

PLCnext Technology

User manual

UM EN PLCNEXT TECHNOLOGY

User manual

PLCnext Technology

UM EN PLCNEXT TECHNOLOGY, Revision 00

2018-08-09

This user manual is valid for:

Designation	Order No.
PLCNEXT TECHNOLOGY	
AXC F 2152	2404267
AXC F 2152 STARTERKIT	1046568

Table of contents

1	General information	7
1.1	Identification of warning notes	7
1.2	Qualification of users	7
1.3	Introduction.....	8
1.4	Information on this document.....	9
1.5	Trademarks / licensing information	9
1.6	Safety notes.....	10
1.7	PLCnext Technology product range	11
2	Structure of PLCnext Technology	13
2.1	System Manager	16
2.2	PLC Manager	16
2.3	Managing components	17
2.4	Configuration files.....	18
2.4.1	acf, esm, gds	18
2.4.2	.tic file	19
2.4.3	Metafiles	19
2.4.4	Generating configuration files with PC Worx Engineer	21
2.4.5	Manual configuration	21
2.5	ESM (Execution and Synchronization Manager)	22
2.5.1	Task configuration with PC Worx Engineer	23
2.5.2	Configuring tasks via configuration files	24
2.6	GDS (Global Data Space)	27
2.6.1	Notes on designating GDS ports	33
2.6.2	Supported data types	34
2.7	RSC (Remote Service Calls)	36
2.8	Operating system	37
2.8.1	Directories of the firmware components in the file system	37
2.8.2	Firewall	38
2.8.3	NTP (Network Time Protocol)	40
2.8.4	OpenVPN™ client	41
2.8.5	IPsec (StrongSwan)	41
2.8.6	Text editors	42
2.8.7	User rights	42
3	Structure of a C++ program	45
3.1	ILibrary or LibraryBase	46
3.2	Several component types in the same library	47
3.3	IComponent or ComponentBase	47

3.3.1	IProgramComponent and IProgramProvider	49
3.3.2	IProgram or ProgramBase	50
3.3.3	IControllerComponent	50
3.4	PLM (Program Library Manager)	51
3.4.1	Requirements on the components	51
3.4.2	Functions	52
3.4.3	Configuration	52
3.5	ACF (Application Component Framework)	52
3.5.1	Libraries	53
3.5.2	Components	54
3.5.3	Configuration	55
3.6	Common classes	56
3.6.1	Threading	57
3.6.2	Ipc	58
3.6.3	Chrono	58
3.6.4	Io	58
3.6.5	Net	58
3.6.6	Runtime	59
3.7	Template Loggable	59
3.8	RSC Axioline services	61
3.9	RSC PROFINET services	62
3.10	RSC Device Interface services	64
3.11	RSC GDS services	68
4	Creating programs with C++	69
4.1	Installing PLCnext Technology SDK and add-in for Eclipse®	69
4.1.1	Requirements	69
4.1.2	Installation	70
4.2	Creating a new project	75
4.3	Creating a program	79
4.4	Importing generated libraries into PC Worx Engineer	86
4.5	Remote debugging	89
4.6	Sample programs	90
5	Creating function blocks and functions with C#	91
5.1	Installing the Visual Studio® extension	92
5.1.1	Requirements	92
5.1.2	Installation	93
5.2	Creating a firmware library	94
5.3	Importing the C# library in PC Worx Engineer	96
5.4	Root rights	98

5.5	Remote debugging of C# code with Visual Studio®	100
5.5.1	Opening a port and deactivating TLS	101
5.5.2	Disabling user authentication	102
5.5.3	Debug mode	103
5.6	Supported functions of the eCLR programming systems.....	105
5.6.1	C# language functions	105
5.6.2	Base class libraries	106
5.6.3	eCLR runtime functions	107
5.7	Supported data types	108
6	MATLAB® Simulink®	111

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.



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

1.3 Introduction

PLCnext Technology

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

To meet the changing demands of industry, Phoenix Contact developed the PLCnext Technology as the basis for a new, open control platform. This solution 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 PLC applications was only possible in the IEC 61131-3 programming languages, because determinism is required. 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 – 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 PC Worx Engineer. You can use classic methods in compliance with IEC 61131-3 to create the code, as well as C/C++ and Simulink® code.

PLCnext Technology connects and combines classic 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 downloaded and used regardless of the location and at any time. All performance and energy data 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 classic 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 PC Worx Engineer, MATLAB® Simulink®, Eclipse® and 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 the PLCnext Technology, you can create programs in C++ and import these into PC Worx Engineer. There, you can instantiate the programs and use them in the same way as classic 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. The programs can therefore 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 common in the field of C++ programming (see section “Creating programs with C++” on page 69).

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 these into libraries, import them into PC Worx Engineer, and use them for your programming. You can call up the function blocks or functions in PC Worx Engineer, e.g. in the code worksheet ST (Structured Text) or LD (Ladder Diagram), and use them 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 91).

1.4 Information on 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 applications.

Detailed information on PLCnext 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

The PLCnext controllers work with a Linux operating system.

All license information can be called up using the “Legal Information” link on every page of the 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 the WBM of the respective controller is available in the corresponding user manual. The user manual is available for downloading 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 is available for downloading at phoenixcontact.net/products.

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
AXC F 2152 starterkit including PLCnext Control AXC F 2152, 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. PC Worx Engineer is IEC 61131-3-compliant and its functionality can be expanded using add-ins.	PC WORX ENGINEER 7	1046008
Software add-on for 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 are available for downloading at phoenixcontact.net/products.

Software packages required for programming in C++ with Eclipse are available for downloading from the following categories in the download area of the AXC F 2152 controller:

- “Configuration software”: You will find the Phoenix Contact add-in for the Eclipse® development environment here.
- “Software”: You will find the Phoenix Contact LibraryBuilder for creating a library from Eclipse® for use in PC Worx Engineer as well as the Phoenix Contact SDK (Software Development Kit) here.

Some sample applications for the AXC F 2152 programmed in C++ are available for downloading from “Sample applications” category.

2 Structure of PLCnext Technology

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

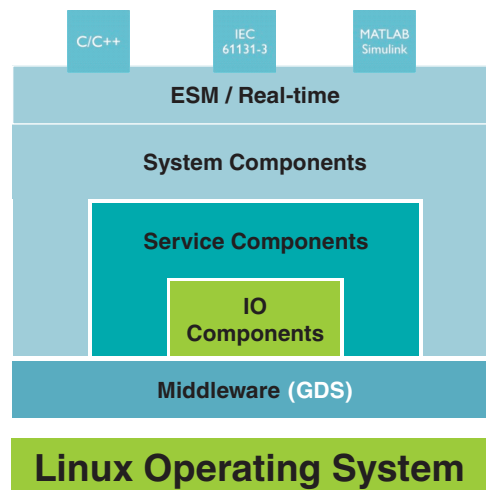


Figure 2-1 PLCnext Technology core components

Middleware

The middleware section decouples the PLCnext Technology firmware platform from the operating system. The GDS (Global Data Space) is part of the middleware and forms a central component of PLCnext Technology. It enables consistent data exchange between different real-time components. You will find further information on the GDS in 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 technologies are supported:

- PROFINET controller
- PROFINET device
- Axioline F master (local bus)

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
- PC Worx Engineer HMI (HTML5 web visualization)
- Accessible via OS
 - DCP
 - SFTP

- VPN
- SSH
- NTP
- Trace controller

System components

All basic functions of PLCnext Technology are implemented in the system components.

- System Manager and PLC Manager
These components load all 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. Programming languages can be combined in any way for creating applications. You will find further information on the ESM in section “ESM (Execution and Synchronization Manager)” on page 22.
- User Manager
The User Manager extends the standard Linux user management function. The various 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 standard with the PLCnext Technology firmware; these are created by the user. These user-defined programs can be created in both classic 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, executed by the firmware.

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

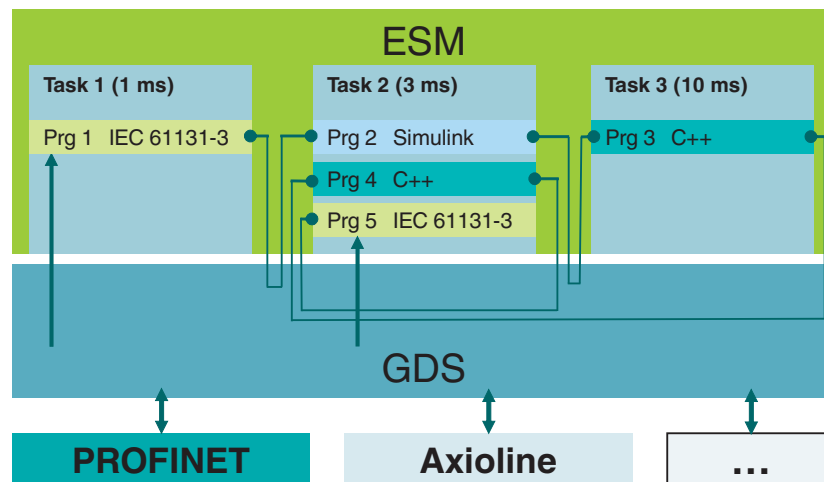


Figure 2-2 Task handling with ESM and GDS

Internal user components

With the modular and open PLCnext Technology control platform, users can add their own components – internal user components – to the system. The System Manager loads and monitors these internal user components (see section 2.1 “System Manager”). All available PLCnext Technology APIs can be used:

- Remote Service Calls (RSC): each component (system, service, I/O components) features a selection of individual services that can be called up and used via RSC (see section “RSC (Remote Service Calls)” on page 36).
- Component interface: the firmware calls up 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 56).

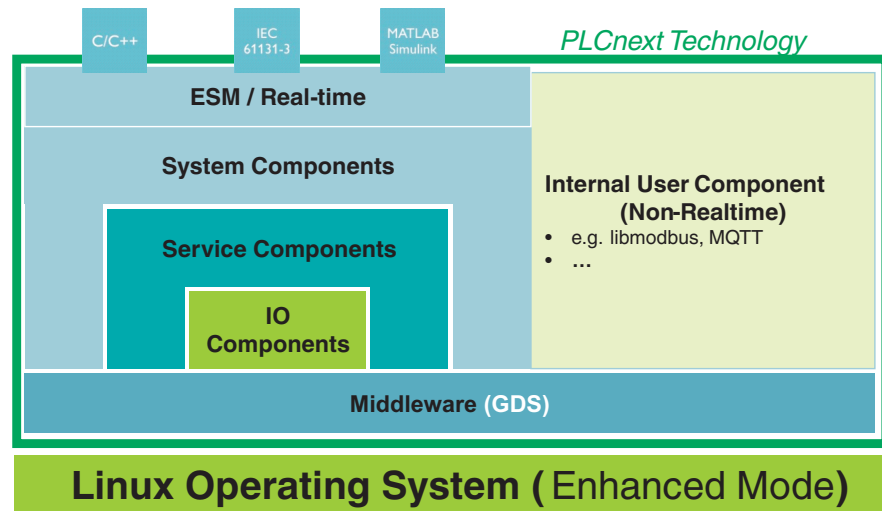


Figure 2-3 PLCnext Technology – Internal user components

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. External user components can be started in along with the PLCnext Technology framework. External user com-

ponents are user-defined components without real-time requirements and independent startup behavior. They are not component parts of the PLCnext firmware and do not have a connection to PLCnext interfaces.

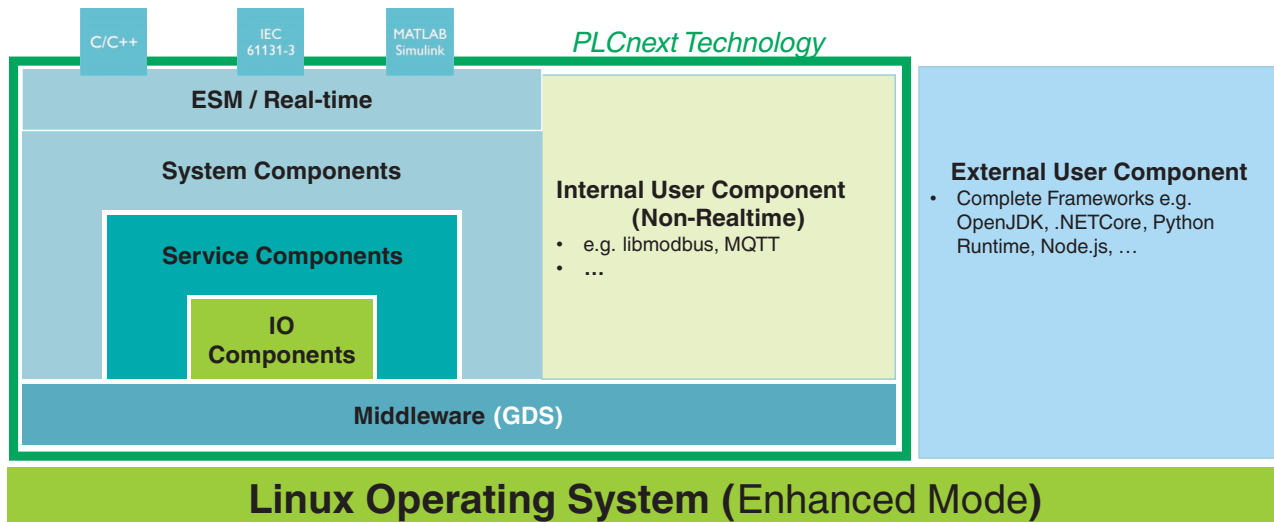


Figure 2-4 PLCnext Technology – External user component

2.1 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, shut down and configured by the System Manager.

2.2 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 may consist of an IEC 61131-3 program that was created and sent using PC Worx Engineer or it can consist of code that has been 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 the boot up and shut down of the real-time system (ESM) as well as the 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:**
With a cold restart, all data is reset.

- **Warm restart:**
With 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:**
With a hot restart, the data is neither reset nor restored.

2.3 Managing components

PLCnext Technology is based on components that are included via the “IComponentInterface” interface (see section “IComponent or ComponentBase” on page 47).

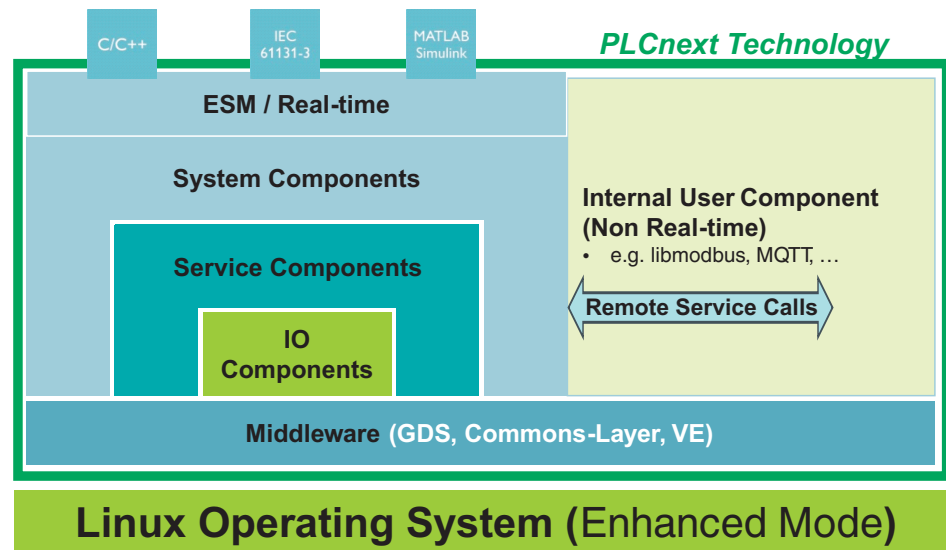


Figure 2-5 PLCnext Technology components

Application Control Framework

The Application Control Framework (ACF) is a part of the System Manager and manages the internal user components. The AFC 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.

You will find further information on the ACF in section “ACF (Application Component Framework)” on page 52.

PLC Manager / Program Library Manager

One firmware component that is managed by the AFC is the PLC Manager (see “PLC Manager” on page 16). This manages the PLC program (real time). The associated components and libraries that are made available for these user programs are managed by the PLC Manager via the PLM (Program Library Manager). The configuration is executed with a file referenced by /opt/plcnext/projects/Default/Plc/Plm/Plm.acf.config. The PLC manager generates the components when loading the program.

You will find further information on the PLC Manager and Program Library Manager in sections “PLC Manager” on page 16 and “IComponent or ComponentBase” on page 47.

- Delimitation ACF and PLM**
- ACF and PLM use the same ILibrary and IComponent interface (see section “ILibrary or LibraryBase” on page 46 and “IComponent or ComponentBase” on page 47).
 - ACF and PLM use the same format for configuration files (see section “Configuration files” on page 18).
 - Only components that are managed via PLM can be made available for programs that can be instantiated in ESM tasks (IProgramProvider, see section 3.3.1 on page 49).
 - Only components that are managed via the PLM can be stopped, changed and started up via download from PC Worx 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.

2.4 Configuration files

With PLCnext Technology, you can load programs and program instances onto the controller even without the PC Worx Engineer software. You can, for example, create a program in Eclipse® with C++ and transfer this directly onto the controller. All of the important and necessary settings can be configured directly in the controller in the configuration files. 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 after 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 PC Worx Engineer is used, all configuration files are created by PC Worx Engineer and downloaded to the controller.

2.4.1 acf, esm, gds

acf.config

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

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

There are two possible components to instantiate:

- Included in Default.acf.config (projects/Default/Default.acf.config)
- Included in Plm.acf.config (projects/Plc/Plm/Plm.acf.config)

esm.config

The esm.config file contains the task configuration, the instantiation of programs and assignment to a processor core (one ESM per processor core). See section “Configuring tasks via configuration files” on page 24.

gds.config

The gds.config configuration file contains the port definition as well as the assignment of IN and OUT ports. See section “GDS configuration using configuration files” on page 31.

2.4.2 .tic file



Phoenix Contact recommends creating the .tic files with the PC Worx 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.

2.4.3 Metafiles

Metadata describes classes/types of components and programs that were created in a PLCnext Technology application.

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). The configuration of the GDS data ports (IN or OUT ports) is defined in the *.progmeta file. This configuration is required for using the program within the PLCnext Technology environment.

When the Eclipse® add-in creates the C++ project, the *.compmeta and *.libmeta metafiles are created automatically. However, you must modify the *.progmeta file, because the port definitions are performed here. The files are interlinked, as shown in Figure 2-6.

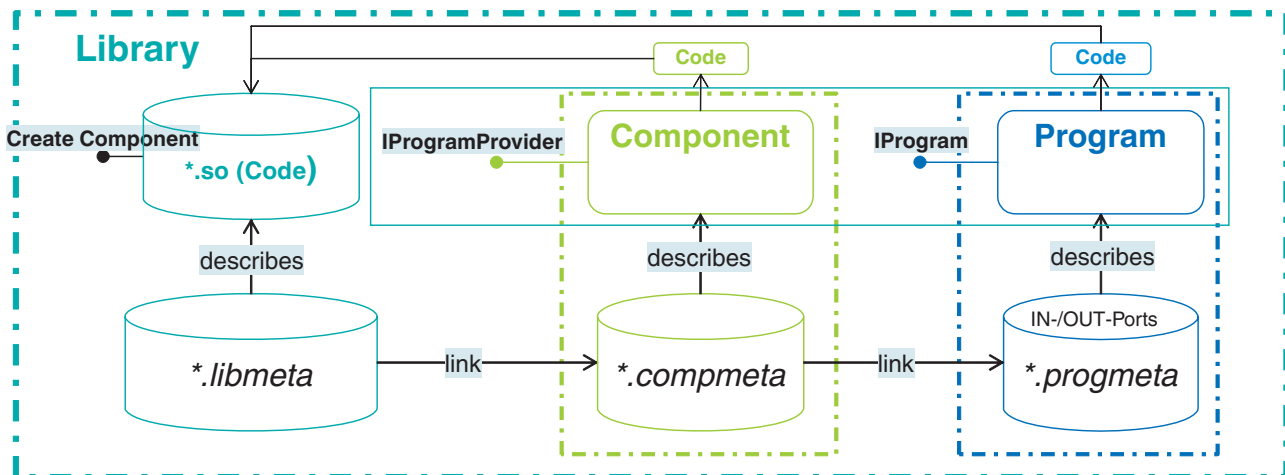
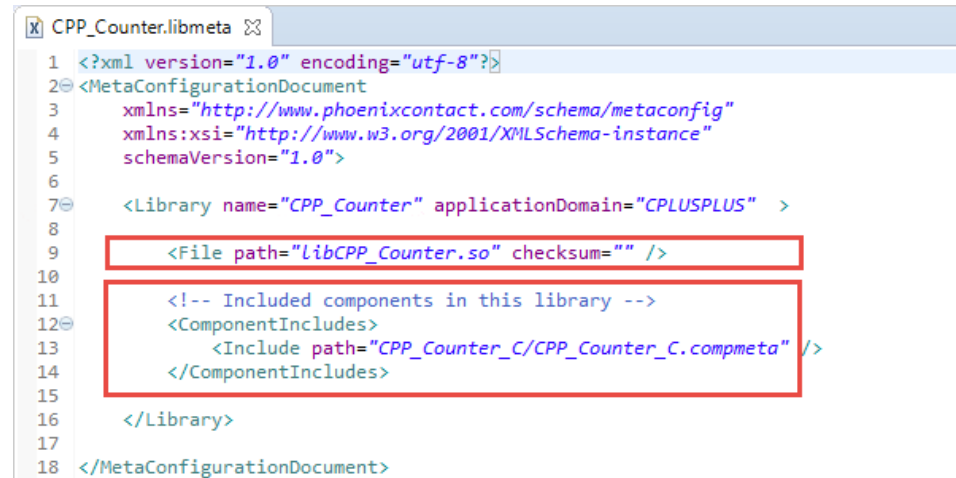


Figure 2-6 Overview of the metafiles

The LibraryBuilder combines the metadata and the code (*.so) into a library which is imported into PC Worx Engineer and which can be used as an IEC 61131-3 program. Tasks and the flow of data are configured directly in PC Worx Engineer. The Phoenix Contact add-in for Eclipse® generates the metafiles. However, you must add additional IN and OUT ports and programs manually.



```

1  <?xml version="1.0" encoding="utf-8"?>
2  <MetaConfigurationDocument
3      xmlns="http://www.phoenixcontact.com/schema/metaconfig"
4      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5      schemaVersion="1.0">
6
7      <Library name="CPP_Counter" applicationDomain="CPLUSPLUS" >
8
9          <File path="libCPP_Counter.so" checksum="" />
10
11          <!-- Included components in this library -->
12          <ComponentIncludes>
13              <Include path="CPP_Counter_C/CPP_Counter_C.compmeta" />
14          </ComponentIncludes>
15      </Library>
16  </MetaConfigurationDocument>

```

Figure 2-7 Example of a component list and a shared object (*.so) in the *.libmeta file



```

1  <?xml version="1.0" encoding="utf-8"?>
2  <MetaConfigurationDocument
3      xmlns="http://www.phoenixcontact.com/schema/metaconfig"
4      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5      schemaVersion="1.0">
6
7      <Component name="CPP_Counter_C" >
8
9          <!-- List of programs referenced in this component -->
10         <ProgramIncludes>
11             <Include path="CPP_Counter_P/CPP_Counter_P.progmeta" />
12         </ProgramIncludes>
13
14         <!-- List of outgoing ports defined by this component. -->
15         <!-- Uncomment this section and add ports -->
16         <Ports>
17
18         </Ports>
19         -->
20     </Component>
21
22 </MetaConfigurationDocument>

```

Figure 2-8 Example of a program list in the *.compmeta file

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <MetaConfigurationDocument schemaVersion="1.0" xmlns="http://www.phoenixcontact.com/schema/metaconfig">
3   <Program name="CPP_Counter_P">
4     <Ports>
5       <Port kind="Input" name="IP_CppEnable_boolean" type="boolean" multiplicity="1"/>
6       <Port kind="Input" name="IP_CppSwitches_uint8" type="uint8" multiplicity="1"/>
7       <Port kind="Output" name="OP_CppCounter_uint8" type="uint8" multiplicity="1"/>
8     </Ports>
9   </Program>
10 </MetaConfigurationDocument>

```

Figure 2-9 Example of the definition of IN and OUT ports in the *.progmeta file

2.4.4 Generating configuration files with PC Worx Engineer

The following description applies for all three *.config types (acf.config, esm.config, gds.config). The configuration files are automatically created when the PC Worx 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 is intended for the download from PC Worx Engineer. The PCWE folder is automatically created and managed by PC Worx Engineer. Do not store any other files here, because PC Worx Engineer completely deletes this folder prior to the download. When the configuration in PC Worx Engineer changes and the project is once again downloaded onto the controller, the previous configuration files are overwritten. Manual changes are thus also lost.

2.4.5 Manual configuration

To make manual changes to the configuration files or to create new configuration files, you can copy the files generated by PC Worx Engineer and rename them. You can subsequently perform your manual configuration in these files. To prevent the "PCWE" folder from being overwritten when the project is downloaded from PC Worx Engineer, create a new folder under /opt/plcnext/projects and save the files there (e.g. /opt/plcnext/projects/Example/). The folder can be used for saving additional configuration files and for the manual, configurative extension of the components.

- Save your own configuration files in the controller file system.
- Create a new folder for this purpose and name it accordingly.
E.g. /opt/plcnext/projects/Example/.
- Include the configuration files via an "Include path" command (see example Figure 2-11).

The "Include" mechanism is used to state which configuration files are to be used for the configuration. If files are to be included that are not in the folder used for inclusion, 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/De-

fault/Plc/GDS (<Include path="*.gds.config" />). If you want to include all config files that are in the respective folder, add an * to the specific file name (e.g. *.gds.config). This results in all files with the same file extension being included.

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

Figure 2-10 Example: "Include path"

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 PC Worx Engineer)
- "Example": /opt/plcnext/projects/Example (created by the user)

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

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

</GdsConfigurationDocument>
```

Figure 2-11 Example: "Include path"

When editing configuration files, observe the information in sections "Configuring tasks via configuration files" on page 24 and "GDS configuration using configuration files" on page 31.

