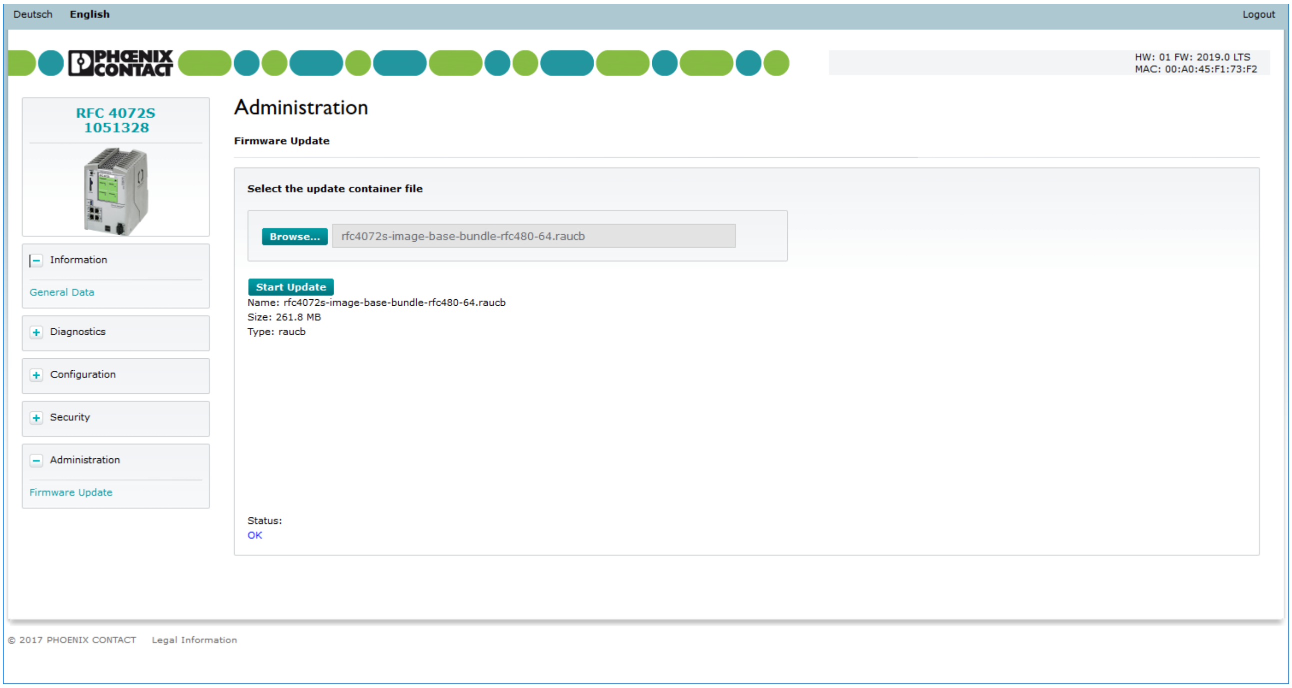
The firmware version currently installed on the controller is displayed on the “General Data” page (see Figure 3-5).

- Before updating the firmware, note the installed firmware version to be able to check later if the firmware update was successful.

Selecting a firmware file

- Click on “Browse...”
- In the file explorer that opens, select the *.raucb firmware file to be installed.
- Click on the “Open” button.

The firmware file to be installed is now displayed in WBM (see Figure 3-33).



HW: 01 FW: 2019.0 LTS
MAC: 00:A0:45:F1:73:F2

Administration

Firmware Update

Select the update container file

rfc4072s-image-base-bundle-rfc480-64.raucb

Name: rfc4072s-image-base-bundle-rfc480-64.raucb
Size: 261.8 MB
Type: raucb

Status:
OK

Figure 3-33 WBM: Firmware file to be installed in WBM

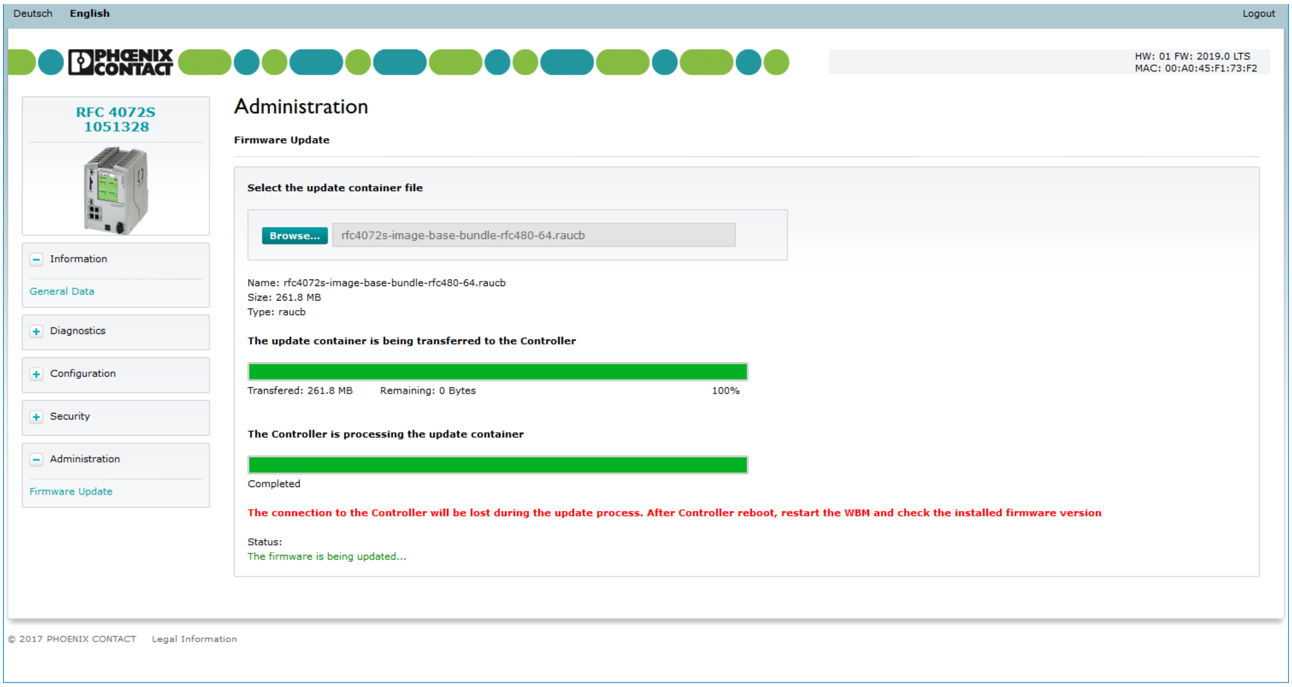
Starting the firmware update

- To start the firmware update, click on the “Start Update” button.

The update file is transferred to the controller.

Once the file was transferred successfully, the firmware update is started.

The status of the file transfer and the status of the update process are displayed in WBM as progress bars.



Administration

Firmware Update

Select the update container file

rfc4072s-image-base-bundle-rfc480-64.raucb

Name: rfc4072s-image-base-bundle-rfc480-64.raucb
Size: 261.8 MB
Type: raucb

The update container is being transferred to the Controller

Transfered: 261.8 MB   Remaining: 0 Bytes   100%

The Controller is processing the update container

Completed

The connection to the Controller will be lost during the update process. After Controller reboot, restart the WBM and check the installed firmware version

Status:
The firmware is being updated...

Figure 3-34 WBM: Status of the file transfer and the update process

After the firmware update, the controller is restarted automatically.

Please note:
The connection to the controller is interrupted during the firmware update. After the firmware update, the WBM pages opened in the browser are no longer up to date.
Once the controller is fully initialized after the restart, you have to log into WBM again to update the WBM pages.
The updated firmware version is displayed on the top right of every WBM page.

Checking the firmware version

- After the controller was restarted, log into WBM.
- Open the “General Data” page.
- Check if the correct firmware version is displayed.

If the previously installed firmware version is displayed after the firmware update (see Section “Noting the installed firmware version”), an error occurred during the firmware update.

- In this case, repeat the firmware update.

4 Transferring variable values to the PROFICLOUD

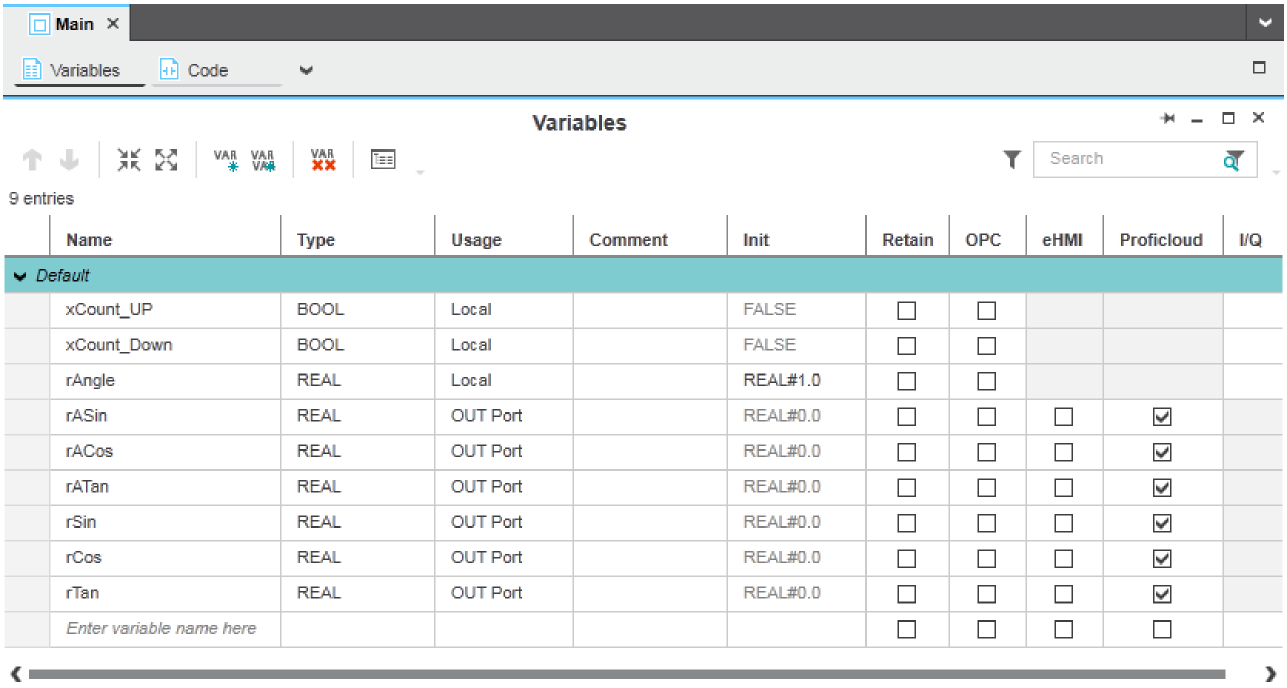
In PLCnext Engineer, you can define variables whose values are to be transferred as a metric to PROFICLOUD. The variable values are stored in PROFICLOUD. The metrics can be represented graphically using the open platform Grafana.

4.1 Creating variables as OUT ports

Variables that are to be transferred from a PLCnext Engineer project to PROFICLOUD have to be created as OUT ports in PLCnext Engineer.

To create a variable in PLCnext Engineer as an OUT port, proceed as follows:

- In the “COMPONENTS” area, click on “Programming”, “Local” and then on “Programs”.
- Double-click on the desired POE from which variables are to be transferred to PROFICLOUD.
- Select the “Variables” editor.
- Enter the variable name and data type.
- In the “Usage” column, select “OUT Port”.
- Enable the check box in the “Proficloud” column.



	Name	Type	Usage	Comment	Init	Retain	OPC	eHMI	Proficloud	I/Q
Default										
	xCount_UP	BOOL	Local		FALSE	□	□			
	xCount_Down	BOOL	Local		FALSE	□	□			
	rAngle	REAL	Local		REAL#1.0	□	□			
	rASin	REAL	OUT Port		REAL#0.0	□	□	□	☑	
	rACos	REAL	OUT Port		REAL#0.0	□	□	□	☑	
	rATan	REAL	OUT Port		REAL#0.0	□	□	□	☑	
	rSin	REAL	OUT Port		REAL#0.0	□	□	□	☑	
	rCos	REAL	OUT Port		REAL#0.0	□	□	□	☑	
	rTan	REAL	OUT Port		REAL#0.0	□	□	□	☑	
	Enter variable name here					□	□	□	□	

Figure 4-1 Creating variables as OUT ports

Note that the “Proficloud” check box has to be enabled for at least one OUT port so that the controller can send data to PROFICLOUD.

If you program your program in C++ using the PLCnext Technology Command Line Interface, you have to define the OUT ports of which the values are to be transferred to PROFICLOUD using attributes `Output` and `Proficloud` (`//#attributes (Output|Proficloud)`), see Section “Creating IN and OUT ports” on page 120.

Time stamp

The controller sets the time stamp for the data that is transferred to PROFICLOUD. In case of an interrupted Internet connection, the time stamp is not affected. When the Internet connection is interrupted, data is buffered in the controller. Make sure that the controller time is set to UTC (for additional information, please refer to Section “System time” on page 65).

4.2 Preparing the controller for PROFICLOUD

Before you can transfer metrics to PROFICLOUD, you have to register the controller in PROFICLOUD and activate the TSD PROFICLOUD service in the WBM of the controller. To do this, proceed as follows:

Activating TSD in WBM

To establish a connection to WBM, proceed as follows:

- Open the web browser on your PC.
- In the address field, enter URL “http://IP address of the controller” (example: “http://192.168.1.10”).

If there is an HMI application on the controller, entering URL “http://IP address of the controller” calls the application. To call WBM in this case, enter URL “http://IP address of the controller/wbm”.

For information on WBM (web-based management) of your controller, please refer to Section “Web-based management (WBM)” on page 73.

- Open the “PROFICLOUD” page in the “Configuration” area.
- Enable the check boxes for “Enable Proficloud Service” and “Enable Time-Series Data (TSD) Service”.

On the “PROFICLOUD” WBM page, you can check the connection status to PROFICLOUD in the “Proficloud Connection State” line of the table.

4.3 Configuring PROFICLOUD

Connection to the PROFICLOUD administration

- Open the web browser on your PC.
- In the address line, enter URL “https://www.proficloud.net”.

Logging in

- Enter your user name and password.
- Click the “Sign In” button to sign into PROFICLOUD.

Adding a controller

- To add the controller as a PROFICLOUD device, select the “TSD Device Manager” PROFICLOUD solution.

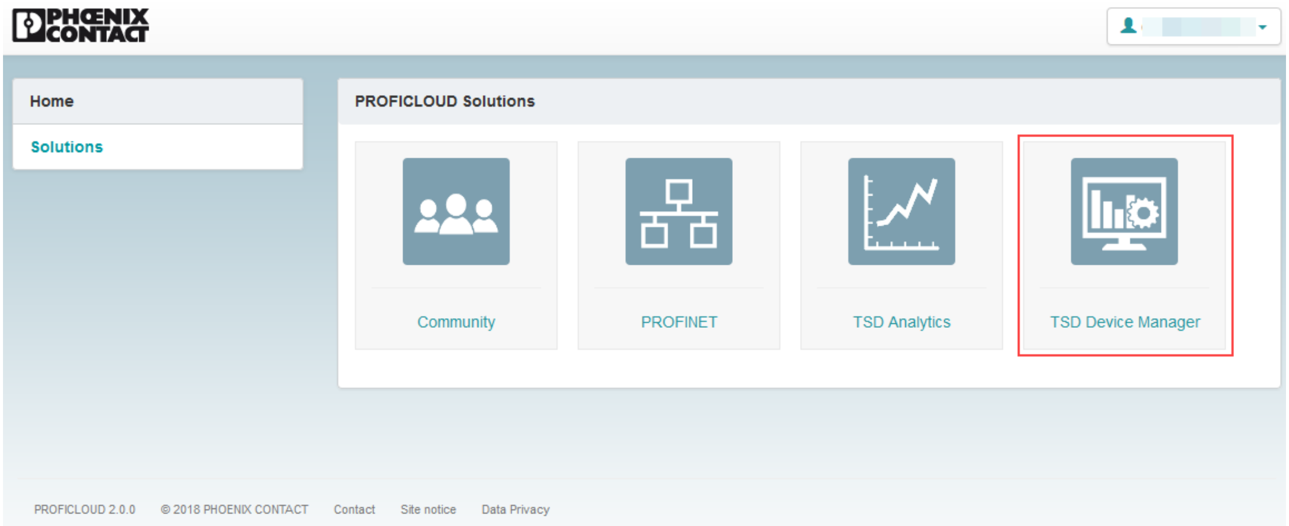


Figure 4-2 Selecting the “TSD Device Manager” PROFICLOUD solution

The “Appliances” page opens.

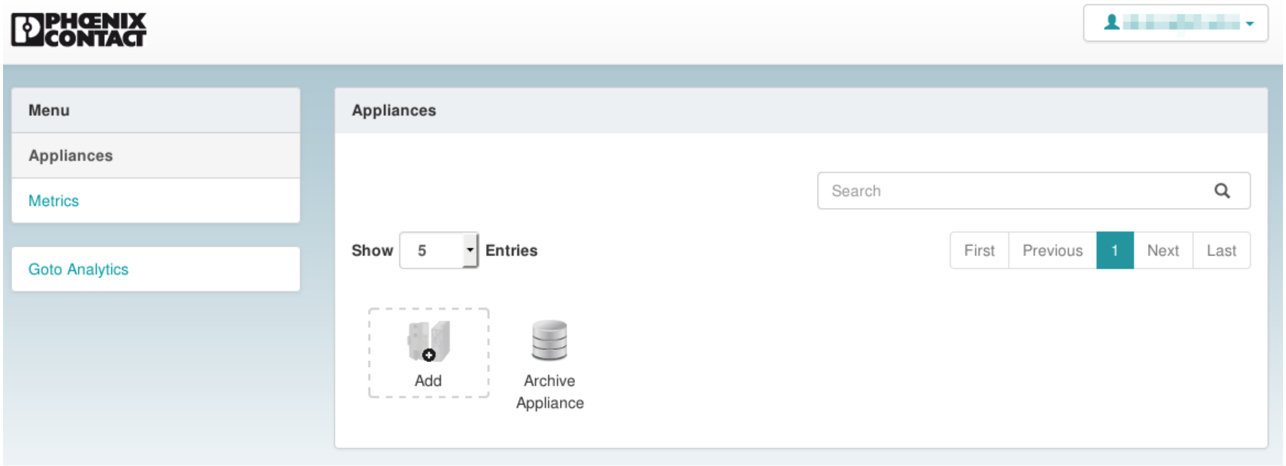
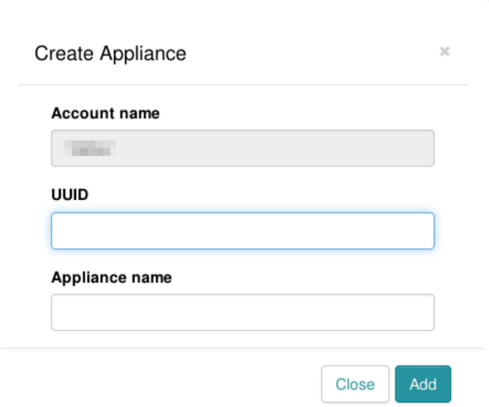


Figure 4-3 “Appliances” page

Registering PLCNEXT TECHNOLOGY

- Click on the “Add” button.
- The “Create Appliance” dialog opens.



Create Appliance

Account name

[redacted]

UUID

Appliance name

Figure 4-4 “Create Appliance” dialog

- Enter the UUID of the controller in the “UUID” input field.

The UUID of the AXC F 2152 is printed on the side of the device, for example. You will also find the UUID in WBM in the “Configuration” area on the “PROFICLOUD” page.

- Enter a unique name for the controller in the “Appliance name” input field.
- Click on the “Add” button to save your entries.

4.4 Displaying an overview of the PROFICLOUD device metrics

When the controller is switched on, the metrics are automatically transferred to PROFICLOUD.

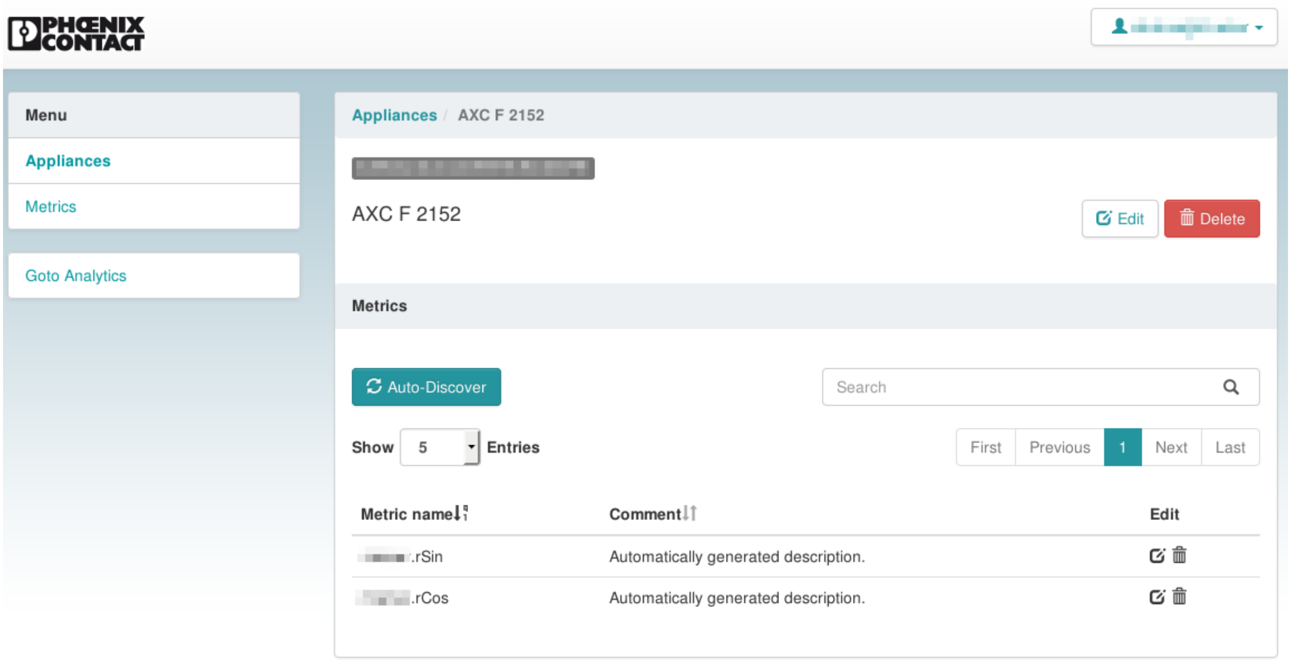
To display an overview of all of the metrics of a PROFICLOUD device, proceed as follows:

- Select the “TSD Device Manager” PROFICLOUD solution.

The “Appliances” page opens.

- On the “Appliances” page, click on the controller of which the metrics are to be displayed.

The “Appliances/device name” page opens.



[redacted]

AXC F 2152

Show 5 Entries

Metric name	Comment	Edit
[redacted].rSin	Automatically generated description.	
[redacted].rCos	Automatically generated description.	

Figure 4-5 “Appliances/device name” page

The metrics received are shown in the “Metrics” area.

4.5 Displaying the metrics graphically in Grafana

The metrics can be represented graphically using the open platform Grafana.

To display a metric graphically in Grafana, proceed as follows:

Establishing a connection to Grafana

- Select the “TSD Device Manager” PROFICLOUD solution.
- Select the “Go to Analytics” entry in the menu.

The Grafana homepage opens.

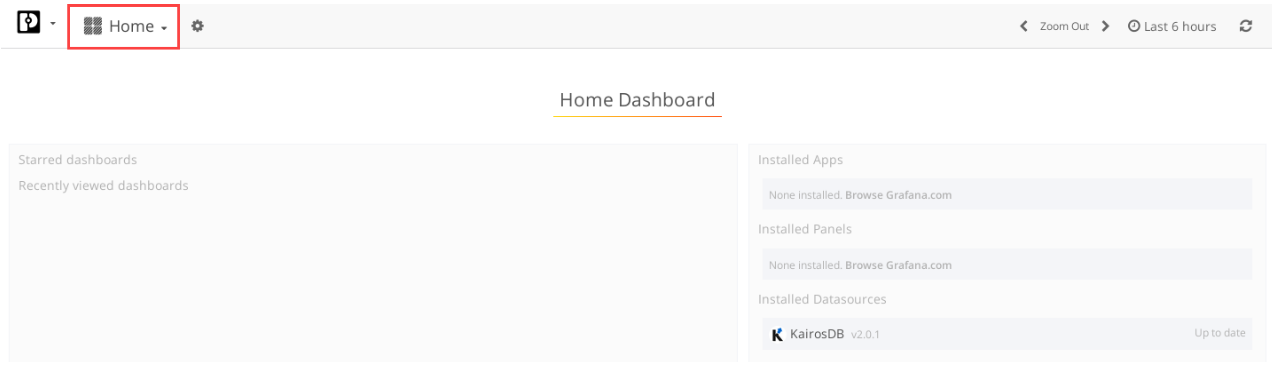


Figure 4-6 Grafana: Homepage

Creating a new dashboard

- Click on the “Home” button.

The “Home” page opens.

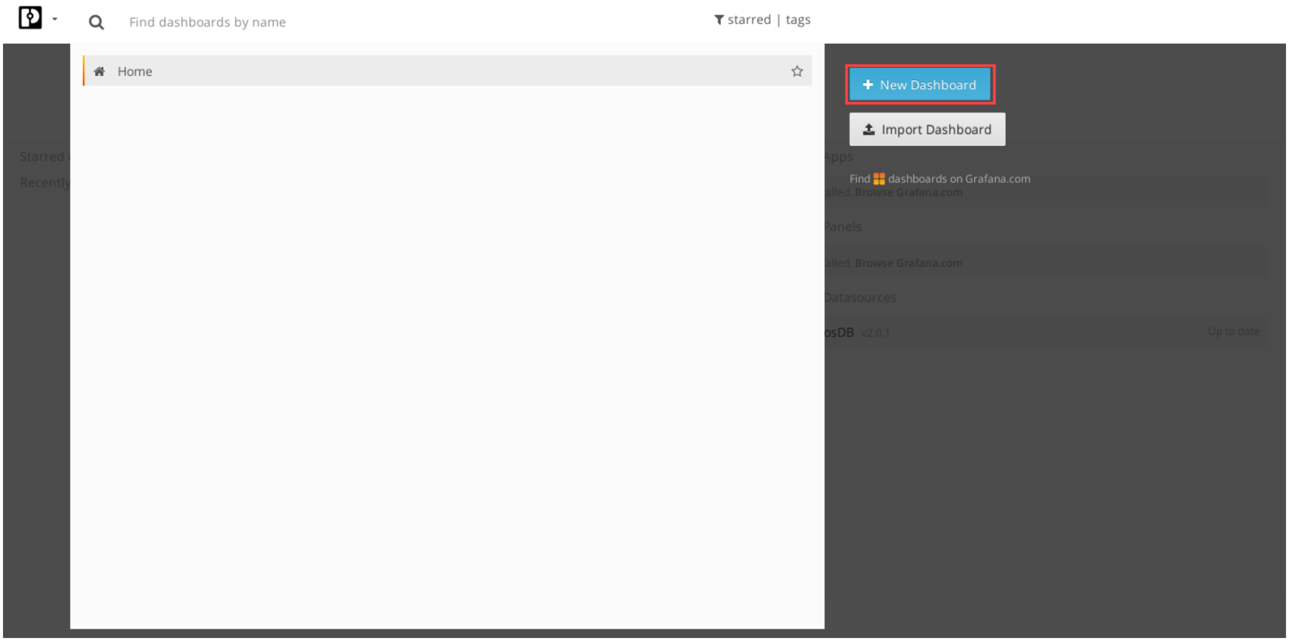


Figure 4-7 Grafana: “Home” page

- Click on the “New Dashboard” button to create a new dashboard.

The “New dashboard” page opens.

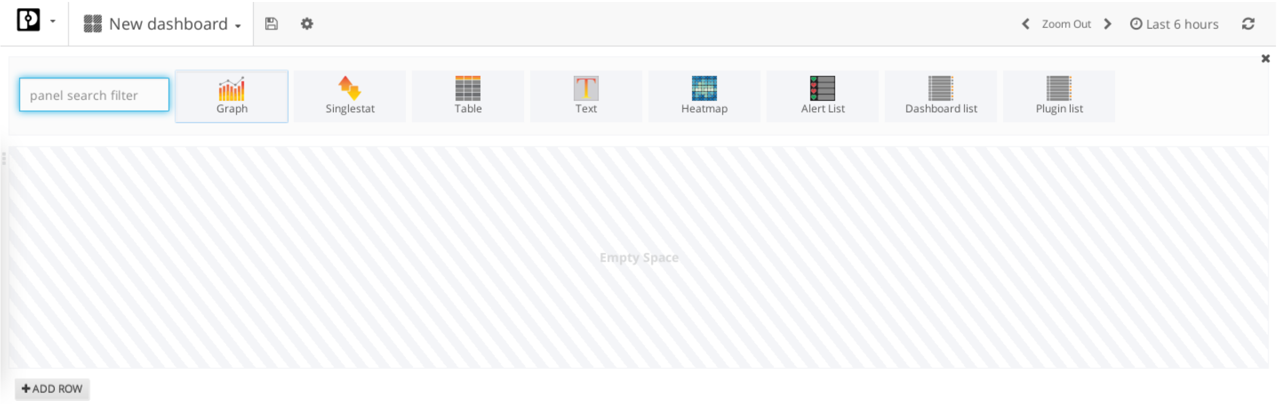


Figure 4-8 Grafana: “New dashboard” page

Selecting the display type

- Click on one of the buttons to select a display type (e.g., graph, etc.).

