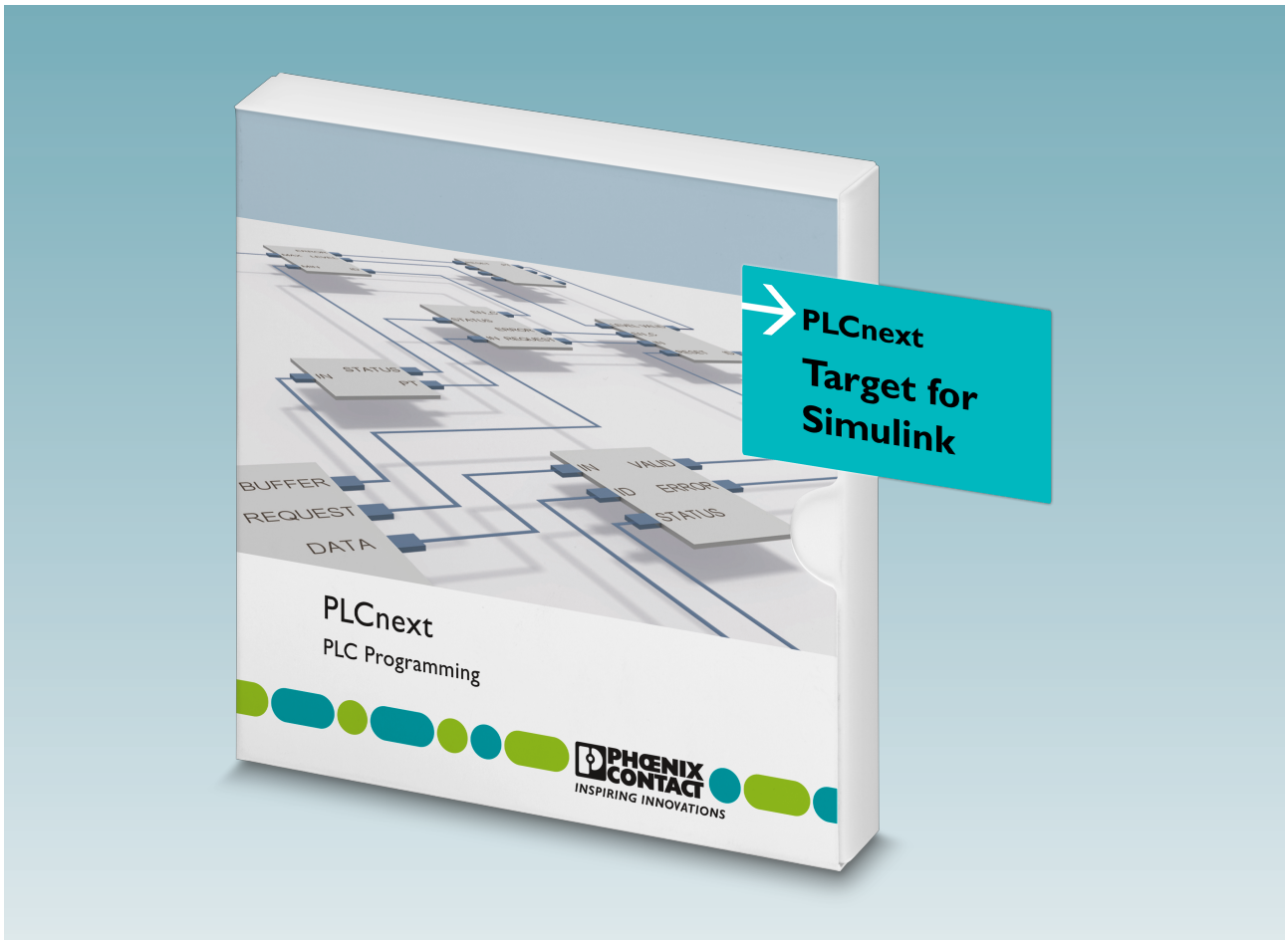


Software add-on for the integration and execution of MATLAB® Simulink® models on Remote Field Controllers and Axioline F controllers

User manual

PHOENIX CONTACT
INSPIRING INNOVATIONS

User manual

Software add-on for the integration and execution of MATLAB® Simulink® models on Remote Field Controllers and Axioline F controllers

UM EN PLCnext TARGET FOR SIMULINK, Revision 16 2024-01-17

This manual is valid for:

Designation	Revision	Item No.
PLCnext Target for Simulink	2.5.1	1326100

106366_en_16

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

Table of contents

1 General information

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

1.1 Identification of warning notes

This symbol indicates hazards that could lead to personal injury.

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

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

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

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

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

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 models and applications, familiarity with MATLAB Simulink, IEC 61131-3, and PLCnext Engineer is also assumed.

1.3 Field of application of the product

You can use the PLCnext Target for Simulink software add-on to execute MATLAB Simulink models on Remote Field Controllers and Axioline F controllers.
To do this, the models created in MATLAB Simulink are imported into the PLCnext Engineer software for PLCnext controllers.

1.4 Trademarks

The terms MATLAB®, Simulink®, Target Language Compiler™ (TLC), Simulink Coder™, and MathWorks® used in this document are registered trademarks of MathWorks.

1.5 PLCnext Target for Simulink licenses

With the purchase of the PLCnext Target for Simulink software add-on, you will receive a single-user license with a license key.

Please note:
If you do not enter a license key during installation, the software add-on is installed in demo mode. The runtime of models compiled in demo mode is limited to one hour.
For further information on demo mode, please refer to Section A 1, “Demo mode”.

1.6 Abbreviations and synonyms used

The following abbreviations and synonyms are used in this document:
- FBD: function block diagram
- TLC: Target Language Compiler
- Model: MATLAB Simulink model
- GDS: global data space
- SDK: software development kit
- PLCnCLI: PLCnext Command Line Interface
- PnE: PLCnext Engineer

2 System requirements

2.1 Supported controllers

The following controllers are supported in conjunction with the listed firmware version:

Table 2-1 Supported controllers

Controller	Item no.	Firmware version
AXC F 1152	1151412	≥2022.6
AXC F 2152	2404267	≥2022.6
AXC F 3152	1069208	≥2022.6
RFC 4072S	1051328	≥2022.6.2
EPC 1502	1185416	≥2022.0
EPC 1522	1185423	≥2022.0
BPC 9120S	1246285	≥2022.0

Definitions:

– PLCnext controller:
  Controller based on the PLCnext Technology. The PLCnext Engineer software is used for programming. Here, you have the option to combine programming in accordance with IEC 61131-3 with various programming languages and development tools.

2.2 Software system requirements

2.2.1 System requirements for creating the model in MATLAB Simulink

To integrate a MATLAB Simulink model into the PLC application, the system requirements that are listed below must be met when the model is created in MATLAB Simulink. The system requirements differ depending on the controller used (PLCnext controller).

- Install the third-party software/features before installing the PLCnext Target for Simulink software add-on.

Table 2-2 System requirements for creating the model in MATLAB Simulink

Software/feature	Description
MATLAB Simulink *	MATLAB Simulink – R2021a, R2021b, R2022a, R2022b, R2023a, R2023b
MATLAB Coder *	Target Language Compiler toolbox for MATLAB
Simulink Coder *	Target Language Compiler toolbox for Simulink
SDKs and TLC templates for supported controllers**	Controller-specific description files for the Target Language Compiler (TLC). They are used to create executable binary files from MATLAB Simulink models.

* Third-party software/feature

** Part of the PLCnext Target for Simulink software add-on

2.2.2 System requirements for integrating the MATLAB Simulink model into the PLC application

If you are using a PLCnext controller, you need the PLCnext Engineer software.

- Before installing the PLCnext Target for Simulink software add-on, install the PLCnext Engineer software.

Table 2-3 System requirements for integrating the MATLAB Simulink model into the PLC application

Software/feature	Description
PLCnext Engineer	Software for PLC programming – ≥2022.6
Microsoft Visual Studio Professional	Visual Studio Compiler with C# support – 2015 **Please note:** The “x64 Compiler Tools” Visual Studio feature is required for the 64-bit version of MATLAB Simulink.

3 Operation overview

You can use the PLCnext Target for Simulink software add-on to integrate models you have created in MATLAB Simulink into the PLCnext Engineer software for PLCnext controllers.

For this, once PLCnext Target for Simulink has been installed, you have to set up and compile the model in MATLAB Simulink for the respective controller (see Section 5).

PLCnext controller

Subsequently, a PLCnext program is available for executing the model on PLCnext controllers. If you select the corresponding model setting for the build, and the requirements are met, the model is also available as a PLCnext model function block.

3.1 Executing the model with a PLCnext program

When you compile the model in MATLAB Simulink, a PLCnext Engineer library (“*.pcwlx”) is generated. This library contains the model as a PLCnext program.

Once you have imported the library into PLCnext Engineer, you can integrate the PLCnext program into your project and execute it on the controller in the same way as an IEC 61131-3 program (see Section 6).

3.2 Executing the model via a PLCnext model function block

When you compile the model in MATLAB Simulink, a PLCnext Engineer library (“*.pcwlx”) is generated. If the PLCN_BuildModelFB model setting is enabled and the model is not built as a multi-tasking model, the library contains the model as a PLCnext model function block.

Once you have imported the library into PLCnext Engineer, you can integrate the PLCnext model function block into your project and execute it as part of an IEC 61131-3 program on the controller (see Section 6).

Note the following restrictions: Only signals for which the grouping into Structs was selected are available in the interface of the generated model function block. For root ports, you should set the PLCN_RPortGenMode setting to “GDS + Engineer”, for example, and enable the PLCN_RPortGroupStruct setting. Furthermore, you cannot use methods for adapting the code generation for PLCnext programs as the PLCnext model function block is not executed from the PLCnext context, but from the eCLR context.

3.2.1 Using the PLCnext subsystem function block library

If your model at the top level only consists of subsystems that each represent a specific function independently of one another, you can build the model as a PLCnext subsystem function block library. Here, each of the subsystems is generated as an independent function block that can be executed from the IEC 61131-3 context either synchronously or asynchronously. From the programmer's point of view, each of the subsystems is a separate function. A subsystem can only be used for this if it does not contain any states (DWorks) and contains only inline parameters. When called, the outputs of the subsystem therefore only depend on the inputs.

This version has a very efficient execution. The step function of the respective subsystem is called up practically directly from the IEC context. External Mode is not available for this version.

Since normal C++ code is ultimately executed, debugging is possible as with the other versions.

4 Installation and updates

4.1 Installing PLCnext Target for Simulink

Please note:

– We recommend that you install Microsoft .Net Framework 3.5 before installing PLCnext Target for Simulink.
See https://www.microsoft.com/en-us/download/details.aspx?id=21.

– A third-party compiler is required for a number of MathWorks products or product functions. Install a compiler before you start installing the PLCnext Target for Simulink.
See https://www.mathworks.com/support/requirements/supported-compilers.html.

To install the PLCnext Target for Simulink software add-on, proceed as follows:

- Close all PLCnext Engineer/PC Worx and MATLAB Simulink applications that are currently running.
- Save the “PLCnext Target for Simulink.exe” setup file to a directory suitable for your application.

The setup file can be downloaded from phoenixcontact.net/product/1326100.

- Double-click on the “PLCnext Target for Simulink.exe” file.
- Select the desired language for the Setup Wizard.

The Setup Wizard is opened.

- Read the End User License Agreement and accept the terms and conditions.
- Select the packages that you wish to install.

Please note:

The software add-on is installed in demo mode. The runtime of models compiled in demo mode is limited to one hour. To enable the full version of the software add-on, once installation is completed, enter your license key in the Phoenix Contact License Manager that is automatically installed with the add-on.

For further information on demo mode, please refer to Section A 1, “Demo mode”.

User-defined setup: selecting packages

The “Package Selection” page is displayed in the Setup Wizard (see Figure 4-1).

This is where you need to select the features you wish to install. Features displayed with the ☑ icon in the structure will be installed automatically. Features displayed with a check box ☐ in the structure can be selected or deselected.

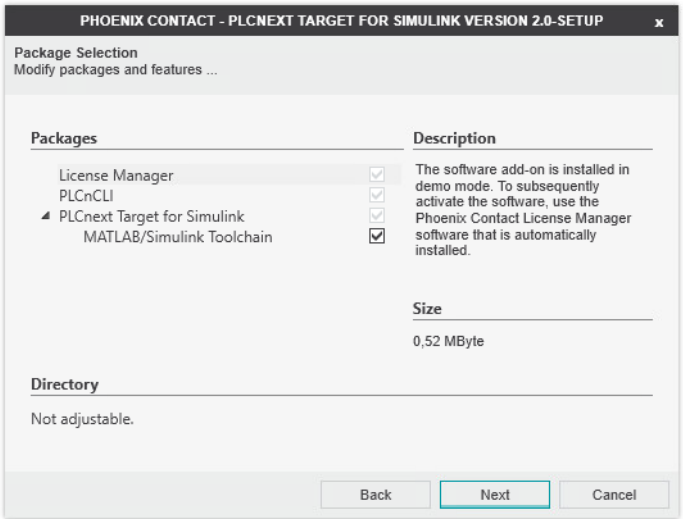


Figure 4-1 Package selection in the Setup Wizard

To add a feature to the installation or exclude it from the installation, change the displayed icon in the structure. To do so, proceed as follows:

- Click on the displayed check box.
- To add a feature to the installation, click on the check box.

If you confirm the check box with a check mark (☑), the package is added to the installation.

- To exclude a feature from the installation, disable the check box.

If you remove the check mark from the check box (☐) the package is not installed.

Please note:
If the system requirements for the relevant features are not met, compatibility problems may arise.