2.5 ESM (Execution and Synchronization Manager)

In addition to the benefits of an open control platform, PLCnext Technology also features task handling. Until now, to benefit from the openness of the Linux operating system, functions such as real-time task handling, watchdog monitoring in the operating system, and process-internal communication had to be created with considerable effort by the user. The Execution and Synchronization Manager (ESM) takes over these tasks for the user. The developer can now concentrate entirely on creating the application. Task handling, monitoring and the chronological sequencing of programs from different programming languages are now performed by the Execution and Synchronization Manager (ESM), one of the most important PLCnext Technology components. 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 features 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 in real time that were created in different programming environments. 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 enables the high-level language programs to be executed 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 up (see also section “GDS (Global Data Space)” on page 27).

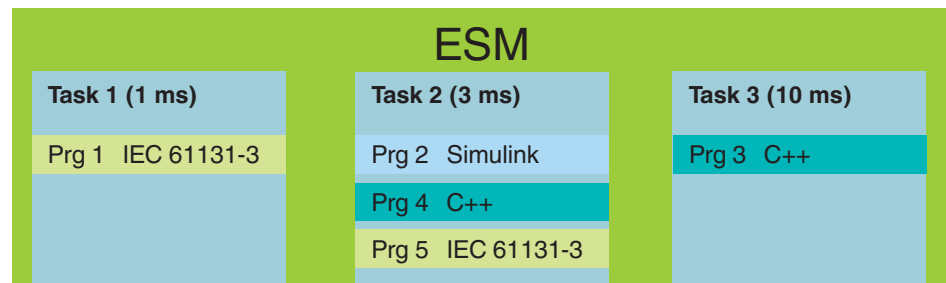


Figure 2-12 Execution and Synchronization Manager

2.5.1 Task configuration with PC Worx Engineer

You can use the PC Worx Engineer software to create and configure tasks conveniently. Here, the developer can proceed as in a classic IEC 61131-3 program. The IEC 61131-3 program, or the program created in a different programming environment and then imported into PC Worx 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 PC Worx 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 PC Worx Engineer is available in the online help, the PC Worx Engineer quick start guide (PC WORX ENGINEER 7, order no.: 1046008), and the AXC F 2152 con-

troller user manual (order no.: 2404267). The documentation for the respective products is available for downloading at phoenixcontact.net/products. You can call up the online help from within PC Worx Engineer.

Name	Task Type	Event Name	Program Type	Interval (ms)	Priority	Watchdog (ms)	Comment
ESM1							
Cyclic 100	Cyclic Task			100	0	100	
MainInstance			Main				
Program1_Moving_Light			Moving_Light_Prog				
Enter program instance name here			Select program type here				
Enter task name here							
ESM2							
Enter task name here							

Figure 2-13 Example: Tasks and program instances in PC Worx Engineer

2.5.2 Configuring tasks via configuration files



When a project is downloaded from PC Worx Engineer to the controller, the existing configuration files are overwritten with the new configuration files. Section “Manual configuration” on page 21 describes how you can safely store your manually modified configuration on 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 PC Worx 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.

```
<?xml version="1.0" encoding="utf-8"?>
<EsmConfigurationDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" s
  <Tasks>
    <CyclicTask name="SquareWave_Cycle" stackSize="0" priority="0" cycleTime="100000000" watchdogTime="100000000" executionTim
    <CyclicTask name="CPP_Cycle" stackSize="0" priority="0" cycleTime="100000000" watchdogTime="100000000" executionTimeThresh
    <CyclicTask name="PCWE_Cycle" stackSize="0" priority="0" cycleTime="100000000" watchdogTime="100000000" executionTimeThres
  </Tasks>
  <EsmTaskRelations>
    <EsmTaskRelation esmName="ESM1" taskName="SquareWave_Cycle" />
    <EsmTaskRelation esmName="ESM1" taskName="CPP_Cycle" />
    <EsmTaskRelation esmName="ESM1" taskName="PCWE_Cycle" />
  </EsmTaskRelations>
  <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>
  <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>
</EsmConfigurationDocument>
```

Figure 2-14 Example of a configuration file

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

Defining a task

A task is defined between the tags <Tasks> and </Tasks>

```
<Tasks>
  <CyclicTask name="SquareWave Cycle" stackSize="0" priority="0" cycleTime="100000000" watchdogTime="100000000" executionTimeThresho
  <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>
```

Figure 2-15 Defining a task

Definition of the task starts with the task type.

- Here, enter the type:
 - “CyclicTask” = Cyclic task
 - “IdleTask” = Idle task
- Define the tasks using the attributes from the following table:

Attribute	Description
“name”	<p>The “name” attribute defines the task name.</p> <ul style="list-style-type: none"> Enter the task name. <p>The name must be unique. It may only be used once per controller.</p>
“priority”	<p>The “priority” attribute defines the task priority.</p> <ul style="list-style-type: none"> Enter a value to specify the task priority. 0: Highest priority 31: Lowest priority 32: Reserved for idle task
“cycleTime”	<p>The “cycleTime” attribute defines the duration of the task (for cyclic tasks only).</p> <ul style="list-style-type: none"> Enter the value in nanoseconds. <p>The minimum value of the AX C F 2152 controller is 500000 ns = 500 µs</p>
“watchdogTime”	<p>The watchdog monitors whether the task was executed within the specified time.</p> <ul style="list-style-type: none"> Enter the value in nanoseconds. The value “0” means that there is no monitoring. <p>The “watchdogTime” attribute can be used for cyclic and idle tasks.</p>

Assigning a task

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

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

Figure 2-16 Assigning a task to an ESM

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

Attribute	Description
"esmName"	The "esmName" attribute defines the name of the ESM to which the respective task is to be assigned. <ul style="list-style-type: none"> 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. <ul style="list-style-type: none"> Enter the name of the task to be assigned.

Instantiating programs

Once the task is defined and assigned to an ESM, programs can be instantiated. Programs are defined between the 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 PC Worx Engineer. The figure originates from a config file generated with PC Worx Engineer.

```
<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>
```

Figure 2-17 Example: Instantiating programs

- Instantiate programs using the attributes from the following table:

Attribute	Description
"name"	The "name" attribute defines the name of the program instance. <ul style="list-style-type: none"> Enter the name of the program instance. The name must be unique within the controller.
"programType"	The "programType" attribute defines the program type. <ul style="list-style-type: none"> Enter the program type.
"componentName"	The name of the component that contains the program is stored in the "componentName" attribute. <ul style="list-style-type: none"> 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 PC Worx Engineer. See section 4.3 "Creating a program"

Assigning program instances to a task

The program instances must be assigned to a task. The assignment is made between the tags <TaskProgramRelations> and </TaskProgramRelations>.

```
<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>
```

Figure 2-18 Assigning program instances to a task

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

Attribute	Explanation
"taskname"	<p>The "taskname" attribute specifies the name of the task to which the program instance is to be assigned.</p> <ul style="list-style-type: none"> Enter the name of the desired task to which the program instance is to be assigned.
"programName"	<p>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.</p> <ul style="list-style-type: none"> Enter the full name into the program instance to be processed.
"order"	<p>The "order" attribute determines the order in which program instances are processed within a task, starting with 0.</p> <ul style="list-style-type: none"> Enter a number to determine the processing sequence. <p>The program instance with the digit 0 is processed first. Use a number once only for each task.</p>

2.6 GDS (Global Data Space)

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 2-19, 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 outputs a new value that is again sent to task 2 as an input value. Once task 1 is finished, the processing of task 2 is continued. However, because the input value has changed in the meantime, there is a data inconsistency which may lead to problems.

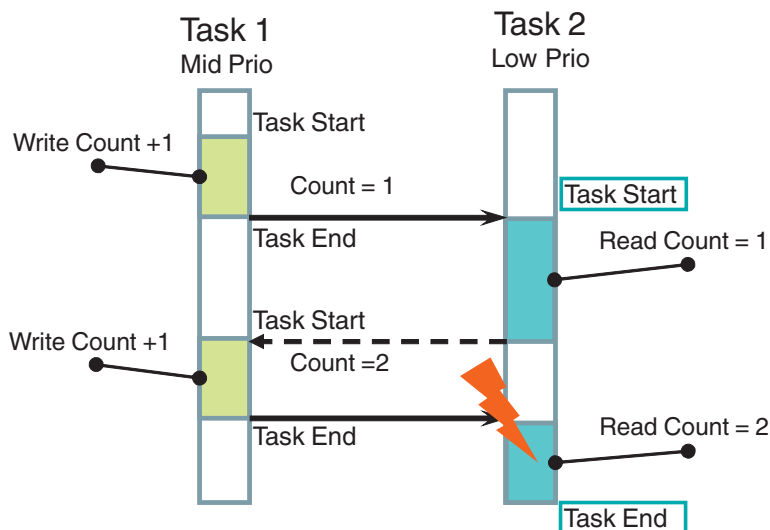


Figure 2-19 Example of data inconsistency

PLCnext Technology is equipped with the Global Data Space (GDS) to ensure seamless processing during data exchange between user programs, fieldbus systems, and system programs. This is used as a central point for storing all data that is exchanged between the programs. All data is stored centrally in the GDS. The tasks refer to GDS variables that do not change while a task cycle is being processed. This means that at the beginning and during the ongoing execution of a task, the same value is available. This approach ensures that each task always uses consistent values.

Port-based communication

With the GDS, it is easy to establish communication relationships between different programs. This communication between the PLCnext Technology programs is realized via ports. A port presents an end point 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 such a 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 the consistency of data and functioning communication relationships at all times.

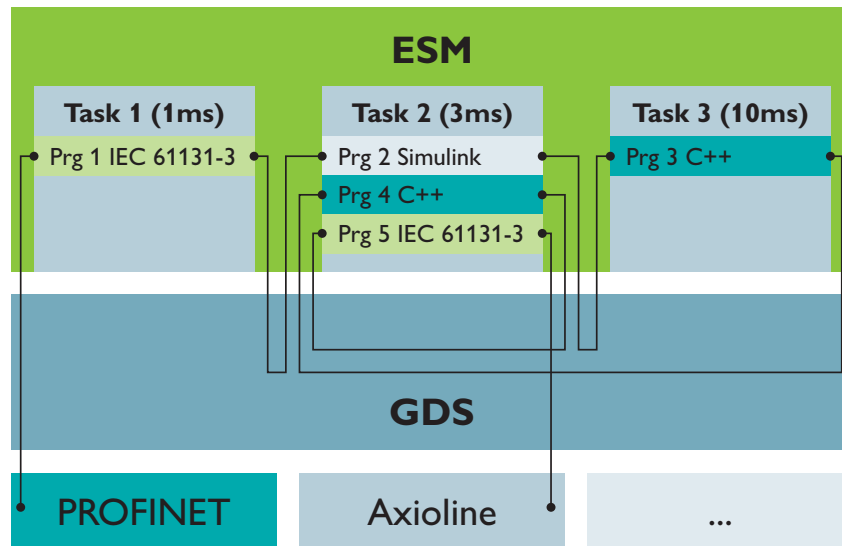


Figure 2-20 Inter-task communication with ESM and GDS

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 up 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 up in the task.

Figure 2-21 “Example: Port communication with task 1 buffer storage unit” shows the behavior of task 1:

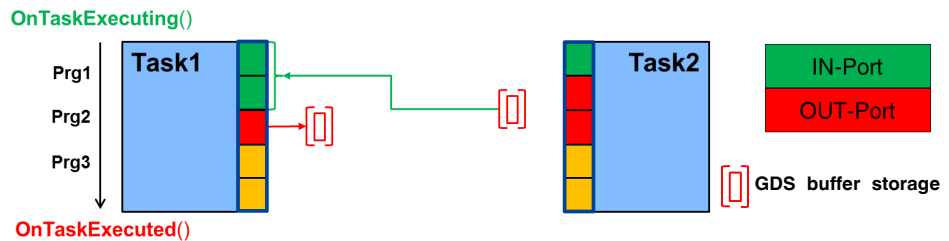


Figure 2-21 Example: Port communication with task 1 buffer storage unit

Data exchange for IN ports and OUT ports takes place at the beginning and at the end of each task execution. This only concerns the 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, a data exchange is only implemented if the program instances are coupled via IN and OUT ports. This ensures that the data is stable and consistent during execution.

Figure 2-22 “Example: Port communication with task 1 and task 2 buffer storage units” shows the behavior of task 1 and task 2:

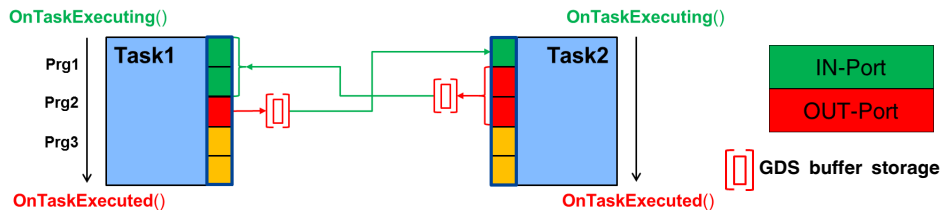


Figure 2-22 Example: Port communication with task 1 and task 2 buffer storage units

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

Figure 2-23 “Example: Fieldbus communication via IN and OUT ports” shows the behavior of the fieldbus communication via ports.

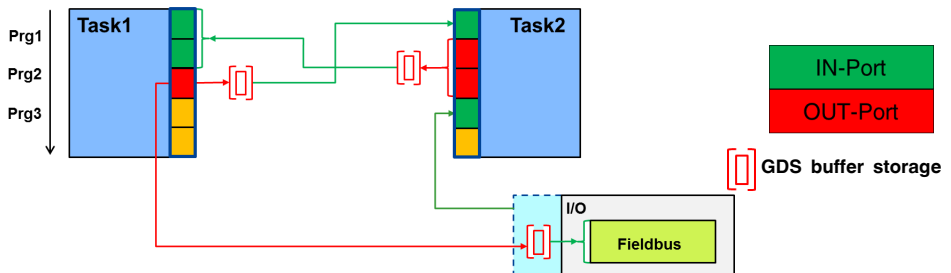


Figure 2-23 Example: Fieldbus communication via IN and OUT ports

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.



You will find further information on connecting ports in the PC Worx Engineer online help (see “Role mapping: Assigning PLCnext ports”).

All components of a PLCnext Technology application that are part of the data exchange and I/O components must be connected via IN and OUT ports.

There are two methods of configuring the GDS:

- With PC Worx Engineer
- With configuration files

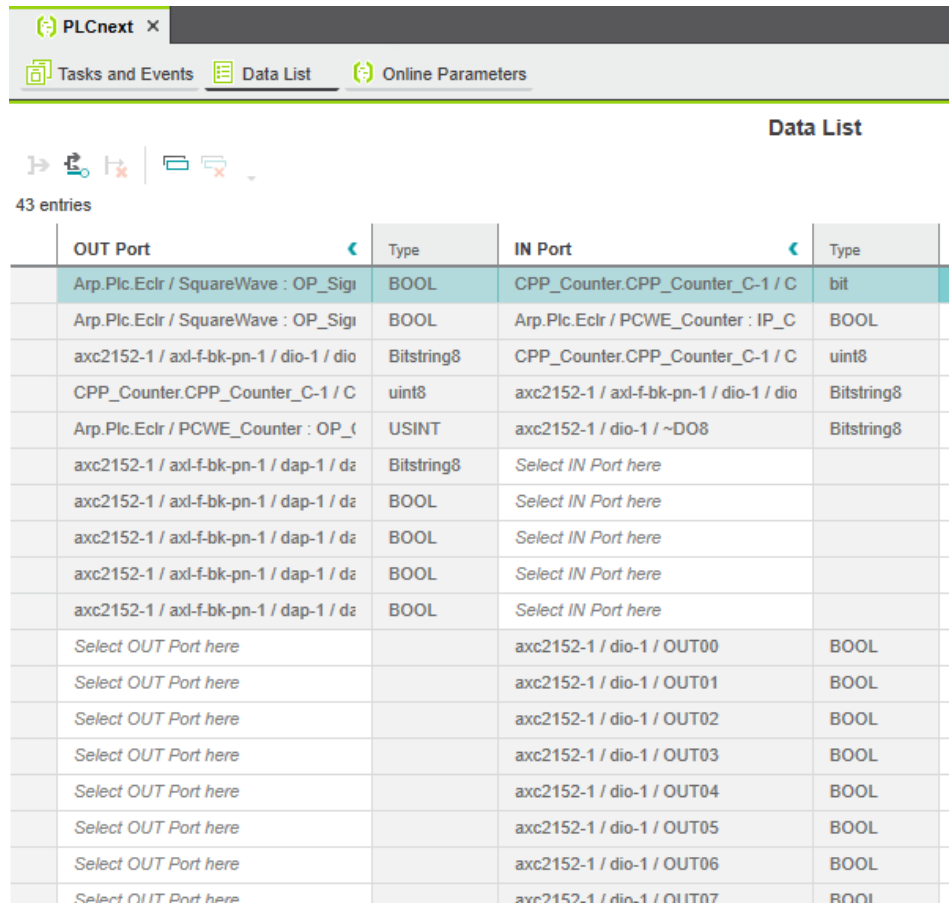
GDS configuration with PC Worx Engineer

- Configure the GDS using the PC Worx Engineer software (see section 4.4 “Importing generated libraries into PC Worx Engineer”).

In the PC Worx Engineer editor, simply select the desired use as IN or OUT ports in the “Data List” of the node “PLCnext” for the respective process data. You can see an overview of all IN and OUT ports available in 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.



The screenshot shows the 'Data List' tab in the PLCnext software. It displays a table with 43 entries, organized into two main sections: OUT Port and IN Port. Each entry includes a description, a type, and a port address. The table is as follows:

OUT Port	Type	IN Port	Type
Arp.Plc.Eclr / SquareWave : OP_Sigi	BOOL	CPP_Counter.CPP_Counter_C-1 / C	bit
Arp.Plc.Eclr / SquareWave : OP_Sigi	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 / d	Bitstring8	Select IN Port here	
axc2152-1 / axl-f-bk-pn-1 / dap-1 / d	BOOL	Select IN Port here	
axc2152-1 / axl-f-bk-pn-1 / dap-1 / d	BOOL	Select IN Port here	
axc2152-1 / axl-f-bk-pn-1 / dap-1 / d	BOOL	Select IN Port here	
axc2152-1 / axl-f-bk-pn-1 / dap-1 / d	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 2-24 Example: List of all available IN and OUT ports

You will find a detailed description of the procedure and additional information on GDS configuration with PC Worx Engineer in the online help, the PC Worx Engineer quick start guide (PC WORX ENGINEER 7, order no.: 1046008) and the AXC F 2152 controller user manual (order no.: 2404267). The documentation for the respective products are available for downloading at phoenixcontact.net/products.

GDS configuration using configuration files



When a project is downloaded from PC Worx Engineer to the controller, the previous configuration files are overwritten with the new configuration files. Section "Manual configuration" on page 21 describes how you can safely store your manually modified configuration on the controller file system, thus preventing the loss of data.

You can also modify the GDS configuration without PC Worx Engineer, using configuration files. All of the important settings can be configured directly in the configuration file *.gds.config on the controller. All configuration files (except *.libmeta) are created by PC Worx Engineer. The configuration can be modified by editing the XML file with any editor. You will find the file in the directory "/opt/plcnext/projects/.../Plc" of your controller (see Section 2.8.1 "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. An OUT port is entered as a “startPort” attribute and an IN port as an “endPort”. Within the *.gds.config, each of the ports has its own unique name that has the following structure:

“ComponentInstanceName/GroupName: VariableName.”

ComponentInstanceName	<p>Components of a PLCnext Technology application can be programs, PROFINET controllers (for instance Arp.Io.Fblo.PNControllerComponent), the IEC 61131-3 runtime (for instance Arp.Plc.Eclr.EclrComponent) and also a C++ application (for instance CppCounterComponent).</p> <p>Names that start with “Arp...” are specified by the PLCnext Technology firmware.</p> <p>The names of the C++ applications are specified by the user during instantiation.</p> <p>Exception: When instantiating C++ or MATLAB® Simulink® programs in PC Worx Engineer, the name of the component instance is specified by PC Worx Engineer.</p> <p>There can be one or multiple instances of each component.</p> <p>In the case of a PROFINET device, it is possible to gain access by entering the product node ID (for more information, refer to the online help within PC Worx Engineer).</p> <ul style="list-style-type: none"> • Enter the name of the instance here.
GroupName	<p>A local bus device is defined based on the position of the device in the local bus. The position numbering starts with the digit 0.</p> <ul style="list-style-type: none"> • Enter this information instead of the GroupName. <p>With program instances, the name is specified.</p> <ul style="list-style-type: none"> • Enter the name of the program instance.
Variable name	<ul style="list-style-type: none"> • Enter the name of the IN and OUT ports or the process data here, depending on the component type.