An example display opens for the type of graphical display selected (see Figure 4-9).

Selecting a metric

- To be able to select the metric to be displayed, click on “Panel Title”.
- Click on the “Edit” button.

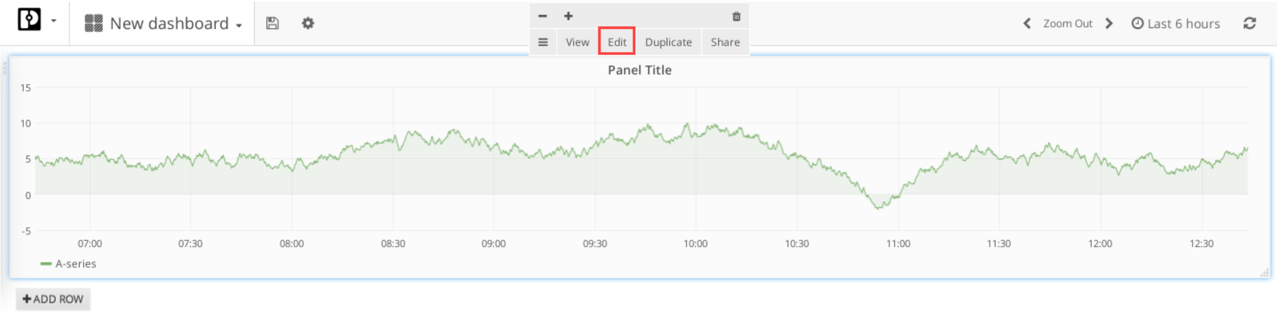
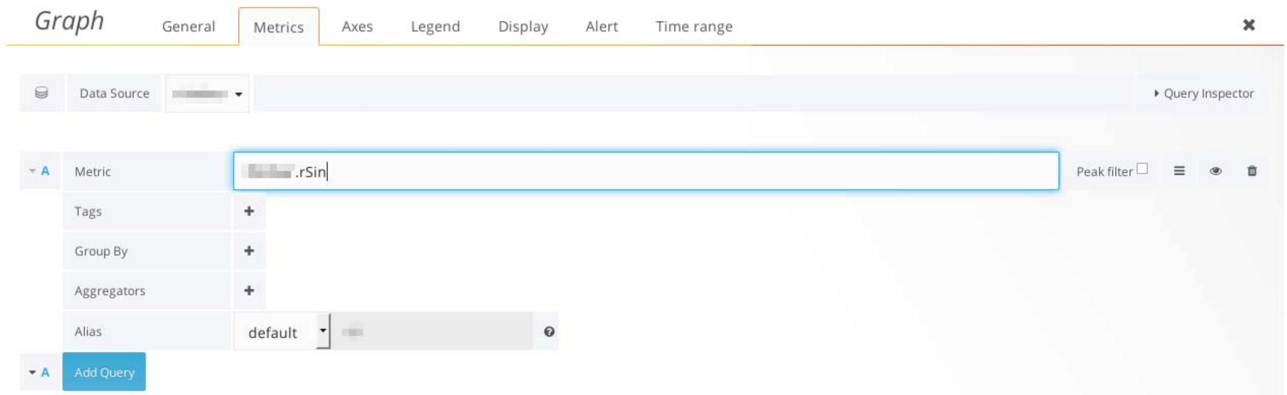


Figure 4-9 Grafana: Example graph; editing Panel Title

An area in which you are able to edit the details of the selected graphical display opens below the example display.

- Switch to the “Metrics” tab.
- Select the metric to be displayed.



Metric: .rSin

Alias: default

Figure 4-10 Grafana: Selecting the metric to be displayed

- Close the bottom area by clicking on the “X” button.

The selected metric is now displayed graphically.

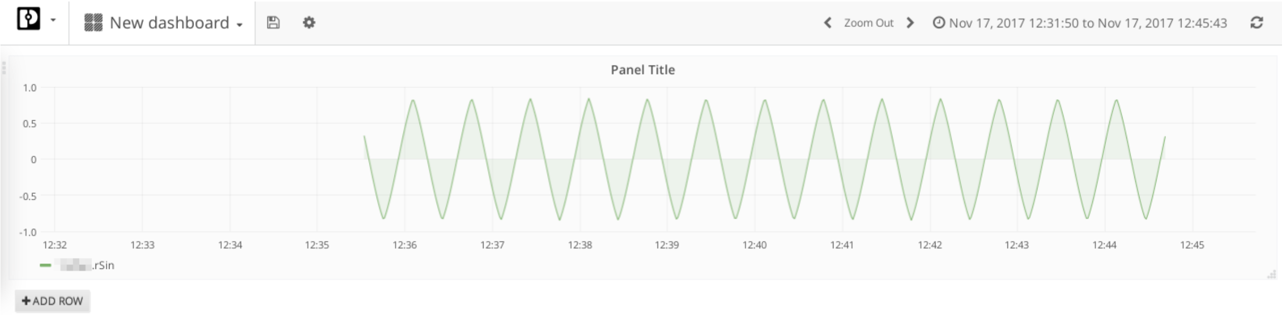


Figure 4-11 Grafana: Graphical display of the selected metric

5 Structure of a C++ program

ℹ If you want to consult the PLCnext Technology SDK C++ header files in parallel while reading this section, first install the necessary tools as described in "Creating programs with C++" on page 151.

ℹ When naming PLCnext Technology components, observe the naming conventions in Section "PLCnext Technology naming conventions" on page 193.

With PLCnext Technology, programs that were created in different programming languages can be used together. For instantiation and calling by the PLCnext Technology firmware, the programs have a uniform basis. This basis applies to all programming languages. PLCnext Technology follows an object-oriented approach. The following base classes are relevant for creating a C++ program for the PLCnext Technology platform.

Library	LibraryBase class	The LibraryBase class is the smallest unit that can be downloaded. It represents an *.so file (shared object). One "Library" can instantiate one or more "Components".
Component	ComponentBase class	The ComponentBase class is a collection of functions (programs) within the PLCnext Technology platform. One "component" can instantiate one or more "programs". The "Program" instances can interact via their shared "Component" instance.
Program	ProgramBase class	With PLCnext Technology, instances of the ProgramBase class can be performed in real time. Here, the IN and OUT ports are published.

The base classes are stored in the Phoenix Contact SDK. Derive from these base classes to create your application.

Some sample applications programmed in C++ can be found at https://github.com/plcnext.

5.1 "ILibrary" and "LibraryBase"

ℹ Using the PLCnCLI (see Section 6.1, "PLCnCLI (PLCnext Command Line Interface)") or the Eclipse® add-in creates the meta configuration files (libmeta, compmeta, progmeta) required for PLCnext Engineer as well as the following functions with a functional implementation during compiling. If you have special requirements that go beyond this, the following descriptions will help you understand the functions.

In a shared object (*.so), there can be exactly one library class (singleton pattern). The PLCnext Technology firmware initializes the *.so after it has been loaded by calling the following function:

```
extern "C" ARP_CXX_SYMBOL_EXPORT ILibrary& ArpDynamicLibraryMain(AppDomain& appDomain);
```

This function is implemented in such a way that the library class is created and returned as an interface. The library object implements the “ILibrary” interface by means of derivation from the “LibraryBase” class. “ILibrary” is defined in header Arp/System/Acf/ILibrary.hpp, and “LibraryBase” in Arp/System/Acf/LibraryBase.hpp. The “LibraryBase” class particularly implements a “ComponentFactory” with the “IComponentFactory” interface.

The PLCnext Technology firmware can thus create instances of the components when loading the PLC program. In order for the PLCnext Technology firmware to be able to determine the necessary information (names of components and program types, program ports), the library class also has to implement the “IMetaLibrary” interface (`#include “Arp/Plc/Commons/Meta/IMetaLibrary.hpp”`). This interface then requires implementation of the “ITypeInfoProvider” interface (`#include “Arp/Plc/Commons/Meta/TypeInfoProvider.hpp”`), whereby the firmware learns the names of the components and program types as well as their IN and OUT ports.

The “MetaLibraryBase” class (`#include “Arp/Plc/Commons/Meta/MetaLibraryBase.hpp”`) therefore expands the “LibraryBase” class by these interfaces.

The component instances the PLCnext Technology firmware creates is defined in the *.acf.config and *.plm.config files.

“AppDomain” and “IApplication

If you use the Eclipse® add-in, the required header files are included automatically.

– “AppDomain”: Arp/System/Core/AppDomain.hpp

– “IApplication”: Arp/System/Acf/IApplication.hpp

The PLCnext Technology firmware is distributed among several operating system processes.

“AppDomain”

An “AppDomain” represents such an operating system process. If a library is to be used in several processes, singleton implementation is performed for each process. The “AppDomain” class is used for this. Since the PLCnext Technology firmware uses the same mechanism for instantiation, initialization and administration of user components as it does for core components, the “AppDomain” is also used here.

“IApplication”

The operating system process is represented by the “IApplication” interface.

5.2 “IComponent” and “ComponentBase”

Using the PLCnCLI (see Section 6.1, “PLCnCLI (PLCnext Command Line Interface)”) or the Eclipse® add-in creates the meta configuration files (libmeta, compmeta, progmeta) required for PLCnext Engineer as well as the following functions with a functional implementation during compiling. If you have special requirements that go beyond this, the following descriptions will help you understand the functions.

The basic integration of a component into PLCnext Technology is implemented by means of derivation from the “ComponentBase” class or via the “IComponent” interface. There are specialized components for various purposes. These specializations are performed by implementing additional interfaces, which are described in the following subsections. To be able to use the classes, you have to include (`#include`) the corresponding header files (.hpp).

– “IComponent”: Arp/System/Acf/IComponent.hpp

– “ComponentBase”: Arp/System/Acf/ComponentBase.hpp

The following “IComponent” operations are called by the ACF or PLM for each component:

- `void Initialize(void)`
- `void SubscribeServices(void)`
- `void LoadSettings(const String& settingsPath)`
- `void SetupSettings(void)`
- `void PublishServices(void)`

In the second phase, the project configuration is loaded and set up. If an exception occurs here, the project configuration is unloaded again and the controller starts with an empty configuration. The ACF or PLM calls the following “IComponent” operations:

- `void LoadConfig(void)`
- `void SetupConfig(void)`

Initialize

The component instance is initialized via the functions specified below. These have to be implemented in the component class. For each instantiated component, the PLCnext Technology firmware calls the following function first:

```
virtual void Initialize(void);
```

SubscribeServices

Resources that have been allocated and initialized for the component in the `Initialize()` function have to be enabled in the `Dispose()` function. Thereafter, the firmware calls the following function for each instantiated component.

```
virtual void SubscribeServices(void);
```

Here, a component can obtain RSC services that have already been registered.

LoadSettings

Subsequently, the `LoadSettings()` function is called. The path to the settings (`settingsPath`) can be specified for the respective component instance in the *.acf.config configuration file. The format and content of this configuration file have to be specified by the respective component (type). The PLCnext Technology firmware does not make any assumptions regarding these.

```
virtual void LoadSettings(const String& settingsPath);
```

SetupSettings

Once the `settingsPath` has been specified, the settings can be applied. The following function is used for this:

```
virtual void SetupSettings(void);
```

PublishServices

The PLM does not call the `PublishServices` function. Creating and registering RSC services is reserved for the core components.

```
virtual void PublishServices(void);
```

LoadConfig/SetupConfig

When the project is subsequently loaded, the following functions are called:

```
virtual void LoadConfig(void);
virtual void SetupConfig(void);
```

ResetConfig

The configuration of the components is reset with the following function:

```
virtual void ResetConfig(void);
```

The interface is identical for the user program and the internal user component. It is simply called at different times.

- The PLC manager manages components that make user programs available. The PLC manager configures them in a file referenced by “/opt/plcnext/projects/Default/Plc/Plm/Plm.acf.config”.
- Internal user components are managed by the ACF (Application Control Framework) and configured respectively in a file referenced by “/opt/plcnext/Default/Default.acf.config”. The ACF generates these components when booting the firmware.

Dispose

A component is stopped after calling the following function:

```
virtual void Dispose(void);
```

5.2.1 “IProgramComponent” and “IProgramProvider”

A component that can instantiate user programs implements the “IProgramComponent” and “IMetaComponent” interfaces by means of derivation from the “ProgramComponentBase” class. The PLCnext Technology firmware requires these interfaces to instantiate programs and receive information on their ports.

The PLCnCLI generates the necessary code. For this, a code is generated for each component, which then implements the “IProgramProvider” interface. If the `//#program` directive is prefixed to the header of a program definition, the PLCnCLI knows which component can instantiate which program. The next directive `//#component()` is used to name the corresponding component.

```
//#program
//#component(SampleComponent)
class SampleProgram : public ProgramBase,
```

The corresponding header files (.hpp) are included (`#include`) so that the classes can be used.

- “IProgramComponent”: Arp/Plc/Esm/IProgramComponent.hpp
- “IProgramProvider”: Arp/Plc/Esm/IProgramProvider.hpp

The component implements the “IProgramComponent” interface by means of derivation from the “ProgramComponentBase” class. In addition, the component creates a private member variable that is passed to the “ProgramComponentBase” class in the constructor of the component. This member variable implements the “IProgramProvider” interface by means of derivation from the “IProgramProviderBase” class.

The “ProgramProvider” makes the following function available for the instantiation of programs:

```
IProgram::Ptr CreateProgramInternal(const String& programName, const String& programType)
```

The PLCnext Technology firmware calls this function when loading the PLC program.

In the process, the following parameters are passed on in the way they are defined in the *.esm.config files:

Attribute	Description
`programName`	Instance name of the program The instance name is configured in the PLCnext Engineer task editor or manually via the *.esm.config file.
`programType`	Class name of the program For each component, the *.compmeta and *.progmeta files describe the programs (type) a component (type) can generate.

5.2.2 “IProgram” and “ProgramBase”

An instantiated user program implements the “IProgram” interface. As a result, a constructor to which the instance name is passed on and the `Execute()` function that is called by the ESM task during each pass are available. The PLCnCLI creates such a user program class that inherits the “IProgram” interface from the “ProgramBase” base class by means of derivation. Furthermore, the PLCnCLI enables reporting the IN and OUT ports of the program to the GDS.

Creating IN and OUT ports

The header file of your program (e.g., MyProgram.hpp) must contain the port definition. The ports are created as `public` according to the following pattern.

For example:

```
public:
   //#port
   //#attributes(Input| ...)
   //#name(...)
   int myInPort;
```

A port is specified by an added attribute. The following attributes are available:

Table 5-1 Attributes for port definition

Attribute	Description
`Input`	The variable is defined as IN port.
`Output`	The variable is defined as OUT port.
`Retain`	The variable name is retained in case of a warm and hot restart (only initialized in case of a cold restart).
`Opc`	The variable is visible for OPC UA.
`Ehmi`	The variable is visible for the PLCnext Engineer HMI.
`Proficloud`	The variable is visible for PROFICLOUD (for OUT ports only).

Multiple attributes are separated by the “|” separator.
E.g.: `//#attributes (Input|Opc|Retain)`

From the attributes, the PLCnCLI generates the meta data. If a variable does not have an attribute, is can only be used internally by the program.

Provide variables you want to connect via the GDS with an `Input` or `Output` attribute, depending on the data direction. You can use all other attributes to control the initialization behavior or enable visibility in OPC UA or a PLCnext Engineer visualization system. You can also use these additional attributes without the `Input` or `Output` attributes. This way, the variable is visible but cannot be connected via the GDS. The `Proficloud` attribute only works in conjunction with the `Output` attribute.

The variable name is also used as the port name. If you want to name the port differently in the GDS, you can use the `//#name()` directive to enter a different port name.

To be able to use the classes, you have to include (`#include`) the corresponding header files (.hpp).

– “IProgram”: Arp/Plc/Esm/IProgram.hpp
– “ProgramBase”: Arp/Plc/Esm/ProgramBase.hpp

5.2.3 “IControllerComponent”

A component created with the PLCnCLI is automatically derived from the “IControllerInterface” interface. An internal user component may need individual lower-priority threads to perform longer tasks outside of the ESM task. To this end, the component can implement the “IControllerComponent” interface in addition to “IComponent”.

The “IControllerComponent” interface defines the following two functions:

```
void Start (void);
void Stop (void);
```

If the component is managed by the PLC manager (“User Program”):

- During a cold, warm or hot restart of the PLC application, the `Start()` function is called after all program objects of the component have been generated. This is an ideal time to start individual, lower-priority threads the programs delegate tasks to.
- The `Stop()` function is called when the PLC application is stopped, before the programs are destroyed. At this point, the created threads can be destroyed again.

If the component is managed by the ACF (Automation Component Framework) (“Internal User Component”):

- When the firmware is started (`boot` or `/etc/init.d/plcnext start`), the `Start()` function is called after all components have been generated. This is an ideal time to start threads which perform the component tasks.
- When the firmware is stopped (`/etc/init.d/plcnext stop`), the `Stop()` function is called before the components are destroyed.

To use this class, you have to include (`#include`) the corresponding header file (.hpp).

- “IProgram”: Arp/System/Acf/IControllerComponent.hpp

5.3 Several component types in the same library

Using the PLCnCLI (see Section 6.1, “PLCnCLI (PLCnext Command Line Interface)”) or the Eclipse® add-in creates the meta configuration files (libmeta, compmeta, progmeta) required for PLCnext Engineer as well as the following functions with a functional implementation during compiling. If you have special requirements that go beyond this, the following descriptions will help you understand the functions.

Initially, the Eclipse® add-in creates one component type in a library. If several component types are to be instantiated in the same library, each component type has to be added to the factory. To this end, every component type is introduced to the factory by calling the `AddFactoryMethod()` function in the constructor of the library object. The PLCnCLI generates the required code if the `//#component` directive is prefixed in the header of the class definition:

```
//#component
class SampleComponent : public ComponentBase,
```

There are two ways to instantiate components:

- “Internal User Components”, managed by the ACF, are instantiated by the Default.acf.config (projects/Default/Default.acf.config) file.

– “User Programs”, managed by the PLM, are instantiated by the Plm.config (/opt/plcnext/projects/Default/Plc/Plm/Plm.config) file.
  When used as PLCnext Engineer library, this is performed automatically by PLCnext Engineer, but it can also be done manually (see Section “Task configuration via configuration files” on page 23).

5.4 PLM (Program Library Manager)

The PLM (Program Library Manager) is part of the PLC manager (see “PLC manager” on page 16). It loads and unloads components during the runtime of the PLCnext Technology firmware. The PLM controls the entire service life of the component instance in accordance with the states of the controller and changes to these states by means of the PLCnext Engineer commands:

- Cold or warm restart
- Hot restart
- Reset
- Download (a reset is implicitly performed prior to the download, however, not during Download Changes)

5.4.1 Functions

The PLM takes on the role of creating, configuring and destroying the components that can instantiate user programs. The application components are controlled as follows:

Table 5-2 “IComponent” (PLM) functions

Calling “IComponent”	User action in PLCnext Engineer
`void Initialize()`	– Restart – Send project
`void SubscribeServices()`	– Restart – Send project
`void LoadSettings(const string & settingsPath)`	– Reboot – Send project
`void SetupSettings()`	– Restart – Send project
`void PublishServices()`	The function is only called by a firmware component, not by the PLM.
`void LoadConfig()`	– Restart – Cold restart – Warm restart – Send project

Table 5-2 “IComponent” (PLM) functions

Calling “IComponent”	User action in PLCnext Engineer
`void SetupConfig()`	– Reboot – Cold restart – Warm restart – Send project
`void ResetConfig()`	– Reboot – Cold restart – Warm restart – Send project
`void Dispose()`	– Send project

5.4.2 Configuration

PLM configuration

The PLM system is configured via a configuration file. “ConfigSettings” provides the path to the configuration file of the application libraries and components.

Configuration of application programs

The AcfConfigurationDocument format is used to configure components that can instantiate user programs. The Library, Component (name, type, library, isEnabled), and Settings (path) elements are evaluated by the PLM (see Section “Configuration files” on page 17).

5.5 ACF (Application Component Framework)

The (ACF) Application Component Framework is the foundation for the PLCnext Technology platform architecture. The ACF is a framework that enables component-based platform development and the configurative composition of the firmware for the devices. It can configuratively distribute the firmware to one or more processes. The ACF enables the configurative integration of user functions into the system. This way, you can extend PLCnext Technology devices with your own functions.

The ACF loads the various shared object files, and starts and manages the components contained therein in the desired sequence. In PLCnext Technology, components are instantiated like classes, i.e., several instances of one component type can exist. Instantiation is performed via configuration files for the ACF.

5.5.1 Libraries

ACF libraries are loaded dynamically by the ACF via configuration. They serve as the configurative extension of the system by means of platform or user functions. The ACF libraries must be available as dynamic libraries (“shared object” or *.so) and contain one or more ACF components. This enables the user to construct the firmware configuratively and add functions.

ACF libraries must be implemented in accordance with a particular template. The specified template is necessary to enable the ACF to access the code dynamically. Implementation consists of the following elements:

- Library class (derived from “LibraryBase”), implemented as a singleton. A library singleton is an instance or function that exists exactly once per library, e.g., “ComponentFactory”.

- At least one component class that implements the “IComponent” interface.
- “ComponentFactory”

See Section ““ILibrary” and “LibraryBase”” on page 116 and Section ““IComponent” and “ComponentBase”” on page 117.

Loading libraries

A library is added to the ACF configuration as shown in the following example:

ACF configuration – adding a library

```
<?xml version="1.0" encoding="utf-8"?>
<AcfConfigurationDocument
    xmlns="http://phoenixcontact.com/schema/acfconfig“
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=“http://www.phoenixcontact.com/schema/acfconfig“
    schemaVersion=“1.0“ >

    <Libraries>
        <Library name="MyLibraryACF.Library“
        binaryPath="$ARP_PROJECTS_DIR$/MyLibrary/libs/libMyLibraryACF.so“ />
    </Libraries>

</AcfConfigurationDocument>
```

The following table lists explanations of the individual attributes:

XML element	Description
`<Libraries>`	List of libraries Here, libraries are listed that are to be loaded to create components.
`<Library>`	Definition of a library
`name`	The name of the library referenced in a component configuration.
`binaryPath`	The binary path of the library that is to be loaded.

5.5.2 Components

Components are classes in the sense of object-oriented programming. They are a part of a library and provide a public interface to the library. They therefore facilitate access to libraries with coherent functionality, and can be instantiated once or several times.

ACF components enable the PLCnext Technology platform to be extended configuratively. The ACF components must be implemented in accordance with a particular template. The ACF components must register with the “ComponentFactory” of the library. Furthermore, they must implement the “IComponent” interface in order for the ACF to dynamically integrate them into the firmware (see Section 5.2 ““IComponent” and “ComponentBase”“).

ACF components are instantiated once or several times. They are assigned a system-wide unique instance name.

Adding a component

Components are added to the ACF configuration as shown in the following example code:

Adding a component to the ACF configuration (example):

```
<?xml version="1.0" encoding="utf-8"?>
<AcfConfigurationDocument
    xmlns="http://phoenixcontact.com/schema/acfconfig“
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=“http://www.phoenixcontact.com/schema/acfconfig“
    schemaVersion=“1.0“ >
        <!-- Components included into /opt/plcnext/projects/Default/Default.acf.config
           are managed by ACF while
           Components included into /opt/plcnext/projects/Default/Plc/Plm/Plm.config
           are managed by PLM
        -->
    <Components>
        <Component name="ACFDemo" type=“MyLibraryACF.MyComponentACF“ library="MyLibraryACF"
         settings=“/opt/plcnext/projects/MyLibrary/MySettings.xml“ />
    </Components>

</AcfConfigurationDocument>
```

The following table lists explanations of the individual attributes:

XML element	Description
`<Components>`	List of components Here, components are listed that are to be created.
`<Component>`	Definition of a component
`name`	Name of the component It must be unique throughout the entire system because all resources in the system are addressed via the component name.
`type`	Type name of the component The name is composed of `<namespace>.<class>` A library can provide several component types. Typically, C++ type names are used here. The name is used in the code to register components with the “ComponentFactory”. This is the only dependency between the system configuration and the source code.
`library`	The name of the library that contains the component and was defined under `<Libraries>`.
`<Settings path>`	Path to the component settings file If the component requires a project configuration, the path to the component-specific project configuration must be specified in the settings file.

5.5.3 Configuration

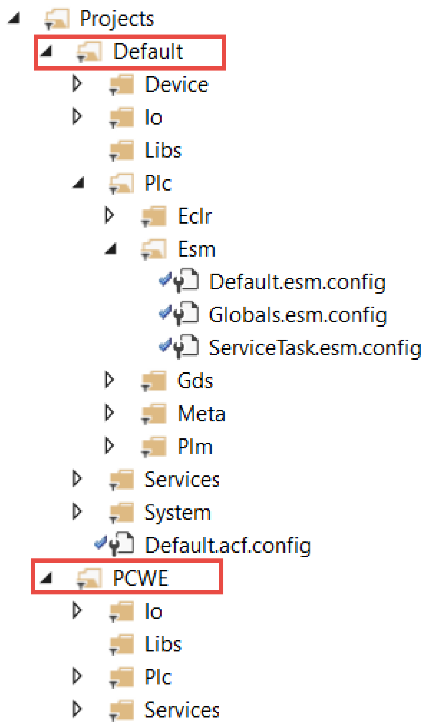
ACF configuration

As there is just one mechanism for the dynamic integration of components, the ACF configuration contains both the firmware settings and the project configuration for the user components. The difference is in the storage location. The respective *.acf.config file is either in the directory for the system settings or for the project configuration.

Project configuration

Transferring the project configuration to the controller can be done by PLCnext Engineer, other development tools (e.g. Eclipse® or Simulink® extension) or manually. Adding user components to the system is done by configuration files of the distinct components as well as the ACF configuration files (*.acf.config). These add the user components to the system. An ARP_PROJECTS_DIR environment variable must be defined in the ACF settings for the project directory (/opt/plcnext/projects). PLCnext Engineer requires this for downloading the project. If an ACF system or service component has a project configuration, the configuration path must be specified in the component settings (*.settings, XML element `<ConfigSettings>` with `path` attribute).

The project directory contains at least two subdirectories: “Default” and “PCWE”. The additional directory structure is constructed in the same way as the Arp component hierarchy.



```
Projects
  Default
    Device
    Io
    Libs
    Plc
      Eclr
      Esm
        Default.esm.config
        Globals.esm.config
        ServiceTask.esm.config
      Gds
      Meta
      Plm
    Services
    System
    Default.acf.config
  PCWE
    Io
    Libs
    Plc
    Services
```

Figure 5-1 Project directory

- PCWE directory
  The “PCWE” folder is intended for the download via PLCnext Engineer. Files that are stored here will be overwritten during the next download via PLCnext Engineer (see Section 2.6.4 “Generating configuration files with PLCnext Engineer“).
- Default directory:
  The “Default” folder is intended for storing further configuration files and for manual, configurative extension of the platform components (see Section 2.6.5 “Manual configuration“).

5.6 Common classes

Common classes provide functions that may be helpful for programming. The PLCnext Technology-specific common classes are made available via the PLCnext Technology SDK. With the help of the SDK, it is possible to generate high-level-language programs in C++ for the PLCnext Technology framework. The SDK provides Arp firmware header files for this (“ARP SDK“). With the help of the ARP SDK, you can use common classes in your program. If you want to use a class, integrate it in your program via an `#include` command. Further information on the common classes and their applications is available directly in the code commentary.

The useful common classes mentioned in the following section are part of the Phoenix Contact SDK, for example. The classes are encapsulated in namespaces according to their subject areas.

5.6.1 Threading

During threading, parts of a program are executed in parallel. The “Threading” namespace provides methods for separating a program into several strings for simultaneous execution, thus improving the performance of the overall system.

NOTE: Error due to changed priority

You can select a priority between 0 and 99. In order to not disturb the structure of the real-time threads, Phoenix Contact recommends priority 0. Otherwise, the stability of the firmware cannot be guaranteed. Programs in ESM tasks are intended for performing time-critical tasks.

“CpuAffinity”

“CpuAffinity” is a bit mask in which one bit is available per processor core. The least significant bit represents processor core 1. If this bit is set, the scheduler may execute the thread on this processor core. Several bits can be set simultaneously. In this case, the scheduler decides on which processor core the thread is started, and whether it is executed during runtime by another processor core. If the value of the parameter is 0, the scheduler can execute the thread on every available processor core.

“Thread”

One instance of this class is used to manage one thread, respectively. You must specify the function or method that is to be executed in a thread during instantiation. If the `Thread::Start` method is called, the thread is executed.

The “Thread” class selects a low priority as standard. Phoenix Contact recommends retaining this priority in order not to endanger the priority structure of the various firmware and operating system tasks.

“WorkerThread”

In contrast to the “Thread” class, an instance of the “WorkerThread” class is executed cyclically, as soon as the `WorkerThread::Start` method is executed. You can define cyclic execution of the thread via the `idletime` parameter. The value is specified in ms.