The “License Manager” feature is saved automatically in the following directory:
“C:\Program Files (x86)\Common Files\PHOENIX CONTACT\Licensing”

The PLCnCLI feature is saved automatically in the following directory:
“C:\Program Files\PHOENIX CONTACT\PLCnCLI 2020.0 LTS”

Please note:
The installation paths for the “License Manager” feature and PLCnCLI feature are not configurable and are saved automatically in the standard path.

Changing the installation path for MATLAB/Simulink features

You can choose the installation path for the features under the "PLCnext Target for Simulink" node. To do so, proceed as follows:

- In the structure, click on "MATLAB/Simulink Toolchain".
  The "Directory" button is enabled.
- Click on the "Directory" button (the default path for the package appears as the preselection) and select the desired installation path.

Please note:
If the system requirements for the relevant features are not met, you are not able to proceed with the installation. Please also refer to Section 2.2.

Completing the installation

- Follow the instructions in the Setup Wizard until you have completed the installation process.

Please note:
For PLCnext controllers, you have to install the controller-specific SDKs after the software add-ons have been installed (see Section 4.5).

4.2 Entering the license key at a later point

You can subsequently enter a license key using the Phoenix Contact License Manager software. In this case, you do not need to reinstall the PLCnext Target for Simulink software add-on.

The Phoenix Contact License Manager software is automatically installed with the software add-on.

4.3 Performing an update

In order to update the PLCnext Target for Simulink software add-on, you have to uninstall the version that is currently installed. To do so, proceed as follows:

- Close all PLCnext Engineer/PC Worx and MATLAB Simulink applications that are currently running.
- Double-click on the "PLCnext Target for Simulink.exe" setup file.
- Follow the instructions in the Setup Wizard.
- On the "Change, repair, or remove installation" page, click on the "Remove" button.
- Follow the instructions in the Setup Wizard until you have completed the uninstallation process.
- Install the new version of the software add-on as described in Section 4.1.

4.4 Performing a repair

To be able to use PLCnext Target for Simulink with further MATLAB Simulink versions after installing them, proceed as follows:

- Close all PLCnext Engineer and MATLAB Simulink applications that are currently running.
- Double-click on the “PLCnext Target for Simulink.exe” setup file.
- Follow the instructions in the Setup Wizard.
- On the “Change, repair, or remove installation” page, click on the “Repair” button.
- Follow the instructions in the Setup Wizard until you have completed the repair process.

4.5 Installing and registering the SDK for PLCnext controllers

If you use a PLCnext controller: After installing PLCnext Target for Simulink, install the controller-specific SDK and register the SDK root directory in MATLAB Simulink. The SDK is installed via the PLCnCLI (PLCnext Command Line Interface), which is automatically installed with the software add-on.

For more detailed information on the PLCnCLI (PLCnext Command Line Interface), please refer to the “PLCnext Technology” user manual (UM EN PLCNEXT TECHNOLOGY).

To this end, the “PLCN.Toolchain.plcncli” is available in MATLAB Simulink. You can display information about the available functions by means of the MATLAB command “doc(‘PLCN.Toolchain.plcncli’)”.

SDK management is best performed using the “SDK Manager”. You can start the “SDK Manager” via the command “PLCN.Toolchain.plcncli.SDKManager();” or via the MATLAB Simulink menu.

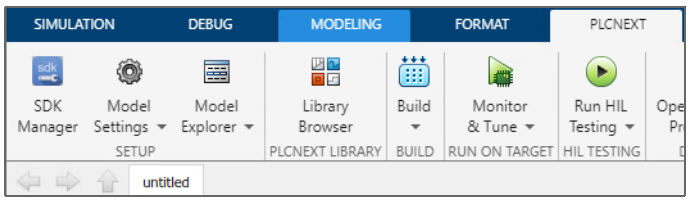


Figure 4-2 Starting the SDK Manager via the MATLAB Simulink menu

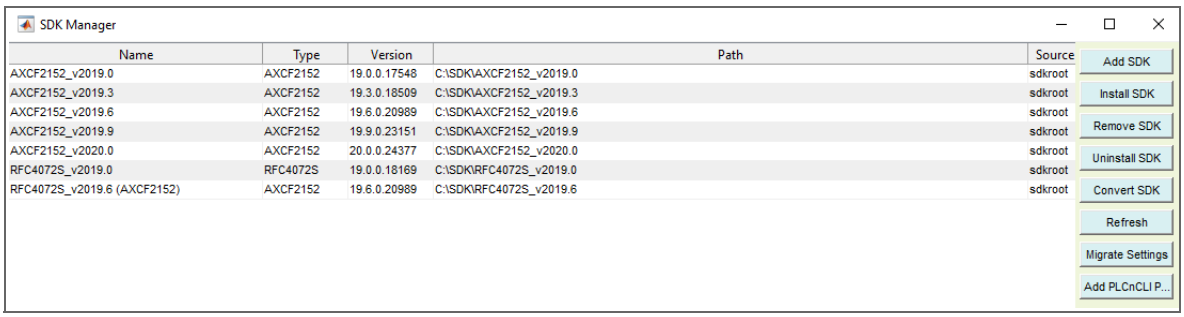


Figure 4-3 SDK Manager

4.5.1 Using the SDK Manager

Add PLCnCLI Path

If the PLCnCLI has not been registered in the Windows path, you can use the “Add PLCnCLI Path” button (which is only available in this case).

Migrate Settings

If no SDK has been registered via the PLCnCLI, you can export SDKs registered in earlier PLCnCLI versions and register them for the current version. To do so, use the “Migrate Settings” button. This button is only available if no SDK has been registered.

Add SDK

You can use the “Add SDK” button to register an SDK that is already on the hard disk for the PLCnCLI tool, in order to use it in Simulink.

Remove SDK

With the “Remove SDK” button, you can remove all PLCnCLI SDKs selected in the SDK Manager from the PLCnCLI tool. However, the SDKs remain on the hard disk and can still be used in Simulink if they are in the SDK root directory (see Section 4.5.2).

Convert SDK

With the “Convert SDK” button, you can register all non-PLCnCLI SDKs selected in the SDK Manager for the PLCnCLI tool. As a result, SDKs that could previously only be used in Simulink also become available to other PLCnext parts (e.g., Eclipse C++ toolchain).

Install SDK

To use an SDK in Simulink, you can use the “Install SDK” button to unpack an SDK archive (“*.tar.gz”) to a selected location on the hard disk and register it for the PLCnCLI tool. The selected installation folder should be empty.

Please note:
The resulting directory hierarchy becomes very deep. Windows limits the path length to 260 characters.

Uninstall SDK

With the “Uninstall SDK” button, you can remove all SDKs selected in the SDK Manager from the PLCnCLI tool and delete them from the hard disk.

Refresh

If the SDK configuration is changed outside the SDK Manager, you can use the “Refresh” button. This makes changes visible and you can use newly added SDKs in Simulink, for example.

Moving/renaming the SDK by changing the “Path” column

You can move SDKs in the SDK Manager by changing the “Path” column. If the SDK has not been registered using the PLCnCLI tool, and is no longer present in the “SDK root” or “targets” directory after having been moved or renamed, the SDK Manager is no longer able to find the SDK.

4.5.2 Using the SDK root directory

Registering the SDK root directory via MATLAB command

To register the SDK root directory in MATLAB Simulink, proceed as follows:

- Enter the following MATLAB command:

```
PLCN.setupSdkRoot
```

- Enter the path to the root directory of the SDK.
  Example: The SDKs are located in the subdirectories of “C:\SDK”. Enter the following command:

```
C:\SDK
```

Within the SDK root directory, each folder has to contain an SDK. To add a new SDK, manually unpack the SDK archive “*.tar.gz” to such a folder.

For the SDKs to be detected and used, you have to run the “PLCN.refreshTargets()” function or restart MATLAB if you have changed a subdirectory or an SDK in this directory.

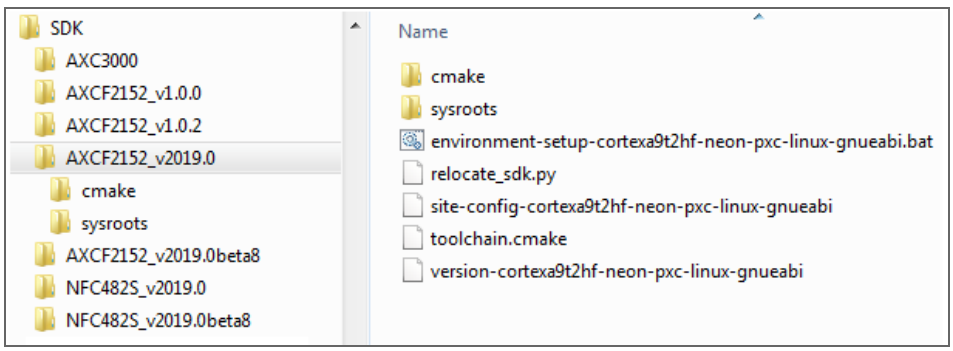


Figure 4-4 Example of an SDK directory

4.6 Upgrading the model configuration for PC Worx Target for Simulink models

You can convert models that have been configured for the old PC Worx Target for Simulink to PLCnext Target for Simulink. To do so, use the command “PLCN.Utils.updatePCWorxModel(ModelName)”. This command changes the target to “pxc_plcn.tlc” and adopts the values of the old HLLI_* Settings to the new PLCN_* configuration.

4.7 Troubleshooting PLCnCLI/EngineeringLibraryBuilder

With certain operating system configurations, a .NET error leads to the operating system deleting certain files from the temporary folder of PLCnCLI/EngineeringLibraryBuilder. This means that the programs can no longer be executed. The typical error message is: “The application to execute does not exist“ with reference to “C:\Users\XXX\AppData\Temp\.net\...”.

In this case, delete the respective subfolder manually from the directory “C:\Users\XXX\AppData\Temp\.net\EngineeringLibraryBuilder” or “C:\Users\XXX\AppData\Temp\.net\plcncli”.

The next time the corresponding application is started, the folders are fully created again via the .NET functionality. You can then execute the program normally again.

4.8 Configuring models for a multi-target build

With the multi-target support feature, you can generate the code and compilations for different targets using several stand-alone model configurations. You can create two model configurations, for example. One for AXC F 2152 and one for RFC 4072. You can compile both model configuration through this batch build. In this way, you create executable code for different architectures.

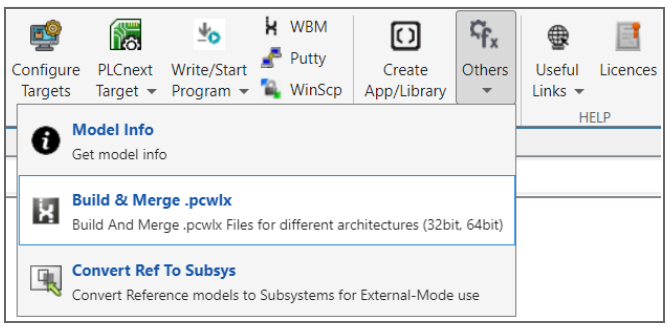


Figure 4-5 Build for multi-targets via the PLCnext tab

Based on the installed SDKs of the different controllers, you can select the targets for which a build is to be generated. Example:

```
set_param('IParam', 'PLCN_MultiTargets', '2021.0.5 (RFC4072S)|2021.0.3 (AXCF2152)')
```

5 Setting up and compiling a model in MATLAB Simulink

This section describes how to set up and compile the model in MATLAB Simulink so that you can later integrate it into the PLCnext Engineer software.

5.1 Notes for model creation in MATLAB Simulink

Please note the restrictions on the path length:

- Windows limits the path length to 260 characters. Because the model name appears several times in the PLCnext Engineer path for the binary files, the model name has to be limited. This also means that the instance path of a data point in PLCnext Engineer is limited. You can change the folder for binary files in the PLCnext Engineer settings area. Shorter paths allow for longer model names.
- You can practically remove this path restriction via the Windows 10 group policy “Activate long Win-32 paths”.

Please note:

- For identifiers, use designations that conform to IEC 61131.
- For Simulink bus types (such as parameters and bus fields) use other names than those for the variables.

Please note:

- In PLCnext Engineer, you can only use model signals of the storage classes “Auto” and “Model default” (previously “SimulinkGlobal”) as GDS signals or for interconnection with other programs.
- To be able to use the parameters, you have to activate the “Parameter Generation” setting on the “PLCN Code Generation” configuration page (see Section 5.2.2).
- For additional information on the storage classes, please refer to the MATLAB Simulink documentation.

Please note:

- If you set the model setting “DefaultParameterBehavior” to the value “Inlined”, you can only use parameters of the storage class “Model default” (previously “SimulinkGlobal”) in PLCnext Engineer.
- If you set the model setting “DefaultParameterBehavior” to the value “Tunable”, you can also use parameters of the storage class “Auto”.

If one of these parameters cannot be used due to the code generation (cannot be adjusted at runtime), there is what is referred to as “loss of tunability”.

- Also refer to the subject “Limitations for Block Parameter Tunability in Generated Code” in the MATLAB Simulink documentation regarding this.
- If you configure the corresponding model setting “ParameterTunabilityLossMsg”, a warning or an error may be output in this case when the model is compiled.

5.2 Setting up a model in MATLAB Simulink

To set up the model for subsequent integration into the PLCnext Engineer software, proceed as follows in MATLAB Simulink:

- In the toolbar, click on the button.

The “Configuration Parameters” window opens (see Figure 5-1).

- In the navigation on the left, click on the “Code Generation” entry.
- In the “Target selection” area, click on the “Browse” button.
- In the “System Target File Browser” window, select the “pxc_plcn.tlc” file.
- Click on the “Apply” button to apply your entries.
- Click on the “OK” button to close the window.