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 as the port and the variable name in the following form. Global variables as well as IN and OUT ports can be used: "runtime system/program instance: Variable name" (e.g.: Arp.Plc.Eclr/MainInstance:OUT_PORT_A)
Data of a high-language project on the controller	Enter the name of the program instance as the port and the variable names in the following form (in accordance with the form 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 as the port and the process data names in the following form: "PROFINET Controller/Node-ID: process data name" (e.g.: Arp.Io.Fblo.PnC/46:~DI8)
Local bus process data	Enter the name of the local bus controller as the port and the process data names in the following form: "Local bus controller/module position: process data name" (e.g.: Arp.Io.Fblo.AxIC/0:~DO8)

Example excerpt from a GDS.config.xml 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" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  <Connectors>
    <Connector startPort="Arp.Plc.Eclr/SquareWave:OP_Signal1" endPort="CPP_Counter.CPP_Counter_C-1/CPP_Counter:IP_CppEnable_b" />
    <Connector startPort="Arp.Plc.Eclr/SquareWave:OP_Signal2" endPort="Arp.Plc.Eclr/PCWE_Counter:IP_CounterEnable" />
    <Connector startPort="Arp.Io.Fblo.PnC/33:~DI8" endPort="CPP_Counter.CPP_Counter_C-1/CPP_Counter:IP_CppSwitches_uint8" />
    <Connector startPort="CPP_Counter.CPP_Counter_C-1/CPP_Counter:OP_CppCounter_uint8" endPort="Arp.Io.Fblo.PnC/33:~DO8" />
    <Connector startPort="Arp.Plc.Eclr/PCWE_Counter:OP_Counter" endPort="Arp.Io.Fblo.AxIC/0:~DO8" />
    <Connector startPort="Arp.Io.Fblo.PnD/:PND_S1_PLC_RUN" endPort="Arp.Plc.Eclr/:PND_S1_PLC_RUN" />
    <Connector startPort="Arp.Io.Fblo.PnD/:PND_S1_VALID_DATA_CYCLE" endPort="Arp.Plc.Eclr/:PND_S1_VALID_DATA_CYCLE" />
    <Connector startPort="Arp.Io.Fblo.PnD/:PND_S1_OUTPUT_STATUS_GOOD" endPort="Arp.Plc.Eclr/:PND_S1_OUTPUT_STATUS_GOOD" />
    <Connector startPort="Arp.Io.Fblo.PnD/:PND_S1_INPUT_STATUS_GOOD" endPort="Arp.Plc.Eclr/:PND_S1_INPUT_STATUS_GOOD" />
    <Connector startPort="Arp.Io.Fblo.PnD/:PND_S1_DATA_LENGTH" endPort="Arp.Plc.Eclr/:PND_S1_DATA_LENGTH" />
    <Connector startPort="Arp.Plc.Eclr/:PND_S1_OUTPUTS" endPort="Arp.Io.Fblo.PnD/:PND_S1_OUTPUTS" />
    <Connector startPort="Arp.Io.Fblo.PnD/:PND_S1_INPUTS" endPort="Arp.Plc.Eclr/:PND_S1_INPUTS" />
    <Connector startPort="Arp.Io.Fblo.AxIC/:AXIO_DIAG_STATUS_REG_HI" endPort="Arp.Plc.Eclr/:AXIO_DIAG_STATUS_REG_HI" />
    <Connector startPort="Arp.Io.Fblo.AxIC/:AXIO_DIAG_STATUS_REG_LOW" endPort="Arp.Plc.Eclr/:AXIO_DIAG_STATUS_REG_LOW" />
    <Connector startPort="Arp.Io.Fblo.AxIC/:AXIO_DIAG_PARAM_REG_HI" endPort="Arp.Plc.Eclr/:AXIO_DIAG_PARAM_REG_HI" />
  </Connectors>
</GdsConfigurationDocument>
```

Figure 2-25 Example *.gds.config

2.6.1 Notes on designating GDS ports

- When designating GDS ports, please observe the following conventions. Correct functioning can only be ensured if you adhere to the following specifications:

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 “_”.

2.6.2 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 PC Worx Engineer, you can only enter permitted combinations of data types.
If you implement the configuration without PC Worx Engineer, but via the XML configuration file, you must ensure that only the data type combinations listed in Table 2-1 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 59).

Table 2-1 Supported data type combinations of C++, Simulink® and PC Worx Engineer programs

C++	Simulink®	PC Worx Engineer	Use in data type Array	Use in data type Struct
boolean	boolean	BOOL	~*	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
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

*Only supported between C++ and C++ programs and between IEC 61131-3 and IEC 61131-3. Not between C++ and IEC 61131-3.

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

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

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

2.7 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 the GDS data using the "IDataAccessService" RSC service.

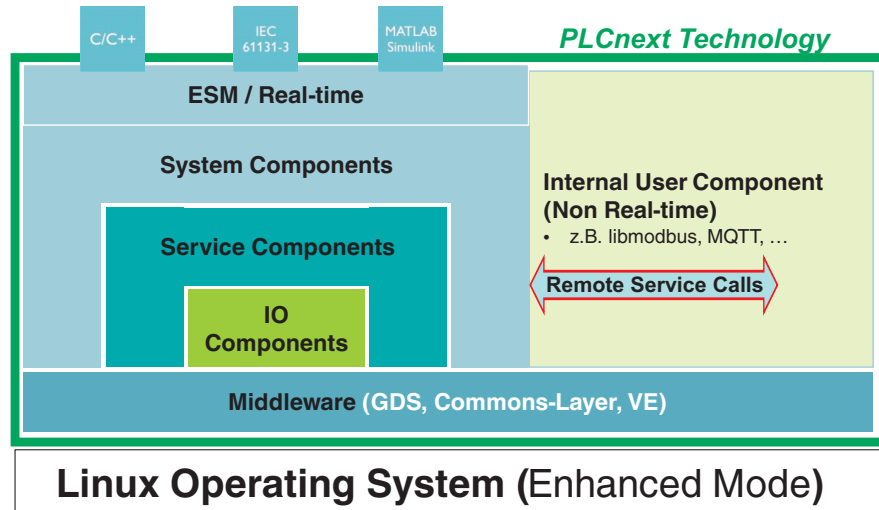


Figure 2-26 PLCnext Technology – RSC services

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. Embed the "ServiceManager" class into the header file via the "#include" command.

```
#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"
```

Figure 2-27 Example header file: #include ServiceManager.hpp

Request a service instance via the "SubscribeServices" command. With the "GetService" method, you can define the interface that you want to use. An RSC service is called up by the ServiceManager as follows:

```
void ExampleComponent::SubscribeServices()
{
    // Get service handle
    this->deviceStatusService = ServiceManager::GetService<IDeviceStatusService>();
}
```

Figure 2-28 Example: Calling up an RSC service



Please note that the execution of RSC services can take up a large amount of time (in particular Axioline and PROFINET services). For this reason, avoid direct call up 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 from Axioline devices (see section 3.8 on page 61)
- PROFINET services: Read and write access to data and information from PROFINET devices (see section 3.9 on page 62)
- Device interface services: Access to information and properties of the operating system and controller hardware (see section 3.10 on page 64)
- GDS services: Read and write access to the GDS data (see section 3.11 on page 68)

2.8 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 applications (e.g. SQL server). PLCnext Technology uses this 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
- OpenVPN
- StrongSwan
- SSH/SFTP
- NTP
- DNS

2.8.1 Directories of the firmware components in the file system

PLCnext Technology controllers work with a Linux operating system. You can access the controller via SFTP or via SSH, view the directories and files on 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. Settings that you have configured yourself (e.g. network configuration, project bus configuration, PC Worx Engineer project, etc.) are stored as directories and files on the SD card. Directo-

ries and files that you modify on the internal parameterization memory are stored on the SD card. The Linux operating system generates an overlay filesystem on the SD card from the directories changed on the internal parameterization memory and files.

Table 2-3 Storage of the FW components in the root file system

Directory in the root file system	Contents
/usr/local/lib	Directory for storing additional open source libraries that are used by customized C++ programs. You will find further information on programming the AXC F 2152 with C++ in the PLCnext Community at plcnext-community.net
/usr/share/common-licenses	License information for the individual Linux packages of the AXC F 2152
/opt/plcnext	Home directory of the Linux user "admin" 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	Log files of the diagnostic logger You will find the Output.log file here. This 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/PCWE	Directory for storing PC Worx Engineer projects. All files and subdirectories in this directory are managed exclusively by PC Worx Engineer. <ul style="list-style-type: none"> Do not make any changes to this directory.

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.

2.8.2 Firewall



The firewall is deactivated by default.

Recommended:

- Activate the firewall

Please note:

If you use the AXC F 2152 as a PROFINET controller, you must authorize all incoming connections via all UDP ports if the firewall is enabled. Otherwise, establishing a connection to certain PROFINET devices is not possible.

The firewall of the PLCnext Technology controller is based on internal Linux mechanisms (network filters) and is configured in the shell using nftables. Access is via SSH (Secure Shell).

Access via SSH requires authentication with a user name and password.



Please note:

Authentication with user name and password is always required for SSH access and cannot be deactivated.

Administrator rights are required for SSH access.

The following access data is set by default with administrator rights:

User name: admin

Password: Printed on the controller.

You can control the firewall with the following shell commands:

Table 2-4 Shell commands for controlling the firewall

Shell command	Description
sudo /etc/init.d/firewall start	Temporarily activates firewall This setting is no longer active after a re-start.
sudo /etc/init.d/firewall stop	Temporarily deactivates firewall This setting is no longer active after a re-start.
sudo /etc/init.d/firewall activate	Permanently activates firewall The firewall remains activated even after the AXC F 2152 is restarted.
sudo /etc/init.d/firewall deactivate	Permanently deactivates firewall The firewall remains deactivated even after a restart of the AXC F 2152.

The following controller firewall rules are stored in the “plcnext-filter” file in the /etc/nftables upon delivery:

Permitted packets/connections	Blocked packets/connections
Outgoing ICMP echo requests and the associated ICMP echo replies Ping commands can be issued by the controller.	Incoming ICMP echo requests The controller cannot be reached using a ping command.
Incoming connections via SSH (port 22) (e.g. for SSH or SFTP connection)	All incoming connections via TCP ports (except for explicitly approved ports, see left column of table)
Incoming HTTPS connections (port 443) Access to the web server (PC Worx Engineer HMI and WBM)	All incoming connections via UDP ports

Permitted packets/connections	Blocked packets/connections
Incoming connections via HTTP (port 80) The connections are diverted directly to port 443.	
Incoming connections via TCP port 41100 Common remoting (TLS-encoded), e.g. via PC Worx Engineer	
Incoming connections via TCP port 17725 The TCP port 17725 is the standard port for the external mode of MATLAB® Simulink®.	
Incoming connections via TCP port 4840 Standard port for connections to the OPC UA server of the controller	
Incoming connections via any TCP and UDP port	

2.8.3 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 ntpd. Execute the script "sudo /etc/init.d/ntpd" and the changes made in the configuration file will be active after the next controller boot up.

In the standard configuration, the operating system time is synchronized with the RTC (Real-Time Clock) 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 (NTP) Demon 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>.

Changing the system time manually

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

User name: admin

Password: Printed on the controller.

- Open the shell.

Requesting the system time

- Request the system time via the "date" command.

Setting the system time

- Enter the shell command "sudo date -s "YYYY-MM-DD hh:mm:ss".
 - YYYY: Year

- MM: Month
- TT: Day
- hh: Hours
- mm: Minutes
- ss: Seconds

Setting the system time in PC Worx Engineer

You can also set the system time using the PC Worx Engineer software.

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

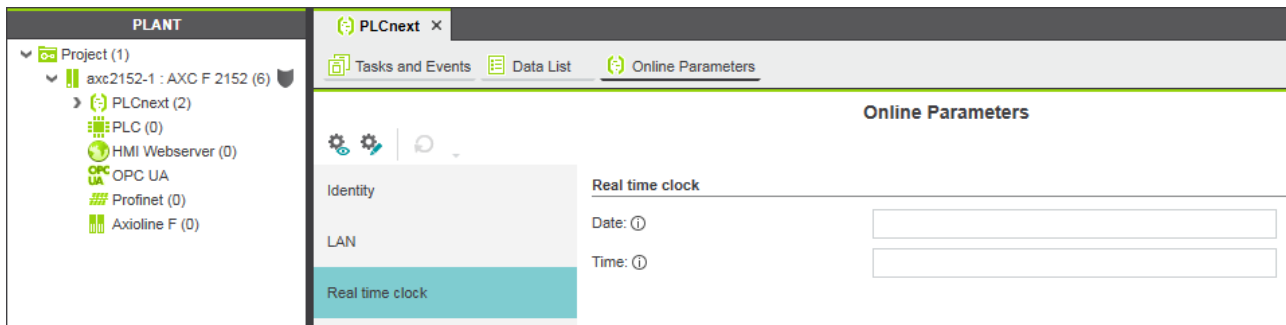


Figure 2-29 Setting the system time in PC Worx Engineer

2.8.4 OpenVPN™ client

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

You will find a description on how to set up a VPN tunnel using openVPN™ in the PLCnext Community at plcnext-community.net.

2.8.5 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.

You will find further details at <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 for starting, stopping and restarting the demon:
 - Start: “sudo ipsec start”
 - Stop: “sudo ipsec stop”
 - Restart: “sudo ipsec restart”
- Use the following command to call up the status: “sudo ipsec status”

Configuration examples

You will find configuration examples at <https://wiki.strongswan.org>.

2.8.6 Text editors

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

2.8.7 User rights

When a PLCnext user logs into the SSH console with the “admin” user role, he is also recognized with the same name and password by the Linux system. The user is thus assigned to the “plcnext” Linux group. Files that this user may read, write to and/or execute are assigned to the “plcnext” group file system.

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 possible:

Table 2-5 User rights

Right	PLCnext group	admin	sudo required
Setting and inspecting IP settings (incl. 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
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		x	x
Reading PLCnext log files	x	x	
Calling up and configuring TOP/HTOP	x	x	
Updating firmware (via RAUC)		x	
Updating firmware via update script		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

Table 2-5 User rights

Right	PLCnext group	admin	sudo required
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

3 Structure of a C++ program



If you want to consult the PLCnext Technology SDK C++ header files in parallel while reading this section, install the necessary tools as described in “Creating programs with C++” on page 69 first.

With PLCnext Technology, programs that were created in different programming languages can be used together. The PLCnext Technology firmware provides the programs with a uniform basis for instantiating and calling up. This basis applies for 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 library class is the smallest unit that can be downloaded. It represents an *.so file (shared object).
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. You create your application by deriving from these base classes.

The structure of an application can be compared to a bookshelf (see Figure 3-1). The bookshelf itself can be compared to the library class, the shelves to the component class and the books to the program class.

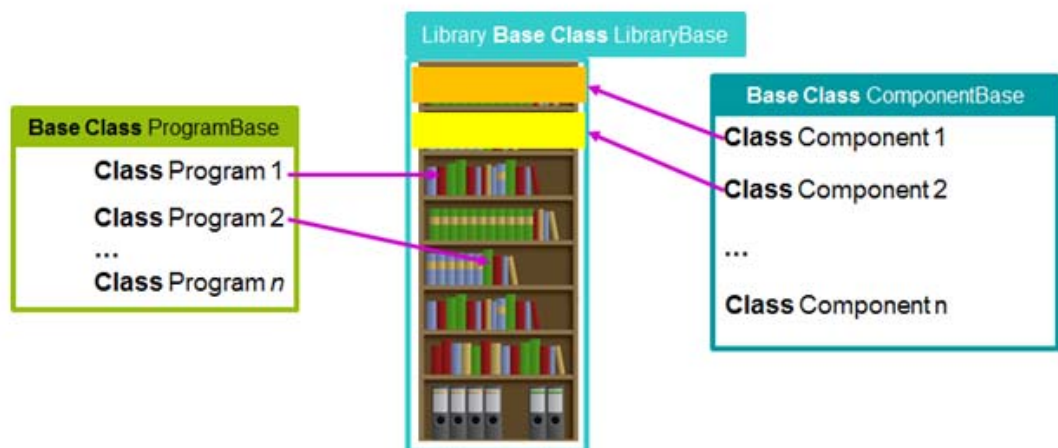


Figure 3-1 Structure of a C++ application

3.1 ILibrary or LibraryBase



If you use the Eclipse® add-in, it creates the following functions with a functional implementation. In many cases, that is sufficient. If you have special requirements that go beyond this, the following descriptions will help you understand the functions.

Each class is defined in an individual header file (*.hpp). To use the class, you have to include it (“#include”).

- “ILibrary”: Arp/System/Acf/ILibrary.hpp
- “LibraryBase”: Arp/System/Acf/LibraryBase.hpp

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

```
extern "C" ARP_CXX_SYMBOL_EXPORT void DynamicLibrary_Main
(AppDomain& appDomain);
```

The PLCnext Technology firmware queries the library object as follows:

```
extern "C" ARP_CXX_SYMBOL_EXPORT ILibrary*
DynamicLibrary_GetInstance(void);
```

The library object implements the “ILibrary” interface by deriving it from the “LibraryBase” class. This in turn implements the following function in particular:

```
IComponentFactory& ILibrary::GetComponentFactory(void);
```

The PLCnext Technology firmware can thus create instances of the components when loading the PLC program.

```
ILibrary.GetComponentFactory().CreateComponent(IApplication&
application, const String&componentType, const
String&componentName);
```

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

The following parameters are transferred in the process:

application	
componentType	Class name of the component Which components (type) a library can generate are described in the *.libmeta and *.compmeta files for the respective library in addition to the implementation.
componentName	Instance name of the component This is automatically configured in PC Worx Engineer during instantiation of programs or manually via the *.acf.config file.

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, the singleton implementation is performed for each process. The “AppDomain” class is used for this. Since the PLCnext Technology firmware for user components uses the same mechanism for instantiation, initialization and administration as it does for core components, the “AppDomain” is also used here.

“IApplication”

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

3.2 Several component types in the same library

The Eclipse® add-in creates exactly 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, the following function is called up in the constructor of the library object for each component type:

```
ComponentFactory::AddFactoryMethod(const String&
ComponentKind, FactoryMethod factoryMethod);
```

- “ComponentKind”:
Enter the name of the C++ class here (without any namespaces used). This name is also used in the configuration files.
- “factoryMethod”:
Enter the following (static) function of the respective component class:

```
IComponent::Ptr Create(IApplication& application, const
String& componentName);
```

There are two possible components to instantiate:

- Default.acf.config (projects/Default/Default.acf.config)
- Plm.acf.config (projects/Plc/Plm/Plm.acf.config). When used as a PC Worx Engineer library, this is performed automatically by PC Worx Engineer, but it can also be performed manually (see section “Configuring tasks via configuration files” on page 24).

3.3 IComponent or ComponentBase

The basic integration of a component into PLCnext 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 that are described in the following subchapters. To be able 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 up for each component by the PLM:

- 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 start operation of the components that implement the IControllerComponent is called up. The System Manager calls up the following IComponent operations:

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

Initialize

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

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

SubscribeServices

Resources that have been allocated and initialized in the “Initialize()” function have to be enabled in the “Dispose()” function. Thereafter, the firmware calls up the following function for each instantiated component. Here, a component can obtain RSC services that have already been registered.

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

LoadSettings

Subsequently, the “LoadSettings()” function is called up. The path to the settings (“settingsPath”) can be specified for the respective component instance in the *.acf.config 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 up 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 up:

```
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 up at different times.

- The PLC Manager manages components that make user programs available. The PLC Manager configures these in a file referenced by “/opt/plcnext/projects/Default/Plc/Plm/Plm.acf.config”.
- Internal user components are managed by the Application Control Framework (ACF) 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 the following functions are called up:

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

3.3.1 IProgramComponent and IProgramProvider

A component that can instantiate user programs is derived from “IProgramComponent”.

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

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

With these, you implement the following function:

```
IProgramProvider* GetProgramProvider(void);
```

The ProgramProvider in turn makes the following function available for instantiating programs. The PLCnext Technology firmware calls up this function when loading the PLC program:

```
IProgram* CreateProgram(const String& programName, const String& programType);
```

In the process, the following parameters are transferred as defined in the *.esm.config files:

programName	Instance name of the program The instance name is configured in the PC Worx Engineer task editor or manually via the *.esm.config file.
programType	Class name of the program Which programs (type) a component (type) can generate are described in *.compmeta and *.progmeta files for the respective component.

The Eclipse® add-in creates such a component class which already implements an IProgramProvider and a user program class.

If a component class is to be able to instantiate several program classes, the following steps are necessary:

- Create the class (*.h*, *.c*)
- For each program class, create a *.progmeta file.
- Integrate this into the *.compmeta file of the component.
- Supplement the program classes in the “CreateProgram()” function.

3.3.2 IProgram or ProgramBase

An instantiated user program implements the “IProgram” interface. As a result, a constructor to which the instance name is transferred and the “Execute()” function that is called up by the ESM task during each pass are available. The Eclipse® add-in creates such a user program class that inherits the “IProgram” interface by means of derivation from the “ProgramBase” base class. Furthermore, the “ProgramBase” base class provides the “AddPortInfo()” function. This makes it possible to make the addresses of the IN and OUT ports known to the GDS (Global Data Space) in the constructor.

To be able 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

```
class ProgramBase : public IProgram
{
public: // construction/destruction
    ProgramBase(const String& programName);
    virtual ~ProgramBase(void) = default;

public: // IProgram implementation
    const String& GetFullName(void) const override;
    bool GetPortInfo(const String& portName, PortInfo& portInfo) const override;
    void SetInitialValues(void) override;

public: // abstract IProgram operations
    virtual void Execute(void) = 0;

protected: // operations
    template<class T>
    void AddPortInfo(const String& portName, T& portValue);
};
```

3.3.3 IControllerComponent

An internal user component may need individual lower-priority threads to perform longer tasks outside of the ESM task. To this end, the internal user component can implement the “IControllerComponent” interface in addition to “IComponent”.

The “IControllerComponent” defines the following two functions:

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

If the component is managed by the PLC Manager:

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

If the component is managed by the Automation Component Framework (ACF):

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

To use the class, you have to include (“#include”) the corresponding header file (.hpp). Both components and programs can be derived from this template class.

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

3.4 PLM (Program Library Manager)

The Program Library Manager (PLM) 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 lifetime of the component instance in accordance with the statuses of the controller and the status changes that result from the PC Worx Engineer command:

- Cold or warm restart
- Hot restart
- Reset
- Download (a reset is implicitly performed prior to the download)

3.4.1 Requirements on the components

- In order that the PLM can manage components that make user programs available, they must implement the “IComponent” interface.
- The name of a component type is transferred to the “ComponentFactory” as a character string during registration of the component-generating function.

```
this->componentFactory.AddFactoryMethod("CppConfigComponent", &CppConfigComponent::Create);
```

The name can be freely selected. Permissible characters are upper and lower case letters, numbers and “_” (underscore). The exact name also has to be entered into the *.compmeta meta file.

If a component has to instantiate programs that have IN or OUT ports, the name has to match the name of the C++ class. If this is not the case, the GDS cannot find the IN or OUT port and ends the loading process with an exception.

3.4.2 Functions

The PLM takes on the role of creating, configuring and destroying the components that can instantiate user programs. Here, the application components are controlled as follows:

Table 3-1 IComponent (PLM) functions

Calling up IComponent	User action in PC Worx Engineer
void Initialize()	<ul style="list-style-type: none"> – Restart – Send project
void SubscribeServices()	<ul style="list-style-type: none"> – Restart – Send project
void LoadSettings(const String& settingsPath)	<ul style="list-style-type: none"> – Restart – Send project
void SetupSettings()	<ul style="list-style-type: none"> – Restart – Send project
void PublishServices()	The function is only called up by a firmware component, not by the PLM.
void LoadConfig()	<ul style="list-style-type: none"> – Restart – Cold restart – Warm restart – Send project
void SetupConfig()	<ul style="list-style-type: none"> – Restart – Cold restart – Warm restart – Send project
void ResetConfig()	<ul style="list-style-type: none"> – Restart – Cold restart – Warm restart – Send project
void Dispose()	<ul style="list-style-type: none"> – Send project

3.4.3 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 format AcfConfigurationDocument is used to configure components that can instantiate user programs. The elements Library, Component (name, type, library, isEnabled), und Settings (path) are evaluated by the PLM (see section “Configuration files” on page 18).

3.5 ACF (Application Component Framework)

The Application Component Framework (ACF) is the foundation for the PLCnext Technology-platform architecture. The AFC is a framework that enables component-based platform development and the configurative composition of the firmware for the

devices. It can distribute the firmware configuratively to one or more processes. The ACF enables the configurative integration of user functions into the system. You can thus extend PLCnext Technology devices with your own functions.

The ACF loads the various share-object files, and starts and manages the components contained therein in the desired sequence. In PLCnext Technology, components are instantiated as classes, such that several instances of one component type can exist. Instantiation is performed via configuration files for the ACF.

3.5.1 Libraries

ACF libraries are loaded dynamically by configuration of the ACF. They serve as the configurative extension of the system through platform or user functionality. The ACF libraries must be available as dynamic libraries (*.dll or *.so) and contain one or more ACF components. This enables the user to construct the firmware configuratively and extend it with functionalities.

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. The implementation consists of:

- 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. the ComponentFactory of the ACF.
- Two exported C functions
- ComponentFactory

See section “Library or LibraryBase” on page 46.