“ThreadSettings”

The “ThreadSettings” class is an auxiliary class for passing on the following thread parameters to a constructor of the “Thread” class:

- Name
- Priority
- CPU affinity (which CPU has been released for the execution of the task)
- Stack size (byte size)

“Mutex”

Using the “Mutex” class, you can prevent data of several threads being changed simultaneously. The “Mutex” class instances can have two states:

- `Locked`
- `Unlocked`

Once the `Mutex:Lock` method has been performed, i.e., the call from the method returns, the data is protected against modification by other threads. This state is retained until the instance calls the `Unlock()` command, therefore rescinding the locked state. Therefore, a call is blocked until the thread which is in the `Lock` state is released again. To prevent a “Deadlock”, i.e., a state in which a locked thread cannot be unlocked again, you can use the “LockGuard” class. This class automates `Lock` and `Unlock` of a “Mutex” instance.

“RwLock”

“RwLock” provides a locking mechanism for increasing performance. This class is useful if several read accesses, but not many write accesses are necessary. The difference between “Mutex” class instances and this class is that the instances of the “RwLock” class permit several simultaneous read accesses to the locked structure. Write access at the same time, however, is not allowed. Instances of this class are therefore suitable for all data that is often read but only rarely updated.

5.6.2 “Ipc” (inter-process communication)

The “Ipc” namespace (inter-process communication) encapsulates classes which can be used to enable the communication between various processes on the same controller.

“Semaphore”

Using the “Semaphore” class, semaphores are implemented in order to synchronize processes or threads. In principle, semaphores are integer counters. If one of the various `Wait` methods is called, the internal counter is lowered by one. If the current value of the counter is zero when a `Wait` method is called, the call is blocked until a different thread on the same semaphore instance calls the `Post` method (the `Post` method increases the internal counter).

“MessageQueue”

Using the “MessageQueue” class, data can be exchanged between processes in the form of messages. The names of “MessageQueue” instances must begin with an “/” because otherwise, the call from the constructor will lead to an exception. The name of a queue corresponds to the file path in Linux.

5.6.3 “Chrono”

The “Chrono” namespace contains classes and functions with which the temporal sequences within an application can be controlled and influenced. This includes high-resolution measurement of time elapsed so far, and also triggering of actions after a predetermined period of time.

“Timer”

The “Timer” class is a high-resolution chronometer for interval-based execution of methods. Instances of this class are used to execute one or more methods periodically in a defined interval. The “Timer” class calculates the next point in time at which the method is to be called. You only have to implement the method or methods that are to be called via the Timer definition.

5.6.4 “Io”

The “Io” namespace encapsulates all the functions necessary for working with files and folders within the file system of the underlying operating system.

“FileStream”

The “FileStream” class is for stream-based editing (opening, writing, reading) of files. The various values in the class define, for example, whether a file is to be overwritten, an already existing file is to be opened, or a new file is to be created.

5.6.5 “Net”

The “Net” namespace encapsulates all the classes and functions that enable network-based communication between processes on the same or separate controllers. For processes that run on the same controller, preferably use the functions from the “Ipc” namespace (see Section ““Ipc” (inter-process communication)” on page 128).

“Socket”

The “Socket” class is an interface for Ethernet-based communication. Use instances of this class to establish an Ethernet-to-peer connection. Currently, the UPD and TCP protocols are supported (IPv4 only).

5.6.6 “Runtime”

The “Runtime” namespace encapsulates functions for manipulation of individual processes that are managed by the Arp firmware.

“SharedLibrary”

Using the “SharedLibrary” class, shared libraries (.so files) are dynamically published or reloaded in applications during runtime. If a library is successfully reloaded with the `SharedLibrary::Load` method, the symbols contained (global variables, classes, methods, functions, etc.) are then known in the current program. The memory area can be requested with `GetFunctionAddress` in order for the functions of the library to be used in the program currently running.

Process

The “Process” class is a high-level API for creating and managing new processes.

5.7 “Template Loggable”

PLCnext Technology provides a log file on the controller file system in which information on the system behavior of the PLCnext Technology firmware, warnings, error messages, and debugging messages are logged. You will therefore find valuable information that can help you in finding the causes of problems.

“Template Loggable”

You can include the “Template Loggable<>” template class to automatically apply a tag to log messages. A tag can be used to determine from which component/program/... the message originates.

To use this class, you have to include (`#include`) the corresponding header file (.hpp).

- “Loggable”: Arp/System/Commons/Logging.h

The following log levels are supported. For each log level, a suitable method can be called.

- Info
- Warning
- Error
- Fatal

Example call:

```
log.info („Info!“);
```

Static call

Alternatively, you can also perform logging without the “Loggable” class. Messages can be written without creating a special logger using the root logger with the static “Log” class. The “root” tag is assigned to the message. The source of the message is thus not visible in the log file.

```
Log::Error („Error!“);
```

It is also possible to pass on and format variables. Placeholders in the form of {x} are used for the variables, where x is the index of the variable.

```
(„Variable a={0} b={1}“, a, b);
```

Diagnostic log file

The “Output.log” diagnostic log file contains status information, warnings, error messages, and debugging messages. You will find the file in the /opt/plcnext/logs folder on the file system of your controller. The file system is accessed via the SFTP protocol. The SFTP client software is required for this (e.g., WinSCP) (see Section “Directories of the firmware components in the file system” on page 63).

The diagnostic log file is configured in such a way that the messages are overwritten once the maximum file size is reached. When an error occurs, it is therefore recommended that the file is called and evaluated as soon as possible.

The diagnostic log file contains the following message types:

- **Error & Fatal**: If messages of the “Error” or “Fatal” type are issued, the controller is stopped. The errors mainly arise during startup or during execution of a user program.
- **Warning**: Warnings indicate potentially occurring errors.
- **Information**: The core components issue messages of type “Information”. These provide an overview of the system status.

Example: “Output.log” diagnostic log file:

```
    A           B                       C                        D                        E
18.05.07    08:24:15.830   MyLibrary.MyComponent                INFO  -   'MyComponent' invoked of object with instance
                                                                          name 'MyLibrary.MyComponent-1'
18.05.07    08:24:15.831   Arp.Plc.Plm.Internal.PlmManager      INFO  -   Component 'MyLibrary.MyComponent-1' from library
                                                                          'MyLibrary' created.
18.05.07    08:24:15.831   MyLibrary.MyComponent                INFO  -   'Initialize' invoked of object with instance name
                                                                          'MyLibrary.MyComponent-1'
18.05.07    08:24:15.832   MyLibrary.MyComponent                INFO  -   'SubscribeServices' invoked of object with in-
                                                                          stance name 'MyLibrary.MyComponent-1'
18.05.07    08:24:15.832   MyLibrary.MyComponent                INFO  -   Component 'AcfDemo' not found!
18.05.07    08:24:15.833   MyLibrary.MyComponent                INFO  -   'LoadSettings' invoked of object with instance
                                                                          name 'MyLibrary.MyComponent-1'
18.05.07    08:24:15.833   MyLibrary.MyComponent                INFO  -   'SetupSettings' invoked of object with instance
                                                                          name 'MyLibrary.MyComponent-1'
18.05.07    08:24:15.834   MyLibrary.MyComponent                INFO  -   'LoadConfig' invoked of object with instance name
                                                                          'MyLibrary.MyComponent-1'
18.05.07    08:24:15.834   MyLibrary.MyComponent                INFO  -   'SetupConfig' invoked of object with instance name
                                                                          'MyLibrary.MyComponent-1'
18.05.07    08:24:15.988   MyLibrary.MyComponent.MyProgram      INFO  -   Added Port 'zaehler (of Data Type 8)' of instance
                                                                          MyLibrary.MyComponent-1/MyProgram1
18.05.07    08:24:15.989   MyLibrary.MyComponent.MyProgram      INFO  -   Added Port 'zaehler (of Data Type 8)' of instance
                                                                          MyLibrary.MyComponent-1/P1
18.05.07    08:24:16.121   Arp.Io.Axioline.AxiolineComponent    INFO  -   Axioline: Load configuration.
18.05.07    08:24:16.127   Arp.Io.Axioline.AxiolineComponent    INFO  -   AxiolineComponent::LoadPlc() Path=/opt/plc-
                                                                          next/projects/PCWE/Io/Arp.Io.AxlC/links.xml
```

A	Date of the message in the format DD.MM.YY
B	Time of the message in the format hh:mm:ss.ms

C	Component that triggers the message
D	Message type (log level)
E	Message, e.g., info text, error message, debugging message

5.8 Using RSC services

You have the option of using the already registered RSC services of the SDK (Software Development Kit) via the “ServiceManager”. It acts as the RSC API and is used to request services.

- Use the `#include` command in the header file to include the “ServiceManager” class and the desired service interface (e.g., “IDeviceStatusService“).

Example header file: `#include` ServiceManager.hpp:

```
#pragma once
#include "Arp/System/Core/Arp.h"
#include "Arp/System/Acf/ComponentBase.hpp"
#include "Arp/System/Acf/IApplication.hpp"
#include "Arp/System/Acf/IControllerComponent.hpp"
#include "Arp/System/Commons/Logging.h"
#include “Arp/System/Rsc/ServiceManager.hpp”
#include "Arp/System/Commons/Threading/WorkerThread.hpp"
#include "Arp/Device/Interface/Services/IDeviceStatusService.hpp"
```

- Initialize a pointer to the object matching the desired service by calling the `GetService()` method of “ServiceManager”.
- Pass on the name of the corresponding interface as the template argument.

You can place this call in the `SubscribeServices()` method of your component, for example:

Header file (.hpp):

```
class ExampleComponent : public .....
{
    ...
private: //services
    IDataAccessService::Ptr dataAccessService;
    ...
}
```

Source file (.cpp):

```
#include....
    using namespace Arp::System::Rsc;
    using namespace Arp ...
void ExampleComponent::SubscribeServices()
{

    // Get service handle
    this->deviceStatusService = ServiceManager::GetService<IDeviceStatusService>();
}
```

Please note that execution of RSC services can take some time (in particular Axioline and PROFINET services). For this reason, avoid direct calls from ESM tasks.

RSC services are available for the following areas. You will find more detailed descriptions in the sections specified.

– **Axioline services**: Read and write access to data and information of Axioline devices (see Section 5.8.1 on page 132)
– **PROFINET services**: Read and write access to data and information of PROFINET devices (see Section 5.8.2 on page 133)
– **Device interface services**: Access to information and properties of the operating system and controller hardware (see Section 5.8.3 on page 135)
– **GDS services**: Read and write access to the GDS data (see Section 5.8.4 on page 140)

5.8.1 RSC Axioline services

> Please note that execution of RSC services can take some time. For this reason, avoid direct calls from ESM tasks.

> The Axioline RSC services are only available to PLCnext Technology controllers that support an Axioline local bus.

The Axioline component can be extended via interfaces for Axioline services. You can use one interface for acyclic communication (`PdiRead`, `PdiWrite`). This interface is available via the RSC protocol. Parameterization data, diagnostics information, and status information (PDI = Parameters, Diagnostics and Information) of an Axioline device can be read or written with the RSC service. Acyclic communication is suitable for the exchange of data that does not recur cyclically.

5.8.1.1 “IAcyclicCommunicationService”

The “IAcyclicCommunicationService” RSC service in the “Arp/Io/Axioline/Services” namespace for acyclic communication makes the following methods available:

– `PdiRead`:
Enables parameters, diagnostics and information of an Axioline device to be read
– `PdiWrite`
Enables parameters, diagnostics and information of an Axioline device to be written

```
PdiResult PdiRead(const PdiParam& pdiParam, std::vector<uint8>& data)
PdiResult PdiWrite(const PdiParam& pdiParam, const std::vector<uint8>& data)
```

Parameters are necessary for executing `PdiRead` and `PdiWrite`. The `PdiParam` structure is used for transmitting the input parameters. The structure has the following elements:

Parameter	Description
uint16 Slot	Device number
uint8 Subslot	Subdevice number
uint16 Index	Object index
uint8 Subindex	Object subindex

The return values are written to the PdiResult structure. The structure has the following elements:

Parameter	Description
uint16 ErrorCode	Error code
uint16 AddInfo	Error code, further information

Data that is read (PdiRead) and data that is to be written (PdiWrite) is transferred in a type unit8 vector (data). A description of the data (maximum size, type, etc.) of the object description is available in the data sheet of the respective Axioline module.

You need the following headers to use the service. Include them via the #include command:

- Arp/Io/Axioline/Services/IAcyclicCommunicationService.hpp
- Arp/Io/Axioline/Services/PdiParam.hpp
- Arp/Io/Axioline/Services/PdiResult.hpp

System-specific information on the Axioline F system is available in the PLCnext Engineer online help system, as well as in the user manuals “Axioline F: System and installation” (UM EN AXL F SYS INST) and “Axioline F: Diagnostic registers and error messages” (UM EN AXL F SYS DIAG).
The user manuals can be downloaded at phoenixcontact.net/qr/2404267/manual. Further information is also available in the data sheets of the respective Axioline modules.

5.8.2 RSC PROFINET services

Please note that execution of RSC services can take some time. For this reason, avoid direct calls from ESM tasks.

The PROFINET component can be extended via interfaces for PROFINET services. You can use one interface for acyclic communication (RecordRead, RecordWrite). The interface is available via the RSC protocol. The parameter data, diagnostics information, and status information (PDI = Parameters, Diagnostics and Information) of a PROFINET device can be read or written with the RSC service. Acyclic communication is suitable for the exchange of data that does not recur cyclically.

5.8.2.1 “IAcyclicCommunicationService”

The “IAcyclicCommunicationService” RSC service in the “Arp/Io/ProfinetStack/Controller/Service” namespace for acyclic communication makes the following methods available:

- RecordRead:
  Enables parameters, diagnostics and information of a PROFINET device to be read
- RecordWrite
  Enables parameters, diagnostics and information of a PROFINET device to be written

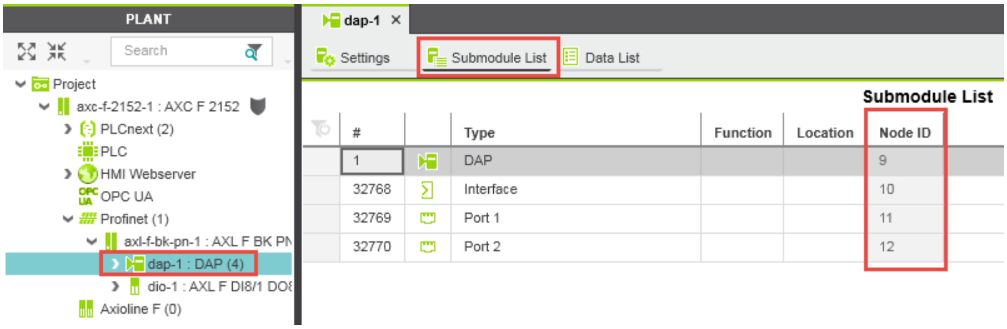
```
RecordResult RecordRead(const RecordParam& recordParam, std::vector<uint8>& data)
RecordResult RecordWrite(const RecordParam& recordParam, const std::vector<uint8>& data)
```

Parameters are necessary for executing RecordRead and RecordWrite. The “RecordParam” structure is used for transmitting the input parameters. The structure provides two ways of addressing a module:

- Version 1: Addressing via the ID
- Version 2: Addressing via an address that is made up of DeviceName, Slot, and Subslot

If one version is selected for addressing, the parameters of the other version must be 0 or empty.

The “RecordParam” structure has the following elements:

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>uint16 Id</td><td>Node ID of the submodule
The ID is assigned automatically. It can be viewed in PLCnext Engineer (version 1). For this, open the submodule list of the PROFINET submodule node.


Submodule List
<table><tr><th>#</th><th></th><th>Type</th><th>Function</th><th>Location</th><th>Node ID</th></tr><tr><td>1</td><td></td><td>DAP</td><td></td><td></td><td>9</td></tr><tr><td>32768</td><td></td><td>Interface</td><td></td><td></td><td>10</td></tr><tr><td>32769</td><td></td><td>Port 1</td><td></td><td></td><td>11</td></tr><tr><td>32770</td><td></td><td>Port 2</td><td></td><td></td><td>12</td></tr></table>
To determine the ID, you can also use the `AddressToID` method of the IAddressConversionService.hpp service.</td></tr>
<tr><td>RscString<512>
DeviceName</td><td>Device name
Name of the device that is to be addressed (version 2)</td></tr>
<tr><td>uint16 Slot</td><td>Device number (version 2)</td></tr>
<tr><td>uint8 Subslot</td><td>Subdevice number (version 2)</td></tr>
<tr><td>uint16 Index</td><td>Object index</td></tr>
<tr><td>uint8 Length</td><td>Maximum data amount
Specifies the maximum amount of data to be written in bytes (“RecordRead”) or the amount of data that is to be written to an object (“RecordWrite”).</td></tr>
</table>

The return values are written to the “RecordResult” structure. The structure has the parameters below. Further information on the parameters is available in the PROFINET specification (version 2.3).

- boolean ServiceDone
- uint8 ErrorCode
- uint8 ErrorDecode
- uint8 ErrorCode1
- uint8 ErrorCode2
- unit16 AddData1
- unit16 AddData2

Table 5-3 ErrorCode1

Error code	Description
0	No errors
0xF0	Internal error The error can be evaluated more precisely via ErrorCode2.
Further error codes	If a value other than 0xF0 or 0 is displayed in the ErrorCode1 element, the error is described in the ErrorCode, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, and AddData2 fields by the PROFINET specification.

Table 5-4 ErrorCode2

Error code	Description
0x03	“InvalidAddress” ID or DeviceName/Slot/Subslot is invalid.
0x15	“RecordReadFailed” RecordRead failed. This error can occur, for example, if several devices simultaneously attempt to execute “RecordRead” on one device.
0x16	„RecordWriteFailed” RecordWrite failed. This error can occur, for example, if several devices simultaneously attempt to execute “RecordWrite” on one device.
0x17	“Timeout” No confirmation was received.

Data that is read (“RecordRead”) or data that is to be written (“RecordWrite”) is transferred in a type unit8 vector (“data”). A description of the data (maximum size, type, etc.) of the object description can be found in the PROFINET specification (version 2.3).

You need the following headers to use the service. If necessary, include them via the `#include` command:

- Arp/Io/ProfinetStack/Controller/Services/IAcyclicCommunicationService.hpp
- Arp/Io/ProfinetStack/Controller/Services/AcyclicCommunicationServiceProxyFactory.hpp
- Arp/Io/ProfinetStack/Controller/Services/RecordParam.hpp
- Arp/Io/ProfinetStack/Controller/Services/RecordResult.hpp

5.8.3 RSC device interface services

Please note that execution of RSC services can take some time. For this reason, avoid direct calls from ESM tasks.

The device interface services provide a range of functions for accessing properties of the operating system and the controller hardware. You can call the information with the following interfaces and defined parameters. The following headers are required to use the service. Integrate these via `#include`, if necessary:

- Arp/Device/Interface/Services/IDeviceInfoService.hpp
- Arp/Device/Interface/Services/IDeviceStatusService.hpp

5.8.3.1 “IDeviceInfoService”

The “IDeviceInfoService” RSC interface enables read access to device information. The status value of a parameter is read with the `GetItem()` method. The status values of several parameters are read with the `GetItems()` method.

```
RscVariant<512>  GetItem(const RscString<512>& identifier)
void       GetItems(GetItemsIdentifiersDelegate identifiersDelegate, GetItemsResultDelegate
           resultDelegate)
```

The following parameters are available for calling information:

Table 5-5 “IDeviceInfoService” - parameters

Parameter	Data type	Description
General.DeviceClass	UInt32	The “DeviceClass” parameter specifies the device class. Currently, only “ProgrammableLogicController” is supported. 0: Undefined 1: ProgrammableLogicController 2. BusCoupler 3: Switch
General.VendorName	String	The “VendorName” parameter indicates the name of the manufacturer.
General.ArticleName	String	The “ArticleName” parameter indicates the device name.
General.ArticleNumber	String	The “ArticleNumber” parameter indicates the order number of the device.
General.SerialNumber	String	The “SerialNumber” parameter indicates the serial number of the device.
General.Firmware.Version	String	The “FirmwareVersion” parameter indicates the firmware version of the device. Here, 5-level notation (Major, Minor, Patch, Build, Status) is used.
General.Firmware.VersionMajor	Byte	“FirmwareVersionMajor” **Info**: The firmware version year is indicated without the first two digits. E.g., “2019” is indicated as “19”.
General.Firmware.VersionMinor	Byte	“FirmwareVersionMinor”
General.Firmware.VersionPatch	Byte	“FirmwareVersionPatch”
General.Firmware.VersionBuild	UInt32	“FirmwareVersionBuild”
General.Firmware.VersionStatus	String	“FirmwareVersionStatus”
General.Firmware.BuildDate	String	“FirmwareBuildDate” ISO 8601 format <YYYY>-<MM>-<DD>
General.Firmware.BuildTime	String	“FirmwareBuildTime” ISO 8601 format <hh>:<mm>:<ss>
General.Hardware.Version	String	The “HardwareVersion” parameter indicates the hardware version of the device.
General.Fpga.Version	String	The “FPGAVersion” parameter indicates the FPGA version of the device. Here, 3-level notation (Major, Minor, Patch) is used.
General.Fpga.VersionMajor	Byte	“FPGAVersionMajor”
General.Fpga.VersionMinor	Byte	“FPGAVersionMinor”
General.Fpga.VersionPatch	Byte	“FPGAVersionPatch”

Table 5-5 “IDeviceInfoService” - parameters

Parameter	Data type	Description
General.UniqueHardwareId	String	Sha256 Hash (32byte) hexadecimal coded as string
General.SPNS.Fpga.Version	String	FPGA version of the SPNS (only for devices with integrated safety controller)
General.SPNS.Fpga.VersionMajor	Byte	SPNS FPGA version Major
General.SPNS.Fpga.VersionMinor	Byte	SPNS FPGA version Minor
General.SPNS.Fpga.BuildVersion	Unsigned32	SPNS FPGA build version
General.SPNS.Firmware.Version	String	SPNS firmware version
General.SPNS.Firmware.VersionMajor	Byte	SPNS firmware version, Major
General.SPNS.Firmware.VersionMinor	Byte	SPNS firmware version, Minor
General.SPNS.Firmware.BuildVersion	Unsigned32	SPNS firmware build version
Interfaces.Ethernet.Count	Byte	The “NoOfNetworInterfaces” parameter indicates the number of network interfaces.
Interfaces.Ethernet.{adapterIndex}.{port}.Mac	String	The “Mac” parameter indicates the MAC address of the selected network interface. AA:BB:CC:DD:EE:FF adapterIndex= 1, 2, ... port = 0 for the interface, MAC port = 1, 2, ... for the MAC port

5.8.3.2 “IDeviceStatusService”

This RSC interface enables read access to status information. The status value of a parameter is read using the `GetItem()` method. The status values of several parameters are read using the `GetItems()` method. Use the `deviceStatusService.GetItem(“Parameters”)` method to call status information.

```
RscVariant<512> GetItem(const RscString<512>& identifier)
void            GetItems(GetItemsIdentifiersDelegate identifiersDelegate, GetItemsResultDelegate
                resultDelegate)
```

The following parameters are available for calling information:

Parameter	Data type	Description
Status.DeviceHealth	Byte	The “DeviceHealth” parameter indicates the operating status of the device. 0: OK 1: WARNING 2: ERROR
Status.Cpu.Load.Percent	Byte	The “CPULoad” parameter indicates the complete processor load of the device as a percentage. 0% ... 100%
Status.Cpu{0}.Load.Percent	Byte	The “CPULoad{Core}” parameter indicates the processor load of the selected processor core of the device as a percentage. 0% ... 100% 0x64 = 100% Core = 0, 1, 2, etc.

Parameter	Data type	Description
Status.Memory.Usage.Percent	Byte	The “MemoryUsage” parameter indicates the complete memory usage of the device as a percentage. 0% ... 100% 0x64 = 100%
Status.ProgramMemoryIEC.Usage.Percent	Byte	The “ProgramMemoryUsage” parameter indicates the program memory usage of the IEC runtime of the device as a percentage. 0% ... 100% 0x64 = 100%
Status.DataMemoryIEC.Usage.Percent	Byte	The “DataMemoryUsage” parameter indicates the data memory usage of the IEC runtime of the device as a percentage. 0% ... 100% 0x64 = 100%
Status.RetainMemory.Usage.Percent	Byte	The “RetainMemoryUsage” parameter indicates the complete retain memory usage of the device as a percentage. 0% ... 100% 0x64 = 100%
Status.RetainMemoryIEC.Usage.Percent	Byte	The “RetainMemoryUsage” parameter indicates the complete retain memory usage of the device as a percentage. 0% ... 100% 0x64 = 100%
Status.Board.Temperature.Centigrade	Int32	The “BoardTemperature” parameter indicates the temperature of the interior of the device in °C.
Status.Board.Humidity	Byte	The “BoardHumidity” parameter indicates the relative humidity in the device. 0% ... 100%
Status.Cpu.Temperature.Centigrade	Int32	The “CPUTemperature” parameter indicates the temperature of the processor in °C (only for RFC 4072S).
Status.KeySwitch.Position	Byte	The “KeySwitch” parameter indicates the position of the run/stop switch (only for RFC 4072S controller). 0: Switch in stop position 1: Switch in run position
Status.RamDisk.{RamDiskIndex}.Usage.Percent	Byte	The parameter indicates the memory usage of the RAM disc(s) as a percentage (only for RFC 4072S controller). 0% ... 100% RamDiskIndex = 1, 2,.... Number of the RAM disc (currently, only one RAM disc is supported, so that the index is always 1)
Status.RamDisk.{RamDiskIndex}.Usage	UInt32	The parameter indicates the absolute memory usage of the RAM disc(s) (only for RFC 4072S controller). RamDiskIndex = 1, 2,....Number of the RAM disc (currently, only one RAM disc is supported, so that the index is always 1)

You can also call status information on the LED states via the “IDeviceStatusService” interface.

The colors of the LEDs are represented as follows, normally in the high word (HW) of the return value:

```
public enum LedColor : ushort
{
    Green = 1,
    Yellow = 2,
    Red = 4
};
```

The states of the LEDs is represented as follows, normally in the low word (LW) of the return value:

```
public enum LedStates : ushort
{
    Off = 0,
    On = 1,
    Flashing_0_5_Hz = 2,
    Flashing_2_Hz = 3,
    Alternating_0_5_Hz = 4,
    Alternating_2_Hz = 5
}
```

Table 5-6 LEDs of the IEC runtime system

Parameter	Data type	Description
Status.Leds.Runtime.Run	UInt32	Runtime RUN LED (HW = color, LW = status)
Status.Leds.Runtime.Fail	UInt32	Runtime FAIL LED (HW = color, LW = status)
Status.Leds.Runtime.Debug	UInt32	Runtime DEBUG LED (HW = color, LW = status)

Table 5-7 Axioline LEDs

Parameter	Data type	Description
Status.Leds.Axio.D	UInt32	AXIO master D LED (HW = color, LW = status)
Status.Leds.Axio.E	UInt32	AXIO master E LED (HW = color, LW = status)

Table 5-8 PROFINET LEDs

Parameter	Data type	Description
Status.Leds.Pnio.Bf_C	UInt32	Pnio controller BF LED (HW = color, LW = status)
Status.Leds.Pnio.Bf_D	UInt32	Pnio device BF LED (HW = color, LW = status)
Status.Leds.Pnio.Sf	UInt32	Pnio controller SF LED (HW = color, LW = status)

You can also call status information on the network states via the “IDeviceStatusService” interface:

Table 5-9 Network interface

Parameter	Data type	Description
Status.Interfaces.Ethernet. {adapterIndex}.{port}Baudrate	Byte	The “Interface Baudrate” parameter indicates the current speed of the interface. 1: 10 Mbps 2: 100 Mbps 3: 1000 Mbps adapterIndex = 1, 2, ... port = 1, 2, ...
Status.Interfaces.Ethernet. {adapterIndex}.{port}.Duplex	Byte	The “Interface Duplex Mode” parameter indicates the current duplex mode of the interface. 1: Half duplex 2: Full duplex adapterIndex = 1, 2, ... port = 1, 2, ...
Status.Interfaces.Ethernet. {adapterIndex}.{port}.Link	Byte	The “Interface Link Status” parameter indicates the link status of the interface. 0: linkDown 1: linkUp adapterIndex = 1, 2, ... port = 1, 2, ...

5.8.4 RSC GDS services

Please note that execution of RSC services can take some time. For this reason, avoid direct calls from ESM tasks.

5.8.4.1 “IDataAccessService”

During runtime, the internal user components have read and write access to the GDS data. The service enables asynchronous reading and writing of one or more ports or even internal variables. For this process, you need the name of the port(s) that is/are to be read. The data read can be written to a database, for example.