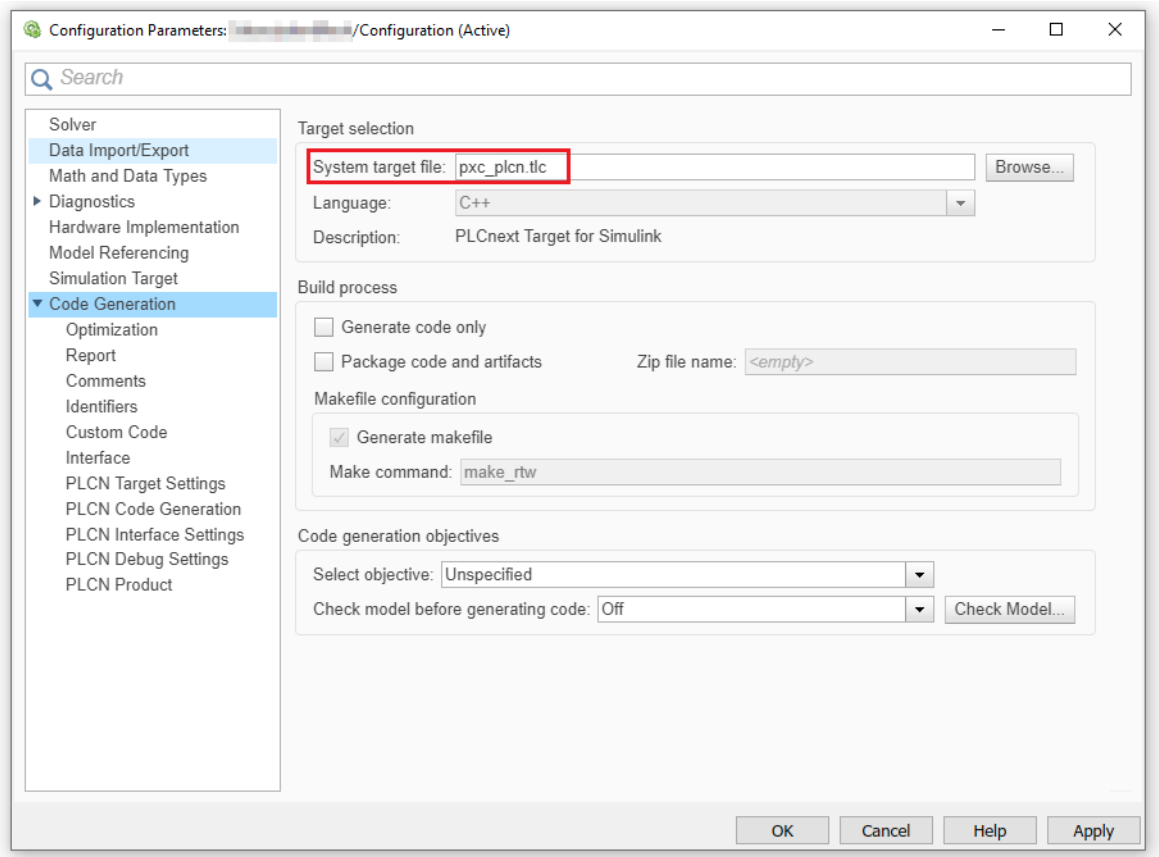


Figure 5-1 “Code Generation” configuration page

When you select the “pxc_plcn.tlc” file, additional configuration pages appear under the “Code Generation” entry in the navigation list on the left.

Setting the solver increment

As the controller is a discrete system with a fixed cycle time, the “Fixed-step” solver type is preset (see Figure 5-2).

To set the solver increment for the model, proceed as follows:

- In the navigation list on the left, click on the “Solver” entry.

- In the “Fixed-step size” input field, enter the same interval that is specified in the task settings of your PLC application in PLCnext Engineer.

The interval that is to be entered is required for offline simulation in MATLAB Simulink. It only has a limited influence on the processing time in the controller.

If you enter a different interval in your PLC application, the model behavior may differ from the simulation.

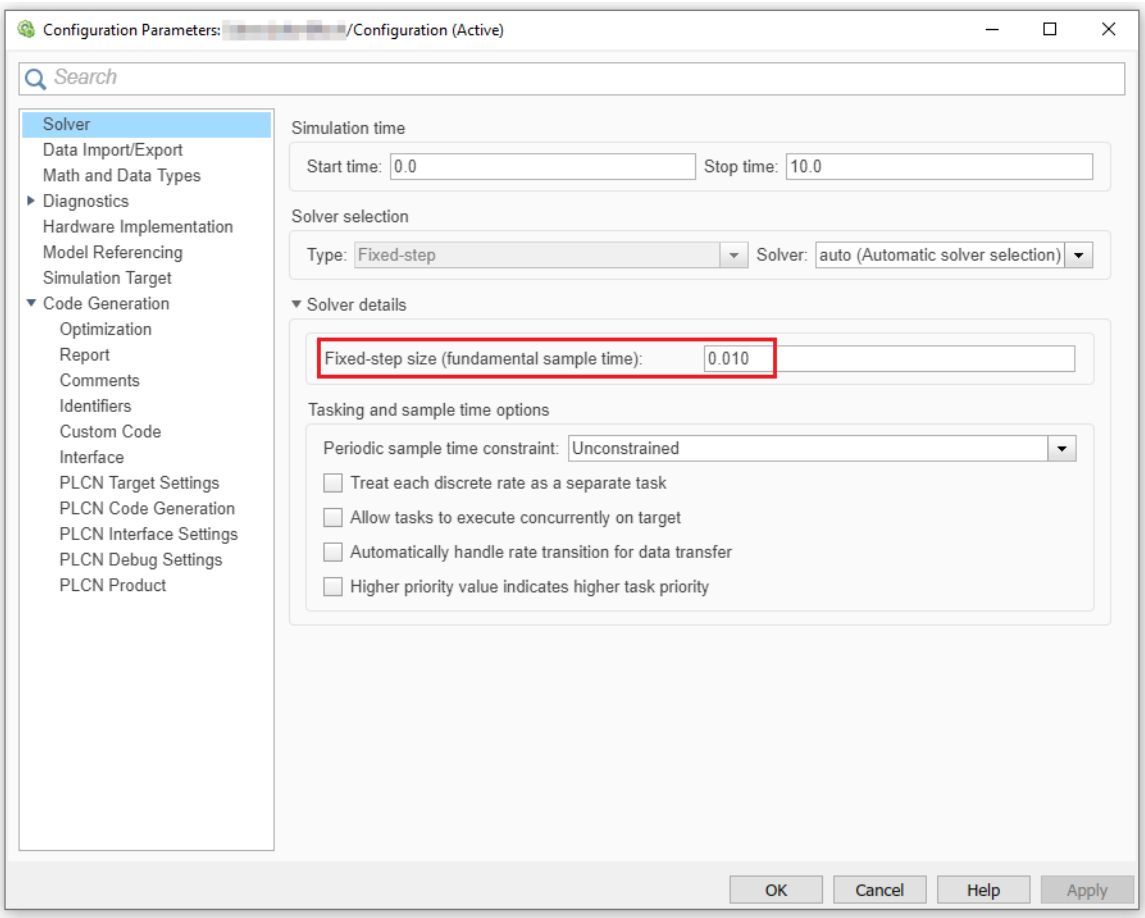


Figure 5-2 “Solver” configuration page

Opening additional configuration pages

For further settings, open the additional configuration pages. To do so, proceed as follows:

- In the navigation list on the left, click on the configuration page that you wish to open (see Section 5.2.1 to 5.2.5).

5.2.1 “PLCN Target Settings” configuration page

For a description of all settings, refer to Table 5-1.

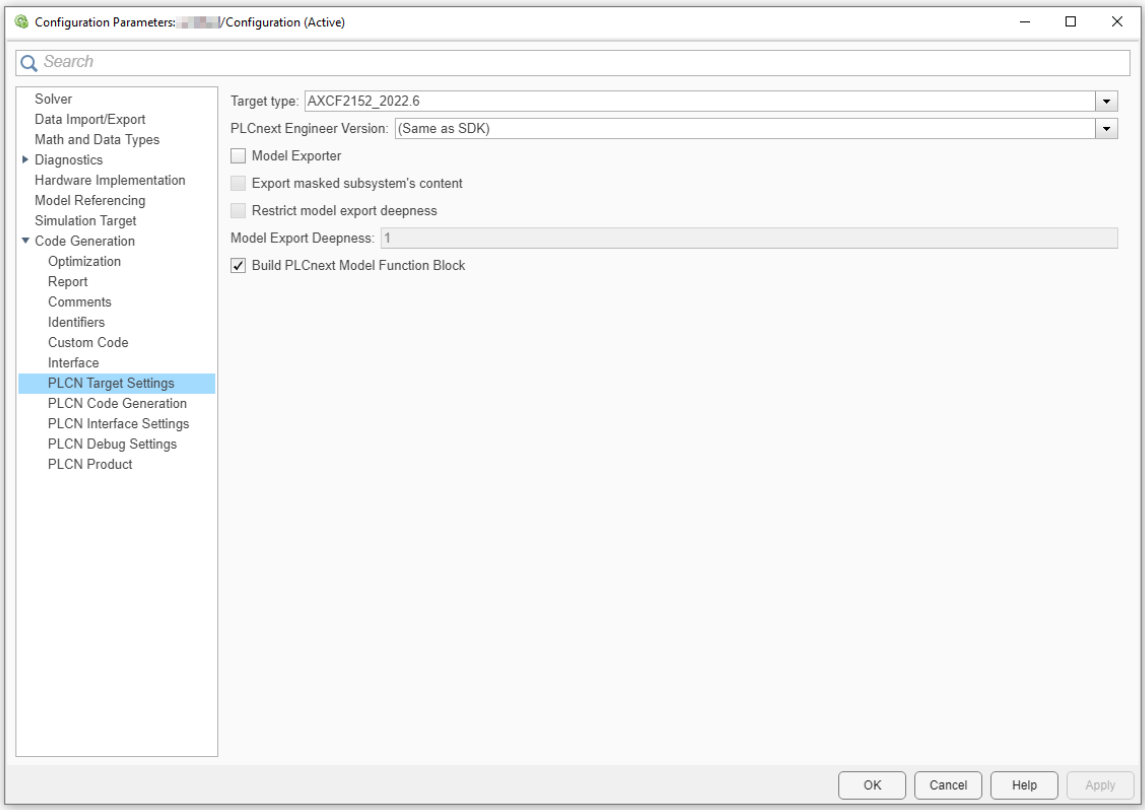


Figure 5-3 “PLCN Target Settings” configuration page

Table 5-1 Description of the settings on the “PLCN Target Settings” configuration page

Setting	Parameter	Default setting	Description
Target type	PLCN_TargetType	Depends on the controllers that are installed	Controller that is being used
PLCnext Engineer Version	PLCN_EngineerVersion	(Same as SDK)	Describes the PLCnext Engineer version for which a model library will be generated. Select the earliest version with which the library is to be compatible. If a library is loaded from a later PLCnext Engineer version, a conversion is carried out for each intermediate version in the background.

Table 5-1 Description of the settings on the “PLCN Target Settings” configuration page

Setting	Parameter	Default setting	Description
Model exporter	PLCN_ExportDiagram	Disabled	If you enable this check box, you can use the model representation in PLCnext Engineer. For the model representation in PLCnext Engineer, you require the “Viewer for Simulink” PLCnext Engineer add-in.
Export masked subsytem's content	PLCN_ExportMasked	Disabled	If you enable this check box, the content of masked subsystems for the “Viewer for Simulink” are exported. For the model representation in PLCnext Engineer, you require the “Viewer for Simulink” PLCnext Engineer add-in.
Restrict model export deepness	PLCN_ExportDeepnessRestricted	Disabled	If you enable this check box, only blocks down to a certain model level are exported. This level can be specified in the following setting, “Model export deepness”.
Model export deepness	PLCN_ExportDeepness	1	Number of model levels to be exported. Example: If you enter the value “1”, only blocks of the uppermost model level are exported.
Build PLCnext Model Function Block	PLCN_BuildModelFB	Enabled	If you enable this check box, the PLCnext model function block is also compiled and stored in the PCWLX library.

5.2.2 “PLCN Code Generation” configuration page

For a description of all settings, refer to Table 5-2.