Loading libraries

A library is added to the ACF configuration as shown in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<AcfConfigurationDocument
  xmlns="http://www.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>
```

Figure 3-2 ACF configuration – adding a library

The following table lists explanations of the individual attributes:

XML element	Description
<Libraries>	A list of libraries. Libraries are listed here that are to be loaded to create components.
<Library>	The definition of a library.

XML element	Description
name	The name of the library referenced in a component configuration.
binaryPath	The binary path of the library that is to be loaded.

3.5.2 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 that the ACF can integrate them into the firmware dynamically (see section 3.3 "IComponent or ComponentBase").

ACF components are instantiated once or several times. They are assigned with a system-wide unique name.

Adding a component

Components are added to the ACF configuration as shown in the following example code:

```
<?xml version="1.0" encoding="UTF-8"?>
<AcfConfigurationDocument
  xmlns="http://www.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.acf.config
       are managed by PLM
  -->
  <Components>
    <Component name="AcfDemo" type="MyComponentAcF" library="MyLibraryAcF.Library"
      settings="/opt/plcnext/projects/MyLibrary/MySettings.xml" />
  </Components>
</AcfConfigurationDocument>
```

Figure 3-3 Example: Adding a component to the ACF configuration

The following table lists explanations of the individual attributes:

XML element	Description
<Components>	A list of the components. Components are listed here that are to be created.
<Component>	The definition of a component.
name	The name of the component. These must be unique throughout the entire system, because all resources in the system are addressed via the component name.

XML element	Description
type	The type name of the component. A library can provide several component types. C++ type names are typically used here. The name is used in the code to register components in the component factory. This is the only dependency between the system configuration and the source code.
library	The name of the library that contains the component and defined under <Libraries>.
<Settings path>	The path to the component settings file. If the component needs a project configuration, the path to the component-specific project configuration must be specified in the settings file.

3.5.3 Configuration

ACF configuration

Because there is just one mechanism for integrating components dynamically, 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

The project configuration is transferred with PC Worx Engineer, other tools (Eclipse, Simulink[®] extension), or manually to the controller. The configuration consists of the configuration files of the individual components and of ACF configuration files (*.acf.config). These add the user components to the system. An environment variable ARP_PROJECTS_DIR must be defined in the ACF settings for the project directory (/opt/plcnext/projects). PC Worx Engineer needs 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 sub-directories “Default” and “PCWE”. The further directory structure is constructed in the same way as the ARP component hierarchy.

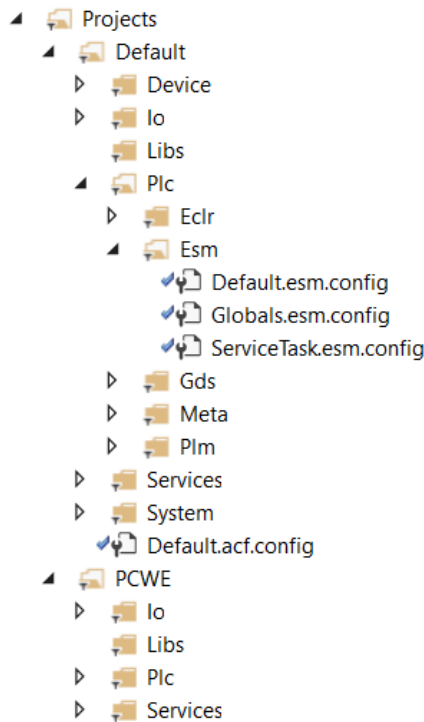


Figure 3-4 Project directory

- PCWE directory:
The “PCWE” folder is intended for the download from PC Worx Engineer. Files that are stored here will be overwritten during the next download from PC Worx Engineer (see 2.4.4 “Generating configuration files with PC Worx Engineer”).
- Default directory:
The “Default” folder is intended for storing further configuration files and for the manual, configurative extension of the platform components (see section 2.4.5 “Manual configuration”).

3.6 Common classes

Common classes provide functions that may be helpful for your 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 this into your program via an “include” command. Further information on the common classes and their applications is available directly in the code commentary.

Among others, the following helpful common classes are component parts of the Phoenix Contact SDK. The classes are encapsulated according to their subject areas in name-spaces.

3.6.1 Threading

With threading, parts of a program are executed in parallel. The “Threading” namespace provides methods of separating a program into several strings for simultaneous execution, thus improving the performance of the overall system.



NOTE: Priority

You can select a priority of between 0 and 99. In order not to disturb the structure of the real-time threads, Phoenix Contact recommends priority 0. Otherwise, the stability of the firmware cannot be guaranteed. Programs are provided in ESM tasks for performing time-critical tasks.

CpuAffinity

The “CpuAffinity” is a bitmask 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 to be started, and whether it can be executed in the runtime of another processor core. If the value of the parameter is 0, the scheduler can execute the thread on all available processor cores.

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 up, the thread is executed.

The “Thread” class selects a low priority as standard. We recommend retaining this priority in order that the priority structure of the various firmware and operating system tasks is not endangered.

WorkerThread

In contrast to the “Thread” class, an instance of the “WorkerThread” class is executed cyclically, as soon as the method “WorkerThread::Start” is executed. You can define the cyclical execution of the thread via the parameter “idletime”. The value is specified in ms.

ThreadSettings

The “ThreadSettings” class is an auxiliary class for passing 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 statuses:

- Locked
- Unlocked

Once the “Mutex:Lock” method has been performed, i.e. the call up from the method returns, the data is protected against modification by other threads. This status is retained until the instance calls up the “Unlock()” command, therefore rescinding the locked status. A call up is thus blocked until the thread which is in the “Lock” status is released again. To prevent a “Deadlock”, i.e. a status in which a locked thread cannot be unlocked again, you can use the class “LockGuard”. This class automates “Lock” and “Unlock” of a “Mutex” instance.

RwLock

This class provides a locking mechanism for increasing performance. This class is useful if several read accesses are necessary, but not many write accesses are necessary. The difference between “Mutex” class instances and this class is that the instances of the “Rw-Lock” class permits several simultaneous read accesses to the locked structure. Write access, however, is forbidden at the same time. Instances of this class are therefore suitable for all data that is often read but only rarely updated.

3.6.2 Ipc

The “Ipc” namespace (Inter-process communication) encapsulates classes which can be used to enable the communication between various processes on the same controller.

Semaphore

With this 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 up, the internal counter is lowered by one. If the current value of the counter is zero when a “Wait” method is called up, the call up is blocked until a different thread on the same semaphore instance calls up the “Post” method (the “Post” method increase the internal counter).

MessageQueue

With 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 up from the constructor will lead to an exception. The name of a queue corresponds to the file path in Linux.

3.6.3 Chrono

The “Chrono” namespace contains classes and functions with which the temporal sequences within an application can be controlled and influenced. This includes the high-resolution measurement of time elapsed so far, and also the triggering of actions after a predetermined period of time.

Timer

The “Timer” class is a high-resolution chronometer for the 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 calculated the next point in time at which the method is to be called up. You only have to implement the method or methods that you are to be called up using via Timer definition.

3.6.4 Io

The namespace “Io” encapsulates all functions necessary for working with files and folders within the file system of the underlying operating system.

FileStream

The “FileStream” class is for the 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.

3.6.5 Net

The “Net” namespace encapsulates all classes and functions that enable network-based communication between processes on the same or separate controllers. Use this for processes that run on the same controller, preferably the functions from the “Ipc” namespace (see section “Ipc” on page 58).

Socket

The “Socket” class is an interface for Ethernet based communication. Use instances of this class to establish an Ethernet connection to peers. Currently, the protocols UDP and TCP are supported (only IPv4).

3.6.6 Runtime

The “Runtime” namespace encapsulates functions for the manipulation of individual processes that are managed by ARP firmware.

SharedLibrary

With the “SharedLibrary” class, shared libraries (.so files) are dynamically published in the runtime in applications or reloaded. If a library is successfully reloaded with the “SharedLibrary::Load” method, the symbols contained (global variables, classes, methods, functions, etc.) are then published in the current program. The memory area can be requested with “GetFunctionAddress” in order that the functions of the library can be used in the program currently running.

Process

The “Process” class is a high-level API for creating and managing new processes.

3.7 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 is logged. You will therefore find valuable information there that can help you in finding the causes of problems.

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

– “Loggable”: Arp/System/Commons/Logging.h

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.

Template Loggable

You can integrate 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.

The following log levels are supported. For each log level, there is a suitable method that can be called up.

- Info
- Warning
- Error
- Fatal

Example call up:

```
log.Info("Info!");
```

Static call up

Alternatively you can also perform logging without the “Loggable” class. Messages can be written without creating a special logger with 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 transfer and format variables. Placeholders in the form {x} are used for the variables; where x is the index of the variable.

```
("Variable a={0} b={1}", a, b);
```

Diagnostics log file

The “Output.log” diagnostics log file contains status information, warnings, error messages and debugging messages. You can find the file in the /opt/plcnx/logs folder on the file system of your controller. The file system is accessed via the SFTP protocol. SFTP client software is required for this (e.g. WinSCP) (see section “Directories of the firmware components in the file system” on page 37).

The diagnostics log file is configured such 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 up and evaluated as soon as possible.

The diagnostics log file contains the following message types:

- **Error & Fatal:** If messages of the type “Error” or “Fatal” are issued, the controller is stopped. The errors mainly arise during startup or during the execution of a user program.
- **Warning:** Warnings indicate potentially occurring errors.
- **Information:** The core components issue messages of the type “Information”. These provide an overview of the system status.

<p>A \</p> <pre>07.05.18 08:24:15.830 MyLibrary.MyComponent 07.05.18 08:24:15.831 Arp.Plc.Plm.Internal.PlmManager 07.05.18 08:24:15.831 MyLibrary.MyComponent 07.05.18 08:24:15.832 MyLibrary.MyComponent 07.05.18 08:24:15.832 MyLibrary.MyComponent 07.05.18 08:24:15.833 MyLibrary.MyComponent 07.05.18 08:24:15.833 MyLibrary.MyComponent 07.05.18 08:24:15.834 MyLibrary.MyComponent 07.05.18 08:24:15.834 MyLibrary.MyComponent 07.05.18 08:24:15.988 MyLibrary.MyComponent.MyProgram 07.05.18 08:24:15.989 MyLibrary.MyComponent.MyProgram 07.05.18 08:24:16.121 Arp.Io.Axioline.AxiolineComponent 07.05.18 08:24:16.127 Arp.Io.Axioline.AxiolineComponent</pre>	<p>D \</p> <pre>INFO - 'MyComponent' invoked of object with instance name 'MyLibrary.MyComponent-1' INFO - Component 'MyLibrary.MyComponent-1' from library 'MyLibrary' created. INFO - 'Initialize' invoked of object with instance name 'MyLibrary.MyComponent-1' INFO - 'SubscribeServices' invoked of object with instance name 'MyLibrary.MyComponent-1' INFO - Component 'AcfDemo' not found! INFO - 'LoadSettings' invoked of object with instance name 'MyLibrary.MyComponent-1' INFO - 'SetupSettings' invoked of object with instance name 'MyLibrary.MyComponent-1' INFO - 'LoadConfig' invoked of object with instance name 'MyLibrary.MyComponent-1' INFO - 'SetupConfig' invoked of object with instance name 'MyLibrary.MyComponent-1' INFO - Added Port 'zaehler (of Data Type 8)' of instance MyLibrary.MyComponent-1/MyProgram1 INFO - Added Port 'zaehler (of Data Type 8)' of instance MyLibrary.MyComponent-1/P1 INFO - Axioline: Load configuration. INFO - AxiolineComponent::LoadPlc() Path= /opt/plcnx/projects/PCWE/Io/Arp.Io.AxlC/links.xml</pre>
<p>B \</p>	<p>E \</p>

Figure 3-5 Example: “Output.log” diagnostics log file

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 triggered the message
D	Message type (log level)
E	Message, e.g. info text, error message, debugging message

3.8 RSC Axioline services



Please note that the execution of RSC services can take up a large amount of time. For this reason, avoid direct call up from ESM tasks.

The Axioline component can be extended via interfaces for Axioline services. You can use one interface for acyclic communication (PdiRead, PdiWrite). This is available via the RSC protocol. The parameter data, diagnostics information, and status information (PDI = Parameters, Diagnostics and Information) of an Axioline device can be read out or written with the RSC service. Acyclic communication is suitable for the exchange of data that does not recur cyclically.

IAcyclicCommunication Service

The “IAcyclicCommunicationService” RSC service in the “Arp/Io/Axioline/Services” namespace for acyclic communication makes the following methods available:

- “PdiRead”:
Enables the parameters, diagnostics and information of an Axioline device to be read off
- “PdiWrite”
Enables the 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” is used for transmitting the input parameters. The structure has the following elements:

Parameter	Description
uint16 Slot	Device number
uint8 Subslot	Sub-device number
uint16 Index	Object index
uint8 Subindex	Object sub-index

The return values are written to the “PdiResult”. The structure has the following elements:

Parameter	Description
uint16 ErrorCode	Error code
uint16 AddInfo	Error code, further information

Data that is read off (“PdiRead”) and data that is to be written (“PdiWrite”) is transferred to uint8-type vector (“data”). A description of the data (maximum size, type, etc.) of the object description is to be found in the data sheet of the respective Axioline module.

You need the following headers to use the service. Integrate these via “#include” if necessary:

- 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 PC Worx 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 are available for downloading at phoenixcontact.net/qr/2404267/manual. Further information is also available in the data sheets of the respective Axioline modules.

3.9 RSC PROFINET services



Please note that the execution of RSC services can take up a large amount of time. For this reason, avoid direct call up from ESM tasks.

The PROFINET component can be extended via interfaces for PROFINET services. You can use one interface for acyclic communication (RecordRead, RecordWrite). This 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 out or written with the RSC service. Acyclic communication is suitable for the exchange of data that does not recur cyclically.

IAcyclicCommunication Service

The “IAcyclicCommunicationService” RSC service in the “Arp/Io/ProfinetStack/Controller/Service” namespace for acyclic communication makes the following methods available:

- “RecordRead”:
Enables the parameters, diagnostics and information of a PROFINET device to be read off
- “RecordWrite”
Enables the parameters, diagnostics and information of a PROFINET device to be written

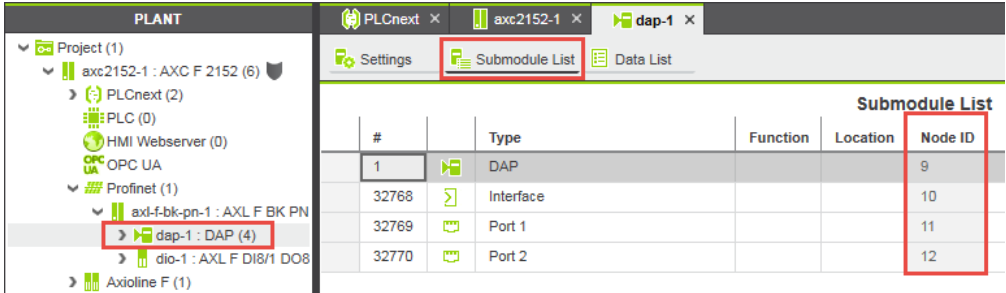
```
RecordResult RecordRead(const RecordParam& recordParam, std::vector<uint8>& data)
RecordResult RecordWrite(const RecordParam& recordParam, const std::vector<uint8>& data)
```

Parameters are necessary for executing “RecordRead” and “RecordWrite”. The “**RecordParam**” 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 Sub-slot.

If one version is selected for addressing, the parameters of the other version must be 0 or empty.

The structure has the following elements:

Parameter	Description
uint16 Id	<p>Node ID of the submodule</p> <p>The ID is assigned automatically. It can be viewed in PC Worx Engineer (version 1). To do so, open the Submodule List of the PROFINET submodule node.</p> 
RscString<512> Device Name	<p>DeviceName</p> <p>Name of the device that is to be addressed (version 2)</p>
uint16 Slot	Device number (version 2)
uint8 Subslot	Sub-device number (version 2)
uint16 Index	Object index
uint8 Length	<p>Maximum data amount</p> <p>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").</p>

The return values are written to the **"RecordResult"** structure. The structure has the following parameters. Further information on the parameters is to be found in the PROFINET specification (version 2.3):

- boolean ServiceDone
- uint8 ErrorCode
- uint8 ErrorDecode
- uint8 ErrorCode1
- uint8 ErrorCode2
- unit16 AddData1
- unit16 AddData2

Table 3-2 ErrorCode1

Error code	Description
0	No errors
0xF0	<p>Internal error</p> <p>The error can be evaluated more precisely via the ErrorCode2.</p>
further error codes	<p>If a value other than 0xF0 or 0 is displayed in the ErrorCode1 element, the error is described in the fields ErrorCode, ErrorDecode, ErrorCode1, ErrorCode2, AddData1 and AddData2 via the PROFINET specification.</p>

Table 3-3 ErrorCode2

Error code	Description
0x03	"InvalidAddress" ID or DeviceName/Slot/Subslot are invalid.
0x15	"RecordReadFailed" RecordRead failed. This error can occur, for example, if several participants attempt to execute "RecordRead" on one device simultaneously.
0x16	"RecordWriteFailed" RecordWrite failed. This error can occur, for example, if several participants attempt to execute "RecordWrite" on one device simultaneously.
0x17	"Timeout" A confirmation has not been received.

Data that is read off ("RecordRead") and data that is to be written ("RecordWrite") is transferred to unit8-type vector ("data"). A description of the data (maximum size, type, etc.) of the object description is to be found in the PROFINET specification (version 2.3).

You need the following headers to use the service. Integrate these via "#include" if necessary:

- 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

3.10 RSC Device Interface services



Please note that the execution of RSC services can take up a large amount of time. For this reason, avoid direct call up 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 up the information with the following interfaces and defined parameters. The following headers are needed to use the service. Integrate these via "#include" if necessary:

- Arp\Device\Interface\Services\IDeviceInfoService.hpp
- Arp\Device\Interface\Services\IDeviceStatusService.hpp

IDeviceInfoService

The IDeviceInfoService RSC interface enables read access to device information. The status value of a parameter is read off with the method "GetItem()". The status values of several parameters are read off with the method "GetItems()".

```
RscVariant<512> GetItem(const RscString<512>& identifier)
void GetItems(GetItemsIdentifiersDelegate identifiersDelegate, GetItemsResultDelegate resultDelegate)
```


The following parameters are available for calling up information:

Table 3-4 IDeviceInfoService – Parameters

Parameter	Data type	Description
General.DeviceClass	UInt32	The “DeviceClass” specifies the device class (currently, only “ProgrammableLogicController” is supported) 0: undefined 1: ProgrammableLogicController 2: BusCoupler 3: Switch
General.VendorName	String	The “VendorName” parameter states the name of the manufacturer.
General.ArticleName	String	The “ArticleName” parameter states the device name.
General.ArticleNumber	String	The “ArticleNumber” parameter states the order number of the device.
General.SerialNumber	String	The “SerialNumber” parameter states the serial number of the device.
General.Firmware.Version	String	The “FirmwareVersion” parameter states the firmware version of the device. Here, the 5-level notation Major, Minor, Patch, Build, Status is used.
General.Firmware.VersionMajor	Byte	“FirmwareVersionMajor”
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 states the hardware version of the device.
General.Fpga.Version	String	The “FPGAVersion” parameter states the FPGA version of the device. Here, the 3-level notation Major, Minor, Patch is used.
General.Fpga.VersionMajor	Byte	“FPGAVersionMajor”
General.Fpga.VersionMinor	Byte	“FPGAVersionMinor”
General.Fpga.VersionPatch	Byte	“FPGAVersionPatch”
Interfaces.Ethernet.Count	Byte	The “NoOfNetworkInterfaces” parameter states the number of network interfaces.
Interfaces.Ethernet.{adapterIndex}.port}.Mac	String	The “Mac-Address” parameter states 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

IDeviceStatusService

This RSC interface enables read access to status information. The status value of a parameter is read off with the method "GetItem()". The status values of several parameters are read off with the method "GetItems()". Use the method "deviceStatusService.GetItem("Parameters")" to call up status information.

```
RscVariant<512> GetItem(const RscString<512>& identifier)
void GetItems(GetItemsIdentifiersDelegate identifiersDelegate, GetItemsResultDelegate resultDelegate)
```

The following parameters are available for calling up information:

Parameter	Data type	Description
Status.DeviceHealth	Byte	The "DeviceHealth" parameter states the operating status of the device. 0: OK 1: WARNING 2: ERROR
Status.Cpu.Load.Percent	Byte	The "CPULoad" parameter states the complete processor utilization of the device as a percentage. 0 ... 100[%]
Status.Cpu{0}.Load.Percent	Byte	The "CPULoad{Core}" parameter states the processor utilization of the selected processor core of the device as a percentage. 0 ... 100[%] 0x64 = 100% Core = 0, 1, 2 etc.
Status.Memory.Usage.Percent	Byte	The "MemoryUsage" parameter states the complete memory utilization of the device as a percentage. 0 ... 100[%] 0x64 = 100%
Status.ProgramMemoryIEC.Usage.Percent	Byte	The "ProgramMemoryUsage" parameter states the program memory utilization of the IEC runtime of the device as a percentage. 0 ... 100[%] 0x64 = 100%
Status.DataMemoryIEC.Usage.Percent	Byte	The "DataMemoryUsage" parameter states the data memory utilization of the IEC runtime of the device as a percentage. 0 ... 100[%] 0x64 = 100%
Status.RetainMemory.Usage.Percent	Byte	The "RetainMemoryUsage" parameter states the complete retain memory utilization of the device as a percentage. 0 ... 100[%] 0x64 = 100%
Status.RetainMemoryIEC.Usage.Percent	Byte	The "RetainMemoryUsage" parameter states the complete retain memory utilization of the device as a percentage. 0 ... 100[%] 0x64 = 100%
Status.Board.Temperature.Centigrade	Int32	The "BoardTemperature" parameter states the temperature of the interior of the device in °C.
Status.Board.Humidity	Byte	The "BoardHumidity" parameter states the relative humidity in the device. 0 ... 100[%]

You can also call up status information on the LED statuses 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 LedColorBites : ushort
{
    Green = 1,
    Yellow = 2,
    Red = 4
};
```

The status of the LEDs are 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 3-5 IEC runtime system LEDs

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 3-6 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 3-7 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 up status information on the network statuses via the “IDeviceStatusService” interface:

Table 3-8 Network interface

Parameter	Data type	Description
Status.Interfaces.Ethernet.{adapterIndex}.{port}.Baudrate	Byte	The “Interface-Baudrate” states 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” states 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-State” states the link status of the interface. 0: linkDown 1: linkUp adapterIndex = 1, 2, ... port = 1, 2, ...

3.11 RSC GDS services



Please note that the execution of RSC services can take up a large amount of time. For this reason, avoid direct call up from ESM tasks.

IDataAccessService

The internal user components can be accessed with this RSC service on the real-time area of the PLCnext Technology platform. You can gain read and write access to the GDS data during the runtime with the “IDataAccessService” interface. The service enables the asynchronous reading and writing of one or more ports or even internal variables. For this access, you need the name of the port(s) that is(are) to be written to or read off. The data read off can, for example, be written to a database.

You need the following header to use the service. Integrate this via “#include” if necessary:

– Arp/Plc/Gds/Services/IDataAccessService.hpp

Methods

- Read Single / Read Multi: These methods are intended for the reading off of data from a port or several ports.
- Write Single / Write Multi: These methods are intended for the writing data to a port or several ports.

```
ReadItem      ReadSingle(const RscString<512>& portName)
               Read(ReadPortNamesDelegate portNamesDelegate, ReadResultDelegate resultDelegate)
DataAccessError WriteSingle(const WriteItem& data)
               Write(WriteDataDelegate dataDelegate, WriteResultDelegate resultDelegate)
```

Each of the ports have a unique name within the GDS that is made up as described in section “GDS configuration using configuration files” on page 31.

4 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 classic IEC-61131-3 programs.

To use programs and program parts created in C++ within the scope of the PLCnext Technology, you need a software development kit (SDK).

Recommended: Use the freely available Eclipse® software with Phoenix Contact add-in for Eclipse® as a development environment for C++ programming. The add-in enables the easy connection to the PLCnext Technology platform.

To use C++ programs in the same way as IEC 61131-3 applications in PC Worx Engineer, the PLCnext Technology LibraryBuilder is required.

If you quickly create a program with Eclipse® and want to make it available by using the Eclipse® add-in for PC Worx Engineer, read this section.

4.1 Installing PLCnext Technology SDK and add-in for Eclipse®

4.1.1 Requirements



An Internet connection is required for installing the components.



A description of the PLCnext Technology SDK and add-in for Eclipse® under Linux is available in the PLCnext-Community under plcnext-community.net.

- Before starting the installation, ensure that the system requirements are met and download the necessary software.

Operating system

- Microsoft® Windows® 7 or
- Microsoft® Windows® 10

Phoenix Contact software

- PLCnext Technology SDK:
The SDK contains all of the important tool chains and libraries required for creating a program.
- Phoenix Contact add-in for Eclipse®:
The add-in for Eclipse® supports you during the creation of metafiles as well as IN and OUT ports.
- EngineeringLibraryBuilder:
The LibraryBuilder generates a library from the C++ program which can be read by PC Worx Engineer via metadata and shared object. The C++ program can be used in the same way as an IEC 61131-3 program in PC Worx Engineer.

Software packages required for programming in C++ with Eclipse are to be found in the download area of the AXC F 2152 controller at phoenixcontact.net/products (order no.: 2404267) in the following categories:

- “Configuration software”: You will find the Phoenix Contact add-in for the Eclipse® development environment here.
- “Software”: You will find the Phoenix Contact LibraryBuilder for creating a library from Eclipse® for use in PC Worx Engineer as well as the Phoenix Contact SDK (Software Development Kit) here.

Some sample applications for the AXC F 2152, programmed in C++, are available in the “Sample applications” category for downloading.

C++ development environment

- Eclipse® Neon 3 / Eclipse IDE for C/C++ Developers
- Download the Eclipse Neon 3 software with the package “Eclipse IDE for C/C++ Developers” from <https://www.eclipse.org>.

Java Runtime Environment

- Oracle® Java™ Runtime Environment
- Download the Java Runtime Environment from <https://java.com/de>.

.NET Framework

- Microsoft® .NET Framework 4.6.1
- Download the .NET Framework from <https://www.microsoft.com>.



If you use PC Worx Engineer 7.x, the .Net Framework is already installed.

4.1.2 Installation

Installing the Java™ Runtime Environment

The Java™ Runtime Environment is required for the Eclipse® development environment.

- Install the Java™ Runtime Environment.
- Follow the instructions of the installation wizard.

Installing the .NET Framework

The .Net Framework 4.6.1 is required for the PLCnext Technology LibraryBuilder. If you use PC Worx Engineer, .Net Framework is already installed on your system.