You need the following header to use the “IDataAccessService” service. If necessary, include it via the `#include` command:

- Arp/Plc/Gds/Services/IDataAccessService.hpp

Service functions for direct data access:

- `Read()`: This function is used to read the values of the variable addresses passed on.
- `ReadSingle()`: This function is used to read the value of the variable address passed on. Only simple variables are supported (no arrays or structures).
- `Write()`: This function is used to write the values passed on to the variables of the variable address passed on.
- `WriteSingle()`: This function is used to write the value passed on to the variables of the variable address passed on. Only simple variables are supported (no arrays or structures).

```
ReadItem    ReadSingle(const RscString<512>& portName)
```

```
           Read(ReadPortNamesDelegate portNamesDelegate, ReadResultDelegate resultDelegate)
DataAccessError WriteSingle(const WriteItem& data)
           Write(WriteDataDelegate dataDelegate, WriteResultDelegate resultDelegate)
```

Each port has a unique name within the GDS that is made up as described in Section "GDS configuration using configuration files" on page 34. You need the complete name (URI) of a port in order to address it. The following port addresses are valid, for example:

- ComponentName-1/ProgramName-1.Variable_Name
- ComponentName-1/Global_Variable_Name
- ComponentName-1/ProgramName-1.Array_Variable_Name
- ComponentName-1/ProgramName-1.Array_Variable_Name[index]
- ComponentName-1/ProgramName-1.Array_Variable_Name[startIndex:endIndex]
- ComponentName-1/ProgramName-1.Struct_Variable_Name.Element1.Leaf
- ComponentName-1/ProgramName-1.Struct_Variable_Name.Element1.LeafArray
- ComponentName-1/ProgramName-1.Struct_Variable_Name.Element1.LeafArray[index]

The following variable types can be read and written:

- Primitive
- DateTime
- StaticString
- IecString
- Enum
- Struct
- Pointer
- Array

5.8.4.2 "ISubscriptionService"

The "ISubscriptionService" offers an alternative to the read functions of "IDataAccessService". The variables of which the values are to be read are only registered once and can then be read continuously. The data can be read more rapidly, and, due to the elimination of variable addressing, also recorded consistently with the task cycle. All variables in the same ESM task are recorded in the same cycle (with the exception of the "DirectRead" subscription type). In addition, "ISubscriptionService" provides time stamps that can be used to assign a clear recording time to each value.

To use the "ISubscriptionService" class, you require the following header file which was declared in the "Arp::Plc::Gds::Services" namespace:

```
#include „Arp/Plc/Gds/Services/ISubscriptionService.hpp“
```

- If necessary, include it via the `#include` command.

Below you will find a description of how to create, start, and read a subscription.

Creating a subscription

First, you have to create a subscription.

- To create a subscription, use the following function of the "ISubscriptionService":

```
uint32  CreateSubscription(SubscriptionKind kind)
```

With this function, the type of the desired subscription is passed on. Select one of the following four types:

Table 5-10 Subscription types

Type	Description
“DirectRead”	The “DirectRead” subscription records the values directly when the `ReadValues()` function is called within the context of the thread to be called. The “HighPerformance”, “RealTime”, and “Recording” subscription types collect the values within the context of the assigned ESM task. The data is read directly from the respective variable. A copy process is only implemented if data is also queried. The read data can originate from different task cycles. This subscription can deliver the best performance with the lowest impact on real time. Possible use: Asynchronous data acquisition of non-time-critical data.
“HighPerformance”	The “HighPerformance” subscription uses a double buffer which contains the last written data of a variable. The buffer enables almost simultaneous writing and reading of data. The “HighPerformance” type is consistent with the task cycle. It uses the least memory and shows the least impact on real time (compared with the “RealTime” and “Recording Subscription” types). Possible use: Standard type for acquiring data.
“RealTime”	The “RealTime” subscription uses a quad buffer which contains the last written data of a variable. The buffer minimizes access times during reading and writing. The “RealTime” type is consistent with the task cycle. It guarantees fast data access but requires four times the amount of memory. Possible use: The subscription is suitable for variables running in very fast tasks and if fast access to read files is required.
“Recording”	The “Recording” subscription uses a ring buffer that can store several data items of a variable. This type is consistent with the task cycle and has a low impact on real time. However, depending on the ring size, it requires a lot of memory. By default, the ring size is set to 10. It can be configured if the `CreateRecordingSubscription()` function is used instead of `CreateSubscription()`. Possible use: The subscription is suitable for variables running in tasks that are faster than the task of the user but the user still needs all values.

Adding variables

- Add the desired variables to the subscription by calling one of the following functions:

```
DataAccessError AddVariable(uint32 subscriptionId, const RscString<512>& variableName)
```

or

```
void AddVariables(uint32 subscriptionId, AddVariablesVariableNamesDelegate variableNamesDelegate,
AddVariablesResultDelegate resultDelegate);
```

Both functions can be called repeatedly for the same subscription. As with “IDataAccessService”, the variables are addressed via their complete name. Some examples are shown in the following table:

Table 5-11 Examples of variable addressing

Variable addressing	Description
ComponentName-1/ProgramName-1.Variable_Name	Program variable
ComponentName-1/Global_Variable_Name	Component variable (global variable)
ComponentName-1/ProgramName-1:Array_Variable_Name[index]	Array element
ComponentName-1/ProgramName-1:Struct_Variable_Name.Element1.Leaf	Structure element (leaf)
ComponentName-1/ProgramName-1:Struct_Variable_Name.Element1.LeafArray[index]	Array element from a structure
ComponentName-1/ProgramName-1:Array_Variable_Name[startIndex:endIndex]	An extract of array elements

Once a variable was added successfully to the subscription, the `DataAccessError::None` value is returned. In case of an error, the following return values might be returned:

Table 5-12 Return values in the event of an error

Name	Description
None	No error
NotExists	The variable does not exist in the system.
NotAuthorized	The user does not have sufficient authorization.
TypeMismatch	During writing, the value type is not suitable for the respective port.
PortNameSyntaxError	The port address is syntactically incorrect.
PortNameSemanticError	The port address is semantically incorrect.
IndexOutOfRange	The address contains an array index that is outside the array.
NotImplemented	The variable or service function has not yet been implemented.
NotSupported	The variable is not supported.
CurrentlyUnavailable	The service is currently unavailable.
UnvalidSubscription	The specified subscription was not found or is invalid.

The following types are currently supported:

- Primitive
- DateTime
- String (currently, only StaticString and IecString are supported)
- StaticString
- IecString
- Enum
- Struct
- Pointer
- Array

Subscribe/unsubscribe

Once you created a subscription and configured it with variables, you can activate it with `Subscribe`.

- To activate the subscription, call the following service function:

```
DataAccessError Subscribe(uint32 subscriptionId, uint64 sampleRate);
```

With call of the function, copying of the variable starts (exception: for the “DirectRead” type, no data is automatically recorded).

Use the `sampleRate` parameter to indicate in which time grid the values are to be recorded. The `sampleRate` can only be a multiple of the interval time of a cyclic ESM task. If a `sampleRate` that does not correspond to this interval time is passed on, the “SubscriptionValue” rounds the value to the next faster value.

Example: Variables from task A and task B are to be recorded:

– Interval time for task A: 10 ms
– Interval time for task B: 8 ms

If you specify 50 ms for `sampleRate`, the following is actually recorded:

– Variables from task A at 50 ms (every fifth cycle)
– Variables from task B at 48 ms (every sixth cycle)

If you specify value 0 for `sampleRate`, all the data is recorded in the interval of the respective task.

– Variables from task A at 10 ms
– Variables from task B at 8 ms

When a subscription was started, you can pause it via `Unsubscribe`.

- For this, call the following service function:

```
DataAccessError Unsubscribe(uint subscriptionId);
```

If a subscription pauses, no new data is recorded. Existing data is available in the subscription.

- To restart recording, call `Subscribe` again.

GetVariableInfos

This service function shows which variables are currently recorded. Therefore, the function only returns information once `Subscribe` is called.

The function only returns information about variable sorting. This information is decisive for reading the data. Each subscription internally sorts the variables, e.g., by assignment to the ESM task. The data of added variables is therefore not read in the order in which is was added. The “Read” functions only provide the raw values of the variables but do not give information on which variable the value is assigned to. At this point, service function information is the only option to assign the values to the respective variables. Variable information is returned in the same order as data for the “Read” functions. Therefore, variable information has to be read before the “Read” functions of a subscription are called for the first time.

A matching “Info” function is available for each “Read” function.

- To query all currently recorded variables, call the following function:

```
DataAccessError GetVariableInfos(uint32 subscriptionId, GetVariableInfosVariableInfoDelegate
variableInfoDelegate);
```

- To query the data, call the associated “Read” function:

```
DataAccessError ReadValues(uint32 subscriptionId, ReadTimeStampedValuesValuesDelegate
valuesDelegate)
```

- If, in addition to the variable values, you also require the time stamps of value acquisition, call the following function:

```
DataAccessError GetTimeStampedVariableInfos(uint32 subscriptionId,
GetTimeStampedVariableInfosVariableInfoDelegate variableInfoDelegate)
```

Information on time stamps is returned in addition to information on the variables. A variable with the name “timestamp” and of the “Arp.Plc.DataType.Int64” data type is always returned as the first element of an ESM task. This is followed by all information of the variable that is associated with the ESM task and can be assigned to the time stamp. If there are variables of several ESM tasks in the subscription, an additional time stamp is returned for each task, which is followed by the associated information. Here, information is also returned in the same order as the data that is returned with the following “Read” function:

```
DataAccessError GetRecordInfos(uint32 subscriptionId, GetRecordInfosRecordInfosDelegate
recordInfosDelegate);
```

ReadValues

Once you have started the recording of a subscription, you can query the acquired data. Different “Read” functions are available for this. The “Read” functions only return the variable values. The values are not assigned to the respective variable. For assigning the values to the respective variables, you have to call the corresponding “Info” function once (see Section “GetVariableInfos” on page 144).

The following “Read” service functions are available:

```
DataAccessError ReadValues(uint32 subscriptionId, ReadTimeStampedValuesValuesDelegate
valuesDelegate)
```

This function returns all the values in a static order. The `GetVariableInfos()` function is used for assigning the variables.

Example:

– Added variables from task A: a1, a2
– Added variables from task B: b1

ReadValues:

Object[]

a2
a1
b1

The following function returns all the values in a static order, including the associated time stamps:

```
DataAccessError ReadTimeStampedValues(uint32 subscriptionId, ReadTimeStampedValuesValuesDelegate
valuesDelegate);
```

The time stamps are always located before the associated variable values. The number of time stamps always corresponds to the number of ESM tasks the variables originate from. By means of the “DateTime” class, which is defined in the Arp namespace and in the Arp/System/Core/DateTime.hpp header file, the value of the “timestamp” variable can be converted into a time stamp. The `GetVariableInfos()` function is used for assigning the variables.

Example:

– Added variables from task A: a1, a2
– Added variables from task B: b1

ReadValues:

```
Object[]
        Timestamp task A
        a2
        a1
        Timestamp task B
        b1
```

The following function returns all the values packed in records:

```
DataAccessError ReadRecords(uint32 subscriptionId, uint16 count, ReadRecordsRecordsDelegate
recordsDelegate);
```

A record, also called data record, only contains the variable data from an ESM task and the associated time stamp. The time stamp is always located before the associated variable values. The variable order is always static and does not change during operation. An ESM task record is created for each ESM task. It contains all the corresponding data records of the respective ESM task. Depending on the subscription configuration, an ESM task record can contain several data records.

E.g.: A subscription of the “Recording” type with a task interval of 100 ms and a capacity of 10 returns 10 data records after one second. The time stamps are 100 ms apart. However, a subscription of the “HighPerformance”, “RealTime”, or “DirectRead” type always returns one data record only. By means of the “Read” function, you can read all the subscription data of the “Recording” type at once.

In addition to the static order of variables in the data records, the order of the ESM task records is static, too. By means of the `GetTimeStampedVariableInfos()` function, each value can be assigned a variable. The variable information describes exactly the first ESM task record and all the data records contained therein, from the first time stamp to the final variable information associated with this time stamp.

By means of the “DateTime” class, which is defined in the Arp namespace and in the Arp/System/Core/DateTime.hpp header file, the value of the “timestamp” variable in the respective data records can be converted into a time stamp.

Example:

- Added variables from task A: a1, a2
- Added variables from task B: b1

- Task A recorded 2 records.
- Task B recorded 1 record.

ReadRecords:

```
Object[] (ESM task records)
    Objects[] (ESM task record A)
        Objects[](data record cycle 1)
            Timestamp
            a2
            a1
        Objects[](data record cycle 2)
```

Timestamp
a2
a1

Objects[] (ESM task record B)

Objects[](data record cycle 1)

Timestamp
b1

Changing a subcription

You can modify two things of an existing subscription:

- The sampling interval the subscription used to record data.
- The variables that are to be recorded.

The subscription type can only be changed if you delete the subscription and create it again.

To change the sample rate, proceed as follows:

- Stop the subscription by calling `Unsubscribe`.
- Execute the subscription by calling `Subscribe` and entering the new sample rate as the parameter.

Several functions are available for changing the variables to be recorded:

- To delete a variable from a subscription, call the following function:

```
DataAccessError RemoveVariable(uint32 subscriptionId, const RscString<512>& variableName),
```

- To add a variable to a subscription, call the following function:

```
DataAccessError AddVariable(uint32 subscriptionId, const RscString<512>& variableName)
```

- To add several variables to a subscription, call the following function:

```
void    AddVariables(uint32 subscriptionId, AddVariablesVariableNamesDelegate
variableNamesDelegate, AddVariablesResultDelegate resultDelegate);
```

The changes can be made during operation and also to a subscription that is currently recording. To apply the changes, execute `Resubscribe`.

- To execute `Resubscribe`, call the following service function:

```
DataAccessError Resubscribe(uint32 subscriptionId, uint64 sampleRate);
```

Once `Resubscribe` was called, the changes are applied to the subscription. As the internal memories are rebuilt, data can be lost.

Deleting a subscription

If a subscription is no longer required, you have to delete it. If you do not delete the subscription, is will be retained until the next restart of the PLC application (warm or cold restart).

To delete a subscription, call the following function:

```
DataAccessError DeleteSubscription(uint32 subscriptionId)
```

The internally reserved memory is enabled and the subscription ID becomes invalid.

5.9 Notifications

The notification manager is used for registering, sending and receiving notifications between components of a controller. The header files required for using the notification manager are provided via the PLCnext Technology SDK (see Section 6.1, “PLCnCLI (PLCnext

Command Line Interface)”). The SDK contains classes for the Notification manager, classes as a basis for user-defined user data (payload), as well as the payload classes of the PLCnext Technology firmware. If you want to use a class, integrate it into your program via an `#include` command, (e.g., `#include <Arp/System/Nm/NotificationManager.hpp>`). Further information on the classes and their applications is available directly in the code commentary.

Defining user data

To enable correct interpretation of the user data of a notification, the structure has to be defined. Use template class “Arp/System/Nm/SpecializedPayload.hpp” for this. User data with up to 50 fields can be used as a character string.

Example code with two fields:

```
// Example payload with two payload fields

#include "Arp/System/Nm/SpecializedPayload.hpp"

// Use the class name as template parameter for SpecializedPayload
class UserLoggedInPayload
    : public Arp::System::Nm::SpecializedPayload<UserLoggedInPayload>
{
public:

    // Inherit base class constructors
    using SpecializedPayload::SpecializedPayload;

    // Define a convenient constructor to create the payload
    // This constructor will be called by a variadic template and perfect forwarding when
    // a notification is sent. Therefore this constructor defines the signature of the
    // function to send a notification.
    // Pass a format string for a user visible representation of the payload to the base
    // class' constructor. This format string will be used by the NotificationLogger to
    // serialize the payload.
    UserLoggedInPayload(const String& username, const String& permissions)
    : SpecializedPayload("User logged in: {0} (permissions={1})")
    {
        // Store the values of the parameters in the underlying payload data structure
        this->SetFieldvalue(this->fieldIndexUsername, username);
        this->SetFieldvalue(this->fieldIndexPermissions, permissions);
    }
    // Add getters for a type-safe access to the payload fields
    String GetUsername() const
    {
        return this->GetFieldValueAs<String>(this->fieldIndexUsername);
    }

    String GetPermissions() const
    {
    return this->GetFieldValueAs<String>(this->fieldIndexPermissions);
    }

private:
    // Define the fields of the payload type. Each field is accessed by an index.
    // The type infomation is used to ensure only valid information is set to the
    // payload fields.
    // Use direct member intialization here to ensure these members are also intialized
    // when the inherited constructors are used.
    // The order of these declarations matters, so don't change them.
    const size_t fieldIndexUsername = this->AddField<String>();
    const size_t fieldIndexPermissions = this->AddField<String>();
};
```

Registering and sending notifications

Before a notification can be sent, it must be registered with the Notification manager. If a component that registered a notification is no longer available, it has to unregister the notification first. Both processes are implemented using the “NotificationRegistration Proxy Object”.

Example:

```
// in UserManager
NotificationManager& nm= NotificationManager::GetInstance();
auto UserLoggedInRegistration = nm.CreateNotificationRegistration<UserLoggedInPayload>(
    "Arp.System.Um.Login", "UserManager", Severity::Info);

UserLoggedInRegistration.SendNotification("hans", "admin");

// notification is deregistered automatically in destructor of UserLoggedInRegistration
```

Metadata of a notification

To register a notification, the sender must specify metadata information:

- `notificationName`:
  The notification name defines under which name the notification is published. The parts of a name are separated by ”.” and should correspond, up to the component level, to the name of the component to be sent, e.g., “Arp.System.Um.UserLogin”, “Arp.Plc.Domain.PlcStateChanged”.
- `senderName`:
  The name of the component sending the notification.
- `Severity`:
  Information on the severity of the notification. Processing is not affected by the severity.
  - Info: General information
  - Warning: Warning for the user
  - Error: Error without serious impact
  - Critical: Error with medium impact
  - Fatal: Error with serious impact
- `PayloadTypeId`:
  Unique identification for the user data type.

⇒ The `NotificationNameId` is returned:
The ID is used to for identify an instance of a notification. It is required for sending and unregistering a notification.

The status of a notification registration can have the following states:

- `Subscribed`: A recipient subscribed but has not yet been registered by a component. Subscribing to a notification that has not yet been registered or has again been unregistered is only possible during startup of the firmware and startup of the controller. Otherwise, an exception will be triggered.
- `Registered`: Registered by a component
- `Unregistered`: Unregistered by a component

The `GetNotificationRegistration()` method returns the above meta information about a notification.

Receiving notifications

To receive notifications, the recipient must subscribe to the notification name with the Notification manager. The NotificatiaonSubscriber object is used for receiving notifications. If a recipient is no longer to receive notifications, it has to unsubscribe from theses notifications. To make sure that this is done at the end of the recipient's life cycle, it is recommended to use a proxy object like the one described for registering of notifications.

Example:

```
// in eHMI
void HandleUserLogins(const Notification& notification)
{
    auto payload = notification.GetPayloadAs<UserLoggedInPayload>();
    // do something with payload.GetUsername() and payload.GetPermissions()
}

// in some long living object of eHMI
NotificationManager& nm = NotificationManager::GetInstance();
auto UserLoggedIn = nm.CreateNotificationSubscriber("Arp.System.Um.Login");
UserLoggedIn.OnNotification += make_delegate(HandleUserLogins);

// notification is unsubscribed automatically in destructor of UserLoggedIn
```

Querying information

The INotificationManagerInfo interface is the interface for querying information about the NotificationManager. The following options are available:

- `GetNotificationName()`: This method returns the notification name.
- `GetNotificationNameId()`: This method returns the ID of a notification name.
- `GetNotificationRegistration()`: This method returns the information about a notification made available during registration (see Section “Metadata of a notification” on page 149.)
- `GetAllKnownNotificationNameIds()`: This method returns a list of NotificationNameIds of all known notifications, independent of their status.
- `GetNotificationsNameIdsByStatus()`: This method returns a list of NotificationNameIds of notifications with a specific status.

6 Creating programs with C++

With PLCnext Technology, you can also use programs created with C++ in the real-time context of a PLC along with conventional IEC 61131-3 programs. To use programs and program parts created in C++ within the scope of PLCnext Technology, you need a Software Development Kit (SDK).

With the PLCnCLI (PLCnext Command Line Interface), Phoenix Contact provides you with a tool featuring the following:

- Administration of the SDKs
- Build environment
- Automatic generation of required metadata and configurations
- Automatic creation of the completed library
- Extensive toolchain based on a console

The PLCnCLI is based on the .NET Core Framework and can be used on Windows and Linux operating systems.

You can use the freely available Eclipse® software with the Phoenix Contact add-in for Eclipse® as the C++ development environment. The add-in allows for easy connection to the PLCnext Technology platform and provides the PLCnCLI functions in Eclipse®.

6.1 PLCnCLI (PLCnext Command Line Interface)

The PLCnCLI (in the following also “CLI”) provides the entire toolchain for C++ programming on the PLCnext Technology platform as well as a template system for creating projects. Based on the templates, you can develop your applications. Use the CLI to unpack and manage the SDKs. CMake is contained as the build environment and each SDK has its own configuration. A parser integrated in PLCnCLI creates the metadata required for PLCnext Technology. The LibraryBuilder contained in PLCnCLI creates a PLCnext Engineer library from the project.

> The use of the Eclipse® add-in requires the installation of PLCnCLI (Section “Installing PLCnCLI” on page 152) as well as the installation of one or several SDKs (Section “Installing SDKs” on page 152). For information on using the Eclipse® add-in, please refer to Section “Eclipse® add-in” on page 156.

6.1.1 System requirements

- Before starting the installation, ensure that the system requirements are met.
- Download the necessary software.

Operating system

The PLCnCLI was tested and released for the following operating systems:

- Microsoft® Windows® 10
- Linux (Ubuntu 18.04.1.LTS)

Phoenix Contact software

- **PLCnCLI (PLCnext Command Line Interface)**:
  The CLI is a command line interface. It can be used, e.g., for generating metadata, C++ header files, PLCnext Engineer libraries, and for adding IN and OUT ports. The functions can be called using simple commands. An integrated help lists the commands and describes their functions.
- **PLCnext Technology SDK**:

  An SDK contains all of the important toolchains and libraries required for creating a program. The SDK is installed via the CLI (see Section 6.1.3).

PLCnCLI and PLCnext Technology SDK are available in the download area of your controller at phoenixcontact.net/products.

Some sample applications programmed in C++ can be found at https://github.com/plcnext.

6.1.2 Installing PLCnCLI

Installing in Windows

- Run the installation file.
- Follow the instructions of the installation wizard.

Phoenix Contact recommends to add the installation directory to the “Path” variable in the environment variables.

Installing in Linux

The following packages are required for installation in Linux:

- **“xz-utils”**
  Required for SDK installation.
- **“python3”**
  Required for SDK installation.
- **“build-essential”**
  The package contains the major part of the toolchain required for C++ programming, e.g., “make”.
- **“libunwind8”**
  The “EngineeringLibraryBuilder” requires this library.
- **“sshpass_1.06-1_amd64.deb”**
  Is used by the debug script.

To check if the packages were installed or are being installed, enter the following command:

```
sudo apt-get install xz-utils python3 build-essential libunwind8
```

- Then, run the “PLCnCLI_Setup.sh” file.
- Read and accept the “Software License Terms”.

You will then receive a folder named “plcncli” to which the PLCnCLI was unpacked. Like for Windows, Phoenix Contact recommends to introduce the “Executable” file to simplify calling the file in the console.

- For this, enter the following command:

```
ln -s [Path to the PLCnCLI installation folder]/plcncli /usr/
local/bin/plcncli
```

E.g.: `ln -s /home/plcncli/plcncli /usr/local/bin/plcncli`

6.1.3 Installing SDKs

- Download the SDK that is suitable for your controller from the download area of your controller at phoenixcontact.net/products.

- Unpack the .zip file.
- Call the CLI in the console using the following command:
  ```
  plcncli.exe install sdk -d [Installation path] -p [Path to archive file]
  ```
  E.g.:
  ```
  plcncli.exe install sdk -d C:\CLI\SDKs\AXCF2152\2019_0\ -p C:\CLI\pxc-glibc-x86_64-axcf2152-image-mingw-cortexa9t2hf-neon-toolchain-2019.0.tar.xz
  ```

If you install several SDKs, Phoenix Contact recommends to use the “target name/firmware version” folder structure. Installing an SDK using the PLCnCLI automatically makes the installation known to the CLI.

Introducing an SDK

- To introduce already unpacked SDKs to the CLI, enter the following command:
  ```
  plcncli.exe set setting -a SdkPaths [Path to SDK]
  ```
  E.g.:
  ```
  plcncli.exe set setting -a SdkPaths C:\CLI\SDKs\AXCF2152\2019_0\
  ```
- To receive a list of all available settings and the set values, enter the following command:
  ```
  plcncli.exe get setting -a
  ```

Uninstalling an SDK

– To uninstall an SDK, enter the following command:
  ```
  plcncli.exe set setting -r [Path to SDK]
  ```

The uninstalled SDK is removed from the PLCnCLI, but will be retained in the directory system.

6.1.4 Functions of the PLCnCLI

CLI features

You can execute many functions directly using the Command Line Interface (CLI). The CLI provides the following features, which you can call by means of simple commands:

- Creating projects
- Managing SDKs
- Generating metafiles (*.progmeta, *.compmeta, *.libmeta)
- Generating C++ frames and header files
- Compiling with CMake
- Creating PLCnext Engineer libraries for several controllers and firmware versions

CLI command structure

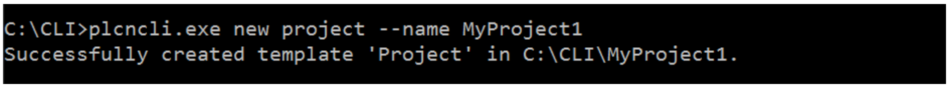
Use the Command Line Interface to call functions by means of simple commands.

- Activate the CLI by calling the plcncli.exe via the shell, e.g., `C:\CLI\plcncli.exe`.

There is a long and a short form for the commands, which you can use as you like:

Long form:

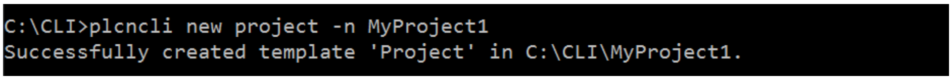
- To add the parameters, prefix “`--`”.
  E.g.: `new project --name MyProject1`



```
C:\CLI>plcncli.exe new project --name MyProject1
Successfully created template 'Project' in C:\CLI\MyProject1.
```

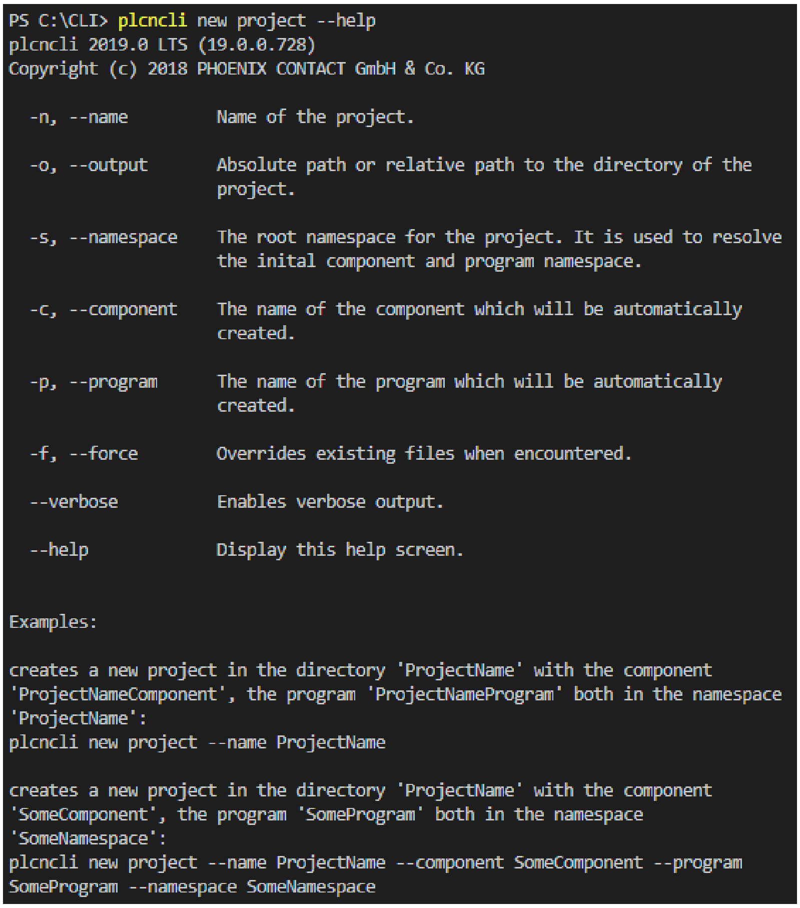
Short form:

- To add the parameters, prefix “`-`”.
  E.g.: `new project -n MyProject1`



```
C:\CLI>plcncli new project -n MyProject1
Successfully created template 'Project' in C:\CLI\MyProject1.
```

The functions are structured hierarchically. For each command, there is another level of commands that specifies the desired function in more detail. Use the `--help` command to call a description of the individual functions.
E.g.: `new project --help`



```
PS C:\CLI> plcncli new project --help
plcncli 2019.0 LTS (19.0.0.728)
Copyright (c) 2018 PHOENIX CONTACT GmbH & Co. KG

  -n, --name         Name of the project.

  -o, --output       Absolute path or relative path to the directory of the
                     project.

  -s, --namespace    The root namespace for the project. It is used to resolve
                     the inital component and program namespace.

  -c, --component    The name of the component which will be automatically
                     created.

  -p, --program      The name of the program which will be automatically
                     created.

  -f, --force        Overrides existing files when encountered.

  --verbose          Enables verbose output.

  --help             Display this help screen.


Examples:

creates a new project in the directory 'ProjectName' with the component
'ProjectNameComponent', the program 'ProjectNameProgram' both in the namespace
'ProjectName':
plcncli new project --name ProjectName

creates a new project in the directory 'ProjectName' with the component
'SomeComponent', the program 'SomeProgram' both in the namespace
'SomeNamespace':
plcncli new project --name ProjectName --component SomeComponent --program
SomeProgram --namespace SomeNamespace
```

Figure 6-1 Description of functions

If you do not specify all the parameters, the system will use default parameters. The values will be displayed after the command was executed.
If you do not specify a directory path via the `--path` option, e.g., when creating a project, a subfolder is created and used in the current directory.