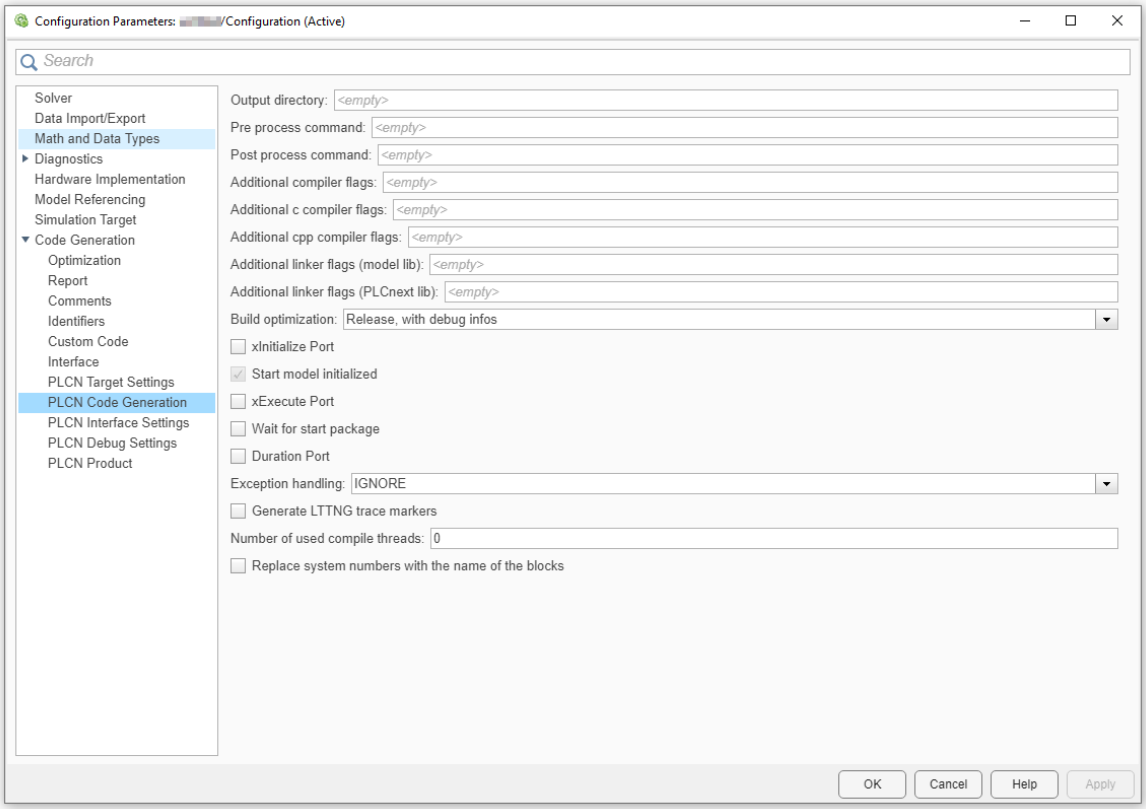


Figure 5-4 “PLCN Code Generation” configuration page

Table 5-2 Description of the settings on the “PLCN Code Generation” configuration page

Setting	Parameter	Default setting	Description
Output directory	PLCN_OutDir	–	Files generated during model compilation (*.pcwlx) are stored in the output directory.
Pre process command	PLCN_Preprocess	–	Here, you can enter a MATLAB command that will be executed before compilation is started.
Post process command	PLCN_Postprocess	–	Here, you can enter a MATLAB command that will be executed after successful compilation.
Additional compiler flags	PLCN_CompileFlags	-	Additional compiler flags that are used to compile “.c” and “cpp” files.
Additional c compiler flags	PLCN_CompileFlagsC	-	Additional compiler flags that are used to compile “.c” files.
Additional cpp compiler flags	PLCN_CompileFlagsCxx	-	Additional compiler flags that are used to compile “cpp” files.

Table 5-2 Description of the settings on the “PLCN Code Generation” configuration page

Setting	Parameter	Default setting	Description
Additional linker flags (model lib)	PLCN_LinkFlags	-	Additional linker flags that are used when linking the model library (“<model>.so” file).
Additional linker flags (PLCnext lib)	PLCN_LinkFlagsPLCN	-	Additional linker flags that are used when linking the program library (“<model>_plcn.so” file).
Build optimization	PLCN_OptFlags	Release, with debug info	Possible settings for adapting the optimization level during compilation: – Fast Build: “-O0”, usually the fastest build, but low runtime efficiency in comparison. – Debug: “-O0 -g”, lack of optimizations and additional debug information make this compilation easy to debug (e.g., using GDB). – Release, with debug infos: “-O2 -g”, compilation is optimized regarding runtime and memory usage; however, it is easy to debug in the event of an error, thanks to additional debug information. – “Release, optimize for code size” “-Os”, compilation is optimized in terms of runtime and memory usage, but the focus is on minimizing the binary sizes. – Release, optimize for code efficiency: “-O3”, compilation is optimized fully in terms of the runtime and system usage.
xInitalize port	PLCN_GenInitializePort	Disabled	If you enable this check box, the execution of the model depends on the value of the xInitialize port.
Start model initilized	PLCN_StartInitialized	Enabled	The model starts in the initialized state. If enabled, the model is already loaded in the initialization phase of the PLCnext library.
xExecution Port	PLCN_GenExecutePort	Disabled	If you enable this check box, the execution of the model depends on the value of the xExecution port.
Wait for Start package	PLCN_WaitForStartPkg	Disabled	If you enable this check box, model execution can only be started and stopped via the MATLAB Simulink interface (External Mode).
Duration port	PLCN_GenDurationPort	Disabled	If you enable this check box, the execution duration for all model programs is output to the diagnostic structure in µs and is updated after each execution.

Table 5-2 Description of the settings on the “PLCN Code Generation” configuration page

Setting	Parameter	Default setting	Description
Exception handling	PLCN_ExceptionHandling	IGNORE	Here, you can modify the behavior of the model for “exception handling”. Possible settings: – IGNORE: Exceptions are not captured. In the event of FPU exceptions (e.g., Division by Zero with the float data type), processing of the model continues and signals can have the values “Inf” and “NaN”. In the event of SIGFPE signals (e.g., Division by Zero with integer data type), the signal is forwarded to the controller and the controller normally switches to an error state. – CATCH-FPU: FPU exceptions of the type “Division by Zero” are captured. In the event of an error, the diagnostic code “0xCF00” is displayed at the “wDiagCode” output.The program execution is stopped. In the event of SIGFPE signals (e.g., Division by Zero with integer data type), the signal is forwarded to the controller and the controller normally switches to an error state. – CATCH_ALL: FPU exceptions of the type “Division by Zero” are captured, as are SIGFPE signals (e.g., “Division by Zero” with integer data type). In the event of an error, the diagnostic code “0xCF00” is displayed at the “wDiagCode” output. If supported by the hardware, both FPU exceptions and SIGFPE signals are captured. The program execution is stopped.
Generate LTTNG trace markers	PLCN_EnableLTTNG	Disabled	If you enable this check box, code markers are generated for the LTTNG diagnostic.
Number of used compile threads	PLCN_NumCompileThreads	0	Limits the number of files that can be compiled at the same time. Use a high value for faster compiling. Use a low value to save resources. – 0: Auto
Replace system numbers with the name of the blocks	PLCN_ReplaceSysNumBlockNames	Disabled	If you enable this check box, the system numbers (e.g., “<S1>” in “<model>.h”) are replaced with the names of the Simulink blocks.

5.2.3 “PLCN Interface Settings” configuration page

On the “PLCN Interface Settings” configuration page, you can set the type and behavior of the model interfaces.

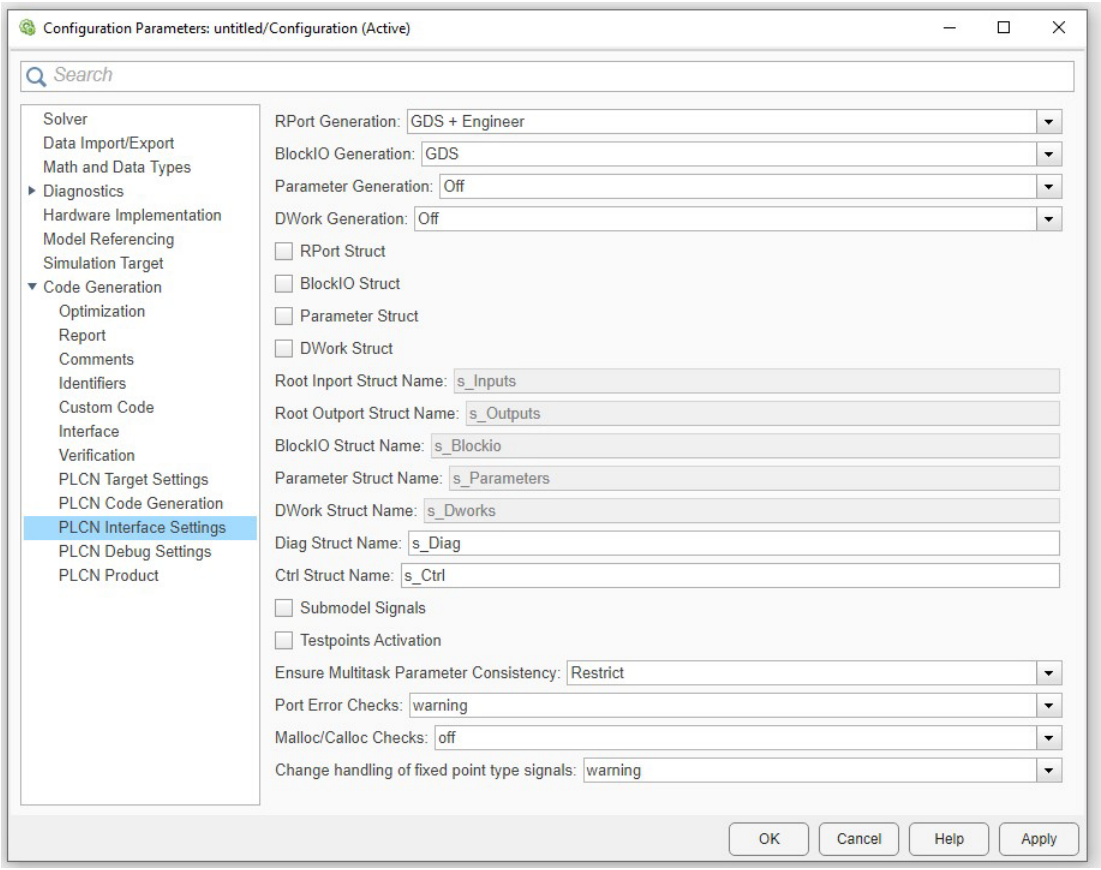


Figure 5-5 “PLCN Interface Settings” configuration page

Table 5-3 Description of the settings on the “PLCN Interface Settings” configuration page

Setting	Parameter	Default setting	Description
RPort Generation	PLCN_RPortGenMode	GDS + Engineer	Here, you can determine how root port signals are provided at the program interface. Possible settings: – Off: Signals of this signal group are not available at the program interface. This does not apply to signals with set meta flags (see Section 5.4). Signals with set meta flags are nevertheless available as GDS signals. – GDS: Signals of this signal group are available as GDS signals. – GDS + Engineer: Signals of this signal group are available as GDS signals and can be interconnected in PLCnext Engineer. **Please note:** You can change the setting for the individual signals by means of the context menu in the model (see Section 5.4).
BlockIO Generation	PLCN_BlockIOGenMode	GDS	Here, you can determine how BlockIO signals are provided at the program interface. Possible settings: See “RPort Generation”.
Parameter Generation	PLCN_ParamGenMode	Off	Here, you can determine how parameters are provided at the program interface. Possible settings: See “RPort Generation”.
DWork Generation	PLCN_DWorkGenMode	Off	Here, you can determine how block statuses (DWorks) are provided at the program interface. Possible settings: See “RPort Generation”.
RPort Struct	PLCN_RPortGroupStruct	Disabled	If you enable this check box, root port signals are combined into structures at the program interfaces.
BlockIO Struct	PLCN_BlockIOGroupStruct	Disabled	If you enable this check box, BlockIO signals are combined into structures at the program interfaces.
Parameter Struct	PLCN_ParamGroupStruct	Disabled	If you enable this check box, “SimulinkGlobal” parameters are combined into structures at the program interfaces.
DWork Struct	PLCN_DWorkGroupStruct	Disabled	If you enable this check box, block statuses (DWorks) are combined into structures at the program interfaces.

Table 5-3 Description of the settings on the “PLCN Interface Settings” configuration page

Setting	Parameter	Default setting	Description
Root Inport Struct Name	PLCN_RPortInStructName	s_Inputs	Name of the Root Inport input structure.
Root Outport Struct Name	PLCN_RPortOutStructName	s_Outputs	Name of the Root Outport output structure.
BlockIO Struct Name	PLCN_BlockIOStructName	s_BlockIO	Name of the BlockIO output structure.
Parameter Struct Name	PLCN_ParamStructName	s_Parameters	Name of the parameter output structure.
DWork Struct Name	PLCN_DWorkStructName	s_Dworks	Name of the DWork output structure.
Diag Struct Name	PLCN_DiagStructName	s_Diag	Name of diagnostic output structure.
Ctrl Struct Name	PLCN_CtrlStructName	s_Ctrl	Name of the program control input structure.
Submodel signals	PLCN_SubmodelSignals	Disabled	If you enable this check box, signals from submodels are added as GDS signals.
Testpoints Activation	PLCN_TestPointsActivation	Disabled	If you select the check box, test points will be set on the ports.
Ensure Multitask Parameter Consistency	PLCN_ParamMultirateHandling	Restrict	Here, you can determine how parameters are handled that belong to various tasks in a multitasking program. Possible settings: – Off Consistency between the parameters is not monitored. – Buffer All parameters are bound to the fastest task. Parameter changes are only adopted in the case of a joint Sample Hit for all tasks. – Restrict Parameters that are attached to different tasks are not permitted and cause an error during compilation.
Port Error Checks	PLCN_PortchkError	warning	Here, you can change the behavior in case a root port signal is renamed or not generated during compilation (see also Section 5.5). Possible settings: – off No message is output. – warning During compilation, a warning message is output. – error During compilation, an error message is output.

Table 5-3 Description of the settings on the “PLCN Interface Settings” configuration page

Setting	Parameter	Default setting	Description
Malloc/Calloc Checks	PLCN_ChkAlloc	off	Here, you can specify whether the code generated by the Simulink Coder in the “<model>.cpp/.c” should be checked for functions for dynamic memory allocation “malloc/calloc”. Possible settings: – off The check is not performed. – warning A warning is issued if one of the functions is found. – error An error is triggered if one of the functions is found.
Change handling of fixed point type signals	PLCN_FixedPointHandling	warning	Here, you can define how fixed point signals are handled. Fixed point signals are represented in the GDS like the underlying integer type. Possible settings: – warning Fixed point signals become GDS signals. A warning is issued. – exclude Fixed point signals do not become GDS signals. A warning is issued. – error Fixed point signals stop the build.

5.2.4 “PLCN Debug Settings” configuration page

On the “PLCN Debug Settings” configuration page, you can configure settings for using the Eclipse programming tool to debug the model code.