- Install the .Net Framework.
- Follow the instructions of the installation wizard.

Installing and configuring Eclipse®

A development environment is required for programming in C++. Phoenix Contact recommends Eclipse® Neon 3. The Phoenix Contact add-in available for Eclipse® enables the easy use of C++ programs in the PLCnext Technology context.

- Execute the installation file for the Eclipse® Neon 3 software.
- In the dialog that opens, select the integrated development environment for C/C++ developers (Eclipse IDE for C/C++ Developers).

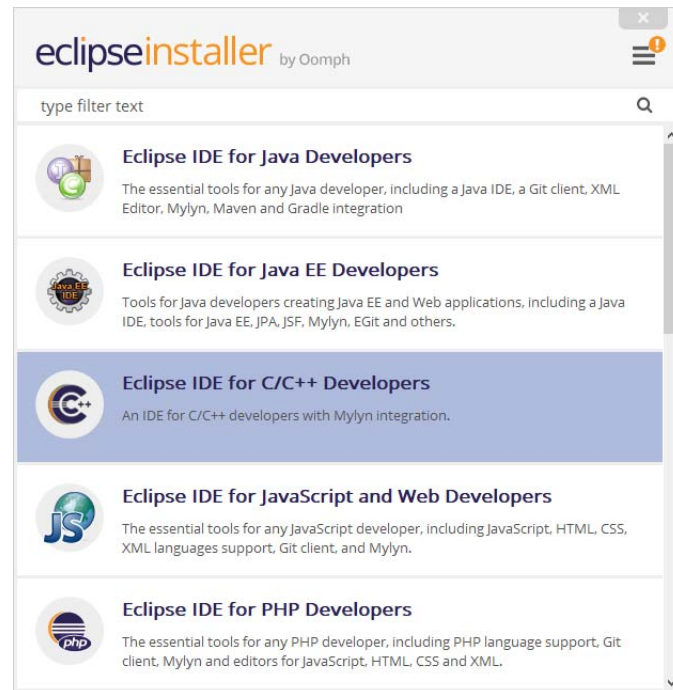


Figure 4-1 Selecting IDE for C/C++ Developers

- Then install the development environment.
- Follow the instructions of the installation wizard.

Installing the Phoenix Contact SDK

- Unzip the downloaded SDK file into a directory of your choice. Please note that the directory name may not contain spaces.
- After unzipping, ensure that the directory contains all files displayed in Figure 4-2.

<input type="checkbox"/> Name	Date modified	Type	Size
sysroots	11/16/2017 2:30 PM	File folder	
environment-setup-cortexa9t2hf-neo...	11/16/2017 2:17 PM	Windows Batch File	3 KB
pxc-glibc-x86_64-axcf2152-image-min...	11/23/2017 5:04 PM	tar Archive	1,124,283 KB
pxc-glibc-x86_64-axcf2152-image-min...	11/23/2017 5:04 PM	xz Archive	198,374 KB
relocate_sdk.py	11/23/2017 5:01 PM	PY File	9 KB
site-config-cortexa9t2hf-neon-pxc-lin...	11/16/2017 2:17 PM	File	55 KB
version-cortexa9t2hf-neon-pxc-linux-...	11/16/2017 2:17 PM	File	1 KB

Figure 4-2 Unzipped Phoenix Contact SDK directory

The SDK is accessed during the creation of a new C++ project (see 4.2 “Creating a new project”).

Updating the SDK**Recommended:**

Always use the latest version of the SDK. Uninstall earlier SDK versions. This is the only way to ensure that the various versions are not mixed.

Installing the Phoenix Contact add-in for Eclipse®

- Unzip the add-in.
- Open the Eclipse® software.
- In Eclipse®, open the menu “Help, Install New Software”.
- Select the path to the directory of the add-in. To do so, click on “Add” in the dialog that opens.
- In the dialog “Add Repository”, click on “Local” and select the directory of the add-in.

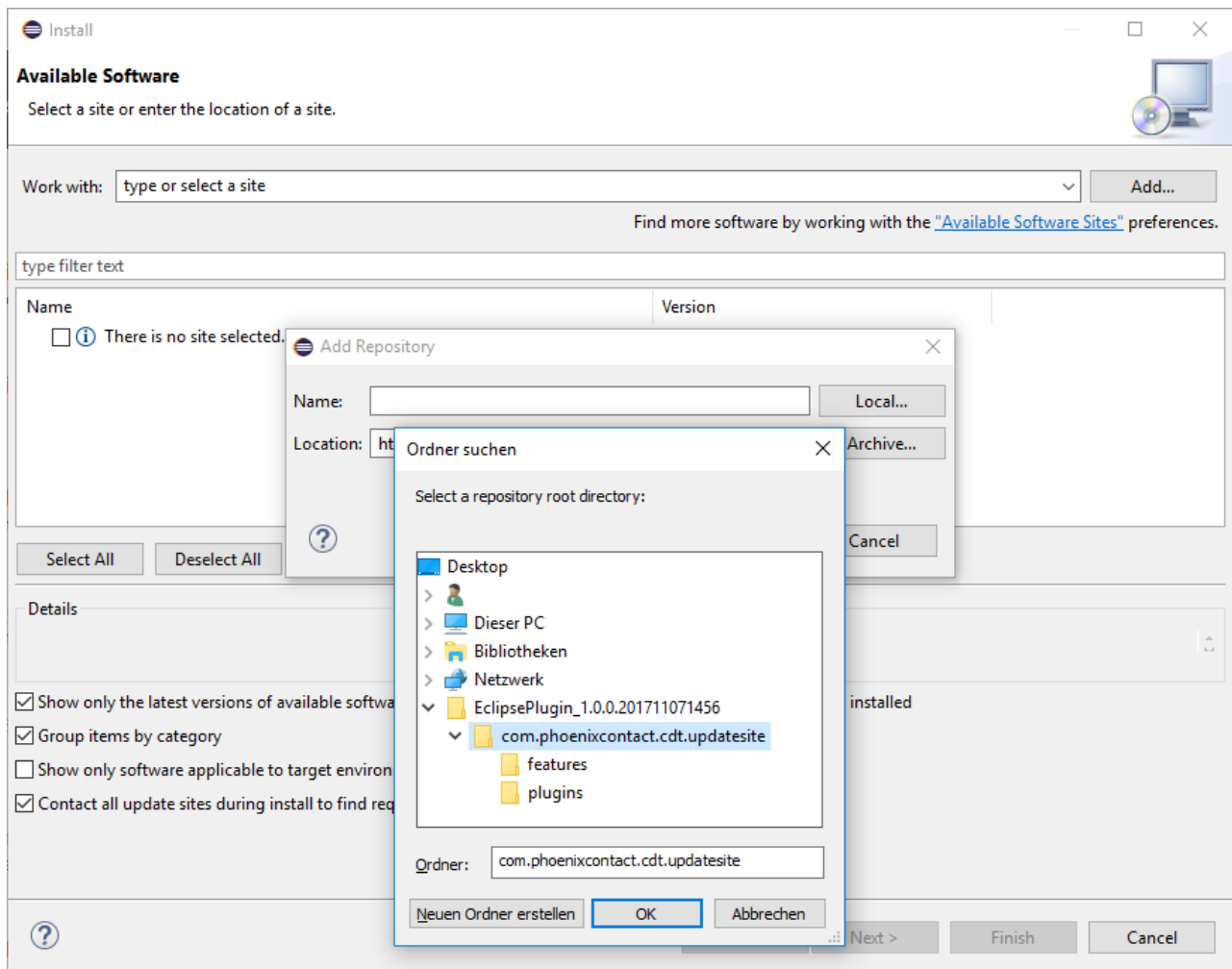


Figure 4-3 Selecting the path to the add-in directory

- In the next dialog, select the components displayed in Figure 4-4.

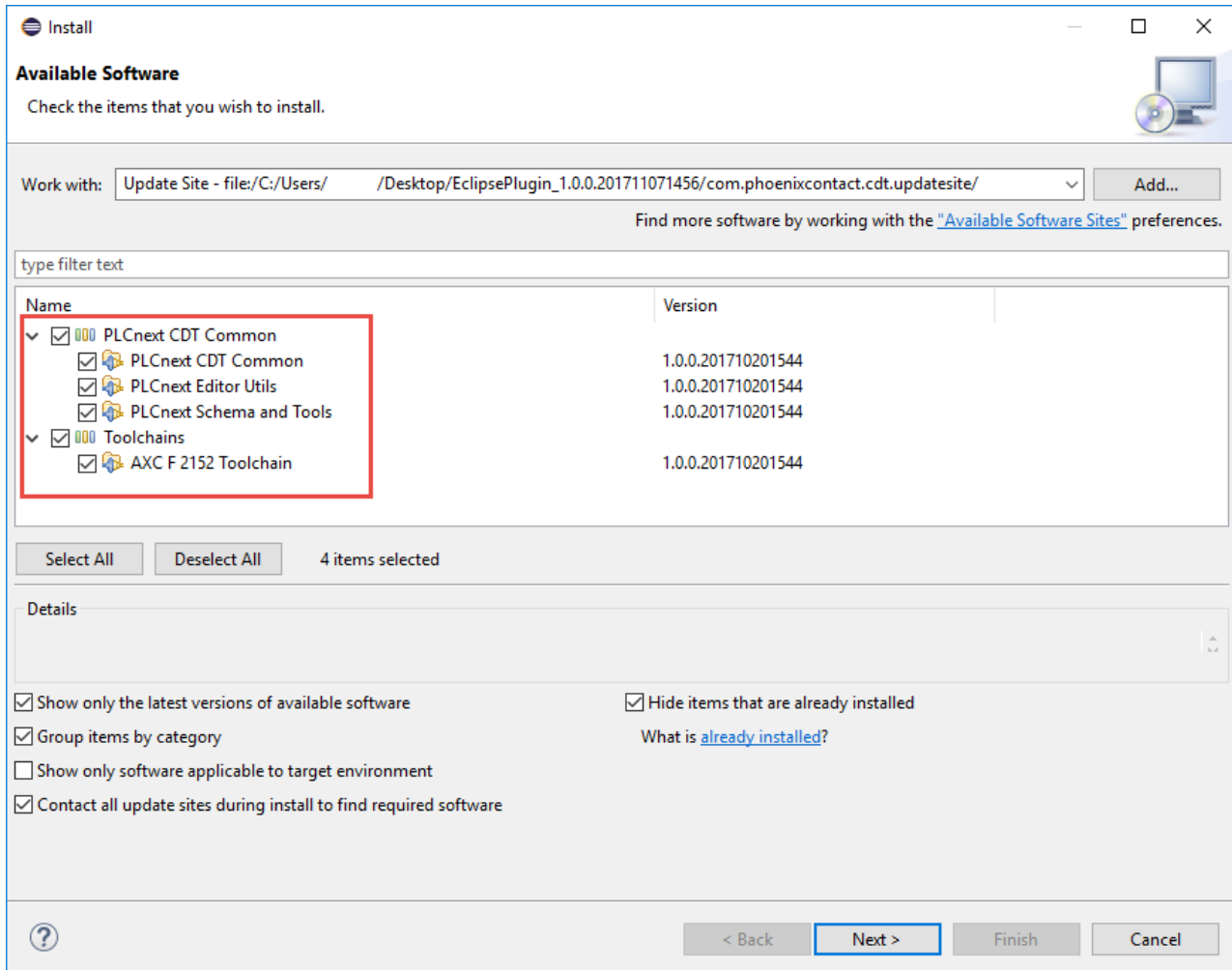


Figure 4-4 Selecting Phoenix Contact add-in components

- Click on the “Next” button.
- Read and accept the licence agreements.
- Click “Finish” to complete the installation.
- Restart Eclipse®.

Performing the add-in update for Eclipse®

To perform any necessary updates, proceed as follows:

- Update the Phoenix Contact add-in for Eclipse® by proceeding as for the initial installation.

You can also copy the update file to the existing directory. Start the update as follows:

- In Eclipse®, open the menu “Help, Check for Updates” or “Help, Install New Software”.

Installing the LibraryBuilder

- Unzip the downloaded LibraryBuilder file into a directory of your choice. Please note that the directory name may not contain spaces.

Now integrate the LibraryBuilder into Eclipse®. This is necessary to import C++ programs as libraries in C++ into PC Worx Engineer.

- In Eclipse®, open the menu “Window, Preferences”.
- Then select “PLCnext”.
- Click on “PCWLX Export”.
- Click on the “Browse” button.
- Select the path to the LibraryBuilder.
- Click on the “Apply” button to apply the path.

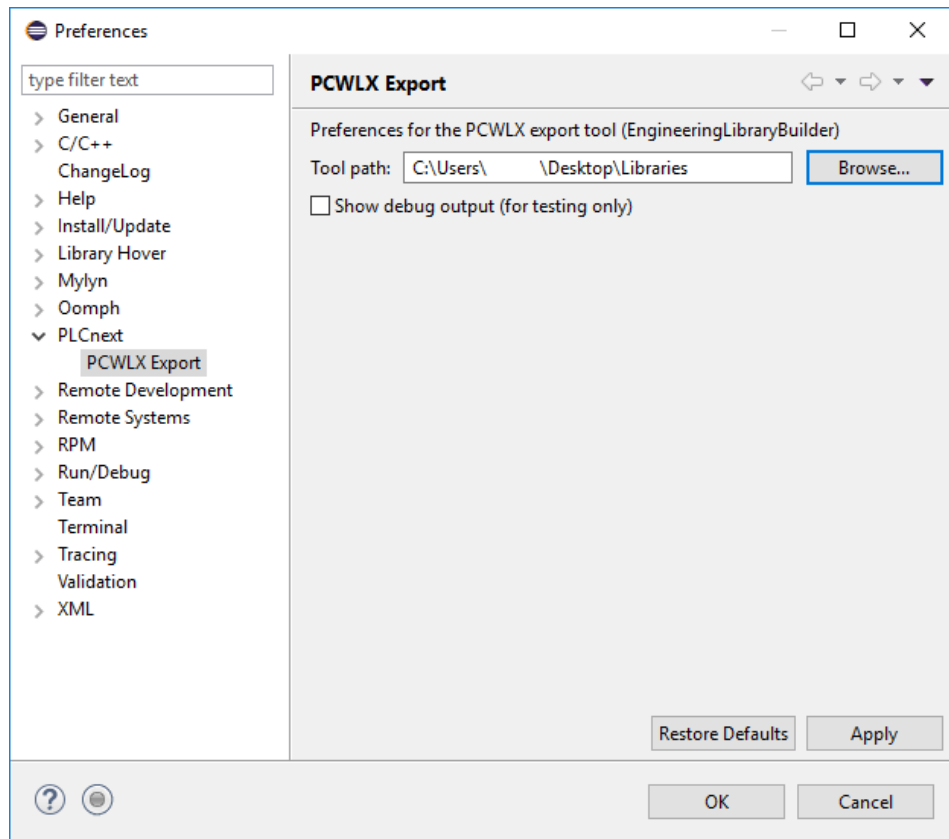


Figure 4-5 Selecting the LibraryBuilder

- Then click on “OK”.

4.2 Creating a new project

This section describes how to create a new C++ project in Eclipse®.

- Open the Eclipse® software.
- Open the menu “File, New, C++ Project”.

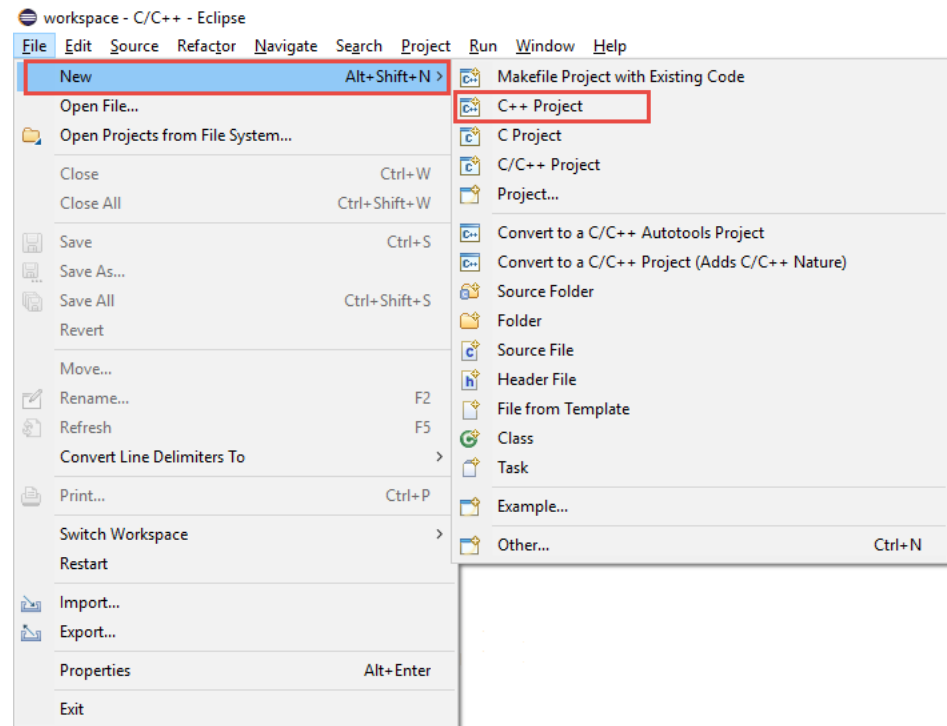


Figure 4-6 Creating a new C++ project

- Enter a project name in the “Project name:” input field.
- In the “Project type” field, select a project type (In the example in Figure 4-7 “Entering a project name and selecting a project type”).

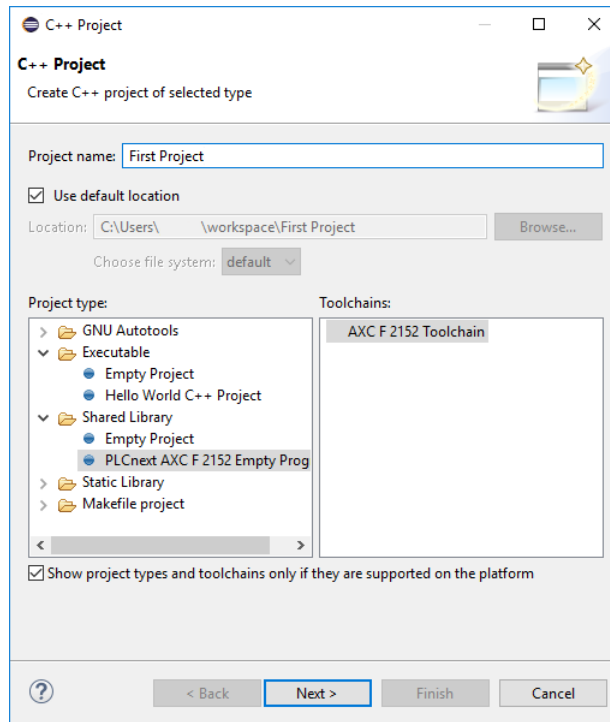


Figure 4-7 Entering a project name and selecting a project type

- Click on the “Next” button.
- If necessary, adapt the basic settings of the C++ project.

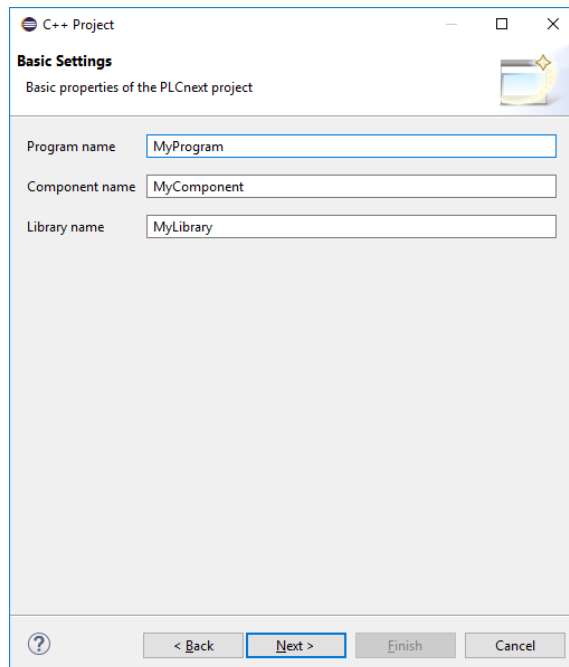


Figure 4-8 Basic settings of the C++ project

- Click on the “Next” button.
- In the “Configurations” area, activate the “Debug” and “Release” check boxes:

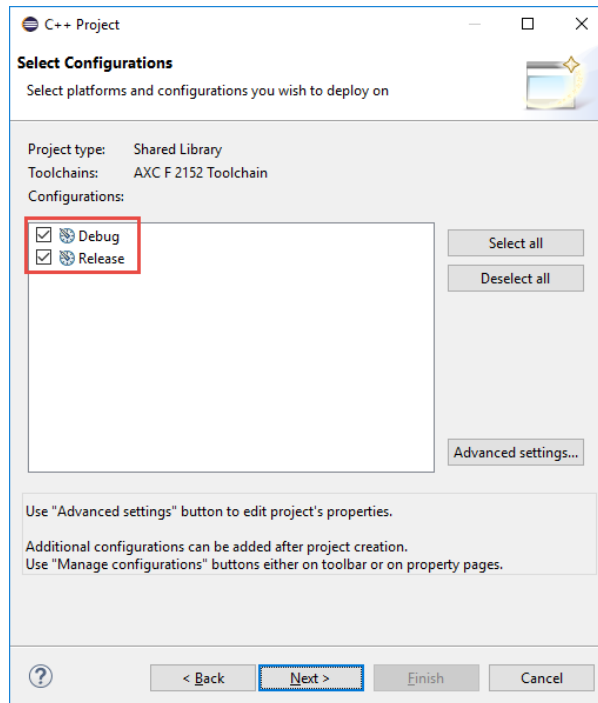


Figure 4-9 Activating the “Debug” and “Release” check boxes

- Click on the “Browse” button.
- Select the path to the directory “sysroots” of the unzipped Phoenix Contact SDK.
- Confirm your selection with “OK”.

- Click the “Finish” button to finish creating the project.

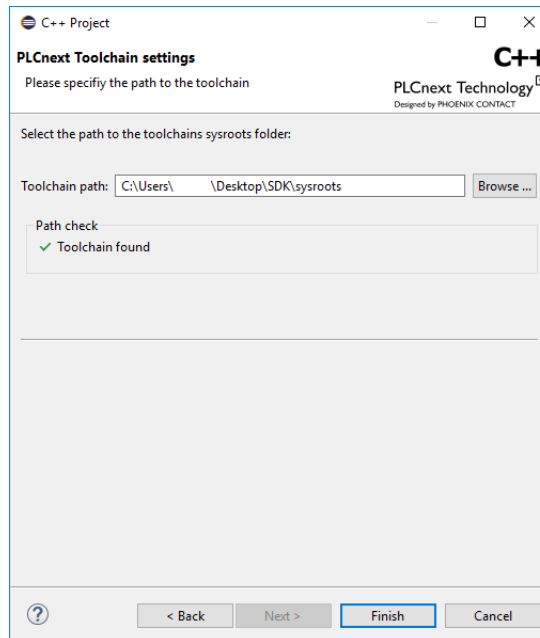


Figure 4-10 Selecting the path to the “sysroots” directory

- To start creating the C++ program, now switch to the “Workbench” view.

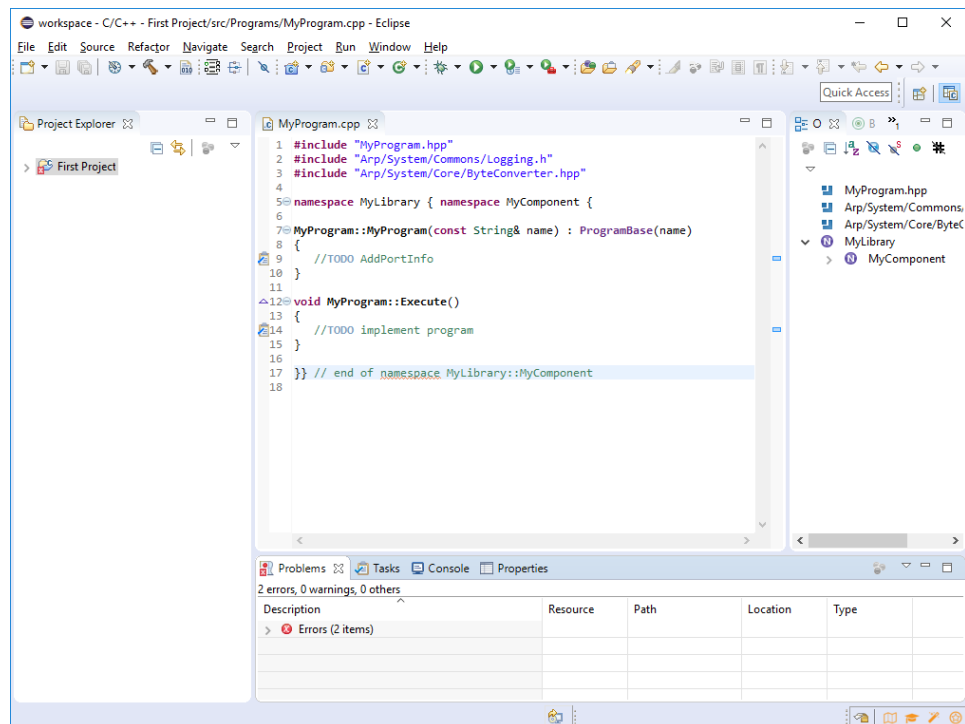


Figure 4-11 “Workbench” view in Eclipse®

4.3 Creating a program

To create a C++ program using Eclipse® which can be imported into PC Worx Engineer, you must first prepare a new project in accordance with the description in section “Creating a new project” on page 75.