CLI command overview

The CLI provides the following commands, which you can use to execute the corresponding functions. To call a command, enter it in the command line after the `plcncli.exe` call.
E.g., `C:\CLI\plcncli.exe new...`:

Table 6-1 CLI command overview

Command	Meaning
`build`	Compile a project
`generate`	Generate libraries, metafiles and code files
`get`	Query information about projects, controllers, etc.
`set`	Make settings, e.g., select a controller
`update`	Update of controllers in the current project
`install`	Install SDK
`new`	Create new files/projects
`help`	Call additional information about a specific command
`version`	Show version information

Each command can be specified using additional parameters. Commands consist of the following:

CLI call	Level 1	Level 2	Level 3
`C:\...plcncli.exe`	`new`	`project`	`-n` or `--name`

If you require information on parameters or additional command levels, you can request them for each level by means of the `--help` command.

Example: Configuring a project

The following example shows an option of how to configure a project up to the complete PLCnext library using the CLI:

- Open a project folder of your choice:
  e.g., C:\Users\<username>\Documents\CLI Projects.
- Generate a new project with the project name “Project1”:
  ```
  \CLI\Plcncli.exe new project –n Project1
  ```
- Go to the “Project1” project folder:
  ```
  cd Project1
  ```
- Select the desired controller for the project (e.g., AXC F 2152):
  ```
  plcncli.exe set targets --add -n AXCF2152
  ```
- Use the following command to generate the necessary metafiles for the project:
  ```
  \CLI\plcncli.exe generate config
  ```
- Use the following command to generate the necessary code files for the project:
  ```
  \CLI\plcncli.exe generate code
  ```
- You can also generate metafiles and code files in only one step:
  ```
  \CLI\plcncli.exe generate all
  ```
- Compile the project.:
  ```
  \CLI\plcncli.exe build
  ```

- Generate a library:
  ```
  \CLI\plcncli.exe generate library
  ```

⇒ The library and the .so files are located in the “bin” folder of the project.

For additional information on programming the code, please refer to Section 5, “Structure of a C++ program”.

Importing the library

You can then import the generated library in PLCnext Engineer, see Section 9, “Importing libraries into PLCnext Engineer”.

6.2 Eclipse® add-in

The Eclipse® add-in for PLCnext C++ programming enables you to use the PLCnCLI functions from you familiar C++ development environment.

6.2.1 Requirements

Installing Eclipse®

Phoenix Contact recommends the Eclipse® CDT Photon (9.5.2) development environment.

- Download the Eclipse® CDT Photon (9.5.2) software with the “Eclipse IDE for C/C++ Developers” package from https://www.eclipse.org.
- Copy and unpack the files to any folder.
- You can execute Eclipse® right away.

> The Java™ Runtime Environment is required for the Eclipse® development environment.
>
> If an error with “error code 13” occurs when Java is started, check if the correct Java version is installed. If you use a 64-bit Eclipse version, you have to use a 64-bit Java version.

Installing PLCnCLI

- Install the PLCnext Command Line Interface according to the information in Section 6.1.2, “Installing PLCnCLI”.

Installing SDKs

- Install the SDKs according to the information in Section 6.1.3, “Installing SDKs”.

6.2.2 Installing/updating/uninstalling the Eclipse® add-in

Installing the add-in

- If necessary, uninstall earlier add-in versions (see “Uninstalling the add-in” on page 158).
- Start Eclipse®.

- Open the “Help, Install New Software...” menu.

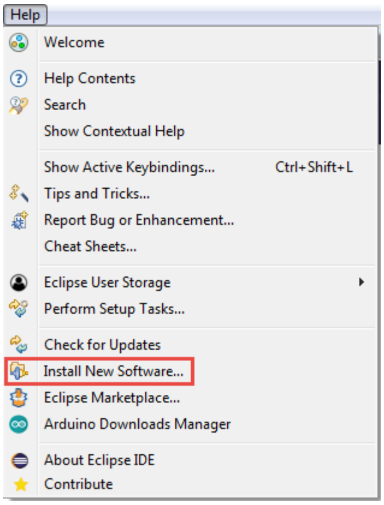


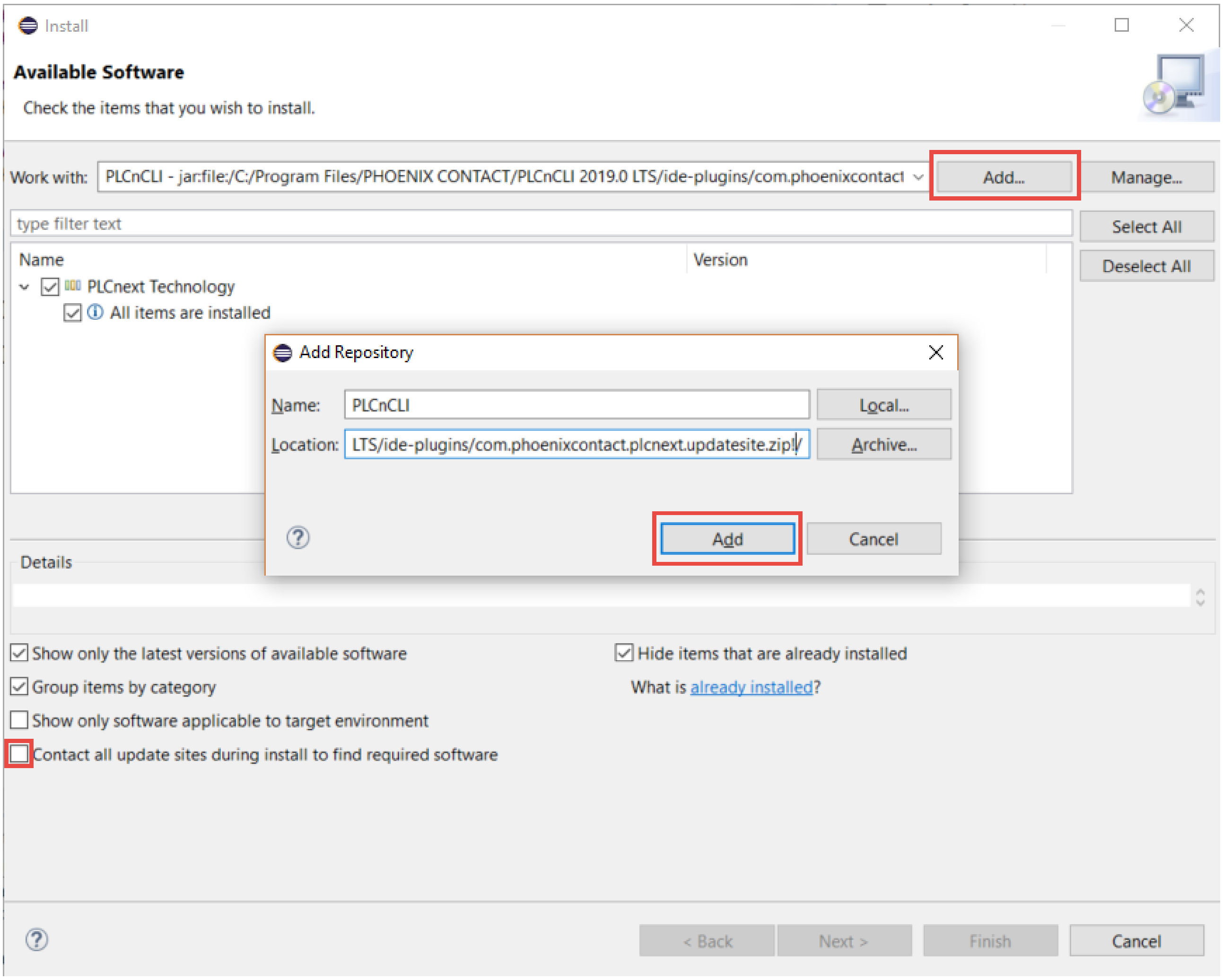
Figure 6-2 “Help, Install New Software...” menu

- To select the path to the directory of the add-in, click on “Add” in the “Install” dialog (see Figure 6-3).
- In the “Add Repository” dialog, click on “Archive” and select the archive of the add-in.

You will find it in the installation folder of the PLCnCLI under .../ide-plugins/com.phoenixcontact.plcnext.updatesite.zip.

- Enter a name of your choice in the “Name” input field and confirm with “Add”.

- Disable the “Contact all update sites during install to find required software” check box.



Install

Available Software

Check the items that you wish to install.

Work with: PLCnCLI - jar:file:/C:/Program Files/PHOENIX CONTACT/PLCnCLI 2019.0 LTS/ide-plugins/com.phoenixcontact

type filter text

Name	Version
☑ PLCnext Technology	
☑ All items are installed	

Add Repository

Name: PLCnCLI

Location: LTS/ide-plugins/com.phoenixcontact.plcnext.updatesite.zip!/

Details

☑ Show only the latest versions of available software

☑ Group items by category

□ Show only software applicable to target environment

□ Contact all update sites during install to find required software

☑ Hide items that are already installed

What is already installed?

Figure 6-3 “Install” dialog

- Click on the “Next” button.
- Read and accept the licence agreements.
- Click “Finish” to complete the installation.
- Restart Eclipse®.

Updating the add-in

- To update the Eclipse® add-in, proceed as described in Section “Installing the add-in” on page 156.

Uninstalling the add-in

- Open the “Help, About Eclipse IDE” menu.

- Click on the “Installation Details” button.

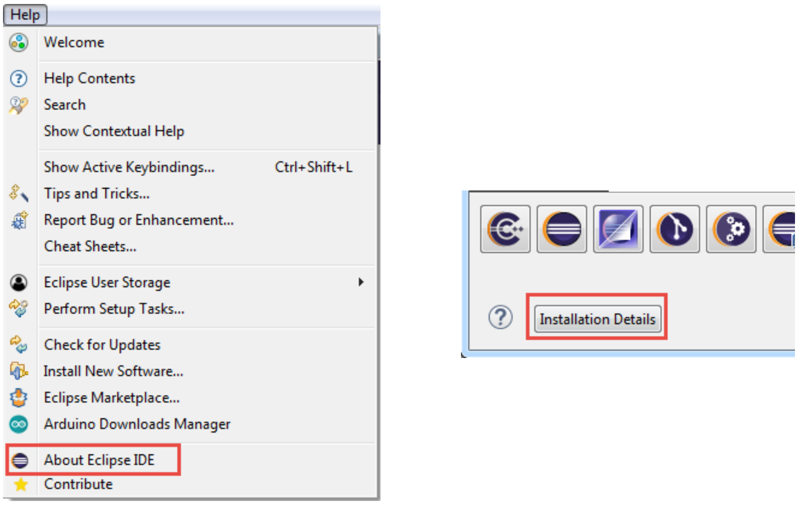


Figure 6-4 “Help, About Eclipse IDE” menu

- Select the earlier add-in versions and uninstall them.

6.2.3 Creating a new C++ project in Eclipse®

This section describes how to create a new C++ project in Eclipse®.

- Open the “File, New, Project...” menu.

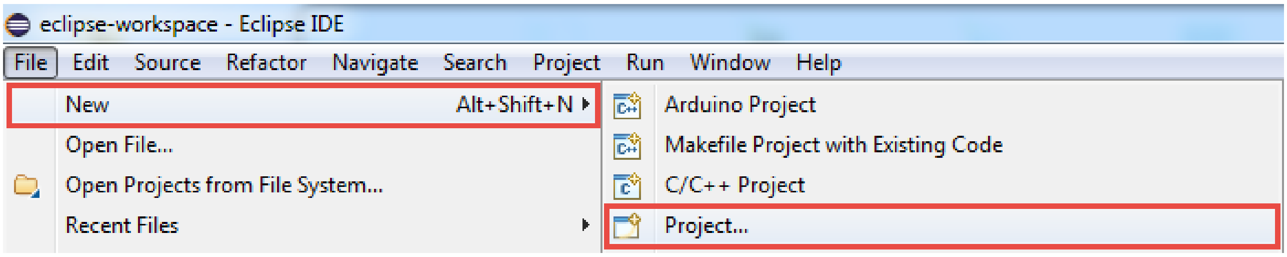


Figure 6-5 Creating a new project

The “New Project” window opens (see Figure 6-6).

- Select the “PLCnext C++ Project” wizard from the selection list.

- Confirm your selection with “Next”.

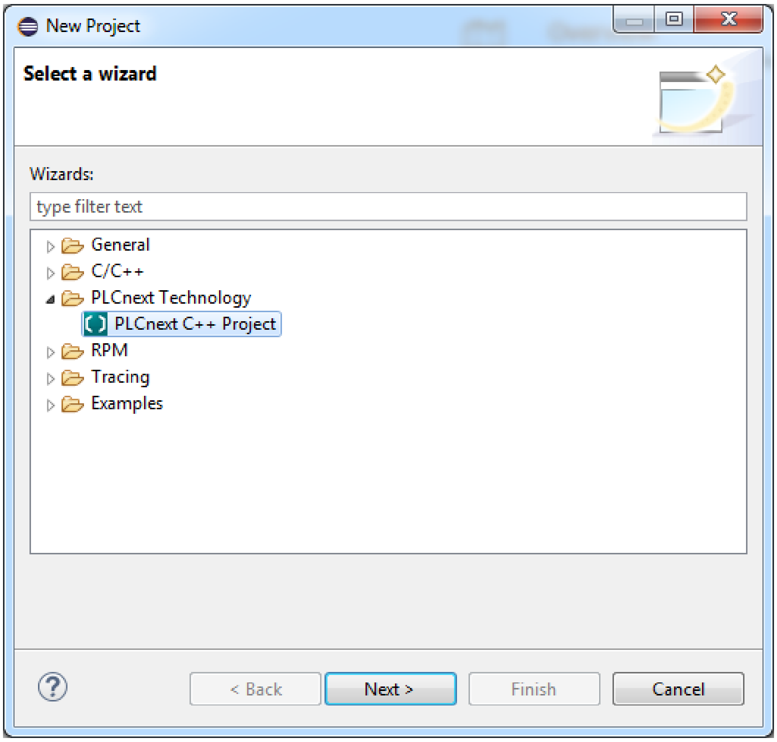
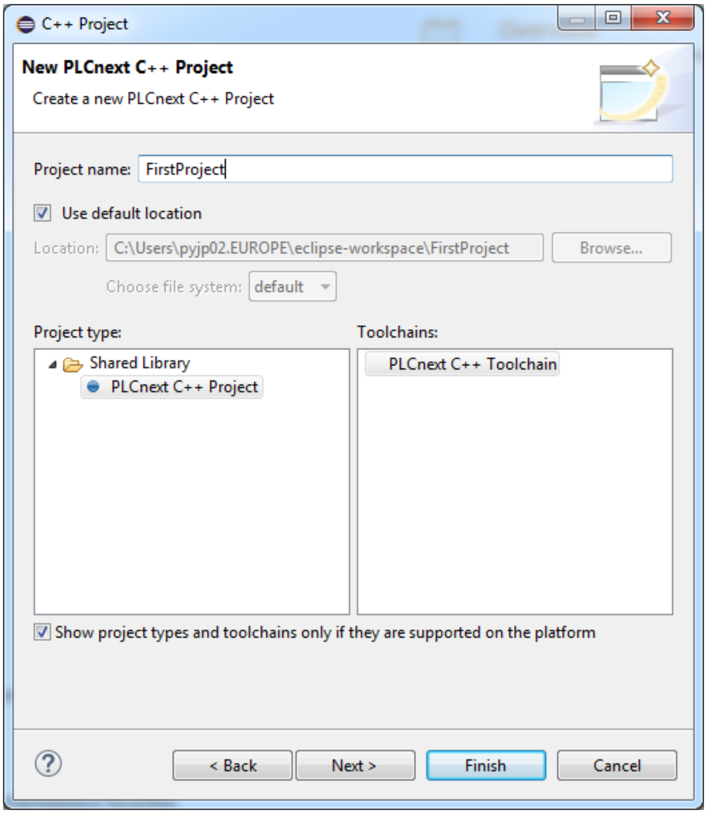


Figure 6-6 Selecting a wizard

- Enter a project name in the “Project name” input field.
- In the “Toolchains” area, select the “PLCnext C++ Toolchain” entry.

- Confirm with “Next”.



C++ Project

New PLCnext C++ Project

Create a new PLCnext C++ Project

Project name: FirstProject

☑ Use default location

Location: C:\Users\pyjp02.EUROPE\eclipse-workspace\FirstProject

Choose file system: default

Project type:

- Shared Library
  - PLCnext C++ Project

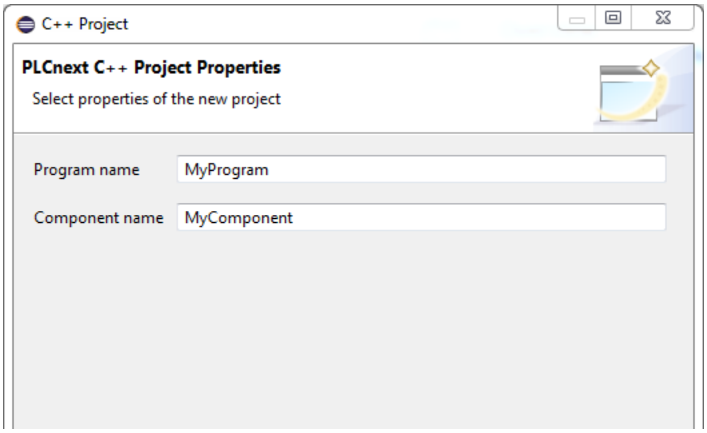
Toolchains:

- PLCnext C++ Toolchain

☑ Show project types and toolchains only if they are supported on the platform

Figure 6-7 Selecting toolchains

- Enter a name for the program and component in the corresponding input fields.



C++ Project

PLCnext C++ Project Properties

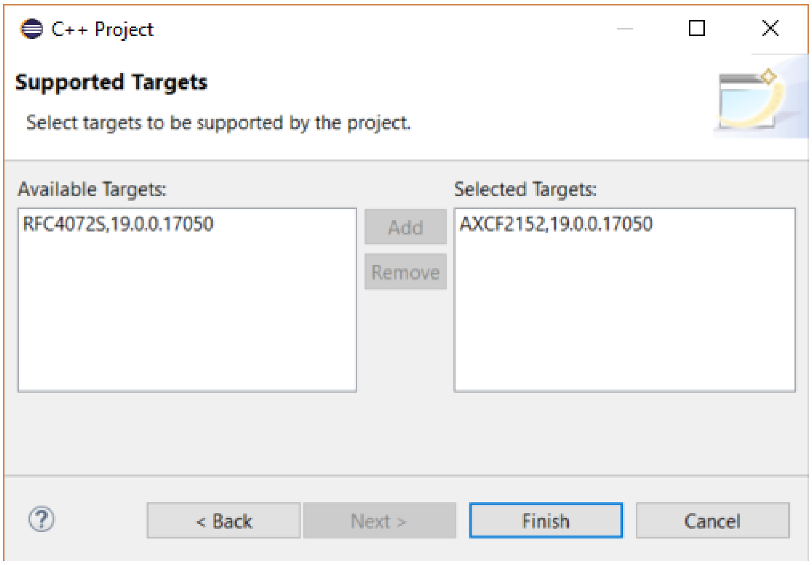
Select properties of the new project

Program name: MyProgram

Component name: MyComponent

Figure 6-8 Entering a name for the program and component

- Select the desired controller and firmware status for the project.



C++ Project

Supported Targets

Select targets to be supported by the project.

Available Targets:

RFC4072S,19.0.0.17050

Selected Targets:

AXCF2152,19.0.0.17050

Figure 6-9 Selecting the controller and firmware status

You can also change the selection of the desired controller and the corresponding firmware status after the project has been created.

- Open the “Project, Properties” menu.
- Select the “PLCnext Targets” option.
- Remove controllers or add new controllers and their firmware version.
- Click the “Finish” button.

You can not start creating the program.

6.2.4 Creating a program

To create a C++ program using Eclipse®, which can be imported as a library into PLCnext Engineer, you must first prepare a new project in accordance with the description in Section 6.2.3 “Creating a new C++ project in Eclipse®“.

The created project has a defined structure in which the C++ program is created. Figure 6-10 shows an example.

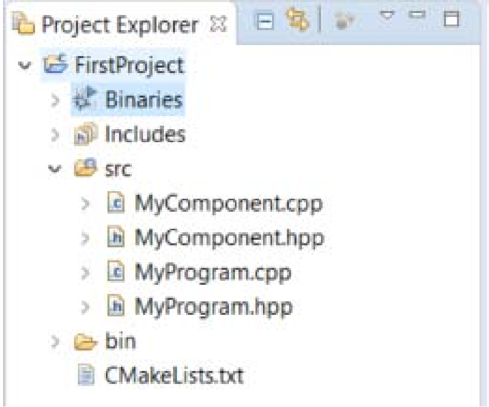


Figure 6-10 Example structure of a C++ project

To create a program, proceed as follows:

- Open the *.cpp source file.
- Program the code to be executed during each ESM task cycle in the "Execute()" function (see comment `//TODO implement program`).

For additional information on programming in C++, please refer to Section "Structure of a C++ program" on page 116.

After instantiation of a program, the assigned ESM task calls the `Execute()` function of the program instance in each cycle.

Example projects and the associated documentation can be downloaded at https://github.com/plcnext.

6.2.5 Compiling the project

Recommended: Make the following setting so that your files are saved automatically before you execute the "Build Project" option:

- In Eclipse®, open the "Window, Preferences" menu.
- Select "General, Workspace, Build".
- Enable the "Save automatically before build" check box.
- Make this setting separately for each workspace.

- Switch to the "Project Explorer" to compile the project.
- Right-click to open the context menu for the project folder (in the example: "FirstProject1").

- In the context menu, click on the “Build Project” option or click on the 🔨 button in the tool bar.

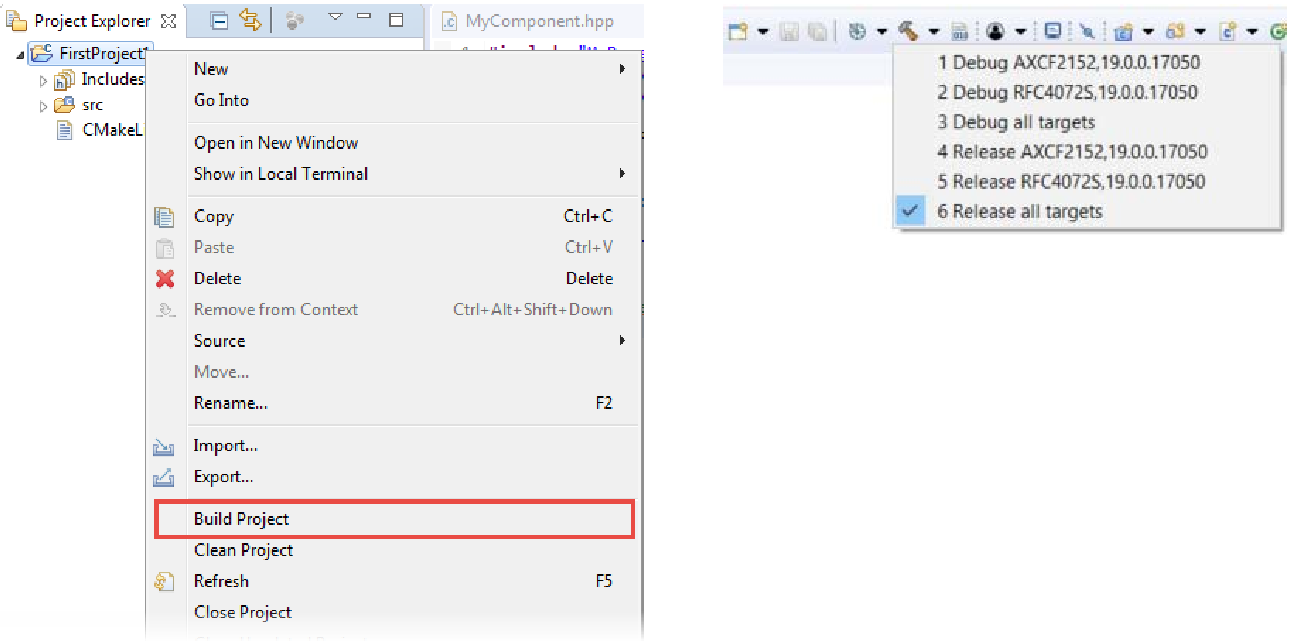
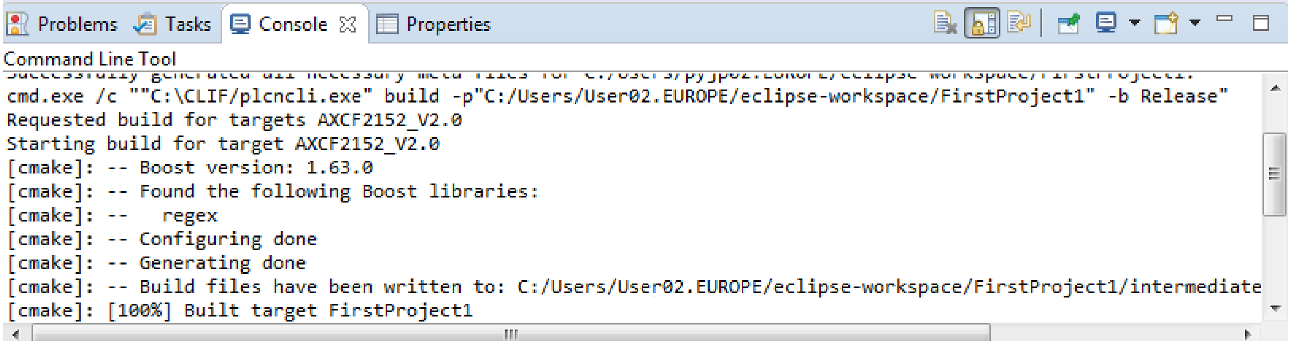


Figure 6-11 “Build Project”

Successful compilation of the project or possible error messages are displayed on the “Console” tab.



```
Successfully generated all necessary meta files for C:/Users/pyjp02.EUROPE/eclipse-workspace/FirstProject1.
cmd.exe /c ""C:\CLIF/plcncli.exe" build -p"C:/Users/User02.EUROPE/eclipse-workspace/FirstProject1" -b Release"
Requested build for targets AXCF2152_V2.0
Starting build for target AXCF2152_V2.0
[cmake]: -- Boost version: 1.63.0
[cmake]: -- Found the following Boost libraries:
[cmake]: --   regex
[cmake]: -- Configuring done
[cmake]: -- Generating done
[cmake]: -- Build files have been written to: C:/Users/User02.EUROPE/eclipse-workspace/FirstProject1/intermediate
[cmake]: [100%] Built target FirstProject1
```

Figure 6-12 Example: “Console”

Note regarding compiler warning

For each defined port, you receive the compiler warning: `offsetof within non-standard-layout type`.

E.g.: `[cmake]: /home/user/Workspace/Counters/intermediate/code/CountersLibrary.meta.cpp:22:47: warning: offsetof within non-standard-layout type ‘Counters::CppRetain’ is undefined [-Winvalid-offsetof]`

You can ignore this warning in conjunction with your defined ports or suppress it using the `-Wno-invalid-offsetof` CMake parameter.

Successful compilation

If compilation was successful, a shared object (*.so) is generated.

Generating a PLCnext Engineer library

To be able to use the program in PLCnext Engineer, you need the pcwlx library, which was generated during compilation. The Library Builder automatically generates the pcwlx library within the framework of compilation ("Build Project"). The files are located in the project folder, e.g., C:/Users/<username>/eclipse-workspace/FirstProject1/Bin. Import the library within PLCnext Engineer and assign the programs to a task (see Section 9, "Importing libraries into PLCnext Engineer").

6.3 Remote debugging

PLCnext Technology has an integrated GDB server. You have the option of setting up a remote debugging session using Eclipse®.

A description on how to set up a remote debugging session for a running process is available in the PLCnext Community.

7 Creating function blocks and functions with C#

The Visual Studio® extension is an extension for the integrated development environment Microsoft® Visual Studio®. The extension allows for using Visual Studio® for the development of eCLR firmware libraries in C# and implementing them on PLCnext Technology devices. Furthermore, you can debug the C# code on PLCnext Technology devices.

ProConOS embedded CLR is the open IEC 61131 control runtime system for different automation tasks. The eCLR programming system is structured as follows:

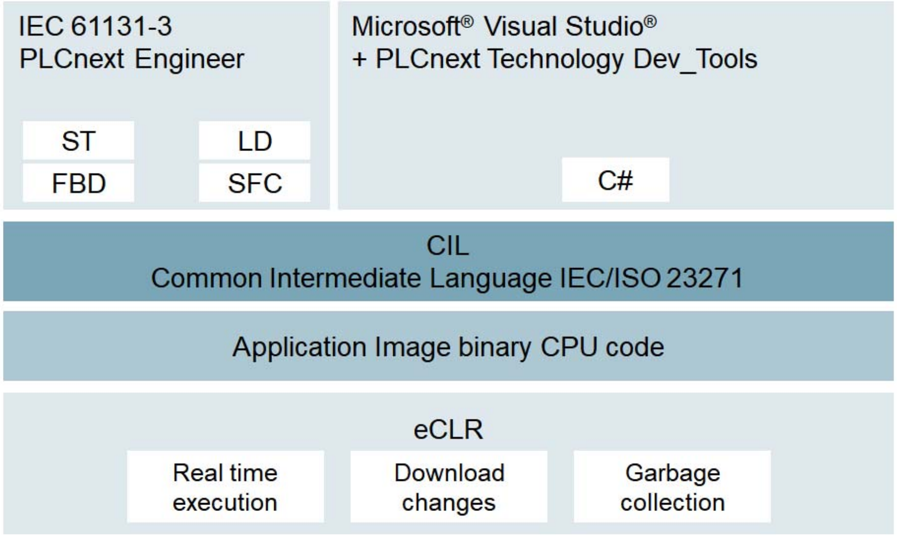

IEC 61131-3
PLCnext Engineer
ST
LD
FBD
SFC
Microsoft® Visual Studio®
+ PLCnext Technology Dev_Tools
C#
CIL
Common Intermediate Language IEC/ISO 23271
Application Image binary CPU code
eCLR
Real time execution
Download changes
Garbage collection

Figure 7-1 Structure of the eCLR programming system

Additional information

In addition to the information in the following sections, further information is available in the online help “eCLR Programming Reference” and the “Readme.txt” text file. Both the online help and the text file are available in every eCLR project in Visual Studio®.