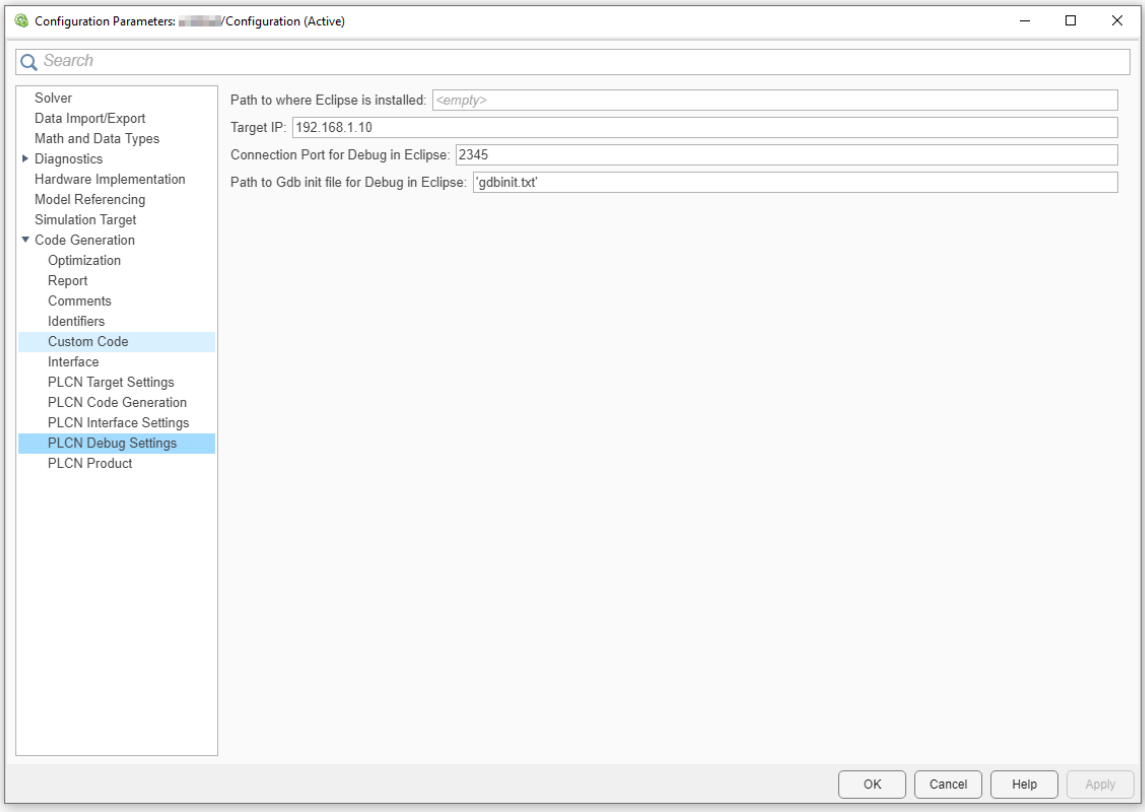


Figure 5-6 “PLCN Debug Settings” configuration page

Table 5-4 Description of the settings on the “PLCN Debug Settings” configuration page

Setting	Parameter	Default setting	Description
Path to where Eclipse is installed	PLCN_PathToEclipseInstall		Path to the Eclipse installation.
Target IP	PLCN_TargetIP	192.168.1.1	Target IP address.
Connection Port for Debug in Eclipse	PLCN_ConnectionPort	2345	Here, you can configure the port for the debug settings in Eclipse.
Path to Gd binit file for Debug in Eclipse	PLCN_PathToGdbInit	C:\gdbinit.txt	Here, you can specify the path to the GDB init file for the debug settings in Eclipse.

5.2.5 “PLCN Product” configuration page

Here, you can specify further file properties of the model. For a description of all settings, refer to Table 5-5.

For PLCnext models, this information is saved in the “<modelname>_plcn.rc” file. This file is included in the PCWLX model library.

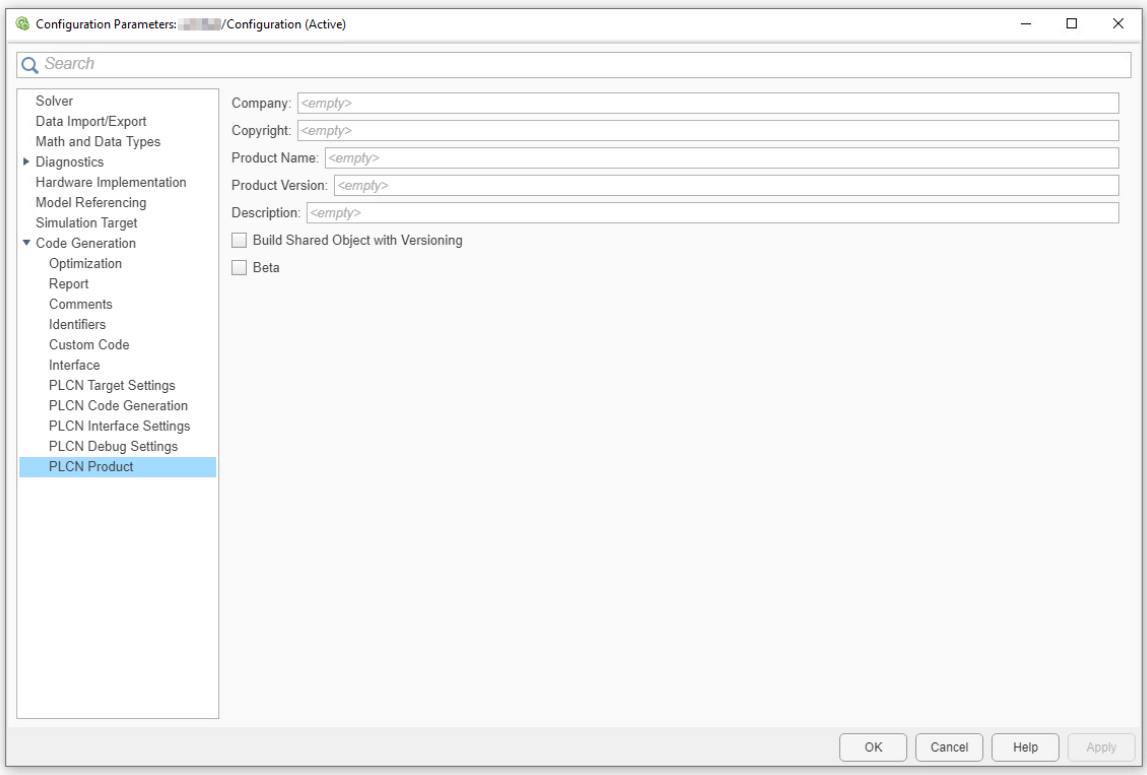


Figure 5-7 “PLCN Product” configuration page

Table 5-5 Description of the settings on the “PLCN Product” configuration page

Setting	Parameter	Description
Company	PLCN_ResCompany	Company information
Copyright	PLCN_ResCopyright	Copyright information
Product name	PLCN_ResProductName	Product name
Product Version	PLCN_ResProductVersion	Product version
Description	PLCN_ResDescription	Description
Build Shared Object with Versioning	PLCN_ResIsVer	If you enable this check box, a model version number following the “lib<Modelname>.so.<major>.<minor>.<patch>[-beta[-L|-S]]” name scheme is applied to the shared objects. L: local build S: server build <major>.<minor>.<patch> is contained in the “Product Version”.
Beta	PLCN_ResIsBeta	If you enable this check box, the model is marked as “Beta”. The default value is “Release”.

You can enter text or a MATLAB command with return value (string) in the input fields.

Alternatively, you can specify the file properties using the MATLAB command *set_param(gcs,<parameter name>,<val>)*.

5.2.6 More model settings

Use the following command to ensure the retain memory assignment is applied when the controller is restarted:

```
set_param('<modelName>', 'PLCN_ShowNeededRetainMem', 'on')
```

To create *.so files in stripped format, select “Release, optimize for code size” in the model setting under “Build optimization”.

5.3 Converting PC Worx Target for Simulink models to PLCnext Target for Simulink

"PLCnext Target for Simulink" must be installed in order to convert models from "PC Worx Target for Simulink".

- You can then use the following MATLAB command for the conversion:

```
PLCN.Utils.updatePCWorxModel(<Modellname>)
```

5.4 Setting meta flags

You can set meta flags via the context menu of a block in MATLAB Simulink. Setting one or several meta flags influences how a signal is provided by the various interfaces.

Table 5-6 Meta flags for the various interfaces

Meta flag	Description
Retain	The signal turns into a remanent signal that is retained during a warm start of the controller and uses the retain function of the controller.
Proficloud	The signal is available via the PROFICLOUD interface.
eHMI	The signal is available via the eHMI interface.
OPCUA	The signal is available via the OPC UA interface.

To set a meta flag, proceed as follows:

For root port blocks, port blocks, and lines

- Right-click on the block or the line.
- In the context menu, use the "Signal interface settings" entry to select the desired meta flag (signal flag).

For other blocks, e.g., delay blocks

- Right-click on the block.
- In the context menu, use the "Parameter interface settings" entry or the "State interface settings" entry to select the desired meta flag (parameter flag or state flag, see example in Figure 5-8).

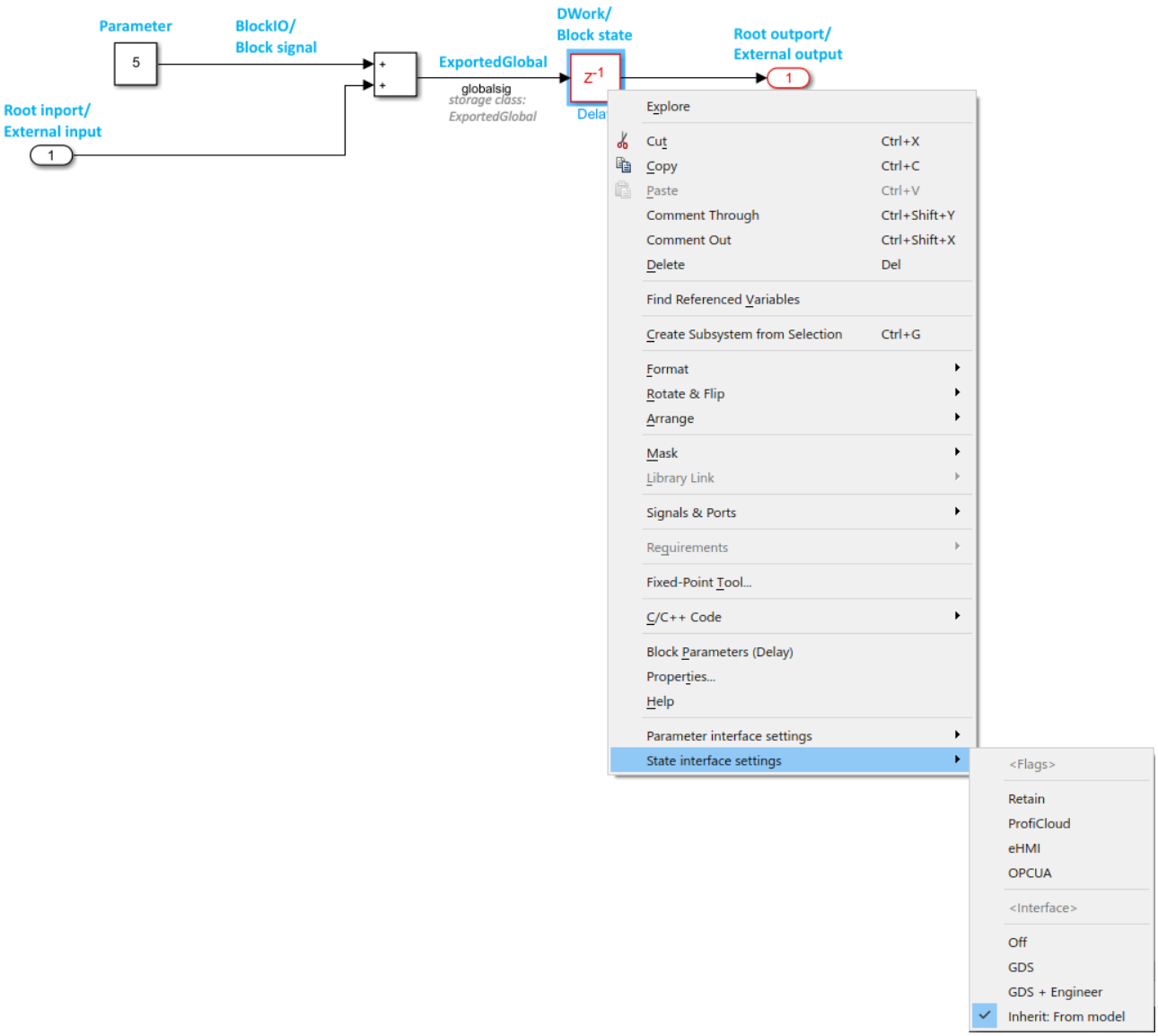


Figure 5-8 Example: setting a meta flag via the context menu of a delay block

Please note:
Setting the meta flag only influences how the signal generated during compilation is provided by the various interfaces. You have to set up the signal generation by configuring the corresponding model settings.

Signal flags

Signal flags are signal-specific. They are bound to the block in which the signal is generated (source block of the signal). The signal flags on the root port blocks are an exception. They are bound directly to the respective root port block.

Parameter flags and state flags

Parameter flags and state flags are not signal-specific. They are bound to the block that the flags are applied to. Here, the selected flags are applied to all parameters and/or states that the Simulink Coder generates for this block.

The meta flags are saved in the “<model>.metadata” file in the same directory as the MATLAB Simulink model (*.slx). During compilation, the “<model>.metalog.csv” log file is created in the “<model>_PLCN” directory. This log file contains the information on which meta flags were set where.