Ensure that the correct path to the “sysroots” directory of the unzipped Phoenix Contact SDK is set.

The created project has a defined structure in which the C++ program is created. Figure 4-12 shows an example.

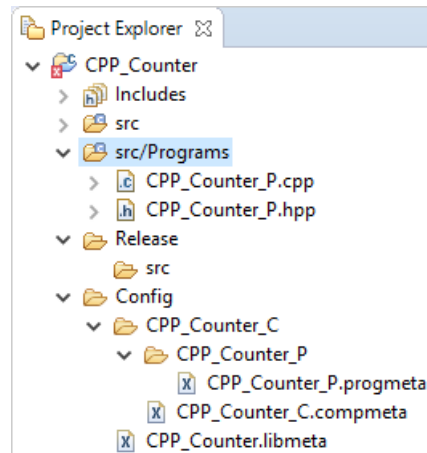


Figure 4-12 Sample structure of a C++ project

Creating variables

The header file of your program (e.g. CPP_Counter_p.hpp) must contain the variable definition.

- Define the variables required for the C++ program as well as their initial values here. Some of the variables are used as IN or OUT ports. You will find further information on the IN and OUT ports and GDS in section “GDS (Global Data Space)” on page 27.
- Create the variables necessary for your project.

```
public: // operations
    void Execute(void)override;

private: // fields
    //Inports
    boolean IP_CppEnable_bit = false;
    uint8 IP_CppSwitches_uint8 = 0;

    //Outports
    uint8 OP_CppCounter_uint8 = 0;

    //Locals
    uint16 local_messageState = 0;
    uint16 local_CounterControl = 0;
    int16 local_CppCounter8 = 0;
};

// inline methods of class ProgramBase

}} // end of namespace CPP_Counter::CPP_Counter_C
```

Figure 4-13 Sample variable declaration with initial values in the header file

- Save the changes to the header file.

Adding IN and OUT ports

Now register the IN and OUT ports for the GDS of the PLCnext Technology firmware. To do so, proceed as follows:

- Switch from the header file (*.hpp) to the source file (*.cpp).
- Right-click in the line behind the comment “//TODO AddPortInfo”.

```
#include "MyProgram.hpp"
#include "Arp/System/Commons/Logging.h"
#include "Arp/System/Core/ByteConverter.hpp"

namespace MyLibrary { namespace MyComponent {

MyProgram::MyProgram(const String& name) : ProgramBase(name)
{
    //TODO AddPortInfo
}

void MyProgram::Execute()
{
    //TODO implement program
}

}} // end of namespace MyLibrary::MyComponent
```

Figure 4-14 Inserting IN and OUT ports in the source file (*.cpp) (1)

- In the context menu that opens, select the entry “PLCnext, Insert Program Ports at position”.

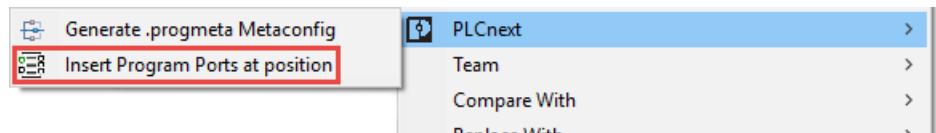


Figure 4-15 Inserting IN and OUT ports in the source file (*.cpp) (2)

The add-in now generates a list of all available variables from the header file. You can define all variables of the list as IN or OUT ports.

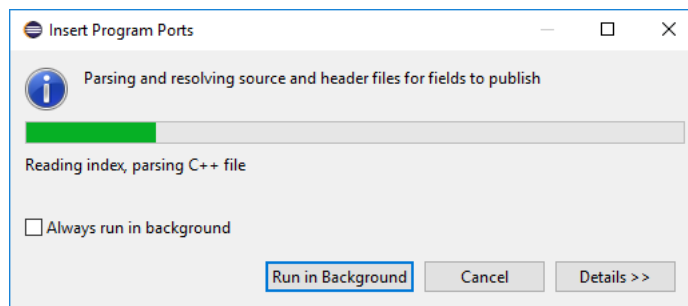


Figure 4-16 List of all available variables is being created

- From the list, select the variables that you want to use as IN or OUT ports.

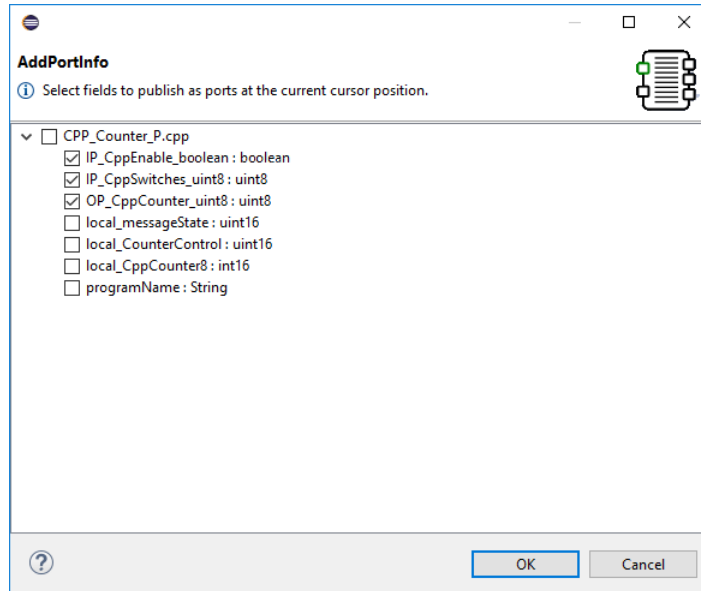


Figure 4-17 Example: List of all available variables

- Confirm your selection with “OK”.

The IN and OUT ports are created at the respective location in the source file with the required syntax. You can furnish these with individual comments if necessary (e.g. “//Inports”).

```
#include "CPP_Counter_P.hpp"
#include "Arp/System/Commons/Logging.h"

namespace CPP_Counter { namespace CPP_Counter_C {

CPP_Counter_P::CPP_Counter_P(const String& name) : ProgramBase(name)
{
    //Inports
    AddPortInfo("IP_CppEnable_boolean",    this->IP_CppEnable_boolean);
    AddPortInfo("IP_CppSwitches_uint8",    this->IP_CppSwitches_uint8);

    //Outports
    AddPortInfo("OP_CppCounter_uint8",     this->OP_CppCounter_uint8);
}

void CPP_Counter_P::Execute()
{
    //Take the first two Bytes of the Inputs to control the counter state.
    local_CounterControl = IP_CppSwitches_uint8 & (uint16) 3;

    if (true == this->IP_CppEnable_boolean)
    {
```

Figure 4-18 Sample IN and OUT ports in the source file *.cpp

The definition of IN and OUT ports consists of two parameters:

Generating the *.progmeta file

- **Parameter 1:** Name of the IN port or OUT port. The name is used to create the reference to the GDS configuration and to the *.progmeta file. The name must match in all three positions. However, the name can deviate from the variable name (see “Notes on designating GDS ports” on page 33).
- **Parameter 2:** Reference to the variable to be used as port. The data direction (IN or OUT port) is determined in the *.progmeta file.
- Save the changes to the source file.

If you defined the variables as IN and OUT ports in the source file, you have to add and configure the IN and OUT ports in the *.progmeta file. You can also perform this task with the add-in. Generate the IN and OUT ports in the *.progmeta file using the Phoenix Contact add-in for Eclipse®. The add-in draws the required data from the existing *.progmeta file and extends it with the found call ups of “AddPortInfo()” in the *.cpp source file. For newly added ports, the data direction has to be added (see Figure 4-20). The file is read in by both the firmware and PC Worx Engineer. Both need the data direction, name and data type.

To generate the file, proceed as follows:

- Right-click anywhere in the open editor of the *.cpp file (* = file name).
- In the context menu that opens, select the entry “PLCnext, Generate .progmeta Metaconfig”.

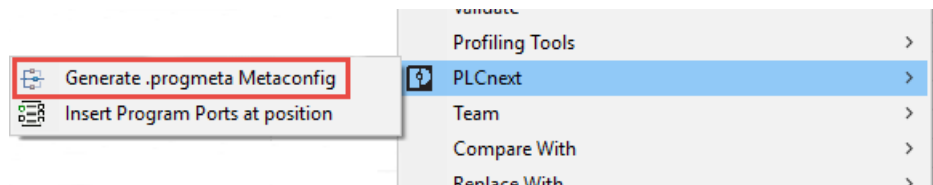


Figure 4-19 Generating the *.progmeta file

The *.progmeta file is being generated. Now define whether the generated ports are IN or OUT ports (INTERNAL ports are intended for a later extension).

- Open the newly generated *.progmeta file.
- Open the “Design” tab.
- In the “Node” column, navigate to the “Ports” entry.
- In the “Content” column, right-click to open the context menu in the “kind” line of a port.
- Choose the type of the desired port.
- Repeat this procedure for all IN and OUT ports used.

The add-in enters the data type in the “type” line.

- Inspect the data type in the “type” input field.

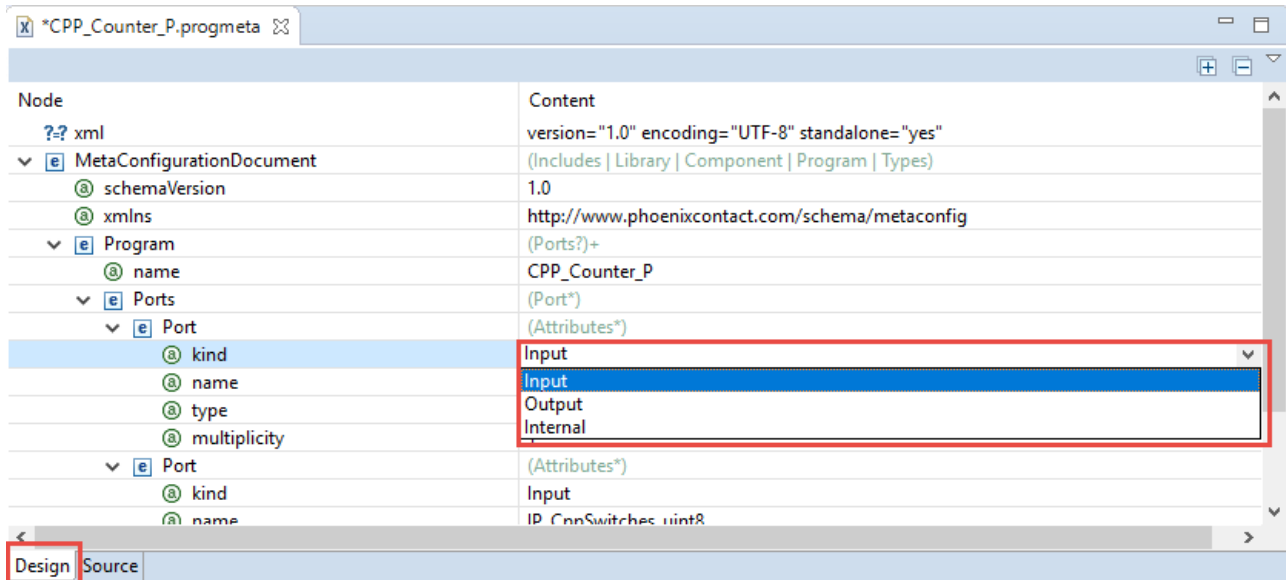


Figure 4-20 Example *.progmeta file in the “Design” tab

All settings made in the “Design” tab are applied to the source text of the *.progmeta file. You can also edit directly in the source text.

- Switch to the “Source” tab to view the settings in the source code of the *.progmeta file.

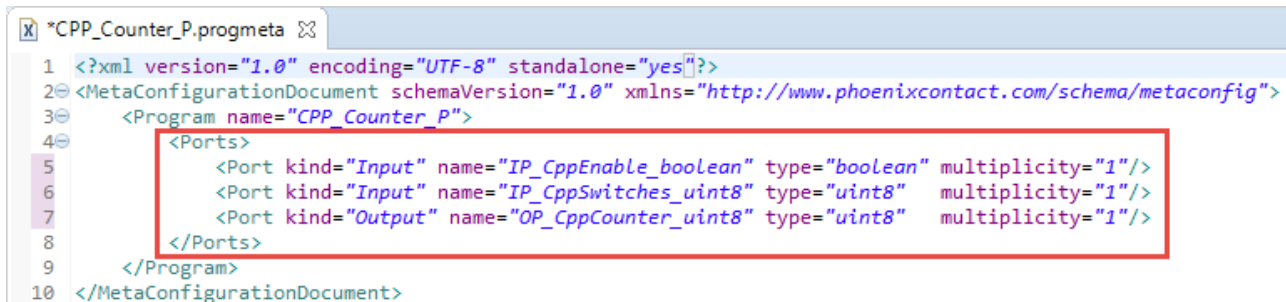


Figure 4-21 Sample settings in the source code of the *.progmeta file on the tab “Source”

Creating a C++ program

- Open the *.cpp source file.
- Program the code to be executed in each ESM task pass in the “Execute()” function (see “//TODO implement program”).

After the instantiation of a program, the assigned ESM task calls up the “Execute()” function of the program instance in each cycle.



Sample projects and the associated documentation are available for downloading on the AXC F 2152 controller page (order no.: 2404267): phoenixcontact.net/products.

Compiling a project

- Switch to the “Project Explorer” to compile the project.

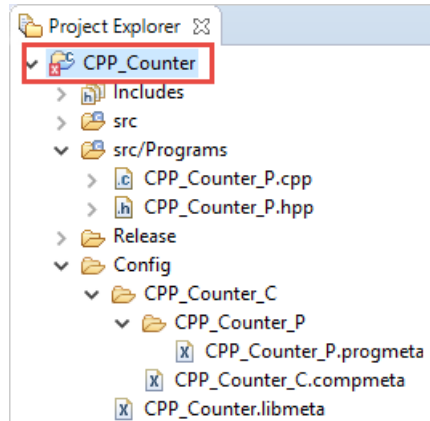


Figure 4-22 Project folder in the “Project Explorer”

- Right-click to open the context menu for the project folder (in the example: “CPP_Counter”).
- In the context menu, click on “Build Project”.

To view whether the compilation of the project was successful or to view possible error messages, switch to the “Console” tab.

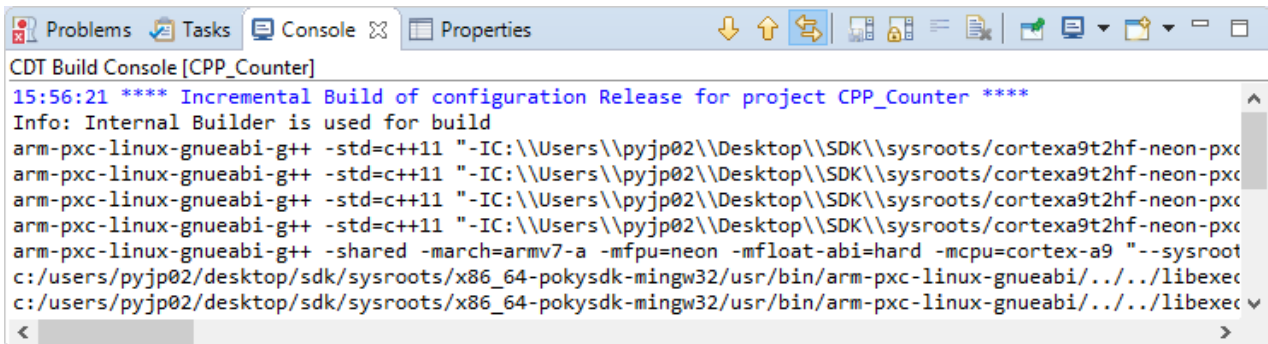


Figure 4-23 Example: “Console”

If the compilation was successful, a shared object (*.so) is generated. You will find this in the “Release” folder of the Project Explorer.

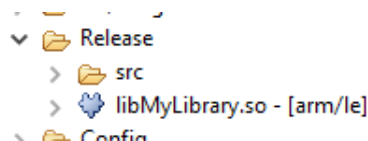


Figure 4-24 Example: Shared object (*.so) in the Project Explorer

Generating a PC Worx Engineer library

In order that you can use the C++ project in PC Worx Engineer, you must create a library from the C++ project. This library can then be imported into PC Worx Engineer. To generate a library from the C++ project, proceed as follows:

- Right-click to open the context menu for the project folder (in the example: “CPP_Counter”).

- Select “PLCnext, Export PCWLX”.

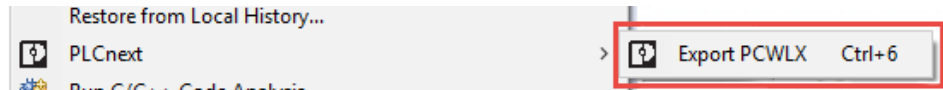


Figure 4-25 Generating a PC Worx Engineer library

- Click on the “Browse” button.
- Select the path to the directory to which the *.pcwlx library is to be stored.

Recommended: Standard path for storing PC Worx Engineer:
 C:\Users\Public\Documents\PC Worx Engineer\Libraries

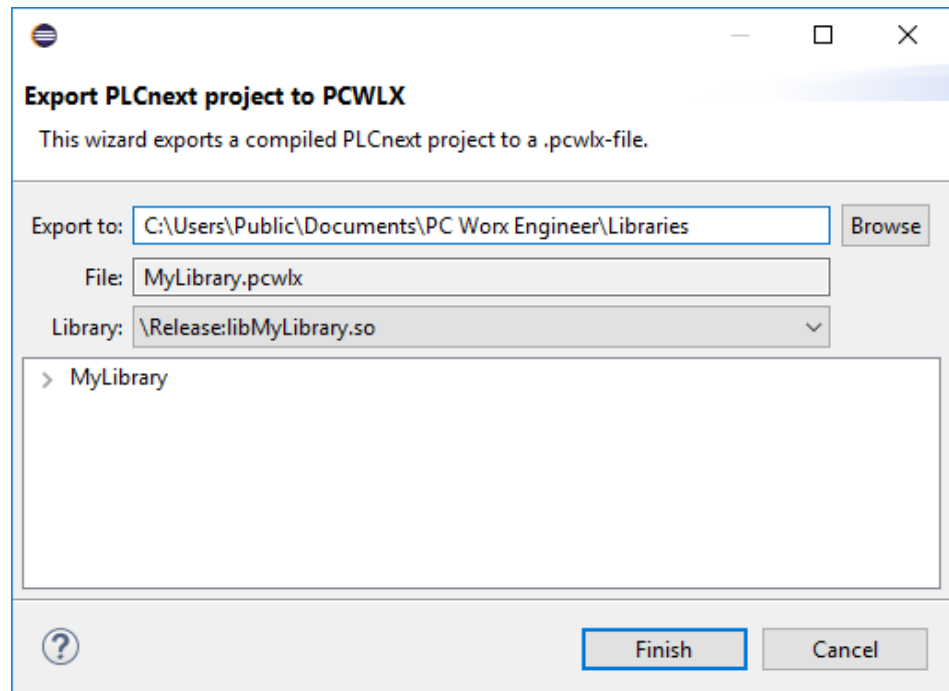


Figure 4-26 Selecting a memory path for the *.pcwlx library to be generated

- Click on the “Finish” button to generate the *.pcwlx library.

4.4 Importing generated libraries into PC Worx Engineer



You will find additional information on PC Worx Engineer in the online help and the quick start guide (PC WORX ENGINEER 7, order no.: 1046008).

Once you have generated a library from a C++ program, import this library into the PC Worx Engineer software. Imported libraries are handled and processed in tasks in the same way as IEC 61131-3 programs in PC Worx Engineer.

To import a library into PC Worx Engineer, proceed as follows:

- Open your PC Worx Engineer project or create a new AXC F 2152 template.
- In the “COMPONENTS” section, click on “References”.
- Right-click on “Libraries”.

Adding a library

- Right-click to open the “Add Library” context menu.

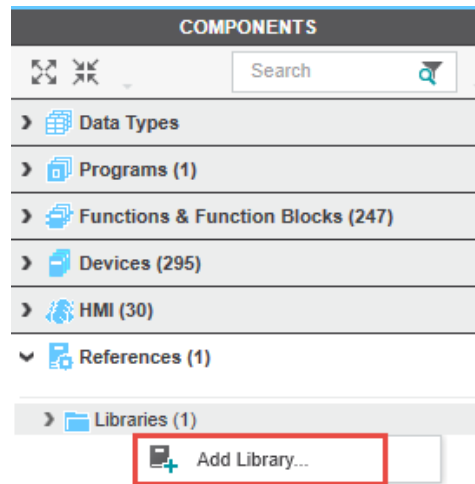


Figure 4-27 “Add library” context menu

- In the dialog that opens, select the desired *.pcwlx library.
- Click on “Open”.

If you have created the *.pcwlx library in Linux PC Worx Engineer converts this into the Windows® format. You have the option of creating a backup copy of the Linux version.

Components and programs are now available in the section “COMPONENTS” under “References”.

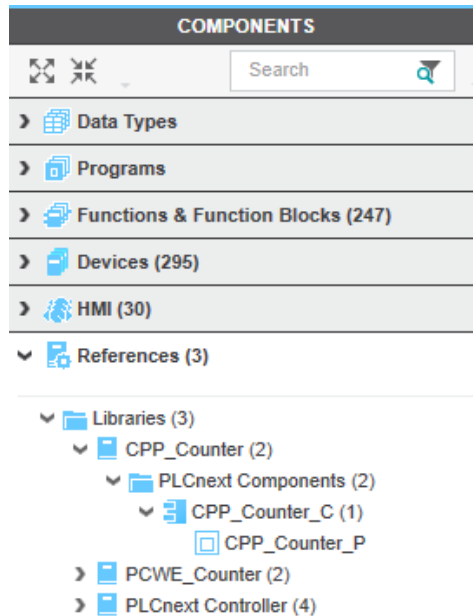


Figure 4-28 PC Worx Engineer – Added library

Instantiating a program

To instantiate the program contained in the library, proceed as follows:

- In the section “PLANT”, select the node “PLCnext”.
- Open the “Tasks and Events” editor.

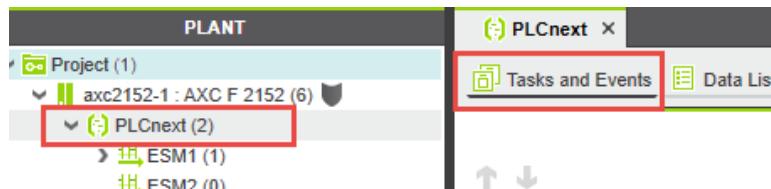


Figure 4-29 Opening the “Tasks and Events” editor

- In the editor, create a new task or open an existing task.

- Drag and drop the library from the section “COMPONENTS, References” to the “ES-MX” column of the task. This way you generate a program instance.

Name	Task Type	Event Name	Program Type	
▼ 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 4-30 Example: Program instance CPP_Counter_P1 in the “Task and Event” editor

Assigning IN/OUT ports

- In the section “PLANT”, select the node “PLCnext”.
- 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

- In order 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, duplicate it.
- Right-click on the OUT port and select the option “Duplicate Out port” in the context menu.