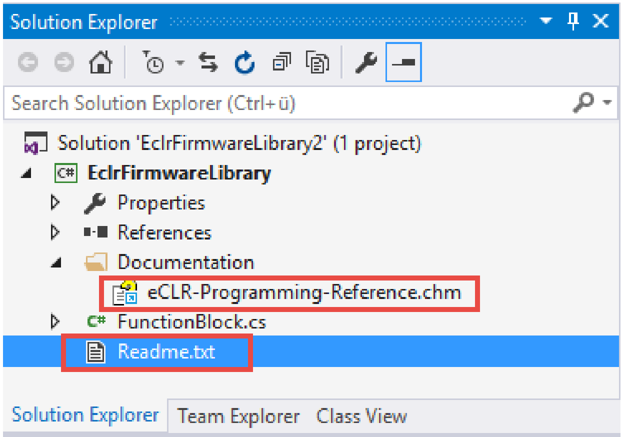


Figure 7-2 “eCLR-Programming-Reference.chm” and “Readme.txt”

The eCLR programming system consists of the following components:

CIL compiler

The CIL compiler is responsible for translating the CIL code (CIL = Common Intermediate Language). The eCLR CIL compiler is an ahead-of-time compiler. This means that the CIL code has been fully translated for the platform before being transferred to the controller (more importantly, before execution). This fulfills an essential requirement for the real-time capability of the system. The functional scope of the CIL compiler enables the use of many C # language elements and namespaces of the Base Class Library, which are listed in Section "C# language functions" on page 178.

eCLR core libraries

The core libraries consist of eCLR base class libraries (mscorlib.dll, System.dll, System.Core.dll) and some eCLR-specific libraries (eclrlib.dll, pcoslib.dll). The eCLR base class libraries implement the .NET Framework class libraries. An overview of the supported functions is available in Section "eCLR Base Class Libraries Reference" of the "eCLR Programming Reference" online help (see "Additional information" on page 166).

eCLR runtime

The eCLR runtime executes the compiled IL code on the controller. The runtime is responsible for object and memory management, metadata processing (e.g., for debugging), as well as for the transition of managed API calls to native implementation specific to the operating system. An overview of the eCLR runtime is available in Section "eCLR runtime functions" on page 181.

7.1 Installing the Visual Studio® extension

7.1.1 System requirements

- Before starting the installation, ensure that the system requirements are met and download the necessary software.

Operating system

– Microsoft® Windows® 7
– Microsoft® Windows® 8.1
– Microsoft® Windows® 10

Phoenix Contact software

– PLCnext Engineer (Order No. 1046008):

To use the libraries created with C# for PLCnext Technology devices, you need the engineering software platform for automation controllers.

– PLCnext Technology controller with eCLR as of version 2019.0 LTS (e.g., AXC F 2152)
– Visual Studio® extension installer

The Visual Studio® extension (tools for Visual Studio 2015 for PLCnext Technology C# programming) are available in the download area of your controller or the PLCnext Engineer software (Order No. 1046008) at phoenixcontact.net/products.

C# development environment

– Microsoft® Visual Studio® 2015 and 2017
Versions: Enterprise, Pro, Community

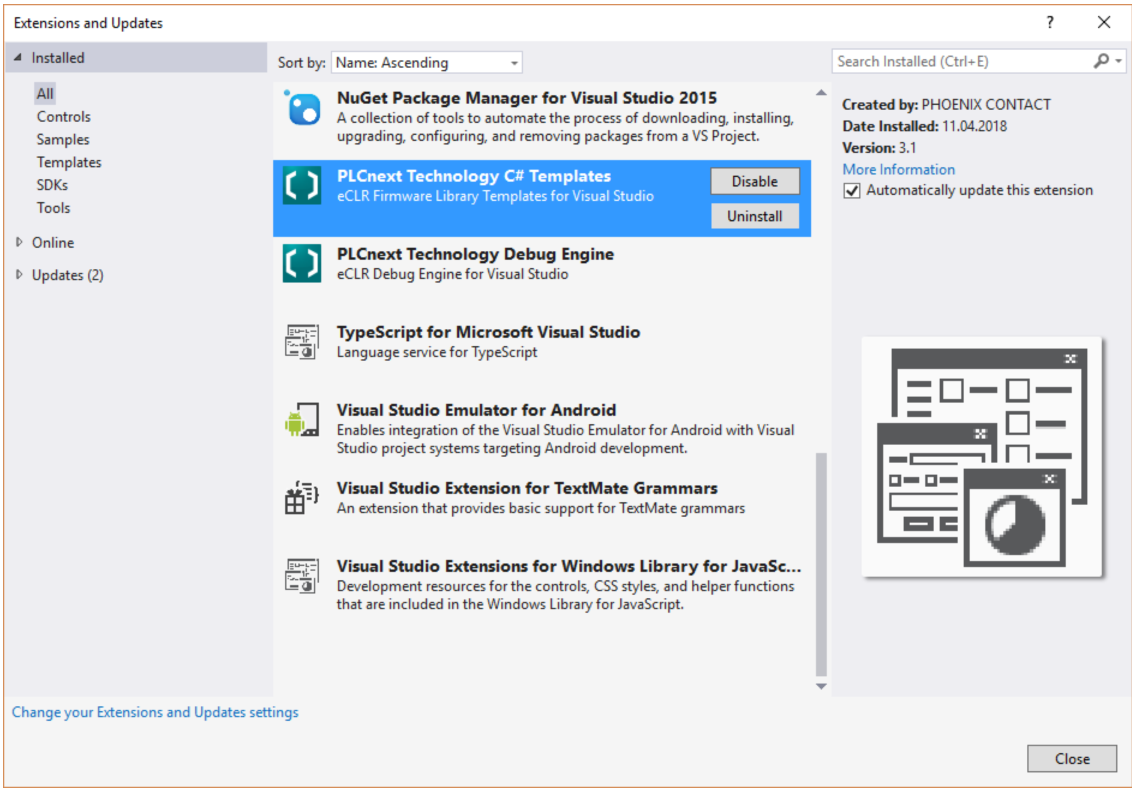
7.1.2 Installation

You need the Microsoft® Visual Studio® development environment for programming in C#. The Phoenix Contact extension available for Visual Studio® makes it easy to use C# programs in the PLCnext Technology context.

Installing the Visual Studio® extension

- To install the Visual Studio® extension, run the Windows installation program of the Visual Studio® extension.
  This adds the project and element templates and the debug module to Visual Studio®.
- Check if installation was successful by opening the “Tools, Extensions and Updates” menu in the Visual Studio® development environment.

If the extensions have been successfully installed, they are displayed as shown in Figure 7-3:



NuGet Package Manager for Visual Studio 2015
A collection of tools to automate the process of downloading, installing, upgrading, configuring, and removing packages from a VS Project.

PLCnext Technology C# Templates
eCLR Firmware Library Templates for Visual Studio

PLCnext Technology Debug Engine
eCLR Debug Engine for Visual Studio

TypeScript for Microsoft Visual Studio
Language service for TypeScript

Visual Studio Emulator for Android
Enables integration of the Visual Studio Emulator for Android with Visual Studio project systems targeting Android development.

Visual Studio Extension for TextMate Grammars
An extension that provides basic support for TextMate grammars

Visual Studio Extensions for Windows Library for JavaSc...
Development resources for the controls, CSS styles, and helper functions that are included in the Windows Library for JavaScript.

Created by: PHOENIX CONTACT
Date Installed: 11.04.2018
Version: 3.1

☑ Automatically update this extension

Figure 7-3 “Extensions and Updates” menu

Uninstalling the Visual Studio® extension

If you want to uninstall the Visual Studio® extension, proceed as follows:

- Run the Windows installation program of the Visual Studio® extension.
- In the next dialog, select “Remove”.

You can also use the Windows® “Programs and Features” menu to uninstall the Visual Studio® extension.

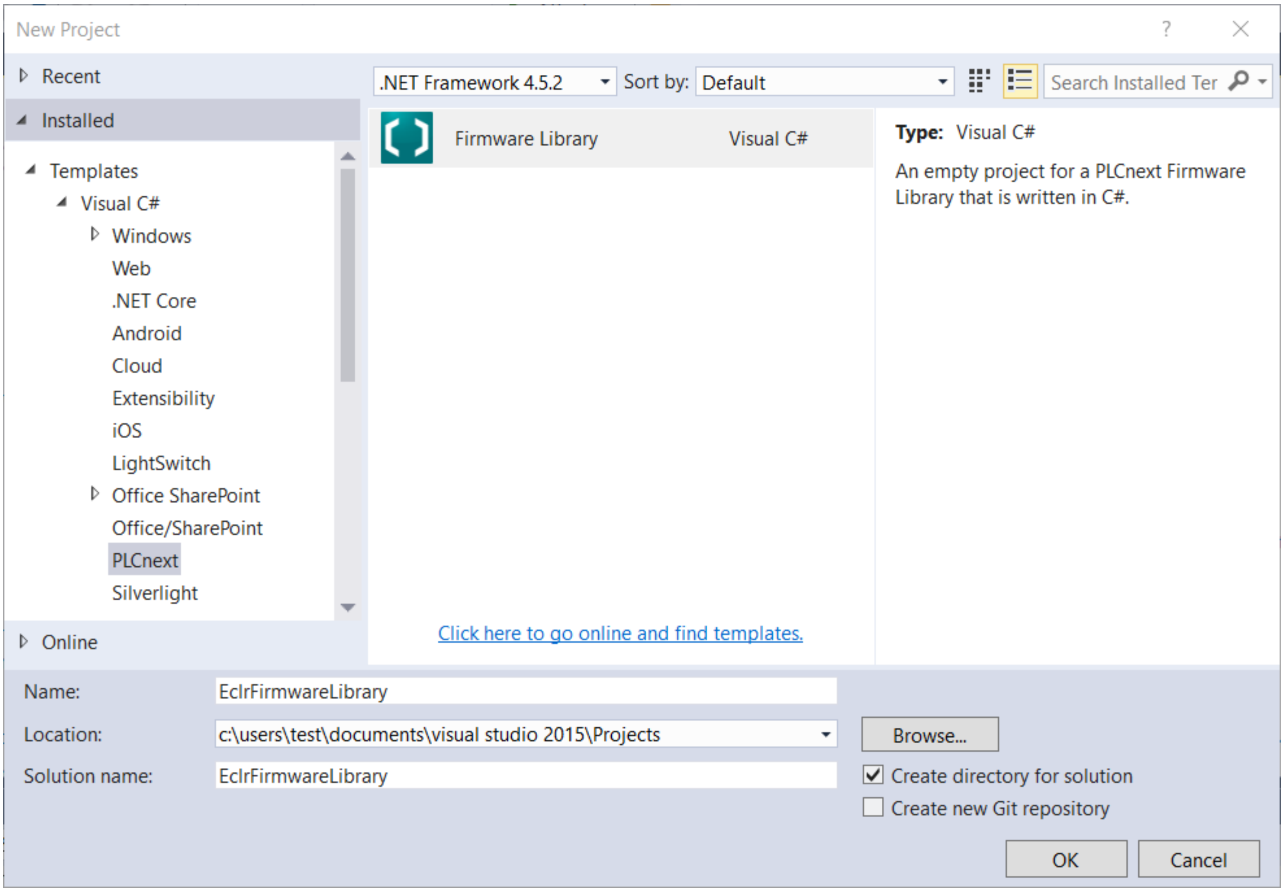
- Select the extension from the list.
- Click on “Uninstall”.

7.2 Creating a firmware library

With Visual Studio® you can create functions and function blocks in C#, which you can subsequently import using the PLCnext Engineer software, and use on a PLCnext Technology device. You first have to create a new project. To do this, proceed as follows:

- In Visual Studio®, open the “File, New, Project...” menu.
- In the dialog that opens, select “Installed, Templates, Visual C#, PLCnext”.
- Select the “Firmware Library” eCLR template from the list.
- Select a storage path and click on “OK”.

A new, empty eCLR library project is created.



Type: Visual C#

An empty project for a PLCnext Firmware Library that is written in C#.

Name: EclrFirmwareLibrary

Location: c:\users\test\documents\visual studio 2015\Projects

Solution name: EclrFirmwareLibrary

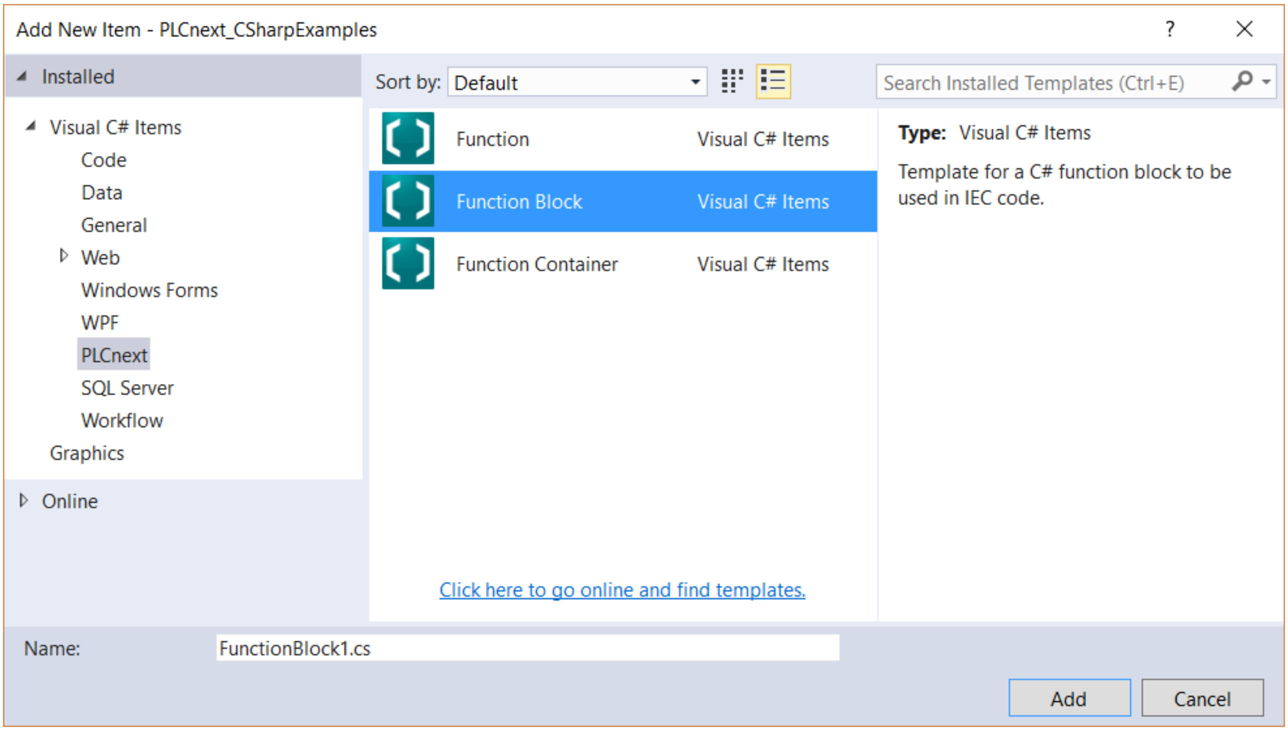
☑ Create directory for solution

□ Create new Git repository

Figure 7-4 Creating a new PLCnext library project

- Open the “Project, Add New Item” menu.
- In the dialog that opens, select “Installed, Visual C# Items, eCLR”.
- Select the “Function Block” or “Function” element from the list.
- Click on “Add”.

⇒ By doing so you create a new template for a function or a function block.


Add New Item - PLCnext_CSharpExamples

Sort by: Default

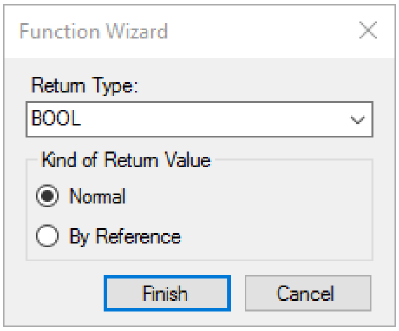
Type: Visual C# Items

Template for a C# function block to be used in IEC code.

Name: FunctionBlock1.cs

Figure 7-5 Adding a “Function Block”

- When you create a function, select a return value in the following dialog.


Function Wizard

Return Type: BOOL

Kind of Return Value

● Normal

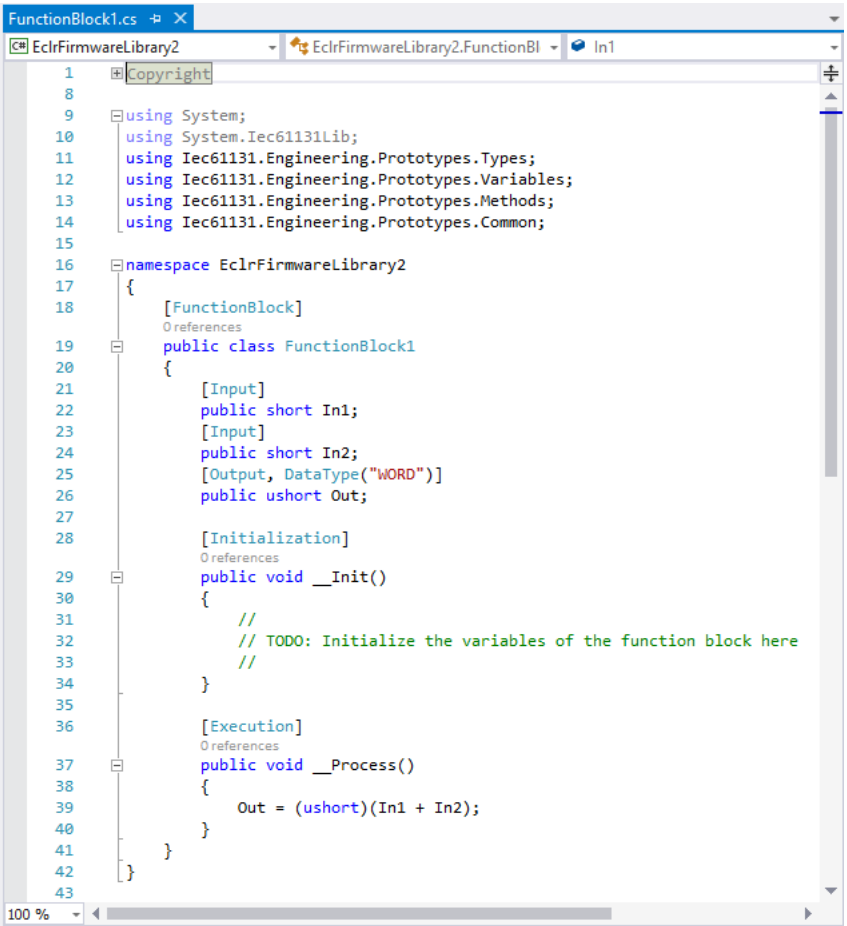
○ By Reference

Figure 7-6 Adding a new “Function” element: selecting a return value

“Kind of Return Value”:
“By Reference” is recommended for complex data types, and is mandatory for generic data types (ANY).

- Create your function block or function in the template.
- Observe the comments in the template.

- Create the entire project by pressing <F6>. Depending on your configuration, a *.pcwlx library is created in the release or debug directory of the project.



```
Copyright

using System;
using System.Iec61131Lib;
using Iec61131.Engineering.Prototypes.Types;
using Iec61131.Engineering.Prototypes.Variables;
using Iec61131.Engineering.Prototypes.Methods;
using Iec61131.Engineering.Prototypes.Common;

namespace EclrFirmwareLibrary2
{
    [FunctionBlock]
    0 references
    public class FunctionBlock1
    {
        [Input]
        public short In1;
        [Input]
        public short In2;
        [Output, DataType("WORD")]
        public ushort Out;

        [Initialization]
        0 references
        public void __Init()
        {
            //
            // TODO: Initialize the variables of the function block here
            //
        }

        [Execution]
        0 references
        public void __Process()
        {
            Out = (ushort)(In1 + In2);
        }
    }
}
```

Figure 7-7 Example: eCLR function block template

- Also observe the information in the “Known Issues and Constraints” section of the online help (see “Additional information” on page 166).

7.3 Remote debugging of C# code with Visual Studio®

NOTE: Risk of damage to equipment

If safety functions are switched off, the controller must not be used for live operation.

Using the debugging function can result in an unsafe process interruption.

- Ensure that there is no risk of damage to the equipment or personal injury.

Because PLCnext Technology includes an implementation of eCLR, you have the option to establish a C# remote session. The following sections describe the basic steps needed for setting up a remote debug session and establishing a direct connection to a running application by means of Visual Studio®.

Software used

– PuTTY
– WinSCP
– Microsoft® Visual Studio® 2015 and 2017
– Visual Studio® extension

Please note:

– The following description is based on the default settings. Consider all changes made by the user.

7.3.1 Disabling user authentication

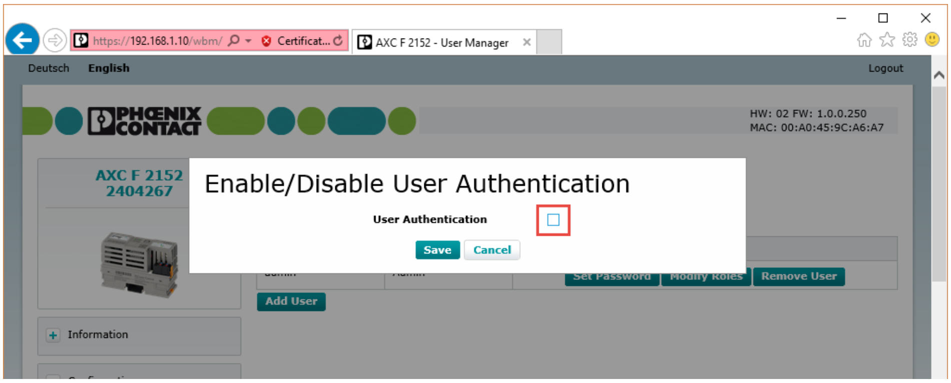
NOTE: Risk of damage to equipment

If safety functions are switched off, the controller must not be used for live operation.

Currently, the C# debugging function does not feature an interface for logging on to the controller. To debug the C# code, you have to deactivate the user authentication in WBM (web-based management) of the controller.

For additional information, please refer to Section 3.9.1 on page 82.

- Deactivate user authentication of your controller via WBM.



https://192.168.1.10/wbm/

AXC F 2152 - User Manager

HW: 02 FW: 1.0.0.250
MAC: 00:A0:45:9C:A6:A7

AXC F 2152
2404267

Enable/Disable User Authentication

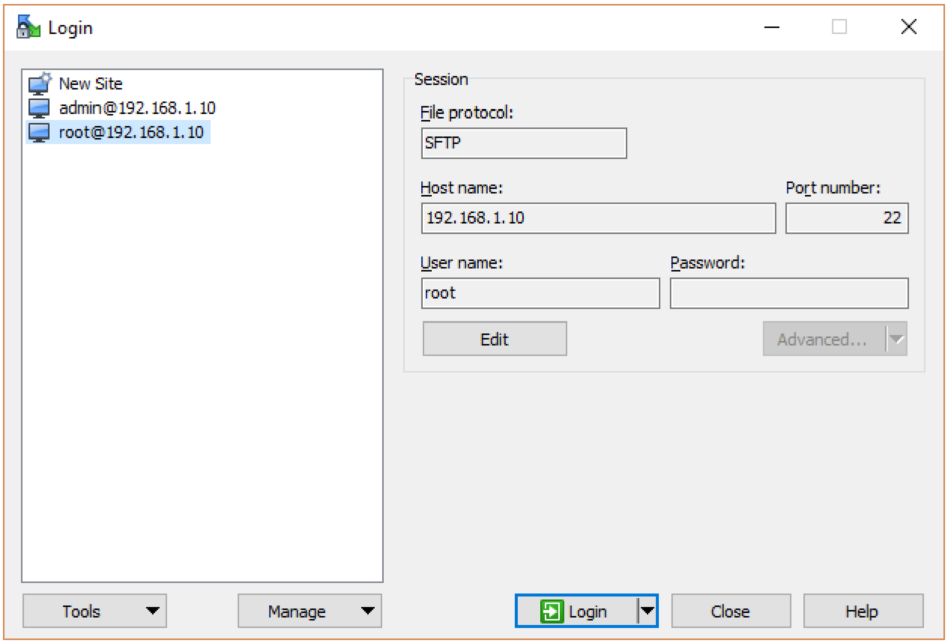
User Authentication □

Figure 7-8 Example WBM for AXC F 2152: Deactivating user authentication

7.3.2 Opening a port and deactivating TLS (Transport Layer Security)

Currently, C# debugging via a secure communication connection is not supported. For this reason, another port has to be opened for communication with the controller. TLS (Transport Layer Security) has to be deactivated on this port.

- Open WinSCP.
- Establish a connection to the IP address of the controller and a user with root rights (see Section 2.14.7, “Root rights”).



New Site
admin@192.168.1.10
root@192.168.1.10

Session

File protocol: SFTP

Host name: 192.168.1.10

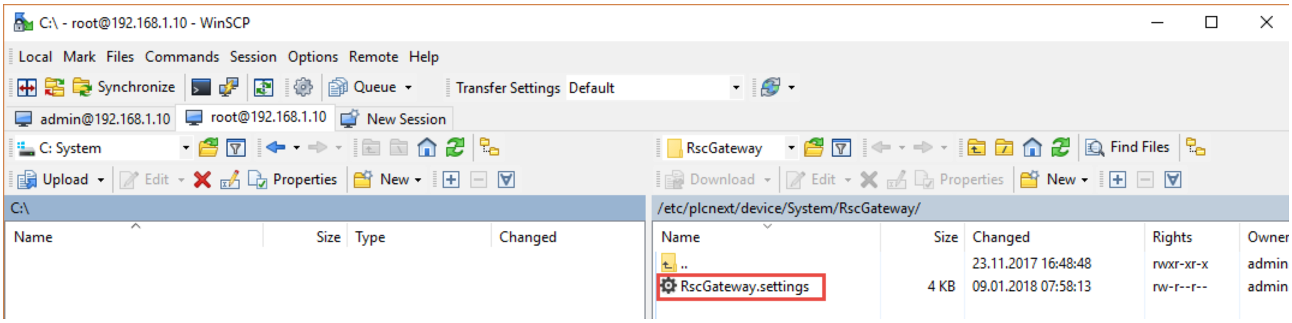
Port number: 22

User name: root

Password:

Figure 7-9 Establishing a connection

- Open the /etc/plcnext/device/System/RscGateway/ folder.
- Open the “RscGateway.settings” file.



C:\

Name	Size	Type	Changed

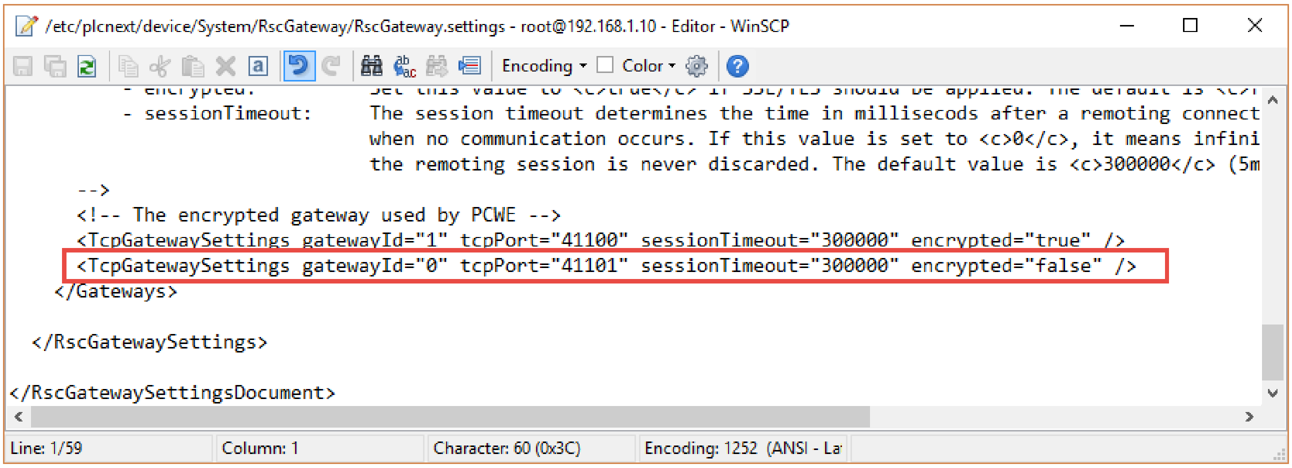
/etc/plcnext/device/System/RscGateway/

Name	Size	Changed	Rights	Owner
..		23.11.2017 16:48:48	rwxr-xr-x	admin
RscGateway.settings	4 KB	09.01.2018 07:58:13	rw-r--r--	admin

Figure 7-10 “RscGateway” directory with “RscGateway.settings” file

- Scroll to the line with the “TcpGatewaySettings”.