Meta flags for providing a signal

Additionally, you can use the context menu to specify whether the signal is to be provided at the program interface. By default, the “Inherit: From model” setting is active, i.e., the setting specified for the respective signal group is applied (see “PLCN Code Generation” configuration page, Section 5.2.2).

- If you want to change the setting for an individual signal, select one of the following options in the context menu:

Table 5-7 Interface options for providing a signal

Meta flag	Description
Off	This signal is not available at the program interface.
GDS	The signal is available as a GDS signal.
GDS + Engineer	The signal is available as a GDS signal and can be connected in PLCnext Engineer.
Inherit: From model	Default setting: Whether the signal is available at the program interface depends on the setting for the respective signal group on the “PLCN Code Generation” configuration page (see Section 5.2.2).

5.4.1 Setting meta flags via programming

The following functions are available for setting the meta flags:

– PLCN.MetaManager.setMetaFlags
– PLCN.MetaManager.getMetaFlags
– PLCN.MetaManager.setMetaInterface
– PLCN.MetaManager.getMetaInterface

The meta flags or interfaces for multi-ports or multi-blocks are set via “PLCN.MetaManager.setMetaInterfacesBlksOrLines“ or “PLCN.MetaManager.setMetaFlagsBlksOrLines” commands. To replace the meta flags or interfaces for the selected part of the model, the commands are also called as follows:

```
PLCN.MetaManager.setMetaInterfacesBlksOrLines(find_system(bdroot,'Find-
All','on','selected','on'), ‘GDS’)
```

or

```
PLCN.MetaManager.setMetaFlagsBlksOrLines(find_system(bdroot,'Find-
All','on','selected','on'), 32)
```

The exact use of the functions can be queried using the “doc” or “help” command in MATLAB.

5.4.2 Synchronization of meta flags from model libraries

If meta flags for Simulink blocks from block libraries are to be used, there are two possible methods for implementing this. The easiest method is to set the flags directly, at the respective blocks in the main model.

As an alternative, however, you can also set the flags at the respective block in the block library. In this case, you have to synchronize the library flags and model flags afterwards. This is done by using the function “PLCN.MetaManager.syncLibraryMetaData”.
E.g., “PLCN.MetaManager.syncLibraryMetaData(bdroot,false);”.

For a description of the function call, use the “doc” or “help” commands in MATLAB.

5.5 Notes on naming model signals

When creating the model signals in MATLAB Simulink, keep in mind how the names affect the model signals that are later provided in the GDS. Below you will find information on how to name the various signal types.

For more detailed information on the GDS (Global Data Space), please refer to the “PLCnext Technology” user manual (UM EN PLCNEXT TECHNOLOGY).

Root port/ExportedGlobal signals

Root port signals are generated with the names of the root port blocks from the model. Signals with the ExportedGlobal storage class are generated with the respective signal name.

Root port signals and ExportedGlobal signals share a namespace. If there are naming conflicts, the Simulink Coder automatically performs renaming. ExportedGlobal signals have priority over root port signals.
Example: If there is a root outport block named “SignalX” and an ExportedGlobal signal with the same name, the root output block is provided with an underscore followed by letters as a suffix, e.g., “SignalX_i”.

If root port signals are renamed as a result of name conflicts or if the model configuration is not provided as a GDS signal, a warning is displayed during compilation (default setting). You can use the “Port Error Checks” setting on the “PLCN Interface Settings” configuration page to change the notification behavior.

BlockIO signals:

BlockIO signals are generated with the name of the respective line, or, if the name of the line has not been set, with the name of the source block. Additionally, the “:BlockIO” suffix is added.

BlockIO signals share a namespace. If there are naming conflicts, automatic renaming is performed. An underscore followed by letters or numbers is added to the signal name.
Example: “SignalX:BlockIO”, “SignalX_1:BlockIO”.

Parameter signals:

Parameter signals are generated with the name of the source block, followed by an underscore and the parameter name. Additionally, the “Param” suffix is added.

Parameter signals share a namespace. If there are naming conflicts, automatic renaming is performed. An underscore followed by letters or numbers is added to the signal name.
Example: “Constant_Value:Param”, “Constant_Value_1:Param”.

Parameters explicitly designated by the user, for example of the type “Simulink.Parameters”, are made available with the original names.

DState signals:

DState signals are generated with the name of the source block, followed by an underscore and the DState name. Additionally, the ":DState" suffix is added.

DState signals share a namespace. If there are naming conflicts, automatic renaming is performed. An underscore followed by letters or numbers is added to the signal name.
Example: "SignalX:DState", "SignalX_1:DState".

Effects on the GDS signal names after changes to the model

For BlockIO, parameter, and DState signals, it is ensured that the signals are always provided with the same signal names in the GDS when the model changes.
Example: A parameter signal name contains the name of the source block. If the source block is renamed, the signal name generated for it by the Simulink Coder changes, too. To prevent changes to the interface and, for example, required reconnection in PLCnext Engineer, it is ensured that the GDS signal name stays the same in spite of renaming.

Additionally, GDS signal names of deleted model parts cannot be reused.

The data required for this is saved to the "<model>.hashdata" file in the same directory as the MATLAB Simulink model (*.slx). You can delete this file to reset the GDS signal names.

5.6 Compiling the model

- Click on the button in the toolbar to compile the model in MATLAB Simulink.

The following files are generated during the compilation process, for example:

Executing the model with a PLCnext program

Table 5-8 Files generated for a PLCnext program

File name	Description	Storage location
<model>.pcwlx	PLCnext Engineer library: contains the model as a program	“<model>_PLCN” directory or user-defined output directory, see Section 5.2.2
<model>.hashdata	The information in this file ensures that the GDS signal names are neither changed nor used again, see Section 5.5.	Saved in the same directory as the MATLAB Simulink model (*.slx)
<model>.metalog.csv	Contains the information on which meta flags were set where, see Section 5.4. The meta flags themselves are saved in the “<model>.metadata” file in the same directory as the MATLAB Simulink model (*.slx).	“<model>_PLCN” directory

Please note:
It is not possible to generate the files individually.

6 Executing the model in PLCnext Engineer

This section describes how to integrate the model compiled in MATLAB Simulink into PLCnext Engineer and execute it on the controller using the generated PLCnext program.

For further information on the PLCnext Engineer software, please refer to the user manual for the controller used or to the quick-start guide for the PLCnext Engineer software (item no. 1046008).

The documents can be downloaded at phoenixcontact.net/products.

Requirements:

– You have created a project in PLCnext Engineer.

6.1 General use

6.1.1 Importing the generated library

Import the PLCnext Engineer library generated during the model compilation process into the PLCnext Engineer software. You can merge multiple libraries with different architectures (32 bit and 64 bit) into one library using the following command:

```
PLCN.Utils.mergePcwlx({['bin' filesep '32bit_arch' filesep
'<model>.pcwlx'], ['bin' filesep '64bit_arch' filesep '<model>.pcwlx']})
```

To import the generated library into PLCnext Engineer, proceed as follows:

- In the “COMPONENTS” area, open the “Libraries (x)” section.
- Right-click on “Libraries (x)”.
- In the context menu, select “Add User Library...”.

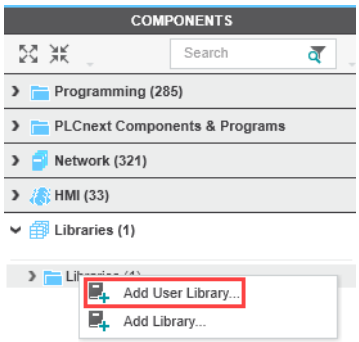


Figure 6-1 Importing the library

- In the file explorer, select the “<model>.pcwlx” file.
- Click the “Open” button.

6.1.2 Instantiating a program

Programs are instantiated in the “Tasks and Events” editor. To open the “Tasks and Events” editor, proceed as follows:

- In the “PLANT” area, double-click on the “PLCnext (x)” node.

The “/ PLCnext” editor group opens.

- Select the “Tasks and Events” editor.
- Drag and drop the “<model>Program” model program or your IEC 61131-3 program from the “COMPONENTS, Libraries (x)” area into the “Tasks and Events” editor.
- Drag and drop the “<model>Program” model program below the desired task (in Figure 6-2: “Cyclic100”).

⇒ The instance of the model program is displayed in the “PLANT” area below the “PLCnext (x)” node.

- If necessary, link the IN and OUT ports of the model to the IN and OUT ports of other programs in the “Port List” editor.

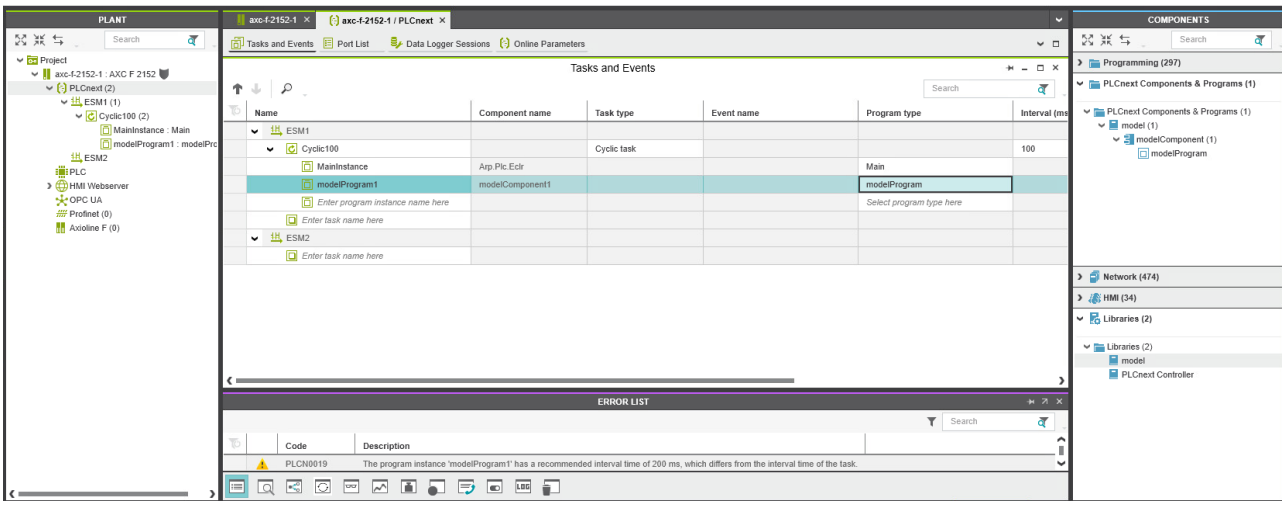


Figure 6-2 Example: instantiating the “modelProgram” model program

6.1.3 Transferring a project to the controller

To transfer the project to the controller, proceed as follows:

- In the “PLANT” area, double-click on the controller node.

The controller editor group opens.

- Select the “Cockpit” editor.
- Click on the button (“Write project to Controller and start execution. (F5)”).

⇒ The project is transferred to the controller and executed.

6.1.4 Executing several models simultaneously

Please note:

The number of models that can be executed simultaneously depends on the memory available on the runtime system. For additional information on the runtime system, please refer to the “Technical data” section in the user manual for the relevant controller.

6.2 Executing models as a PLCnext program

The imported library contains the model as a PLCnext program. The “<model>Program” PLCnext program is displayed in the “PLCnext Components and Programs (x)” section.

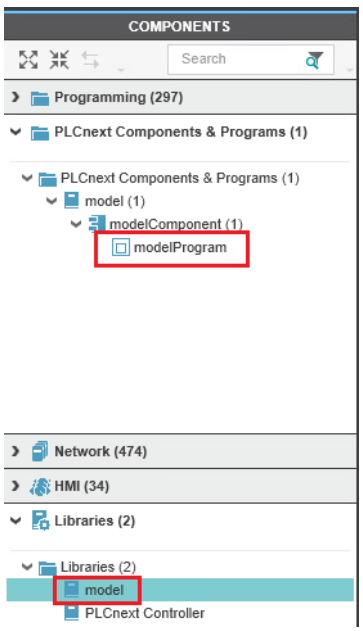


Figure 6-3 Imported library and model program in the “COMPONENTS” area

6.3 Executing models as a PLCnext model function block

The imported library contains the model as a PLCnext model function block. The “<model>Blk” PLCnext model function block is displayed in the “Programming” section.

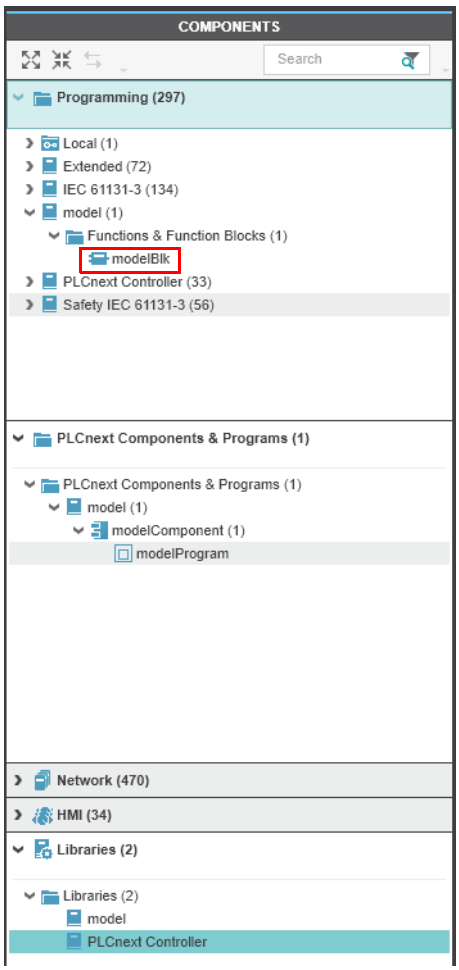


Figure 6-4 Imported library and model program in the “Programming” area

6.3.1 Integrating a function block into an IEC 61131-3 program

To be able to execute the PLCnext model function block, you must use the function block in the context of an IEC 61131-3 program.