Figure 4-31 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

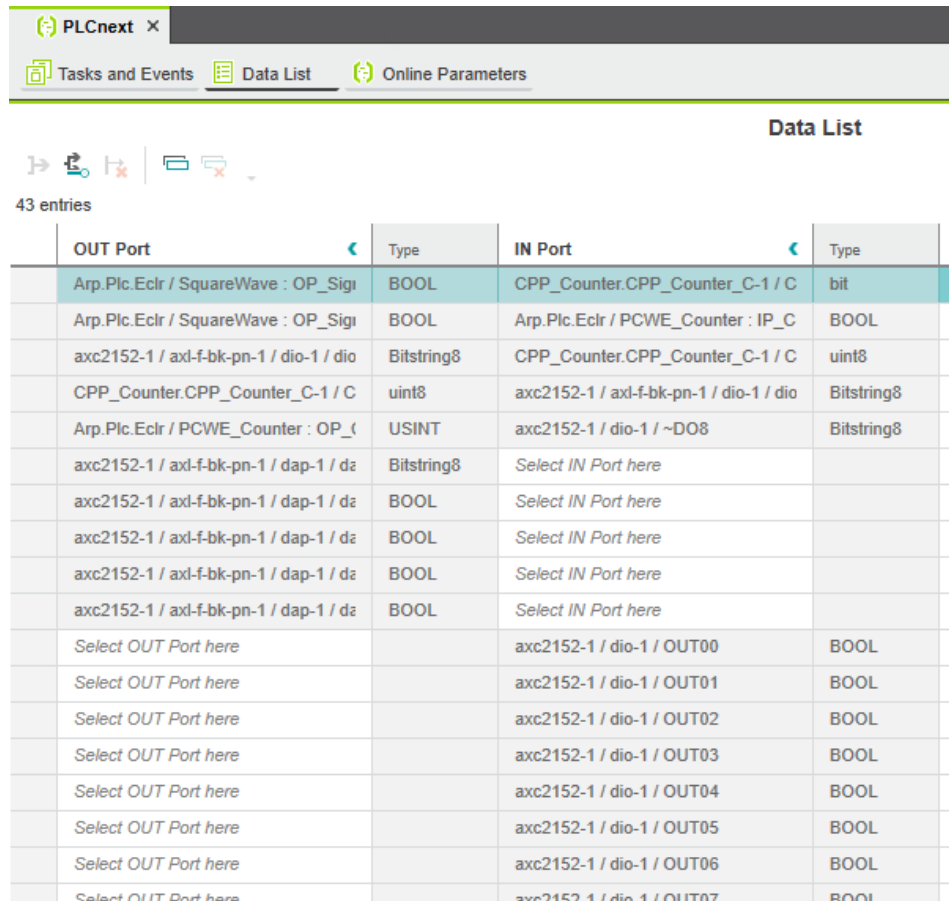
- In order to assign an OUT port to an IN port, click on “Select OUT Port here” in the “OUT Port” column.

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 2.6.2 “Supported data types”.



OUT Port	Type	IN Port	Type
Arp.Plc.Eclr / SquareWave : OP_Sigi	BOOL	CPP_Counter.CPP_Counter_C-1 / C	bit
Arp.Plc.Eclr / SquareWave : OP_Sigi	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 / d	Bitstring8	Select IN Port here	
axc2152-1 / axl-f-bk-pn-1 / dap-1 / d	BOOL	Select IN Port here	
axc2152-1 / axl-f-bk-pn-1 / dap-1 / d	BOOL	Select IN Port here	
axc2152-1 / axl-f-bk-pn-1 / dap-1 / d	BOOL	Select IN Port here	
axc2152-1 / axl-f-bk-pn-1 / dap-1 / d	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 4-32 “Data List” editor

Once you have assigned all IN and OUT ports to be used, transfer the PC Worx Engineer project to the controller. Additional information on PC Worx Engineer is available in the on-line help, the quick start guide (PC WORX ENGINEER 7, order no.: 1046008) and the user manual for the AXC F 2152 controller (order no.: 2404267).

4.5 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](#).

4.6 Sample programs

Several example programs created with Eclipse® and the Phoenix Contact add-in for Eclipse® are available in the download area of the AXC F 2152 controller under the address phoenixcontact.net/products (Order No.: 2404267). The examples illustrate aspects of programming with C++ within the framework of the PLCnext Technology that may be of help to you when creating your own programs.

5 Creating function blocks and functions with C#

The Visual Studio® extension is an extension for the integrated development environment Microsoft® Visual Studio®. The extension makes it possible to use Visual Studio® for the development of eCLR firmware libraries in C# and to implement these 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:

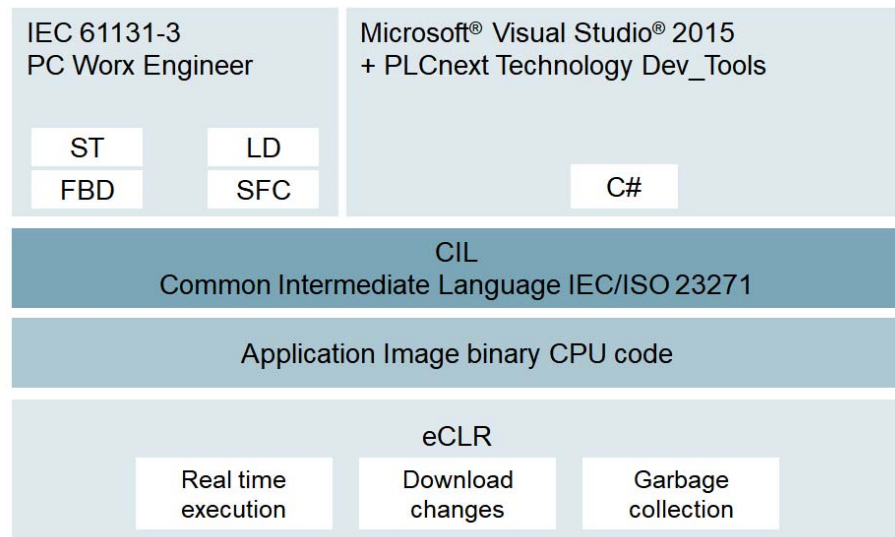


Figure 5-1 Structure of the programming system

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 to be found in every eCLR project in Visual Studio®.

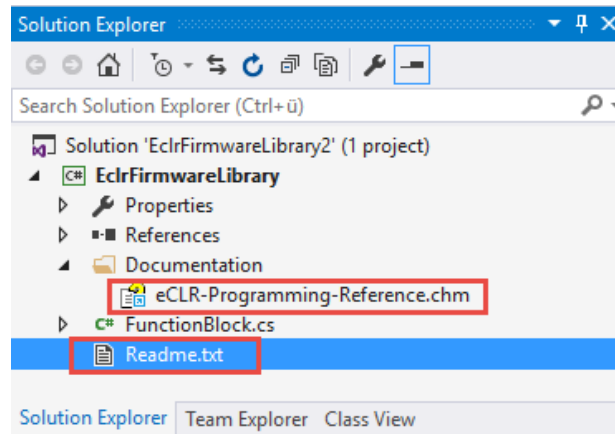


Figure 5-2 “eCLR Programming Reference” and “Readme.txt”

The eCLR programming system consists of the following components:

CIL compiler

The CIL compiler is responsible for translating the CIL code (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 target system (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 Chapter “C# language functions” on page 105.

eCLR core libraries

The core libraries consists of eCLR base class libraries (mscorlib.dll, System.dll, System.Core.dll) and some eCLR-specific libraries (eclrlib.dll, pcslib.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 91).

eCLR runtime

The eCLR runtime executes the compiled IL code on the target system. The runtime is responsible for the object and memory management, metadata processing (e.g. for debugging), as well as for the transition of managed API call ups to operating-system-specific native implementation. An overview of the eCLR runtime is available in section “eCLR runtime functions” on page 107.

5.1 Installing the Visual Studio® extension

5.1.1 Requirements

- Before starting the installation, ensure that the system requirements are met and download the necessary software.

Operating system

- Windows® 7
- Windows® 8.1
- Windows® 10

Phoenix Contact software

- PC Worx Engineer (at least Version 7.2.2, Order No.: 1046008):

You need the engineering software platform for automation controllers to use the libraries created with C# for PLCnext Technology devices.

- PLCnext Technology target with eCLR, at least Version 3.1 (e.g. AXC F 2152)
- Visual Studio® extension installer

The extension is to be found in the download area of the AXC F 2152 controller (Order No.: 2404267) or the PC Worx Engineer software (Order No.: 1046008) under the address phoenixcontact.net/products (PLCnext Technology Development Tools for Visual Studio).

C# development environment

- Microsoft® Visual Studio® 2015
Versions: Enterprise, Pro, Community

5.1.2 Installation

You need the Microsoft® Visual Studio® 2015 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 Visual Studio®

- Install the Microsoft® Visual Studio® 2015 development environment.
- Follow the instructions of the installation wizard.

Installing the Visual Studio® extension

- 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®.
- To ensure that the installation has been completed correctly, open the Visual Studio® development environment.
- Open the “Tools, Extensions and Updates” menu.

If the extensions have been successfully installed, they are displayed as shown in Figure 5-3:

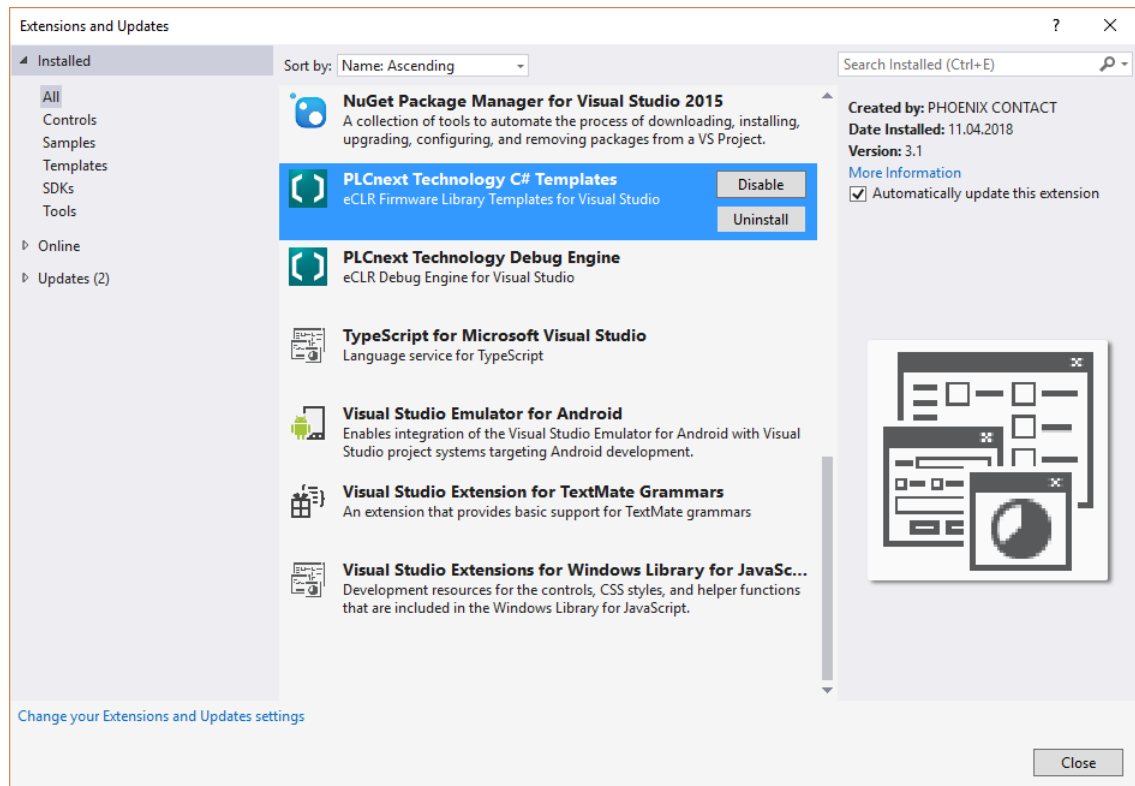


Figure 5-3 “Extensions and Updates” menu

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® menu “Programs and Features” to uninstall the Visual Studio® extension.

- Select the extension from the list.
- Click on “Uninstall”.

5.2 Creating a firmware library

With Visual Studio® you can create functions and function blocks in C# that you can subsequently import using the PC Worx Engineer software and which you can use on a PLCnext Technology device. You first have to create a new project. Proceed as follows:

- Open the “File, New, Project...” menu in Visual Studio®.
- In the dialog that opens, select “Installed, Templates, Visual C#, eCLR”.
- Select the eCLR template “Firmware Library” from the list.
- Select a storage path and click on “OK”.

This creates a new, empty eCLR library project.

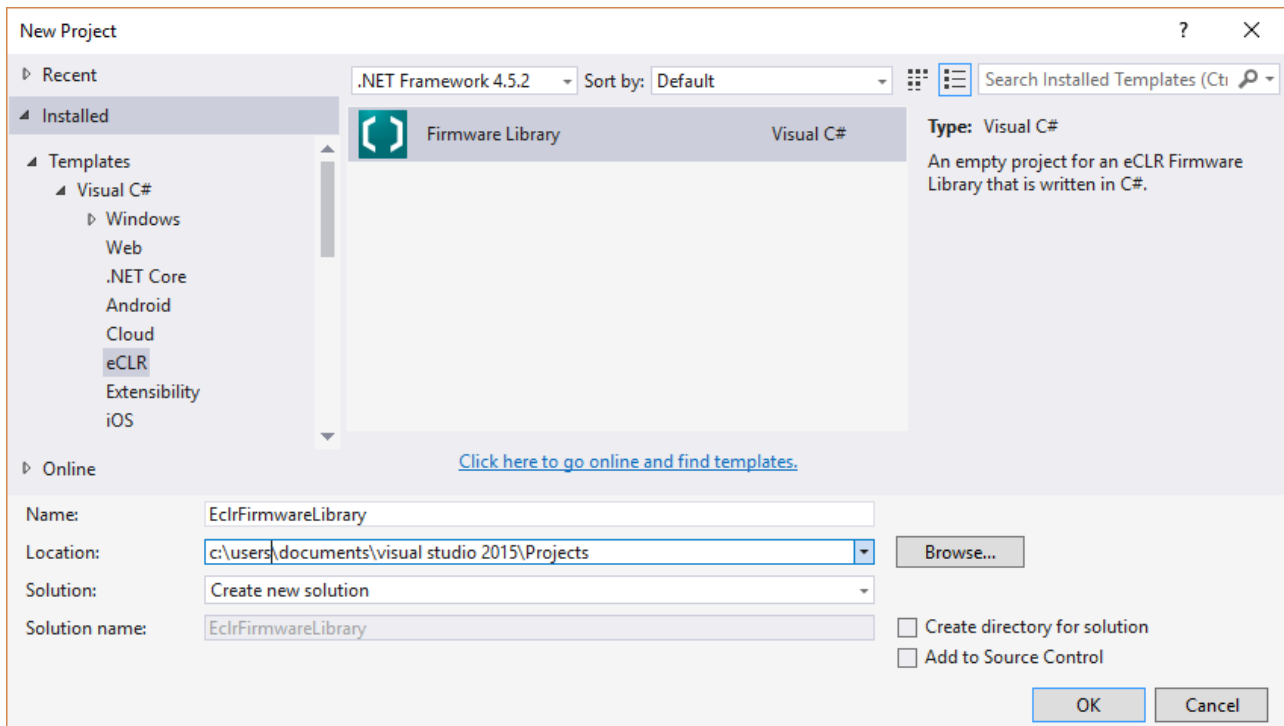


Figure 5-4 Creating a new, empty eCLR library project

- 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 an eCLR function or a function block.

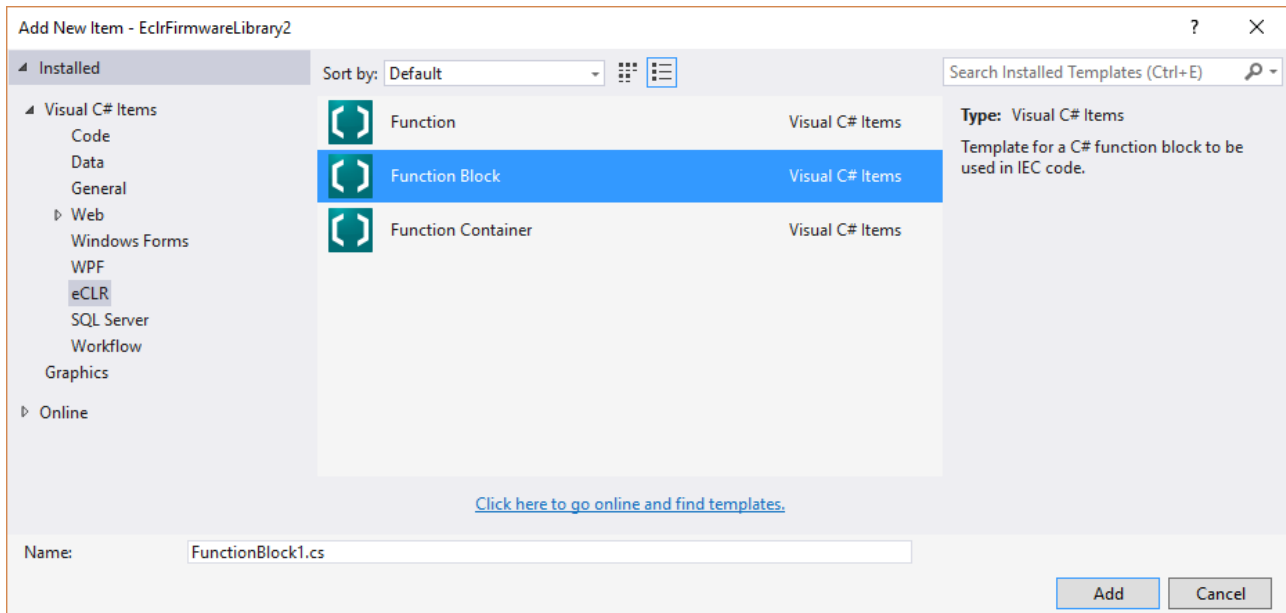


Figure 5-5 Adding a new “Function Block” element

- When you create a function, you select a return value in the following dialog.

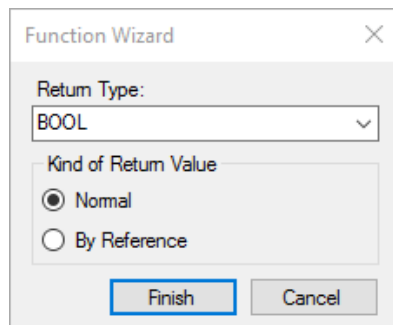


Figure 5-6 Adding a new “Function” element: Selecting a return value

“Kind of Return Value”:

“By Reference” is recommended for complex data types, and must be used 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.

```

1  Copyright
8
9  using System;
10 using System.Iec61131Lib;
11 using Iec61131.Engineering.Prototypes.Types;
12 using Iec61131.Engineering.Prototypes.Variables;
13 using Iec61131.Engineering.Prototypes.Methods;
14 using Iec61131.Engineering.Prototypes.Common;
15
16 namespace EclrFirmwareLibrary2
17 {
18     [FunctionBlock]
19     public class FunctionBlock1
20     {
21         [Input]
22         public short In1;
23         [Input]
24         public short In2;
25         [Output, DataType("WORD")]
26         public ushort Out;
27
28         [Initialization]
29         public void __Init()
30         {
31             //
32             // TODO: Initialize the variables of the function block here
33             //
34         }
35
36         [Execution]
37         public void __Process()
38         {
39             Out = (ushort)(In1 + In2);
40         }
41     }
42 }
43

```

Figure 5-7 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 91).

5.3 Importing the C# library in PC Worx Engineer



Additional information on PC Worx Engineer is available in the online help and the quick start guide. This is available for downloading at phoenixcontact.net/products (PC WORX ENGINEER 7, Order No.: 1046008).

Once you have generated a *.pcwlx library from a C# library, import this library into the PC Worx Engineer software. In PC Worx Engineer, imported libraries are treated as functions or function blocks in accordance with IEC 61131-3, and are processed in tasks.

To import a library into PC Worx Engineer, proceed as follows:

- Open your PC Worx Engineer project or create a new AXC F 2152 template.
- Save the project by selecting the “File, Save Project As...” menu.

- Select a path.
- In the “COMPONENTS” section, click on “References”.
- Right-click on “Libraries”.

Adding a library

- Right-click to open the “Add Library” context menu.

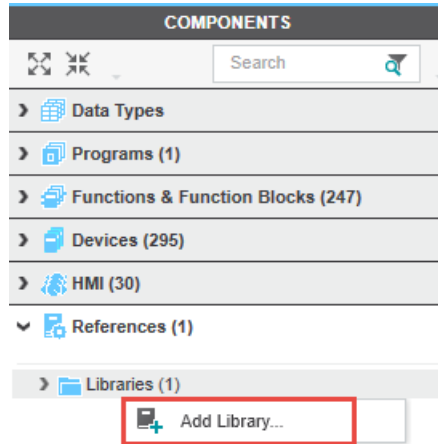


Figure 5-8 “Add library” context menu

- In the dialog that opens, select the desired *.pcwlx library. You will find this in the folder of your C# firmware project, in the subfolder bin/Debug or bin/Release (depending on the setting in Visual Studio®).
- Click on “Open”.

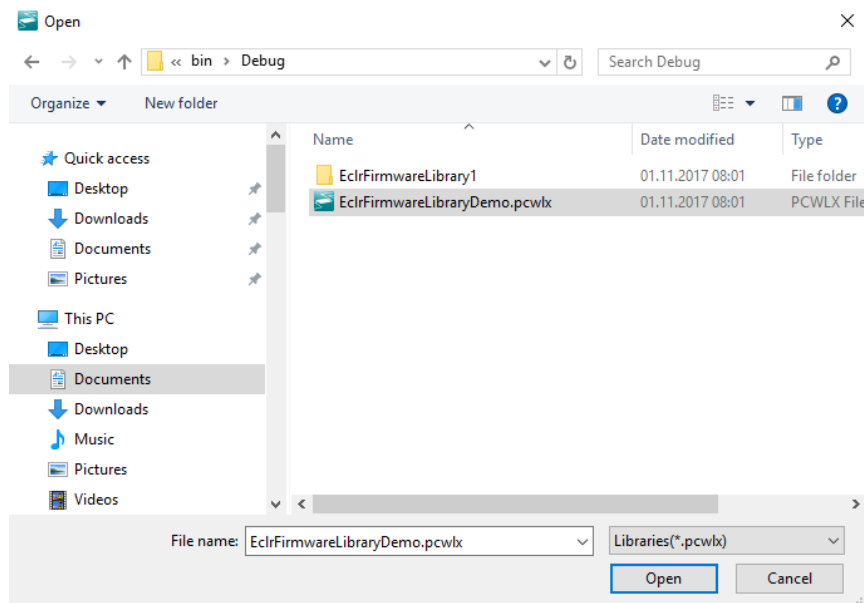


Figure 5-9 Example: Opening a pcwlx library

The library is now a part of the PC Worx Engineer project. The POU's contained therein (Program Organization Unit) 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 “References” to your code worksheet.

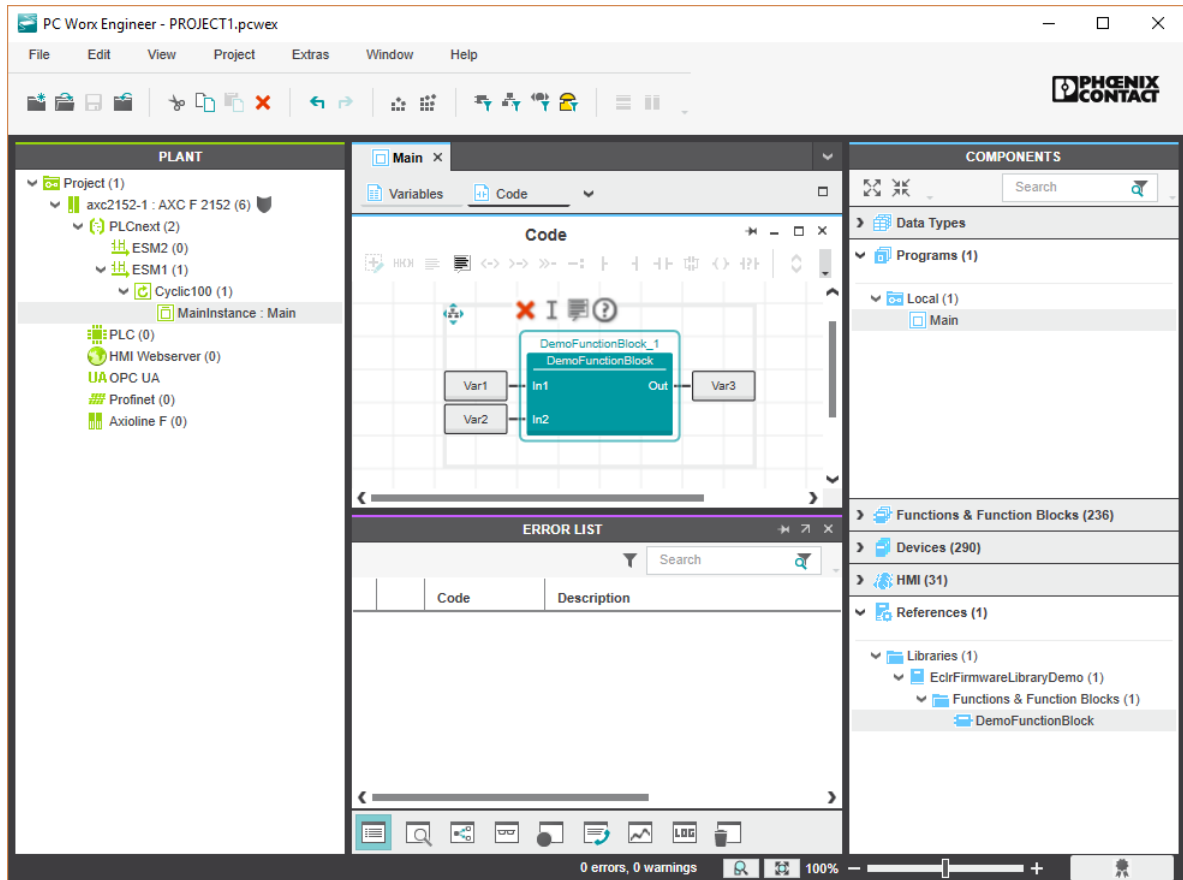


Figure 5-10 Inserting a function block

Assigning inputs/outputs

Additional information is to be found in the Readme.txt file (see “Additional information” on page 91).