- Insert the line
  `<TcpGatewaySettings gatewayId=”0” tcpPort="41101” sessionTimeout="300000” encrypted="false” />`
  as shown in the figure below:


/etc/plcnext/device/System/RscGateway/RscGateway.settings - root@192.168.1.10 - Editor - WinSCP
Encoding
Color
Line: 1/59
Column: 1
Character: 60 (0x3C)
Encoding: 1252 (ANSI - La

```
        - encrypted:        Set this value to <c>true</c> if SSL/TLS should be applied. The default is <c>t
        - sessionTimeout:   The session timeout determines the time in millisecods after a remoting connect
                            when no communication occurs. If this value is set to <c>0</c>, it means infini
                            the remoting session is never discarded. The default value is <c>300000</c> (5m
      -->
      <!-- The encrypted gateway used by PCWE -->
      <TcpGatewaySettings gatewayId="1" tcpPort="41100" sessionTimeout="300000" encrypted="true" />
      <TcpGatewaySettings gatewayId="0" tcpPort="41101" sessionTimeout="300000" encrypted="false" />
    </Gateways>

  </RscGatewaySettings>

</RscGatewaySettingsDocument>
```

Figure 7-11 “RscGateway.settings”

- Save and close the file.
- Restart the controller.

7.3.3 Debug mode

Creating/opening a project

- Create a new project in Visual Studio®. Set it up.
- Follow the operating instructions in Section “Creating a firmware library” on page 169
  Or
- Open an existing project

Importing a library in PLCnext Engineer

- Import the library in PLCnext Engineer (see Section “Importing libraries into PLCnext Engineer” on page 184).

Activating debug mode

To activate debug mode, proceed as follows:

- Connect PLCnext Engineer to the controller. Additional information on PLCnext Engineer is available in the online help and in the quick start guide. The quick start guide can be downloaded at phoenixcontact.net/products.
- In the “PLANT” area, right-click on the controller.

- In the context menu, select “Debug On/Off”.

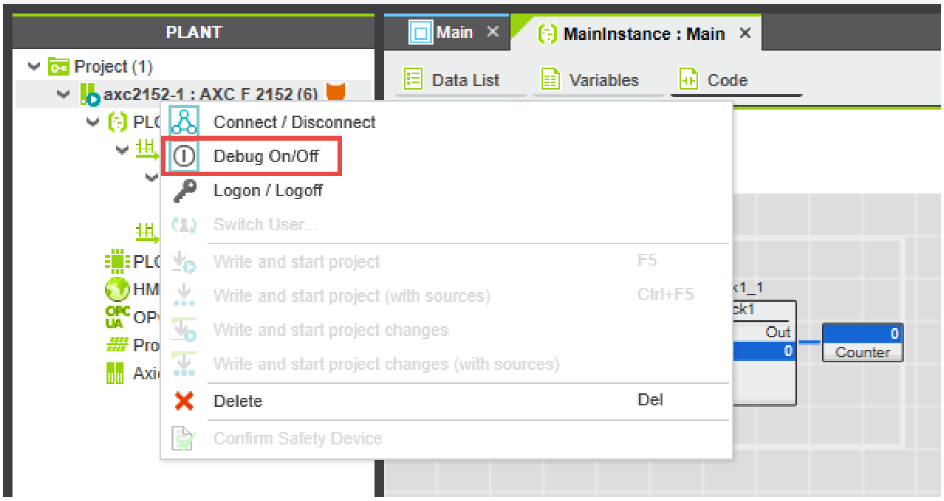
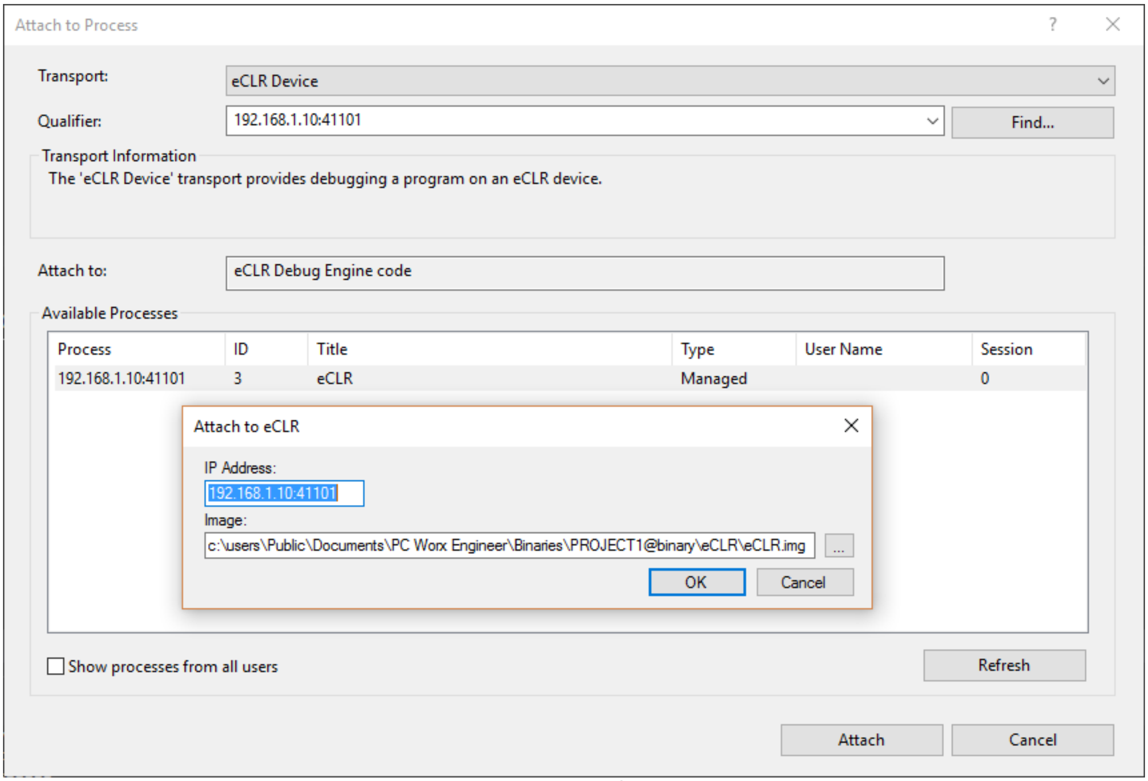


Figure 7-12 Activating debug mode in PLCnext Engineer

Attaching Visual Studio® to the process

- In Visual Studio® open the “Debug, Attach to Process...” menu.
- From the “Transport” drop-down list, select the “eCLR Device” entry.
- Click on the “Find...” button.
- In the “Attach to eCLR” dialog that opens, enter the IP address of the controller (e.g., 192.168.1.10:41101) and port number 41101.
- Select the image under the following path: “c:\users\Public\Documents\PLCnext Engineer\Version\Binaries\PROJECT1@binary\eCLR\”.

A shortcut to this path is available in the “Select Image File” dialog. It can be found under “Microsoft Visual Studio 2015”, “Binaries”.



Attach to Process

Transport: eCLR Device

Qualifier: 192.168.1.10:41101

Transport Information

The 'eCLR Device' transport provides debugging a program on an eCLR device.

Attach to: eCLR Debug Engine code

Available Processes

Process	ID	Title	Type	User Name	Session
192.168.1.10:41101	3	eCLR	Managed		0

Attach to eCLR

IP Address: 192.168.1.10:41101

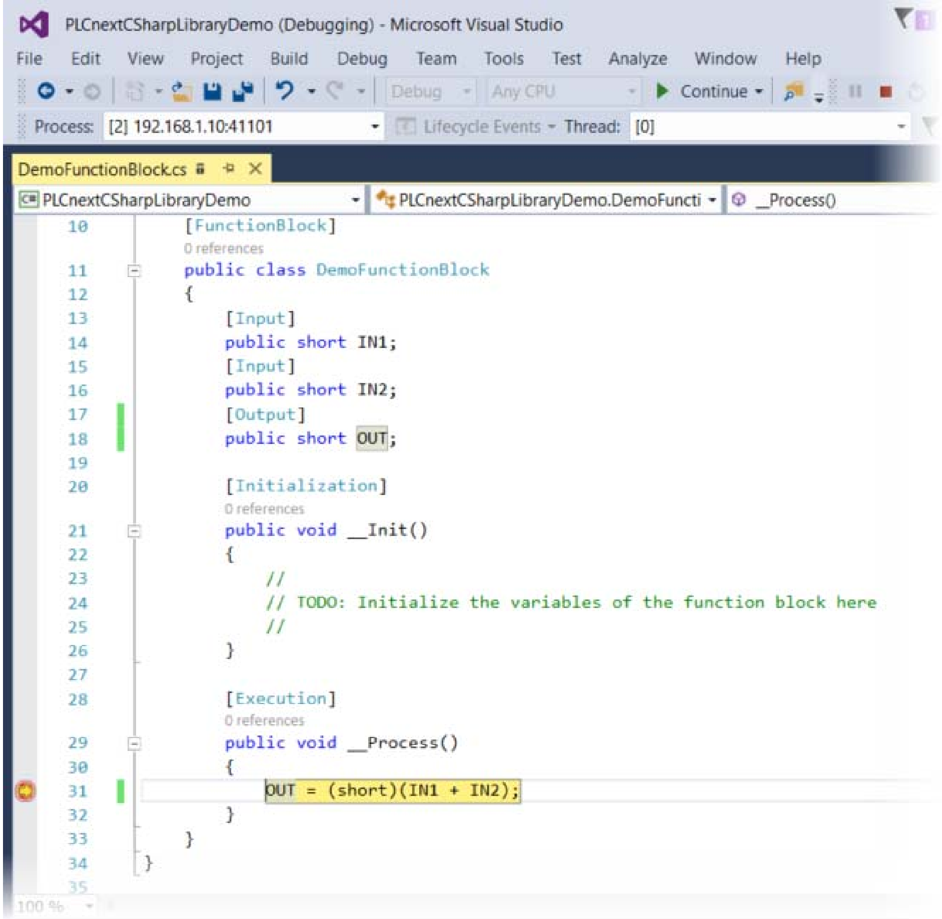
Image: c:\users\Public\Documents\PC Worx Engineer\Binaries\PROJECT1@binary\eCLR\eCLR.img

☐ Show processes from all users

Figure 7-13 Attaching Visual Studio® to the process

- Select the image currently running on the controller.
- Confirm your selection with “OK”.
- Click on the “Attach” button.
- Then add breakpoints in Visual Studio.

Execution of the code is stopped at the breakpoint. You can now evaluate the current variable values.



```
    [FunctionBlock]
    0 references
    public class DemoFunctionBlock
    {
        [Input]
        public short IN1;
        [Input]
        public short IN2;
        [Output]
        public short OUT;

        [Initialization]
        0 references
        public void __Init()
        {
            //
            // TODO: Initialize the variables of the function block here
            //
        }

        [Execution]
        0 references
        public void __Process()
        {
            OUT = (short)(IN1 + IN2);
        }
    }
}
```

Figure 7-14 Setting breakpoints in Visual Studio

- Use the functions in Visual Studio to jump to the next breakpoint or continue processing of the code.

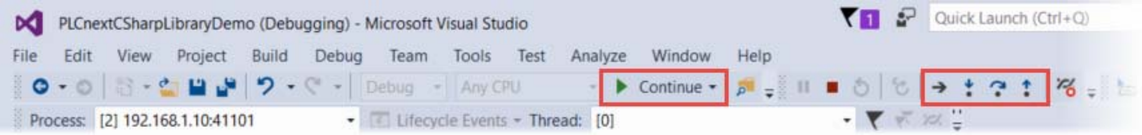
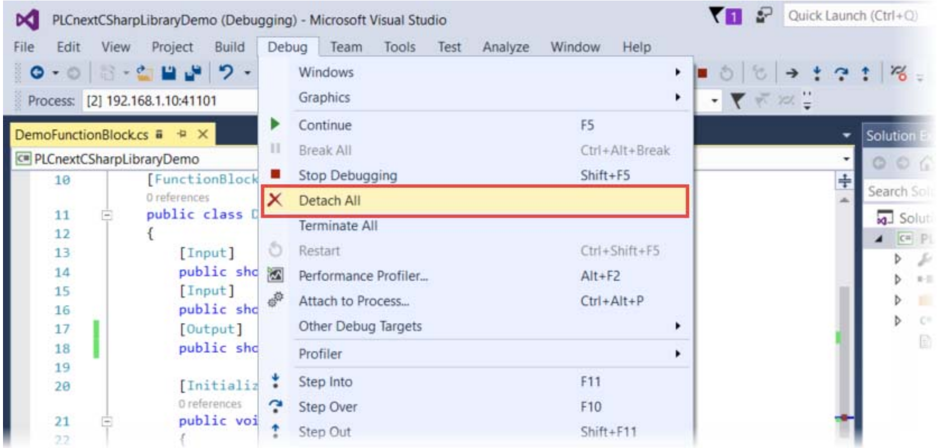


Figure 7-15 Continuing code processing or jumping to the next breakpoint

If you want to make changes to the code within the framework of debugging, you have to disconnect the process connection (“Detach“):

- In Visual Studio, select the “Debug, Detach All” menu.



```
[FunctionBlock
0 references
public class D
{
    [Input]
    public sho
    [Input]
    public sho
    [Output]
    public sho

    [Initializ
    0 references
    public voi
    {
```

Figure 7-16 Visual Studio® - “Detach All”

- Make changes to the code and save them.
- Recompile the project. To do so, open the “Build, Rebuild Project name” menu.
- Then, switch to PLCnext Engineer.
- PLCnext Engineer recognizes the change. Save the change.

> If the inputs/outputs have been changed, errors might occur in PLCnext Engineer. If necessary, correct them in PLCnext Engineer.

- Transfer the project to the controller.
- Then, reconnect Visual Studio to the process (see Section “Attaching Visual Studio® to the process” on page 175).

For information on the debug functions in PLCnext Engineer and in Microsoft Visual Studio®, please refer to the associated documentation.

7.4 Supported functions of the PLCnext Technology C# programming system

7.4.1 C# language functions

Types

All the integrated types are supported with the exception of decimal.

Type system

- Namespaces, structs, classes, interfaces, enumerations, nested types
- Indexers, properties, operations
- Events, delegates, MultiCastDelegates
- Arrays
- Constructors, static constructors, destructors (finalizers)
- Boxing, unboxing, static casts

- Nullable

Polymorphy

- Virtual mechanism (virtual, overwrite, abstract)
- Dynamic casts (as)

Modifiers and keywords

- public, internal, protected, private
- readonly, const, sealed, unsafe
- params, ref, out
- base, this
- explicit, implicit, operator

Operators

- new, sizeof, typeof, as, is
- All unary operators
- All binary operators
- Prefix, postfix and conditional operator
- Cast and index operator

Control structures and statements

- if, else
- switch, case, default (also on strings)
- for, do, while, foreach, break, continue
- goto, return
- using, fixed
- lock

Exceptions

- throw
- try, catch, finally

For additional information, please refer to Section “Known issues and constraints” on page 181.

7.4.2 Base class libraries

Below you will find a list of the most important classes that have been implemented in the eCLR base class library, separated by namespaces. A complete list of the implemented classes with additional information is available in Section “eCLR Base Class Libraries Reference” of the “eCLR-Programming-Reference.chm” online help (see “Additional information” on page 166).

System

All integrated types, as well as the following:

- Char, Boolean, Int32, Single, etc., with the exception of decimal
- All common exception types

- Common interfaces such as:
  IDisposable, IComparable, ICloneable, IFormatProvider, IFormattable
- String, array, object, valuetype
- Version, DateTime, TimeSpan, TimeZone
- BitConverter, Convert
- GC
- Console
- Uri, UriBuilder

System.Collections

- IEnumerator, IEnumerable, ICollection, IList, IDictionary, IComparer
- ArrayList, Hashtable, Queue
- Comparer, DictionaryEntry

System.Collections.Generic

- IEnumerator<T>, IEnumerable<T>, ICollection<T>, ICollection<T>, IList<T>, IDictionary<TKey,TValue>, IComparer<T>
- Dictionary<TKey,TValue>, List<T>, Queue<T>
- Comparer<T>, KeyValuePair<TKey,TValue>

System.Globalization

- Calendar, CalendarWeekRule
- CultureInfo
  Supported cultures: de-DE, en-US, en-GB, Invariant
  Types that support the localized formatting/parsing: all number types, DateTime, TimeSpan
- DateTimeFormatInfo, NumberFormatInfo, NumberStyles, DateTimeStyles

System.IO

- Path, File
- Stream, FileStream, MemoryStream
- BinaryReader, BinaryWriter
- TextReader, TextWriter, StreamReader, StreamWriter, StringReader, StringWriter

System.Security.Cryptography

- CryptoConfig
- HashAlgorithm, MD5, MD5CryptoServiceProvider

System.Text

- StringBuilder
- Encoding, ASCIIEncoding, UTF8Encoding, UnicodeEncoding (big endian and little endian)

System.Threading

- Thread, ThreadPool, ThreadStart, ThreadState
- Monitor (lock)
- WaitHandle, EventWaitHandle, AutoResetEvent, ManualResetEvent, WaitCallback

- Timer, TimerCallback

System.Net

- IPAdress, IPEndpoint, Endpoint, SocketAddress
- WebClient, HttpWebRequest, HttpWebResponse

System.Net.Sockets

- NetworkStream
- AddressFamily, ProtocolType
- Socket, SocketType

7.4.3 eCLR runtime functions

Garbage collection

Memory management is performed by the eCLR. The garbage collector should explicitly only be called at a specific point in time (application of GC.Collect).

Implicit finalization

The finalizer of each applicable .NET class (implementing ~T()) is called when teh garbage collector releases the instance. The garbage collector collect function is called automatically. The time of the implicit call cannot be predicted.

Implicit initialization

All the integrated types and value types (struct) are implicitly initialized with their default values. Reference type instances are implicitly initialized with the value “zero”.

Debug support

- Defining breakpoints
- Evaluation of instance values and local variables, as well as arguments from the current method
- Providing call stack information

Known issues and constraints

For detailed information, please refer to Section “Known Issues and Constraints” of the Visual Studio® extension online help (see “Additional information” on page 166).

7.5 Supported data types

The following table illustrates how the IEC 61131-3 data types are linked to the .Net Framework and C#. Variables of data types that are marked with a “+” in the “Attribute data type” column must have the optional “DataType” attribute for unique assignment.

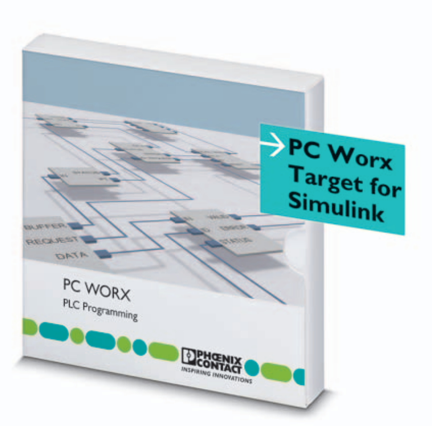
Table 7-1 Supported IEC 61131-3, .Net Framework, and C# data types

IEC 61131-3	.Net Framework	C#	Attribute data type
BOOL	System.Boolean	bool	-
SINT	System.SByte	sbyte	-
INT	System.Int16	short	-
DINT	System.Int32	int	-
LINT	System.Int64	long	-
USINT	System.Byte	byte	-
UINT	System.UInt16	ushort	-
UDINT	System.UInt32	uint	-
ULINT	System.UInt64	ulong	-
REAL	System.Single	float	-
LREAL	System.Double	double	-
TIME	System.UInt32	uint	+
LTIME	System.Int64	long	+
LDATE	System.Int64	long	+
LTOD	System.Int64	long	+
LDT	System.Int64	long	+
BYTE	System.Byte	byte	+
WORD	System.UInt16	ushort	+
DWORD	System.UInt32	uint	+
LWORD	System.UInt64	ulong	+
STRING	System.Iec61131Lib.IecStringEx		-
ANY	System.Iec61131Lib.Any		+
ANY_MAGNITUDE	System.Iec61131Lib.Any		+
ANY_NUM	System.Iec61131Lib.Any		+
ANY_INT	System.Iec61131Lib.Any		+
ANY_SIGNED	System.Iec61131Lib.Any		+
ANY_UNSIGNED	System.Iec61131Lib.Any		+
ANY_REAL	System.Iec61131Lib.Any		+
ANY_BIT	System.Iec61131Lib.Any		+
ANY_ELEMENTARY	System.Iec61131Lib.Any		+

8 Matlab® Simulink®

Matlab® Simulink® is a software program for the model-based development of dynamic systems.

The “PC Worx Target for Simulink” software add-on (Order No. 2400041) enables the conversion of Simulink® models into device-specific code for use with Remote Field und Axioline controllers. The models converted from Simulink® can be integrated into the PC Worx and PLCnext Engineer development environments.



The model-based design and versatile simulation possibilities of Simulink® can therefore also be used for automation projects within PLCnext Technology. A structured model implementation can be ensured, thanks to the automatic creation of executable code. The import and configuration options in PLCnext Engineer also enable Simulink® models to be operated together with programs that have been created in IEC 61131-3 languages or C++. The combination with other high-level-language programs in the same task and execution in the real-time context is therefore also possible for Simulink® models. Furthermore, you can also implement monitoring of model parameters, and optimization of the models during execution via “external mode”.

For additional information on the add-on and its use, please refer to:

– PC Worx Target for Simulink product page (Order No. 2400041):
  phoenixcontact.com/webcode/#1955
– Tutorial videos in the PLCnext Community:
  plcnext-community.net/index.php?option=com_content&view=category&layout=blog&id=71&Itemid=339&lang=en

9 Importing libraries into PLCnext Engineer

You will find additional information on PLCnext Engineer in the online help and the quick start guide (PLCNEXT ENGINEER, Order No. 1046008).

Once you have generated a *.pcwlx library from a C++ or C# library, import this library into the PLCnext Engineer software. In PLCnext Engineer, imported libraries are treated as an ICE 61131-3 program or as a function or function block, and are processed in tasks.

To import a library into PLCnext Engineer, proceed as follows:

- Open your PLCnext Engineer project or create a new AXC F 2152 template.
- In the “COMPONENTS” area, click on “References”.
- Right-click on “Libraries”.

Adding a library

- Right-click to open the “Add Library” context menu.

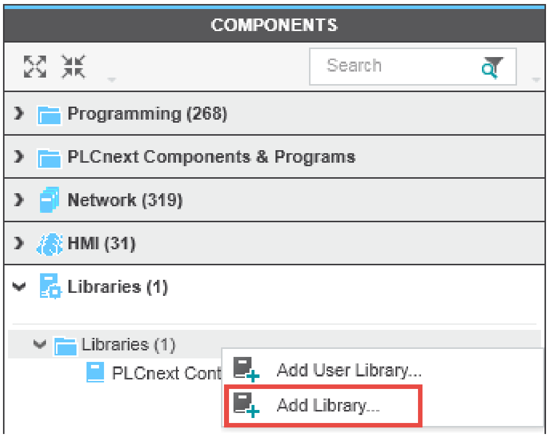


Figure 9-1 “Add library” context menu

- In the dialog that opens, select the desired *.pcwlx library.
- Click on “Open”.

The component and program or the function or function block are now available in the “COMPONENTS” area under “Libraries”.

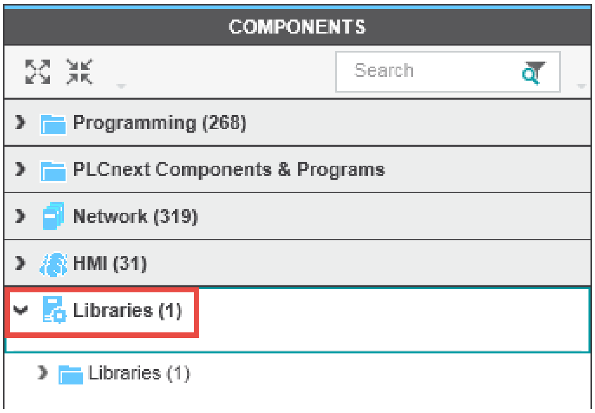


Figure 9-2 PLCnext Engineer - Libraries

Adding functions/function blocks

The library is now a part of the PLCnext Engineer project. The POUs (Program Organization Units) contained therein can be used in the project, for example in the FBD (Function Block Diagram).

- Open the “Main” code worksheet via the “COMPONENTS” area.
- Open the “Code” program editor.
- Add the new C# function block by dragging it from the “COMPONENTS” area under “Libraries” to your code worksheet.

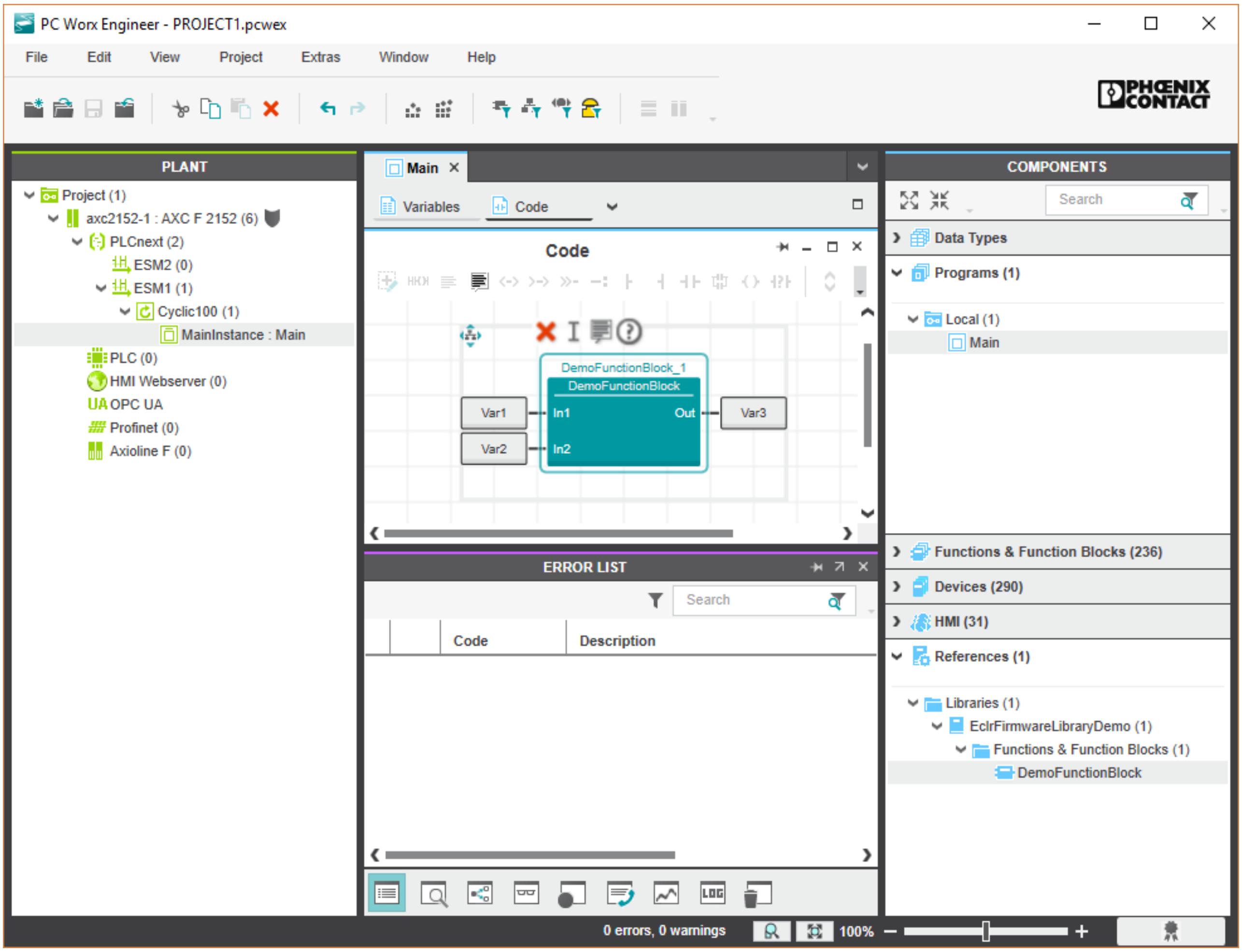


Figure 9-3 Inserting a function block

Assigning inputs/outputs

Additional information is available in the Readme.txt file (see “Additional information” on page 166).

Instantiating a program

To instantiate a program contained in a library, proceed as follows:

- In the “PLANT” area, select the “PLCnext” node.

- Open the “Tasks and Events” editor.