- First, create a new program.
- Right-click on? “Add Program” in the “Programming, Local, Programs” area.
- Name the program and open it with a double-click.
- Select the type of the first worksheet (e.g., LD Code Worksheet).
- Drag the PLCnext model function block into the worksheet.
- Create new variables for the ports marked in red or connect the ports.
- Then follow the steps for instantiating a program in Section 6.1.2.

6.3.2 Executing models as a PLCnext subsystem function block library

The individual subsystems from the model subsystem function block library are each available as stand-alone IEC 61131-3 function blocks (“<LibraryName> <SubsystemName>”). You can use these in exactly the same way as model function blocks.

7 Architecture

7.1 Program control and program diagnostics

In addition to the signals arising from the Simulink model, a control inport and a diagnostic outport are attached to the PLCnext programs, depending on the model settings.

Both ports each form a “Struct” with various elements.

7.1.1 Control inport (default name: s_Ctrl)

xExecute

Controls the execution of the model (depending on the model setting PLCN_GenExecutePort).

The xExecute port is ignored if CoSimulation is enabled or if the model setting PLCN_WaitForStartPkg is used.

xInitialize

Controls the loading of the model (depending on the model settings PLCN_GenInitializePort and PLCN_StartInitialized).

7.1.2 Diagnostic outport (default name: s_Diag)

wDiagCode

Describes the general status of the program.
See Section “Diagnostics and troubleshooting”.

dwAddDiagCode

Extended information on program state 1.
See Section “Diagnostics and troubleshooting”.

dwAddDiagCode2

Extended information on program state 2.
See Section “Diagnostics and troubleshooting”.

dwChecksum

The interface checksum is formed using the root inports, root outports, and parameters, as well as their names and data types. This is the same as the checksum used for classic controllers and HLLI.

dwModelChecksum

This checksum is formed over the entire model, from the 4 model checksums provided by Simulink (MATLAB command: Simulink.BlockDiagram.getChecksum).

modelStateDescription

The state of the model, reason, and code count of the runtime error is provided via this string in case of an exception.
See Section “Diagnostics and troubleshooting”.

duration

Execution duration of the last executed program cycle in µs (depending on the model setting PLCN_GenDurationPort).

7.2 Control and diagnostics – model subsystem library function blocks

In addition to the signals resulting from the Simulink model, control inports and diagnostic outports are attached to the PLCnext subsystem library function blocks. This depends on the model settings.

7.2.1 Control inport

usiPriority

Controls whether the function block is executed synchronously or asynchronously. A value of 0 means “synchronous execution”. For a value between 1 and 99, the function block is executed asynchronously with the corresponding PLCnext Arp Thread priority.

7.2.2 Diagnostic outports

wDiagCode

Describes the general status of the program.
See Section “Diagnostics and troubleshooting”.

dwAddDiagCode

Extended information on program state 1.
See Section “Diagnostics and troubleshooting”.

7.3 Output compilation

The result of the build process in MATLAB Simulink is the <model>.pcwlx library that is read in and used by PLCnext Engineer. Along with various interface description files required by PLCnext Engineer, this contains the actual executable model code in the form of three dynamic C++ libraries (*.so).

7.3.1 PLCnext program interface library (<model>_plcn.so / lib<model>.so)

This library contains the PLCnext components, the PLCnext program(s), and in general the code necessary for PLCnext.

This library is always loaded, and just once. Because the “shared object” with the “RTLD_GLOBAL” flag is loaded, all exported icons defined here for the PLCnext process are visible globally.

7.3.2 Model library (<model>.so)

This library contains the actual model code.

Depending on the setting, the library is either loaded during the generation of the PLCnext program (for multitask, during generation of the first program) or, depending on the “xInitialize Port” of the program, loaded or unloaded. Here, each program instance loads its own copy of the model library. This “shared object” is also loaded with the “RTLD_GLOBAL” flag, but the compilation process ensures that most icons are not exported. As a result, icons are not visible globally.

If model references are used, a static library is generated for each submodel and also linked in the model library.

Furthermore, “shared utility code” (outsourced code), which is used by several models (main model or model reference) at the same time, can also be generated. This is also generated as a static library and linked in the model library.

7.3.3 Shared library (<model>_shared.so)

This library contains shared outsourced code.

7.4 Model multi-instancing

Due to the architecture of the two separate dynamic libraries, a PLCnext program can be multi-instanced by distributing it several times to the cores and tasks in PLCnext Engineer.

7.5 Exchanging models for PLC runtime

If the model setting “PLCN_GenInitializePort” is used, the “xInitialize” port in the program can be used to control whether the model library is loaded or unloaded. While the model library is unloaded, the “*.so” file in the controller can be exchanged directly while the remainder of the controller continues to run. After the exchange is complete, the model can be re-initialized and will then execute the changed model code.

This feature can be used if nothing is changed at the model interface (root ports, parameters, External Mode, etc.). Otherwise, the changed model library will not be loaded.

8 Diagnostics and troubleshooting

8.1 Diagnostic concept

The wDiagCode output contained in the diagnostic structure issues diagnostic codes regarding the model, i.e., all model programs.

In addition to the diagnostic code at the wDiagCode output, the dwAddDiagCode output can provide additional diagnostic information. The dwAddDiagCode2 output is not currently used and serves as a reserve for additional error information. In the event of an error, the diagnostic description and the error location are forwarded to the diagnostic structure via the modelStateDescription string. The diagnostic description is added to the log as an additional log entry.

The description for this can be found in Table 8-1 in the description for the two diagnostic codes.

8.2 Diagnostic codes PLCnext program and model function blocks

Table 8-1 Diagnostic codes for PLCnext programs and model function blocks

wDiagCode	dwAddDiagCode	Description
0000_{hex} 0_{dec}	–	Model is inactive
8100_{hex} 33024_{dec}	–	Model is initialized. The compiled model is loaded asynchronously into the controller RAM.
8200_{hex} 33280_{dec}	–	Model is ready for operation. The binary file (*.so) of the MATLAB/Simulink model has been loaded and the function block is waiting for the *xExecute* input.
	00000001_{hex} 1_{dec}	Model processing is blocked by the model configuration “Wait for start package”. You can only start model processing via the MATLAB Simulink interface.
	00000002_{hex} 2_{dec}	Model processing is blocked by the “CoSimulation”.
8300_{hex} 33536_{dec}	–	The compiled model is executed.
	00000002_{hex} 2_{dec}	The compiled model is executed as part of the “CoSimulation”.
8900_{hex} 35072_{dec}	–	The model is removed.
9100_{hex} 37120_{dec}	–	See 8100_{hex}, demo mode

Table 8-1 Diagnostic codes for PLCnext programs and model function blocks

wDiagCode	dwAddDiagCode	Description
9200_{hex} 37376_{dec}	–	See 8200_{hex}, demo mode
9300_{hex} 37632_{dec}	–	See 8300_{hex}, demo mode
9900_{hex} 39168_{dec}	–	See 8900_{hex}, demo mode
$C001_{hex}$ 49153_{dec}	–	The model library could not be loaded. Either it could not be found or the dynamic link was faulty. The precise error message is issued in the controller log as an error.
$C002_{hex}$ 49154_{dec}	–	The model library could not be loaded because the model interface is not as expected. See Section 7.5, “Exchanging models for PLC runtime”.
$C900_{hex}$ 51456_{dec}	–	The demo mode runtime was exceeded. The model is no longer processed; however, it remains in the controller RAM until the next reset.
$CF00_{hex}$ 52992_{dec}	[Exception code]	An exception occurred in the model. The dwAddDiagCode output displays the type of the exception, see Section 8.4, “Exception handling”.

8.3 Diagnostic codes PLCnext subsystem library function block

You can use the wDiagCode and dwAddDiagCode ports in PLCnext Engineer in decimal or hexadecimal format.

Table 8-2 Diagnostic codes for PLCnext subsystem library function block

wDiagCode	dwAddDiagCode	Description
0001_{hex} 1_{dec}	0	The function was started synchronously. The function has finished and the output can be used.
0010_{hex} 16_{dec}	0	The function was started asynchronously. The function has not finished and the output is not yet available.
0003_{hex} 3_{dec}	0	The function was started asynchronously in a previous cycle. The function has finished and the output can be used. The function was not restarted.
0030_{hex} 48_{dec}	0	The function was started synchronously in a previous cycle. The function has not finished and the output is not yet available. The function was not restarted.

Table 8-2 Diagnostic codes for PLCnext subsystem library function block

wDiagCode	dwAddDiagCode	Description
$CF00_{hex}$ 52992_{dec}	[Exception code]	An exception occurred in the model. The dwAddDiagCode output displays the type of the exception, see Section 8.4, “Exception handling”.

8.4 Exception handling

The behavior in the event of an error can be set via the model setting PLCN_ExceptionHandling. For PLCnext controllers, this can be used to capture floating point unit (FPU) exceptions and the SIGFPE signal as errors. An overview of the captured errors is given in table 8-3.

8.4.1 Options

With the “IGNORE” setting, errors are not captured. In the case of FPU calculations, the values “Inf” or “NaN” may be generated. In the case of integer calculations, an error signal is generated and sent to the controller, which responds by switching to the error state.

With the “CATCH_FPU” setting, only FPU exceptions are captured. If such an exception arises, the current model step is executed to completion, and the values “Inf” or “NaN” may be issued as the result of the exception. The model execution is then stopped and the respective error displayed at the diagnostic port. In the case of integer calculations, an error signal is generated and sent to the controller. This leads to the controller responding by switching to the error state.

With the “CATCH_ALL” setting, only FPU exceptions are captured and, if the respective hardware of the controller supports it, are upvalued to a SIGFPE signal. Without the upvalue function, the behavior with floating point calculations is the same as with the “CATCH_FPU” setting. With SIGFPE signals, the current model step is not continued. The model execution is stopped immediately and the respective error displayed at the diagnostic port. Because SIGFPE signals, unlike simple FPU exceptions, are triggered immediately, you can easily determine and inspect the location of the occurrence via debugging, using Eclipse for example.

8.4.2 Configuration of FPU exceptions

With active Exception Handling, only “FE_DIVBYZERO” and “FE_INVALID” are captured by default as FPU exceptions. The model setting? “PLCN_ExceptionHandlingFPETypes” can be used to change this behavior. This setting can contain the values “all”/“on” for all 5 FPU exceptions, “none”/“off” for no FPU exceptions, or any combination of the 5 FPU exception types listed in the “FPU exception” column, separated with the character “|”. For example, “FE_DIVBYZERO | FE_INVALID”.

8.4.3 Integer division by zero

For integer calculations, the code generation is generally performed very defensively by the Simulink Coder. In the case of divisions, a check is performed for divisor 0. In the case of S functions, this only applies if the creator wrote the code just as defensively. Otherwise, SIGFPE errors of the type “FPE_INTDIV” can occur.

8.4.4 Exception codes

Table 8-3 Extended exception codes for PLCnext programs

dwAddDiagCode	FPU exception	SIGFPE type	Description	Example	Default
$C000008E_{hex}$ 3221225614_{dec}	FE_DIVBYZERO	FPE_FLTDIV	A Division By Zero error occurred in an earlier floating point operation.	1.0/0.0 Typical result: Inf	CATCH_FPU CATCH_ALL
$C0000090_{hex}$ 3221225616_{dec}	FE_INVALID	FPE_FLTINV	Invalid operation in an earlier floating point operation.	sqrt(-1) 0.0/0.0 Typical result: NaN	CATCH_FPU CATCH_ALL
$C0000091_{hex}$ 3221225617_{dec}	FE_OVERFLOW	FPE_FLTOVF	The result of the earlier floating point operation was too large for it to be shown.	DBL_MAX*2.0 Typical result: Inf	-
$C0000093_{hex}$ 3221225619_{dec}	FE_UNDERFLOW	FPE_FLTUND	The result of the earlier floating point operation was subnormal with a loss of precision.	nextafter(DBL_MIN /pow(2.0,52),0.0) Typical result: 0.0	-
$C000008F_{hex}$ 3221225615_{dec}	FE_INEXACT	FPE_FLTRES	Imprecise result: Rounding was necessary to ensure that the result of an earlier floating point operation could be saved.	1.0/10.0 Typical result: 0.1	-
$C0000094_{hex}$ 3221225620_{dec}		FPE_INTDIV	Integer division by zero.	1/0	CATCH_ALL
$C0000095_{hex}$ 3221225621_{dec}		FPE_INTOVF	Integer overflow.	-	CATCH_ALL
$C0000096_{hex}$ 3221225622_{dec}		FPE_FLTSUB	Subscript is out of range.	-	CATCH_ALL
$C0000097_{hex}$ 3221225623_{dec}		FPE_FLTUNK	Undiagnosed floating point exception.	-	CATCH_ALL
$C0000098_{hex}$ 3221225624_{dec}		FPE_CONDTRAP	Trap condition	-	CATCH_ALL
$C0000099_{hex}$ 3221225625_{dec}		(FPE signal type unknown)	Integer division by zero.	-	CATCH_ALL

9 Extending the Simulink Library Browser

After PLCnext Target for Simulink has been installed, various new blocks are available to the user in the “Simulink Library Browser” (see figure 9-1).