5.4 Root rights

PLCnext Technology controllers are supplied with a preset admin user. This enables access to the most important functions. To configure settings for remote debugging of C# code, you need advanced rights. To this end, first create a user with root rights.



NOTE: Risk of damage to equipment

If safety functions are switched off, using the controller for live operation is not permitted. Ensure that there is no risk of equipment damage or personal injury.

Software used

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

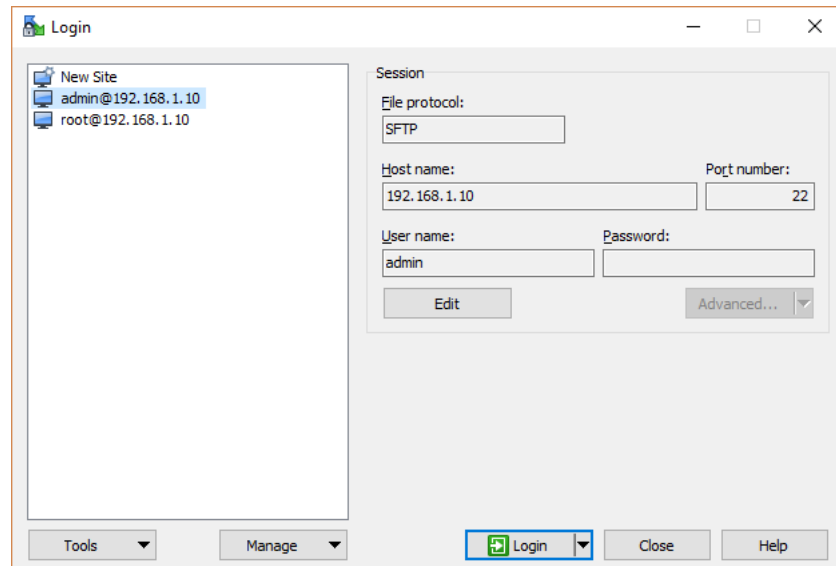


Figure 5-11 Connecting WinSCP with the controller (1)

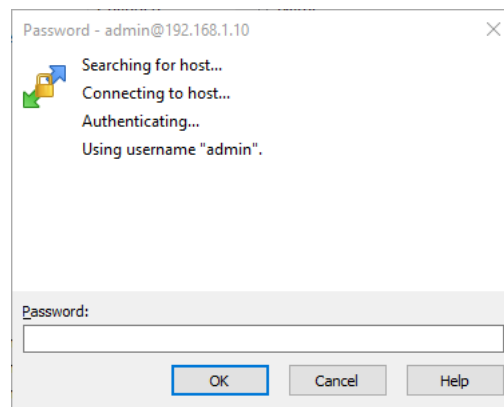


Figure 5-12 Connecting WinSCP with the controller (2)

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

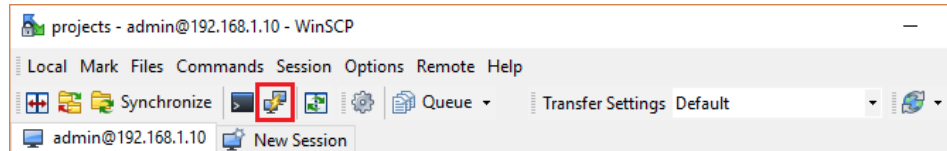


Figure 5-13 Opening PuTTY

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



Figure 5-14 Logging in as admin user

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

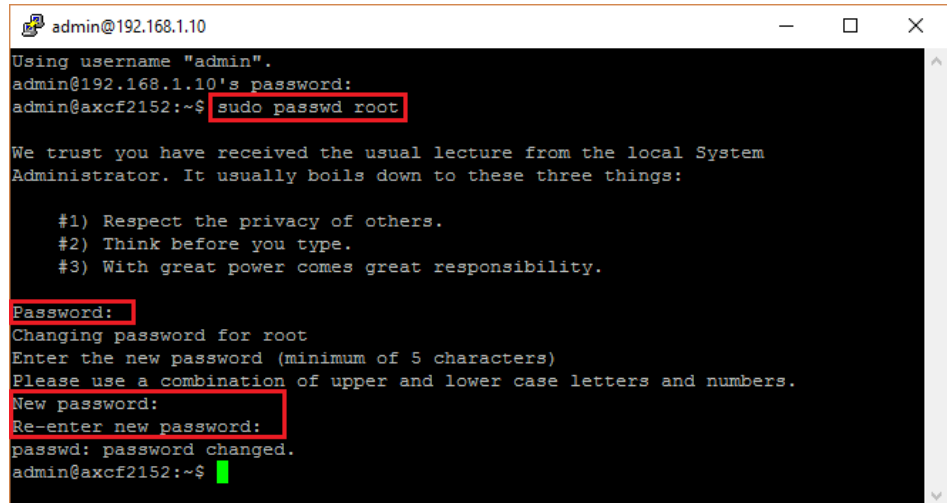


Figure 5-15 Creating a new root user

- To end the connection, enter the “exit” command.

5.5 Remote debugging of C# code with Visual Studio®

Because PLCnext Technology includes an implementation of eCLR, you have the option of establishing a C# remote session. The following sections describe the basic steps needed for setting up a remote debug session and establishing a direct connection with a running application by means of Visual Studio®.

Software used

- PuTTY
- WinSCP
- Microsoft® Visual Studio® 2015
- Visual Studio® extension



Please note:

- The following description is based on the default settings. Take all changes made by the user into account.
- Do not use the controller in live operation when safety functions have been disabled.

Using the debugging functionality can result in an unsafe process interruption. Ensure that there is no risk of equipment damage or personal injury.

5.5.1 Opening a port and deactivating TLS

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. Transport Layer Security (TLS) has to be deactivated on this port.

- Open WinSCP.
- Establish a connection using the IP address of the controller and the user with root rights.

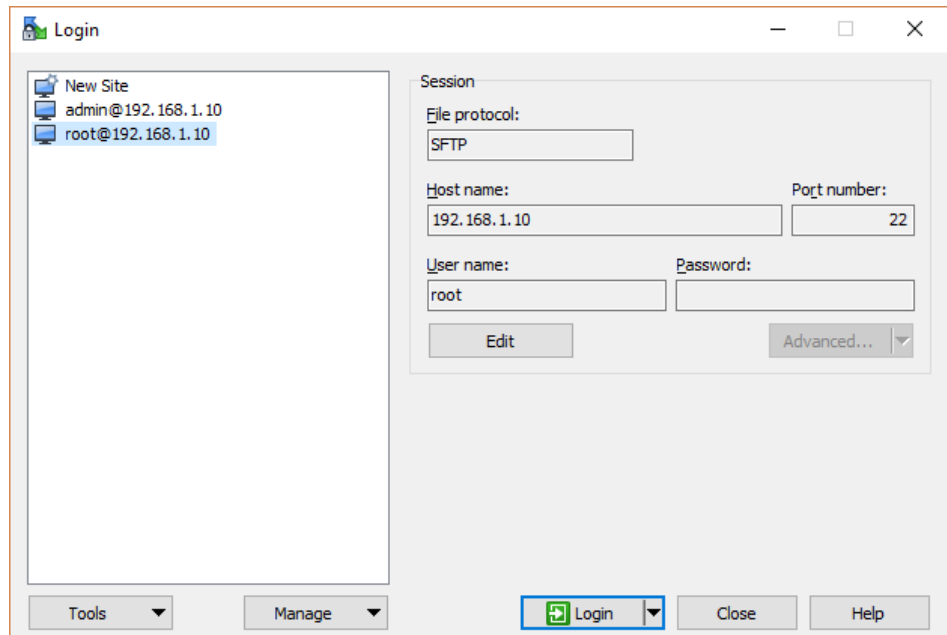


Figure 5-16 Establishing a connection

- Open the “/etc/plcnext/device/System/RscGateway/” folder.

- Open the “RscGateway.settings” file.

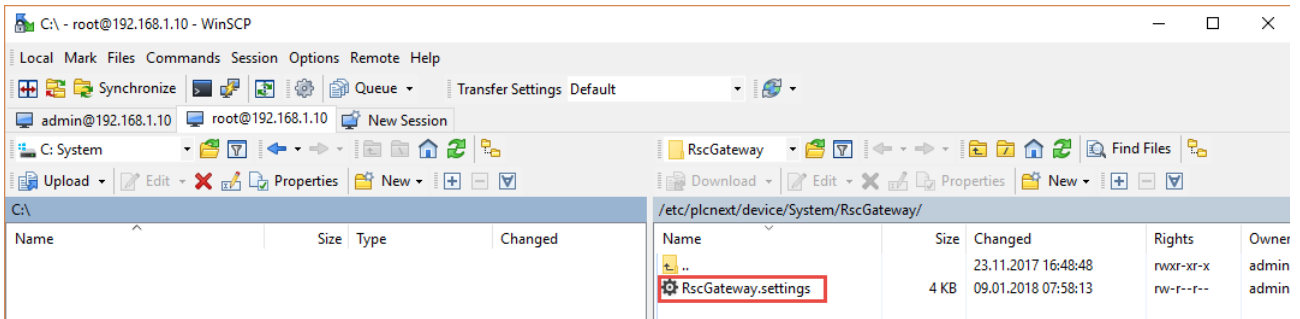


Figure 5-17 “RscGateway” directory with “RscGateway.settings” file

- Insert the line `<TcpGatewaySettings gatewayId="0" tcpPort="41101" sessionTimeout="300000" encrypted="false" />` as shown in the figure below.

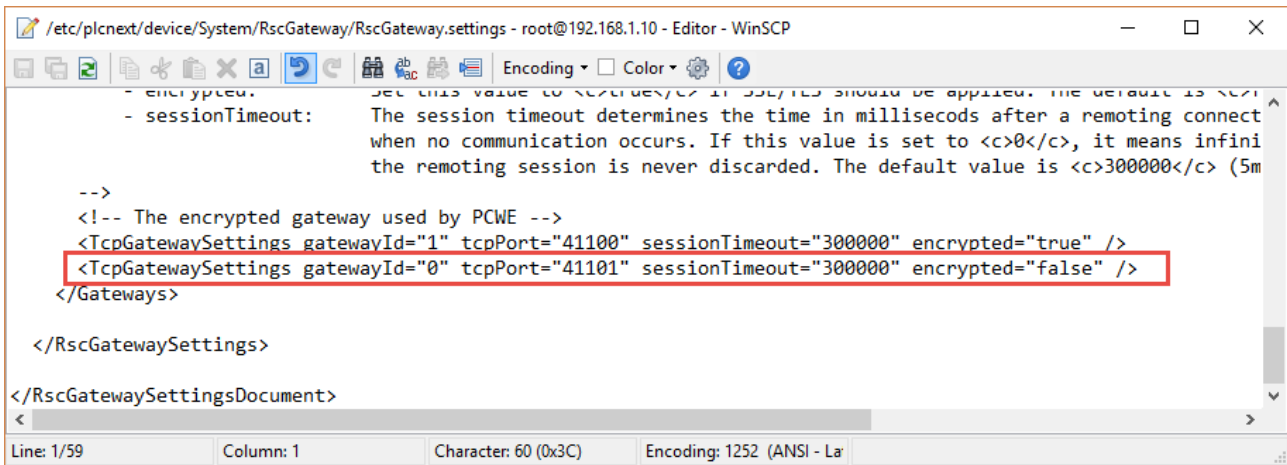


Figure 5-18 “RscGateway.settings”

- Save the file.
- Restart the controller.

5.5.2 Disabling user authentication



NOTE: Risk of damage to equipment

If safety functions are switched off, using the controller for live operation is not permitted.

The C# debugging function does not currently have an interface for logging on to the controller. To debug the C# code, you have to deactivate the user authentication in the WBM of the controller.

For information on the WBM (Web-Based Management) of your controller, please refer to the corresponding user manual.

- Deactivate the user authentication of your controller via the WBM.

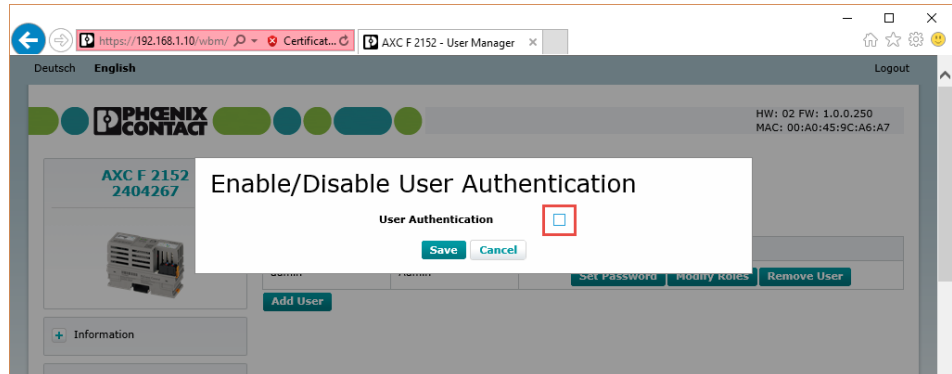


Figure 5-19 Example WBM AXCF 2152: Deactivating the user authentication

5.5.3 Debug mode

Creating/opening a project

- Create a new project in Visual Studio®. Set this up.
- Follow the operating instructions in section “Creating a firmware library” on page 94, or
- Open an existing project.

Importing a library into PC Worx Engineer

- Import the project into PC Worx Engineer.
- Follow the operating instructions in section “Importing the C# library in PC Worx Engineer” on page 96.

Activating debug mode

- Connect PC Worx Engineer with the controller. Additional information on PC Worx Engineer is available in the online help and in the quick start guide. This is available for downloading at phoenixcontact.net/products.
- Activate the debug mode by right-clicking on the controller in the “PLANT” area.
- In the context menu, select “Debug On/Off”.

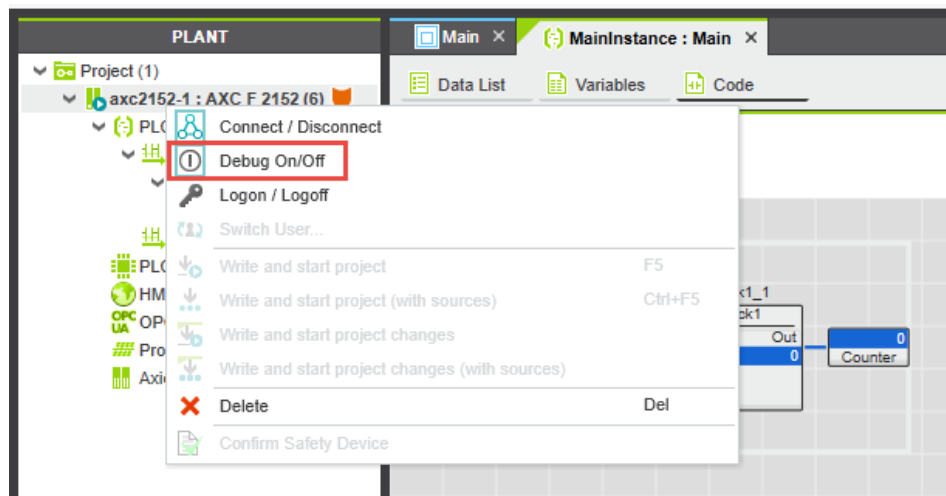


Figure 5-20 Activating debug mode in PC Worx Engineer.

Attaching Visual Studio® to the process

- In Visual Studio® open the “Debug, Attach to Process...” menu.
- In the “Transport” drop-down list, select the “eCLR Device” entry.
- Click on the “Find...” button.
- In the dialog that opens, enter the IP address of the controller and the port number 41101 (e.g. 192.168.1.10:41101).
- Select the image under the following path: “C:\Users\Public\Documents\PC Worx Engineer\Binaries\PROJECT1@binary\CLR\”. A shortcut to this path is available in the “Select Image File” dialog. This is available under “Microsoft Visual Studio 2015”, “Binaries”.

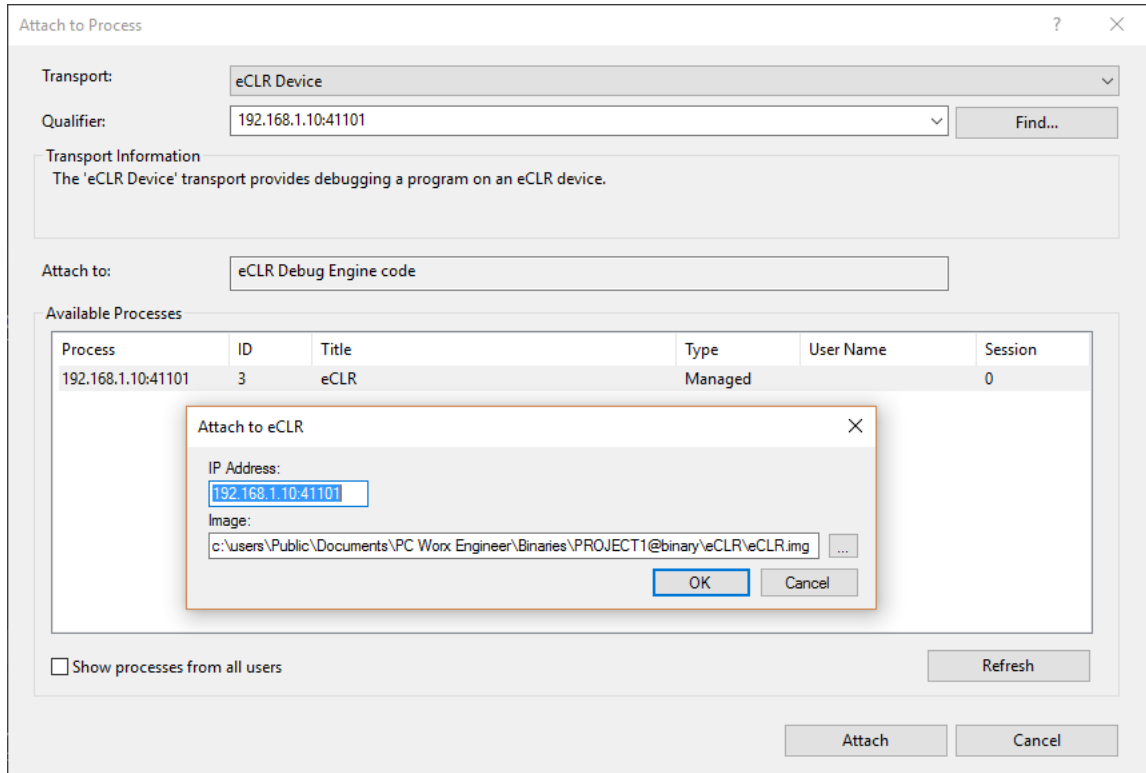


Figure 5-21 Attaching Visual Studio® to the process

- Confirm your selection with “OK”.
- Click on the “Attach” button.

Activating breakpoints

- If you want to use breakpoint functions in Visual Studio®, you first have to activate all breakpoints in the controller cockpit in PC Worx Engineer. Subsequently, you can use breakpoints in both programs.

- Click the “Enable/disable all BP” button to activate breakpoints.

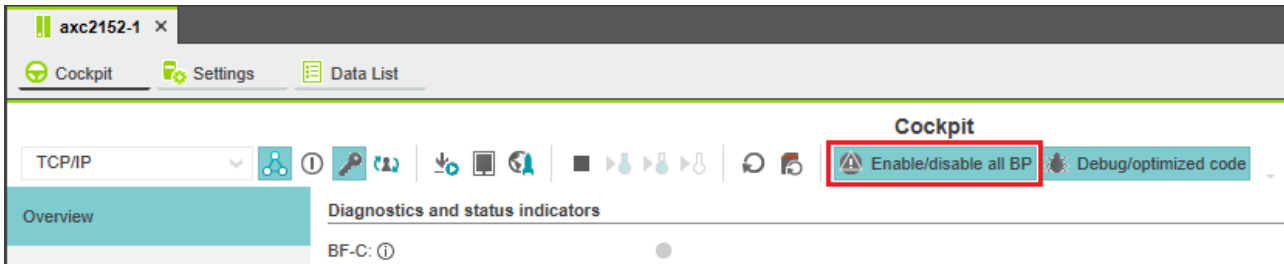


Figure 5-22 Activating all breakpoints in PC Worx Engineer

For information on the debug functions in PC Worx Engineer and in Microsoft Visual Studio®, please refer to the associated documentation.

5.6 Supported functions of the eCLR programming systems

5.6.1 C# language functions

Types

All 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 further information, see section “Known issues and constraints” on page 108.

5.6.2 Base class libraries

The following table contains 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 the section “eCLR Base Class Libraries Reference” of the “eCLR Programming Reference” online help (see “Additional information” on page 91).

System

All integrated system types, as well as the following:

- Char, Boolean, Int32, Single, etc., with the exception of decimal
- All standard 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, Time-Span
- 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

- IPAddress, IPEndPoint, Endpoint, SocketAddress
- WebClient, HttpWebRequest, HttpWebResponse

System.Net.Sockets

- NetworkStream
- AddressFamily, ProtocolType
- Socket, SocketType

5.6.3 eCLR runtime functions

Garbage Collection

The memory management is performed via eCLR. The Garbage Collector should explicitly be called up at a specific point in time (application of GC.Collect).

Implicit Finalization

The Finalizer of each applicable .NET class (implementing ~T()) is called up when the instance is released by the Garbage Collector. The Garbage Collector collect function is called up automatically. The time of the implicit call up cannot be predicted.

Implicit initialization

All 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

Detailed information is to be found in the identically named online help section of the Visual Studio® extension (see “Additional information” on page 91).

5.7 Supported data types

The following table illustrates how the IEC 61131-3 data types are linked with the .Net Framework and C#. Variables of data types that are marked with a “+” in the “Attribute data type” column must have the optional “Dataae.g.Type” attribute for unique assignment.

Table 5-1 Supported data types IEC 61131-3, .Net Framework and C#

IEC 61131-3	.Net Framework	C#	Attribut 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	+

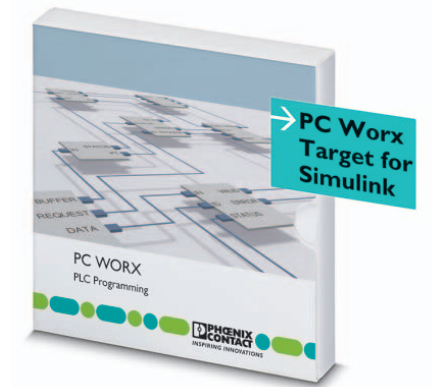
Table 5-1 Supported data types IEC 61131-3, .Net Framework and C#

IEC 61131-3	.Net Framework	C#	Attribut DataType
WORD	System.UInt16	ushort	+
DWORD	System.UInt32	uint	+
LWORD	System.UInt64	ulong	+
STRING	System.Iec61131Lib.Iec-StringEx		-
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_ELEME- NTARY	System.Iec61131Lib.Any		+

6 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 Axio-line controllers. You can integrate the models converted from Simulink® into the PC Worx and PC Worx Engineer development environments.



The model-based design and versatile simulation possibilities of Simulink® can therefore also be used within the PLCnext Technology platform for automation projects. A structured model implementation can be ensured thanks to the automatic creation of executable code. The import and configuration options in PC Worx Engineer also enable Simulink® models to be operated together with programs that have been created in IEC-61131-3 languages and C++. The combination with other high-level-language programs in the same task and the execution in the real-time context is therefore also possible for Simulink® models. Furthermore, you can also realize monitoring and model parameters, and even the optimization of the model during execution, via the “external mode”.

Further information on the add-on and its use is available at:

- 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

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.

This manual, including all illustrations contained herein, is copyright protected. Any changes to the contents or the publication of extracts of this document is prohibited.

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