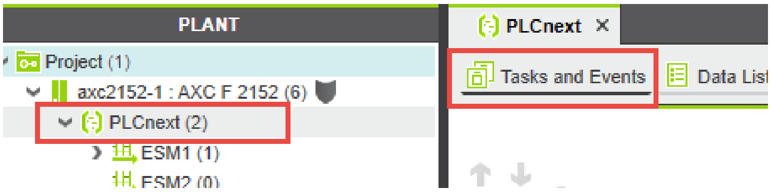
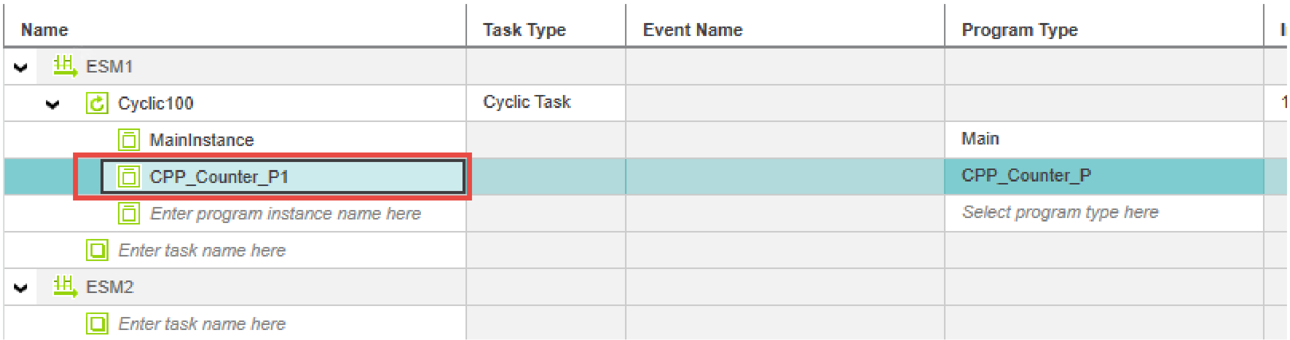


Figure 9-4 Opening the “Tasks and Events” editor

- In the editor, create a new task or open an existing task.
- Drag and drop the program from the “COMPONENTS” area to the “Name” column below the desired tasks. This way you generate a program instance.

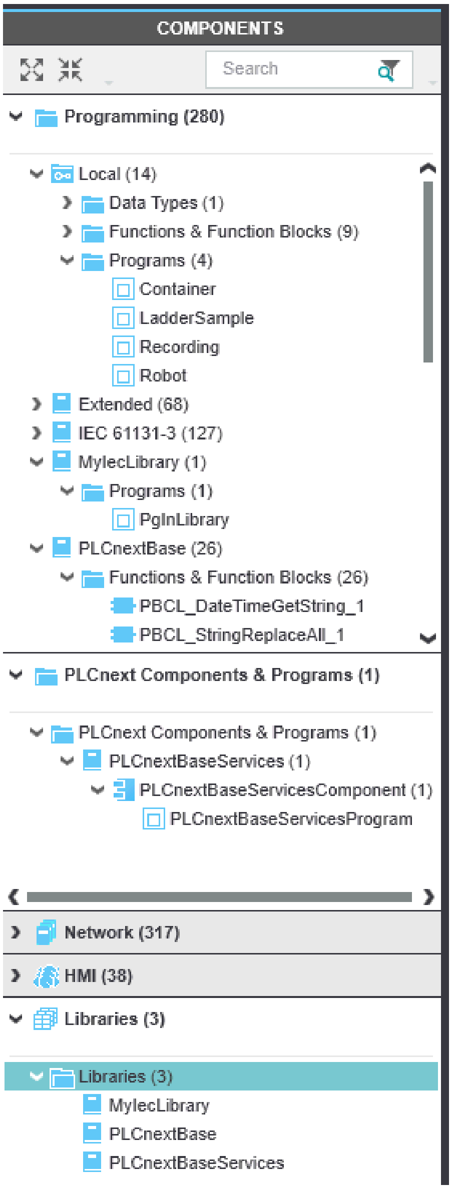


Name	Task Type	Event Name	Program Type	I
ESM1				
Cyclic100	Cyclic Task			1
MainInstance			Main	
CPP_Counter_P1			CPP_Counter_P	
Enter program instance name here			*Select program type here*	
Enter task name here				
ESM2				
Enter task name here				

Figure 9-5 Example: CPP_Counter_P1 program instance in the “Task and Event” editor

The programs in the “COMPONENTS” area are displayed at different locations, depending on the programming language they were written in.

– Programs that were created in IEC 61131 can be found under “Programming”. A distinction is made between:
  – “Local, Programs”: Programs from the same project are located here.
  – “Library name”: Programs from an IEC 61131 library are located here.
– Programs that were created in C++ or Matlab® Simulink® are located under “PLCnext Components & Programs”, “Library name”, “Component name”.
– Functions and function blocks created with C# are located under “PLCnextBase”.



COMPONENTS

Search

- Programming (280)
  - Local (14)
    - Data Types (1)
    - Functions & Function Blocks (9)
    - Programs (4)
      - Container
      - LadderSample
      - Recording
      - Robot
  - Extended (68)
  - IEC 61131-3 (127)
  - MylecLibrary (1)
    - Programs (1)
      - PgInLibrary
  - PLCnextBase (26)
    - Functions & Function Blocks (26)
      - PBCL_DateTimeGetString_1
      - PBCL_StringReplaceAll_1
- PLCnext Components & Programs (1)
  - PLCnext Components & Programs (1)
    - PLCnextBaseServices (1)
      - PLCnextBaseServicesComponent (1)
        - PLCnextBaseServicesProgram
- Network (317)
- HMI (38)
- Libraries (3)
  - Libraries (3)
    - MylecLibrary
    - PLCnextBase
    - PLCnextBaseServices

Figure 9-6 Programs, function blocks and functions in PLCnext Engineer

Assigning IN/OUT ports

- In the “PLANT” area, select the “PLCnext” node.
- Open the “Data List” editor.

In the “Data List” editor, all IN and OUT ports saved to the GDS of the controller are displayed. The IN and OUT ports of the newly instantiated program are displayed. To use the IN/OUT ports of the imported library, you must assign the IN and OUT ports of the imported libraries to the IN and OUT ports of other program instances so that consistent data exchange can take place.

Assigning an IN port to an OUT port

- To assign an IN port to an OUT port, click on “Select IN Port here” in the “IN Port” column.

The role picker opens. Only the IN ports that you can actually assign to the respective OUT port are displayed in the role picker.

- Select the IN port that you want to assign to the relevant OUT port in the role picker.
- The IN port is assigned to the OUT port.
- Proceed as described for other IN ports.

Assigning several IN ports to an OUT port

- Assign an IN port to an OUT port as described in the previous section.
- To assign further IN ports to an OUT port, you have to duplicate the OUT port.
- Right-click on the OUT port and select the “Duplicate Output Port” option in the context menu.

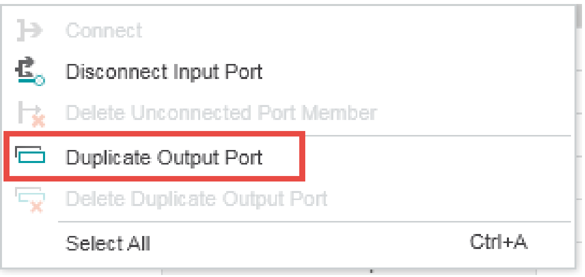


Figure 9-7 Duplicating an output port

- To assign an IN port to the duplicated OUT port, proceed as described in the previous section.

Assigning an OUT port to an IN port

- To assign an OUT port to an IN port, click on “Select OUT Port here” in the “OUT Port” column.

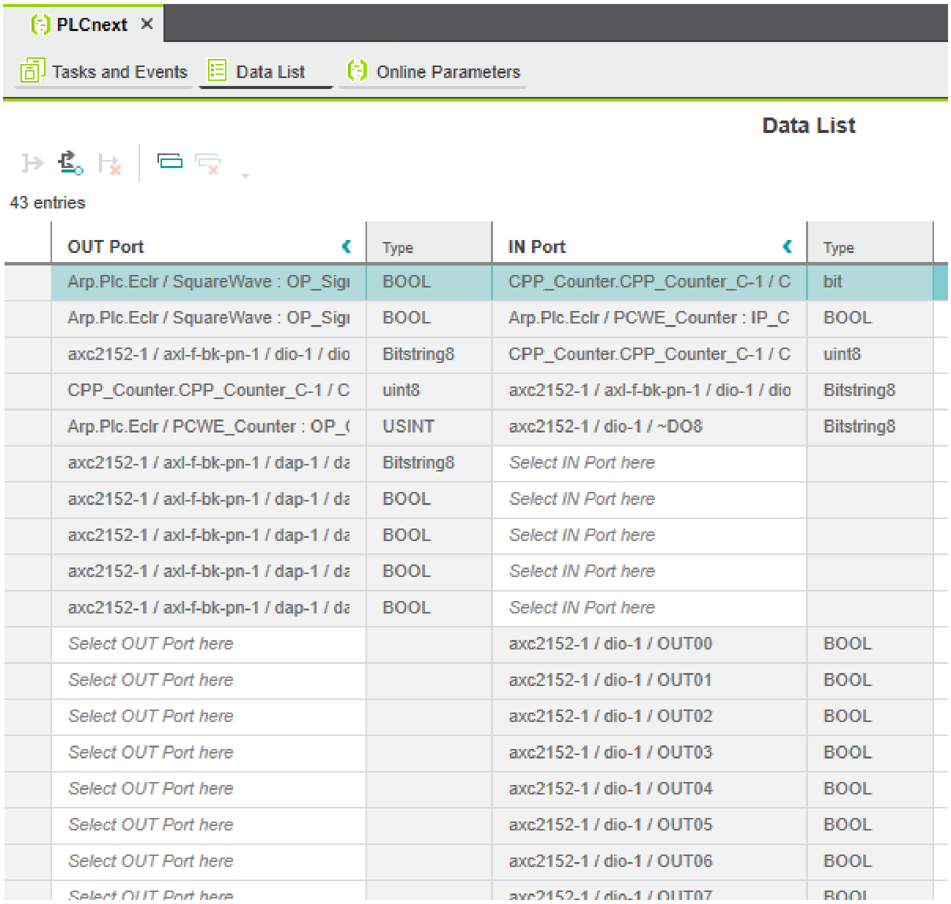
The role picker opens. Only the OUT ports that you can actually assign to the respective IN port are displayed in the role picker.

- Select the OUT port that you want to assign to the relevant IN port in the role picker.

The OUT port is assigned to the IN port.

- Proceed as described for other OUT ports.

- Observe the instructions in Section 2.8.5 “Supported data types“.



Data List

43 entries

	OUT Port	Type	IN Port	Type
	Arp.Plc.Eclr / SquareWave : OP_Sigı	BOOL	CPP_Counter.CPP_Counter_C-1 / C	bit
	Arp.Plc.Eclr / SquareWave : OP_Sigı	BOOL	Arp.Plc.Eclr / PCWE_Counter : IP_C	BOOL
	axc2152-1 / axl-f-bk-pn-1 / dio-1 / dio	Bitstring8	CPP_Counter.CPP_Counter_C-1 / C	uint8
	CPP_Counter.CPP_Counter_C-1 / C	uint8	axc2152-1 / axl-f-bk-pn-1 / dio-1 / dio	Bitstring8
	Arp.Plc.Eclr / PCWE_Counter : OP_(	USINT	axc2152-1 / dio-1 / ~DO8	Bitstring8
	axc2152-1 / axl-f-bk-pn-1 / dap-1 / da	Bitstring8	*Select IN Port here*	
	axc2152-1 / axl-f-bk-pn-1 / dap-1 / da	BOOL	*Select IN Port here*	
	axc2152-1 / axl-f-bk-pn-1 / dap-1 / da	BOOL	*Select IN Port here*	
	axc2152-1 / axl-f-bk-pn-1 / dap-1 / da	BOOL	*Select IN Port here*	
	axc2152-1 / axl-f-bk-pn-1 / dap-1 / da	BOOL	*Select IN Port here*	
	Select OUT Port here		axc2152-1 / dio-1 / OUT00	BOOL
	Select OUT Port here		axc2152-1 / dio-1 / OUT01	BOOL
	Select OUT Port here		axc2152-1 / dio-1 / OUT02	BOOL
	Select OUT Port here		axc2152-1 / dio-1 / OUT03	BOOL
	Select OUT Port here		axc2152-1 / dio-1 / OUT04	BOOL
	Select OUT Port here		axc2152-1 / dio-1 / OUT05	BOOL
	Select OUT Port here		axc2152-1 / dio-1 / OUT06	BOOL
	Select OUT Port here		axc2152-1 / dio-1 / OUT07	BOOL

Figure 9-8 “Data List” editor

Once you have assigned all IN and OUT ports to be used, transfer the PLCnext Engineer project to the controller. Additional information on PLCnext Engineer is available in the online help, the quick start guide (PLCNEXT ENGINEER, Order No. 1046008) and the user manual for the AXC F 2152 controller (Order no. 2404267).

A Appendix

A 1 Available data types

Table A-1 Available data types

Bits	C#	IEC 61131	FDCML	C++	OPC UA
1	BOOL	BOOL	Bit/BOOL	Boolean	Boolean
16	UInt16	WORD	Bitstring16	uint16	UInt16
32	UInt32	DWORD	Bitstring32	uint32	UInt32
64	UInt64	LWORD	Bitstring64	uint64	UInt64
8	Byte	BYTE	Bitstring8	uint8	Byte
8	Byte	BYTE	BYTE	uint8	Byte
32	Int32	DINT	DINT	int32	Int32
32	UInt32	DWORD	DWORD	uint32	UInt32
32	Single	REAL	Float32	float32	Float
64	Double	LREAL	Float64	float64	Double
16	Int16	INT	INT	int16	Int16
64	Int64	LDATE	LDATE	int64	*
64	Int64	LDATE_AND_-TIME	LDATE_AND_-TIME	int64	*
64	Int64	LDT	LDT	int64	*
64	Int64	LINT	LINT	int64	Int64
64	Double	LREAL	LREAL	float64	Double
64	Int64	LTIME	LTIME	int64	*
64	Int64	LTIME_OF_DAY	LTIME_OF_DAY	int64	*
64	Int64	LTOD	LTOD	int64	*
64	UInt64	LWORD	LWORD	uint64	UInt64
8	Byte	BYTE	Octetstring1	uint8	Byte
16	UInt16	WORD	Octetstring2	uint16	UInt16
32	UInt32	DWORD	Octetstring4	uint32	UInt32
64	UInt64	LWORD	Octetstring8	uint64	UInt64
32	Single	REAL	REAL	float32	Float
16	Int16	INT	Signed16	int16	Int16
32	Int32	DINT	Signed32	int32	Int32
64	Int64	LINT	Signed64	int64	Int64
8	SByte	SINT	Signed8	int8	SByte
8	SByte	SINT	SINT	int8	SByte
n*8	String	STRING	StringASCII	StaticString	*
32	Int32	TIME	TIME	int32	*

Table A-1 Available data types

Bits	C#	IEC 61131	FDCML	C++	OPC UA
32	UInt32	UDINT	UDINT	uint32	UInt32
16	UInt16	UINT	UINT	uint16	UInt16
64	UInt64	ULINT	ULINT	uint64	UInt64
16	UInt16	UINT	Unsigned16	uint16	UInt16
32	UInt32	UDINT	Unsigned32	uint32	UInt32
64	UInt64	ULINT	Unsigned64	uint64	UInt64
8	Byte	USINT	Unsigned8	uint8	Byte
8	Byte	USINT	USINT	uint8	Byte
16	UInt16	WORD	WORD	uint16	UInt16

* Not yet implemented

A 2 PLCnext Technology naming conventions

When designating PLCnext Technology components, please observe the following conventions. Correct functioning can only be ensured if you adhere to the following specifications.

A 2.1 GDS ports

A designation

- May not start with a number.
- May not be empty.
- Must consist of at least two characters.
- May not exceed 128 characters.
- May not contain spaces or tabulators.
- May not start or end with “.”.
- May contain “.”.
- May start or end with “_”.

XML restriction `<xs:pattern value=“([_a-zA-Z]+)([0-9a-zA-Z\-\[\]_.]*)([0-9a-zA-Z\[\]_]+)“ />`

A 2.2 Library, components and program

A designation

- May not start with a number.
- May not contain “.”.
- May not be empty.
- Must consist of at least two characters.
- May not exceed 128 characters.
- May not contain spaces or tabulators.
- Has to start with a capital letter

XML restriction `<xs:pattern value=“([a-zA-Z]+)([0-9a-zA-Z\-]*)([0-9a-zA-Z]+)“ />`

A 2.3 Program instances

A designation

- May not start with a number.
- May not be empty.
- Must consist of at least two characters.
- May not exceed 128 characters.
- May not contain spaces or tabulators.

XML restriction `<xs:pattern value=“([a-zA-Z]+)([0-9a-zA-Z\-]*)([0-9a-zA-Z]+)“ />`

A 2.4 Component instances

A designation

- May not start with a number.
- May not be empty.
- Must consist of at least two characters.
- May not exceed 128 characters.
- May not contain spaces or tabulators.

XML restriction `<xs:pattern value="([a-zA-Z]+)([0-9a-zA-Z\-]*)([0-9a-zA-Z]+)" />`

A 2.5 Processes

A designation

- Has to be unique.
- May not contain “.”.
- May not start with a number.
- Must consist of at least two characters.
- May not exceed 128 characters.
- May not contain spaces or tabulators.

XML restriction `<xs:pattern value="|([a-zA-Z]+)([0-9a-zA-Z\-]*)([0-9a-zA-Z]+)" />`

A 2.6 Tasks

A designation

- May not start with a number.
- May not be empty.
- Must consist of at least two characters.
- May not exceed 128 characters.
- May not contain spaces or tabulators.

XML restriction `<xs:pattern value="([a-zA-Z]+)([0-9a-zA-Z\-]*)([0-9a-zA-Z]+)" />`

A 2.7 Namespaces

A designation

- May not contain spaces.
- May not end with “++”.
- Should not contain “_”.

A 3 Explanation of terms

ACF

→ Application Control Framework

Application Control Framework

The ACF is a central part of the PLCnext Technology system architecture. It enables component-based platform development as well as expandability and configurability of the entire system.
→ Section 5.5 “ACF (Application Component Framework)”

API

→ Application Programming Interface

Application Programming Interface

An API is an interface that enables the connection of programs to a software system.

ARP

→ Automation Runtime Platform

Automation Runtime Platform

In the PLCnext Technology firmware, ARP is used as the root name for namespaces.

CLI

→ Command Line Interface

See PLCnCLI

Component

→ Component

Common classes

Common classes are useful classes for operating system functions, e.g., threads, sockets, chrono, etc. The PLCnext Technology common classes are provided via the Phoenix Contact SDK.

Connector

A connector connects an IN port to an OUT port. An OUT port can be connected to several IN ports. One connector is used for each connection. A connector can execute additional tasks such as byte swapping, type casts or bit masking.

Core component

→ Core components

CRL

→ Certificate Revocation List

Data sink

A data sink is the location where data is received and stored.

Debugging

Localization and diagnostics of errors in programs. The debug mode provides different functions that support the user during the function test of the program as well as during error localization and diagnostics (e.g., setting of breakpoints).

Real-time user program

PLC program that can be programmed in IEC 61131-3, C++, or Matlab® Simulink®.

eCLR

ProConOS embedded CLR is the open IEC 61131 control runtime system for different automation tasks.

ESM

→ Execution and Synchronization Manager

ESM task

→ Task

Execution and Synchronization Manager

The Execution and Synchronization Manager is used for real-time task handling. It performs task handling, monitoring and chronological sequencing of programs from different programming languages and enables program execution in a real-time context.
→ Section 2.7 “ESM (Execution and Synchronization Manager)” on page 21

External user components

An external user component is a component that is created by the user. It is not processed within the PLCnext framework but directly under Linux.

GDS

→ Global Data Space

GDS buffer storage units

A GDS buffer storage unit is a memory area used for buffering and transferring data between ESM tasks. It serves as a data exchange area between IN and OUT ports.

Global Data Space

The GDS is the central point for storing all the data that is exchanged between the programs. Data exchange is implemented via IN and OUT ports that the user can link. The GDS enables communication relationships and data exchange between tasks and programs that were created in different programming languages. The GDS also ensures that each task always uses consistent values.
→ Section 2.8 “GDS (Global Data Space)” on page 27

Hot restart

All variable values are retained when the controller is started.

IdentityStore

The IdentityStore is an identity memory for PLCnext Technology devices. It contains authentication tools, e.g., key pairs, certificates, etc.

IN port

→ Port

Internal user component

Internal user components are components created by the user. These components do not run in the real-time context of the PLCnext Technology platform. They extend the range of functions of the PLCnext Technology platform.

Cold restart

When the controller is started, all variables are started with their initialization values. Existing data is reset.

Core components

The core components form the core function of the PLCnext Technology. They are part of the firmware and contain the following components:

- Service components (e.g., ESM, system manager, PLC manager, eCLR, etc.)
- System components (e.g., OPC UA server, WBM, etc.)
- I/O components (e.g., fieldbus manager)
- Middleware (e.g., GDS, RSC, common classes, etc.)

Component

A component is a collection of functions. It may contain one or more programs as well as IN and OUT ports.

Middleware

→ Core components

Notification

Notifications are messages that can be sent and received by components to point out special events. They are identified via a name and can transport user data.

Notification manager

The notification manager enables components to send and receive notifications. It accepts messages from a sender and forwards them to the registered recipient.

User data

User data (payload) is information data in data packets. It does not contain control or protocol information.

NTP

Network Time Protocol

NTP is a protocol for time synchronization in IT systems.

OPC UA server

Open Platform Communications Unified Architecture

OPC UA is a standardized protocol for industrial machine-to-machine communication. Exchange of process data, variable values, read and write access to file systems, etc. between OPC UA server and clients is implemented via the protocol.

OUT port

→ Port

Payload

→ User data

PC Worx Engineer

Engineering software platform for Phoenix Contact automation controllers. PC Worx Engineer complies with IEC 61131-3 and its functions can be expanded using add-ins. The software was called PC Worx Engineer up to version 7.2.3. As of version 2019.0 LTS, the software is called PLCnext Engineer.

PLC manager

The PLC manager is part of the PLCnext Technology firmware. It manages the real-time programs of the PLC and starts and stops them.

PLCnCLI

→ PLCnext Command Line Interface
The PLCnCLI is a command-line interface. It can be used for generating metafiles, C++ header files, for adding IN and OUT ports and generating PLCnext Engineer libraries, for example.

PLCnext Community

Information platform for PLCnext Technology users with a connected forum for direct contact with PLCnext Technology experts.
See plcnext-community.net.

PLCnext Engineer

Engineering software platform for Phoenix Contact automation controllers.
PLCnext Engineer is IEC 61131-3-compliant and its functions can be extended using add-ins.

PLCnext Store

The PLCnext Store is a trade platform for the sale of apps for PLCnext controllers (PLCnext Engineer libraries, programs, solution apps, etc.).

PLCnext Technology

PLCnext Technology is the basis of the new, open control platform from Phoenix Contact. This new technology enables users to work with various well-established software tools at the same time, including Visual Studio®, Eclipse®, Matlab® Simulink®, and PLCnext Engineer. In addition, it facilitates writing of program code in IEC 61131-3 as well as in C++ or C#. Additional functions from third-party manufacturers or the open-source community can be added to the Phoenix Contact automation system, merging into one complete system.

PLM

→ Program Library Manager

POU

→ Program Organization Unit (POU)
Collective term for the terms “function”, “function block”, and “program” defined in IEC 61131-3. They are used for modularizing and structuring the PLC program.

Program Library Manager

The PLM is part of the PLC manager. The PLM manages components providing user programs.

PROFICLOUD

PROFICLOUD is the cloud solution from Phoenix Contact. Locally or globally distributed network devices and even the functions of an industrial PROFINET network can simply and securely be moved to the cloud with PROFICLOUD. Performance data can be accessed rapidly from sites anywhere in the world.

Port

Data between programs is exchanged via the IN and OUT ports. To establish a communication relationship between the programs, IN and OUT ports have to be connected via connectors.

Regular expressions

Regex: Character string for describing quantities of character strings using syntactic rules.

Resource-global variables

According to IEC 61131-3, resource-global variables can be used in all Program Organization Units (POU) of the current resource.

Remote service call

Remote service calls are services for the communication between processes and between devices. They provide the API to enable access to all core components of the PLCnext Technology firmware.

RSC

→ Remote service call

SDK

→ Software Development Kit

Service components

→ Core components

Service manager

The service manager is used to request registered RSC services (components of the SDK) and integrate them into the process.

SFTP

Secure File Transfer Protocol

Shared object

A shared object file (*.so) corresponds to a dynamic library (also called shared libraries). It is downloaded from the firmware to the memory during runtime. Then, the functions included can be called. In PLCnext Technology, a shared object file contains the code for the components and programs to be executed and which where created in C++ or Matlab® Simulink®.

Software Development Kit

The Phoenix Contact SDK contains important toolchains and libraries required for creating programs.

PLC

Programmable logic controller

SSH

Secure shell

System components

→ Core components

System manager

The system manager loads and configures the system components and monitors the stability of the system.

Task

A task is a configuration unit for the execution of program instances (e.g., cyclic execution at fixed intervals).
The following task types are available:

- Cyclic task
- Idle task
- System event task

Task output data

At the end of its execution, a task makes its calculated output data available to other tasks via a buffer storage unit. The task output data is the sum of all OUT port variables that are connected to buffer storage units of I/O systems or to IN ports of program instances in other tasks.

Task input data

When being started, a task actively consumes data from the task output buffer storage units of other tasks, or from the buffer storage units of I/O systems. All user programs within a task therefore use the same input data during the entire task runtime. The consistent input data image is only updated with the next start of the task.

TSD

Time-Series Data

User manager

The user manager manages the different users. Each user has a name and a password. One or several user roles are assigned to each user.

User program

The user creates the user programs in IEC 61131-3, C++, or with Matlab® Simulink®. They are processed in real-time context via the ESM. Through integration into the ESM, a user program becomes real-time-capable. Connection to the data exchange area of the GDS is implemented via IN and OUT ports.

VPN

Virtual Private Network

Warm restart

All variable values marked with “Retain” in PLCnext Engineer are retained when the controller is started.

WBM

→ Web-based management

Web-based management

You can call web-based management via the web browser of your PC. WBM is used to receive information about the controller and to configure it (e.g., managing access data and user roles).

A 4 PROFINET diagnostic code in WBM

The “PROFINET” page in the WBM “Diagnostics” area of your controller is used to view information about the controller and the connected PROFINET devices, as well as information about their Axioline F local bus devices. The displayed diagnostic code consists of several bits that are evaluated individually. The diagnostic state (OK, warning, error) as well as the corresponding diagnostic text are based on an evaluation of the diagnostic bits. The following section is an excerpt from the “AutomationRuntimePlatform/Source/Arp/Io/ProfinetStack/Controller/Services/DiagnosticInfo.hpp” firmware file. It contains the bit description of the diagnostic code:

```
/// <summary>
    /// Special status of this AR, bitwise coded.
    /// - Bit 0: set if not connected
    /// - Bit 1: set if data invalid
    /// - Bit 2: set if diagnosis is available
    /// - Bit 3: set if module difference is available in case of configuration difference
    /// - Bit 4: set  if AR is deactivated
    /// - Bit 5: set if neighborhood information is not available
    /// - Bit 6: neighborhood information not unique
    /// - Bit 7 - Identify.cnf on alias received, but name of the device is already configured in
                 another AR
    /// - Bit 8: Maintenance Required Information is available
    /// - Bit 9: Maintenance Demanded Information is available
    /// - Bit 10: Channel or Manufacturer Specific Diagnosis Information are available
    /// - Bit 11..15: reserved (shall be 0)
    /// </summary>
```

Please observe the following notes

General terms and conditions of use for technical documentation

Phoenix Contact reserves the right to alter, correct, and/or improve the technical documentation and the products described in the technical documentation at its own discretion and without giving prior notice, insofar as this is reasonable for the user. The same applies to any technical changes that serve the purpose of technical progress.

The receipt of technical documentation (in particular user documentation) does not constitute any further duty on the part of Phoenix Contact to furnish information on modifications to products and/or technical documentation. You are responsible to verify the suitability and intended use of the products in your specific application, in particular with regard to observing the applicable standards and regulations. All information made available in the technical data is supplied without any accompanying guarantee, whether expressly mentioned, implied or tacitly assumed.

In general, the provisions of the current standard Terms and Conditions of Phoenix Contact apply exclusively, in particular as concerns any warranty liability.

Phoenix Contact reserves the right to register its own intellectual property rights for the product identifications of Phoenix Contact products that are used here. Registration of such intellectual property rights by third parties is prohibited.

Other product identifications may be afforded legal protection, even where they may not be indicated as such.

How to contact us

Internet

Up-to-date information on Phoenix Contact products and our Terms and Conditions can be found on the Internet at:
phoenixcontact.com

Make sure you always use the latest documentation.
It can be downloaded at:
phoenixcontact.net/products

Subsidiaries

If there are any problems that cannot be solved using the documentation, please contact your Phoenix Contact subsidiary.
Subsidiary contact information is available at phoenixcontact.com.

Published by

PHOENIX CONTACT GmbH & Co. KG
Flachsmarktstraße 8
32825 Blomberg
GERMANY

PHOENIX CONTACT Development and Manufacturing, Inc.
586 Fulling Mill Road
Middletown, PA 17057
USA

Should you have any suggestions or recommendations for improvement of the contents and layout of our manuals, please send your comments to:
tecdoc@phoenixcontact.com

PHOENIX CONTACT GmbH & Co. KG
Flachsmarktstraße 8
32825 Blomberg, Germany
Phone: +49 5235 3-00
Fax: +49 5235 3-41200
E-mail: info@phoenixcontact.com
phoenixcontact.com

108664_en_03
Order No. —03

INSPIRING INNOVATIONS