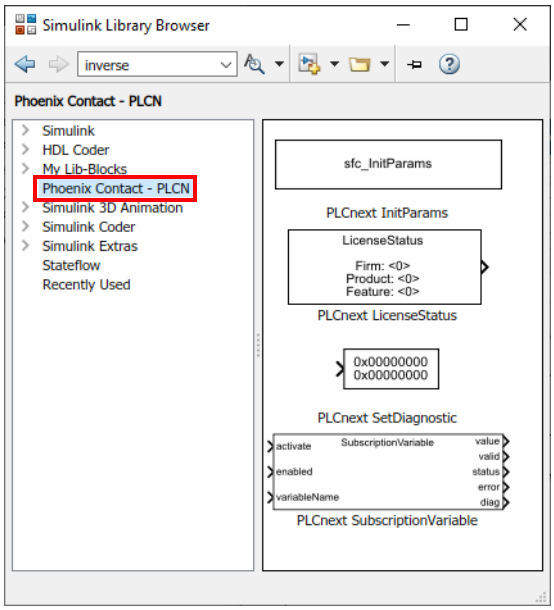


Figure 9-1 “Phoenix Contact - PLCN” Simulink Library Browser

9.1 PLCnext InitParams

With the PLCnext InitParams block, model parameters that were previously marked as static can be read out of a configuration file in the initialization phase of the model. They do not have to be retransmitted cyclically via the model ports. This enables you to configure the model flexibly and efficiently via the configuration file.

9.1.1 Function block

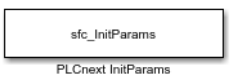


Figure 9-2 “PLCnext InitParams” function block

9.1.2 Configuration

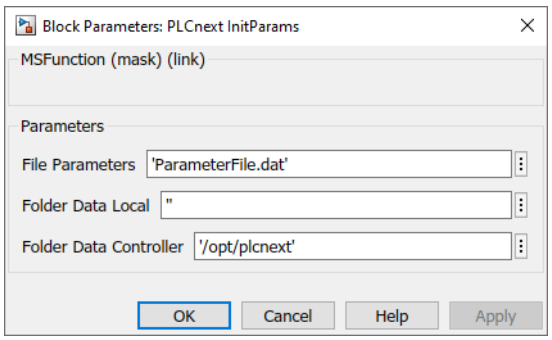


Figure 9-3 “PLCnext LicenceStatus” configuration dialog

“Folder Data Local” is currently not used. The path of the parameter data file is made up of “Folder Data Controller” and “File Parameters” on the controller.

9.1.3 Technical details

See Section “Initializing model parameters from a configuration file” on page 61.

9.2 PLCnext LicenceStatus

With the PLCnext LicenseStatus block, the license for an app from the PLCnext Store can be verified.

Thus, it is possible to bind the execution or the behavior of a model generated in Simulink to a license installed on the target controller. The license can be sold through the PLCnext Store.

For further information, see also https://store.plcnext.help/st/home.htm.

9.2.1 Function block

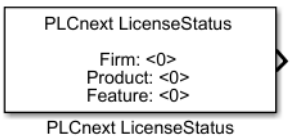


Figure 9-4 “PLCnext LicenceStatus” function block

9.2.1.1 Output parameters

Table 9-1 Output parameters

Parameter	Type	Description	
valid	BOOL	TRUE:	The respective license is available in the controller and is valid.

9.2.2 Configuration

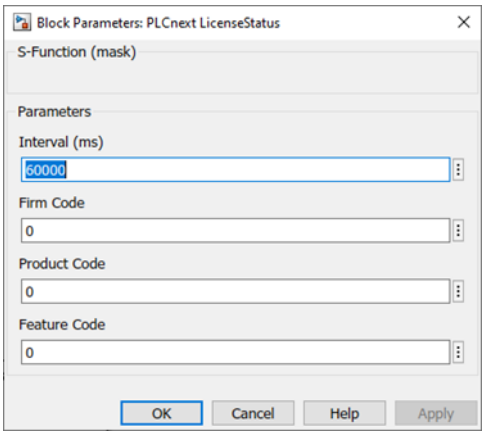


Figure 9-5 “PLCnext LicenceStatus” configuration dialog

Table 9-2 Description of the settings in the “PLCnext LicenceStatus” configuration dialog

Setting	Description
Interval (ms)	Interval in which the license is queried during model execution. This must be between 600000 and UINT32_MAX.
Firm Code	Firm code of the license to be queried.
Product Code	Product code of the license to be queried.
Feature Code	Feature code of the license to be queried.

9.2.3 Technical details

– All licenses within a model are queried at the same time; the minimum value set in all of the “LicenseStatus” blocks is used as the interval.
– The query is made via the “Arp::System::Lm::Services::ILicenseStatusService” Services”.
– The query is made asynchronously via a separate low-priority thread.

9.3 PLCnext SubscriptionVariable

With a PLCnext SubscriptionVariable block, a GDS variable can be read from or written to the controller cyclically.

In contrast to the ports, the assignment to PLCnext variables is made within Simulink and not within the PLCnext Engineer.

Please note:
The use of “IN Ports” is recommended for exchanging data with the controller.

The SubscriptionVariable block has a considerable overhead and a different time behavior, in particular when registering or changing the subscription (activating or deactivating SubscriptionVariable blocks or changing the linked variable).

9.3.1 Function block to read out a GDS variable

Figure 9-6 PLCnext SubscriptionVariable Read function block

Table 9-3 Input parameters

Parameter	Type	Description	
activate	BOOL	With a rising edge, the model is updated with the actual status.	
enabled	BOOL	TRUE:	The variable should be polled.
variableName	STRING	Optional input with which the variable name registered for the runtime can be changed.	

Table 9-4 Output parameters

Parameter	Type	Description	
value	Depending on the setting	Value of the variables	
valid	BOOL	TRUE:	The variable is currently registered.

Table 9-4 Output parameters

Parameter	Type	Description	
status	UINT8	Status of the block.	
		2 5 6 7 8 all other values	Unregistered Registered ErrorUnsubscribe ErrorSubscribe ErrorRead Invalid
error	BOOL	TRUE:	An error has occurred.
diag	UINT8	PLCnext error code when error=true. Contains the PLCnext error code “Arp::Plc::Gds::Services::DataAccessError”	

9.3.2 Function block to write a GDS variable

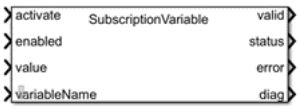


Figure 9-7 PLCnext SubscriptionVariable Write function block

Table 9-5 Input parameters

Parameter	Type	Description	
activate	BOOL	With a rising edge, the model is updated with the actual status.	
enabled	BOOL	TRUE:	The variable should be polled.
value	Depending on the setting	Value that is written to the GDS variable	
variableName	STRING	Optional input with which the variable name registered for the runtime can be changed.	

Table 9-6 Output parameters

Parameter	Type	Description	
valid	BOOL	TRUE:	The variable is currently registered.

Table 9-6 Output parameters

Parameter	Type	Description	
status	UINT8	Status of the block.	
		2 5 6 7 8 all other values	Unregistered Registered ErrorUnsubscribe ErrorSubscribe ErrorRead Invalid
error	BOOL	TRUE:	An error has occurred.
diag	UINT8	PLCnext error code when error=true. Contains the PLCnext error code “Arp::Plc::Gds::Services::DataAccessError”	

9.3.3 Configuration

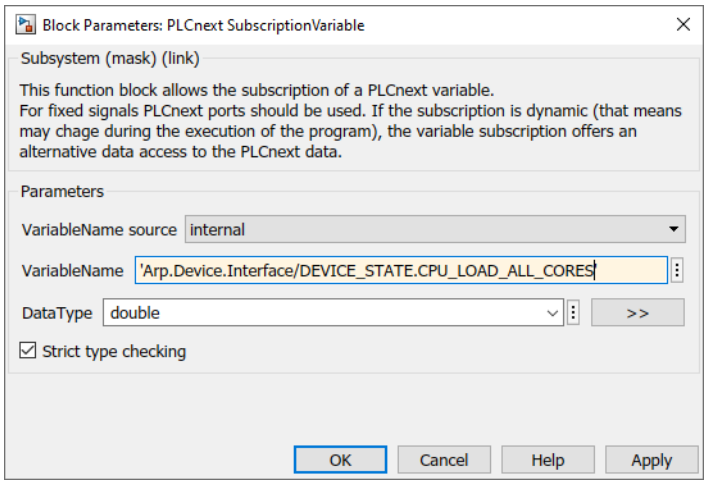


Figure 9-8 “PLCnext SubscriptionVariable” configuration dialog

Table 9-7 Description of the settings in the “PLCnext SubscriptionVariable” configuration dialog

Setting	Description
VariableName source	The variable name can be set via the block input or permanently defined in the block. With the “external” option, the “VariableName” input field is hidden and the “variableName” block input shown.
VariableName	Name of the PLCnext variable whose value is to be read out. The input field is only active if “VariableName source” has been set to “internal”.

Table 9-7 Description of the settings in the “PLCnext SubscriptionVariable” configuration dialog

Setting	Description
DataType	Data type of the “value” output.
Strict type checking	If the data type of the variable specified does not match the data type of the block exactly, the block is switched to the error state: error=true status=8 (ErrorRead) diag= (TypeMismatch) If “Strict type checking” is not enabled, the block only switches to the error state if the data types cannot be mapped onto each other. Integer values can be mapped onto floating point values, but not the other way around. Unsigned values can be mapped onto signed values, as long as their range can accommodate the possible input values.

9.3.4 Technical details

- Variables are managed in a joint subscription per task (for performance reasons).
- With multi-rate models, an individual subscription is used for each rate which bundles all SubscriptionVariable blocks of this rate and queries them together.
- A HighPerformance subscription is used: The value is read at the end of the ESM task in which the Simulink program is embedded, and is then available for the next cycle. The consistency of the read data is ensured.
- The same variable cannot be included in a subscription more than once. In this case, one of the “SubscriptionVariables” is set to the error state.
- If the block switches to an error state, the “value” value is not changed; rather, it remains at the last valid value.

10 Initializing model parameters from a configuration file

During compilation, an attempt is made to read in the “<ModelName>_StaticParameters.txt” or “StaticParameters.txt” with this prioritization. This file must be located in the same folder as the model file. Parameter names in the file must be separated by spaces. The parameter names can refer to all named parameters (normal MATLAB variables, “Simulink.Parameter” objects or “slexpr”). If such a definition list is used in connection with the InitParams S function, the listed parameters are not generated with “GDS + Engineer” visibility, but rather as “Invisible GDS Ports” at most. The parameters therefore cannot be interconnected. However, they are still available as signals for the Model Viewer or OPCUA. In addition, a map is generated as the interface code, which can be used by the S function code. The map contains the actually generated global parameters with the name as the key, the pointer, the size, and the PLCnext data type as values.

If the InitParams S function is available in the model, the model loads static parameters from a file on the controller in the initialization phase and overwrites the model parameters with this data. Various error cases are captured when reading the file on the controller and writing the parameters.
Hard error: The message is written to the controller's log as an “Error Log”, and writing of all parameters is aborted.
Soft error: The message is written to the controller's log as an “Error Log”, and only the corresponding parameter is not written.
Error cases:

- File not found. Hard error.
- CRC mismatch. The CRC contained in the file does not match the CRC of the file content (apart from the CRC). Hard error.
- The size of the data stored in the file is too large or too small. Not actually possible. Hard error.
- The data type of a parameter does not match the expected type. This can happen in particular if the Auto data type is used in Simulink. Soft error.
- The data size of a parameter does not match the expected size. Not actually possible. Soft error.

With the “PLCN.InitParams.createModelParameterData (ModelName, PathOutdata)” function, parameter data can be generated from a model for which a “StaticParameters.txt” is available. You then have to manually transfer the parameter data to the controller. Here, all parameters in the list are evaluated according to their type.

If a parameter cannot be evaluated, the process is interrupted. If the parameter is nevertheless used in the model, the model cannot be built.

For parameters of the type “auto”, a warning is issued, stating that it is better to use a specific type. Here, double is assumed for the data. This depends on the model configuration and is determined dynamically during code generation.

A warning is issued for parameters of the type “Struct”. The parameter is skipped, because these are not supported.

The file also contains the sizes for the parameter values saved as binary data as well as the respective PLCnext data type numbers and the individual data sizes per parameter name.

In addition, a checksum is formed for everything except for the checksum itself at the beginning of the file (CRC32).

11 External Mode

External Mode enables you to access the model in MATLAB Simulink while it is being executed on the controller.

Please note:
External Mode is only available for a complete model (“Build Model” function in MATLAB Simulink). External Mode is not available for a subsystem.

11.1 Configuring the model in MATLAB Simulink for External Mode

Configuring with MATLAB command

Rather than configuring manually, we recommend using the MATLAB command “PLCN.ExtMode.setupExtModeArgs”. This command is called up with the model name, IP, port, and instance path as parameters. The instance path parameter is optional.

Example:

```
PLCN.ExtMode.setupExtModeArgs('model','192.168.1.10',17725,' modelComponent1/modelProgram1')
```

Manual configuration

- In the toolbar, click on the button.

The “Configuration Parameters” window opens.

- In the navigation list on the left, click on the “Interface” entry.
- Enable the “External mode” check box (see Figure 11-1).
- Enter the following values in inverted commas in the “MEX-file arguments” input field:
  - IP address of the controller.
  - “Verbosity” for the External Mode log in Simulink, either “0” or “1”.
    “0” stands for default and “1” for verbose, which means more log messages are displayed when External Mode is enabled in Simulink.
  - Port for the connection as a number.
  - Timeout intervals for connections as a number. The default value is “5”.
  - Instance path of the model with which the connection is to be established.

  Example: 192.168.1.10',1,17725,5,'simpleCounterComponent1/simpleCounterProgram1'

Please note:
You must enter the IP address. The values “Verbosity”, “Port”, “Timeout interval”, and “Instance path” of the model are optional.

- Click on the “Apply” button to apply your entries.

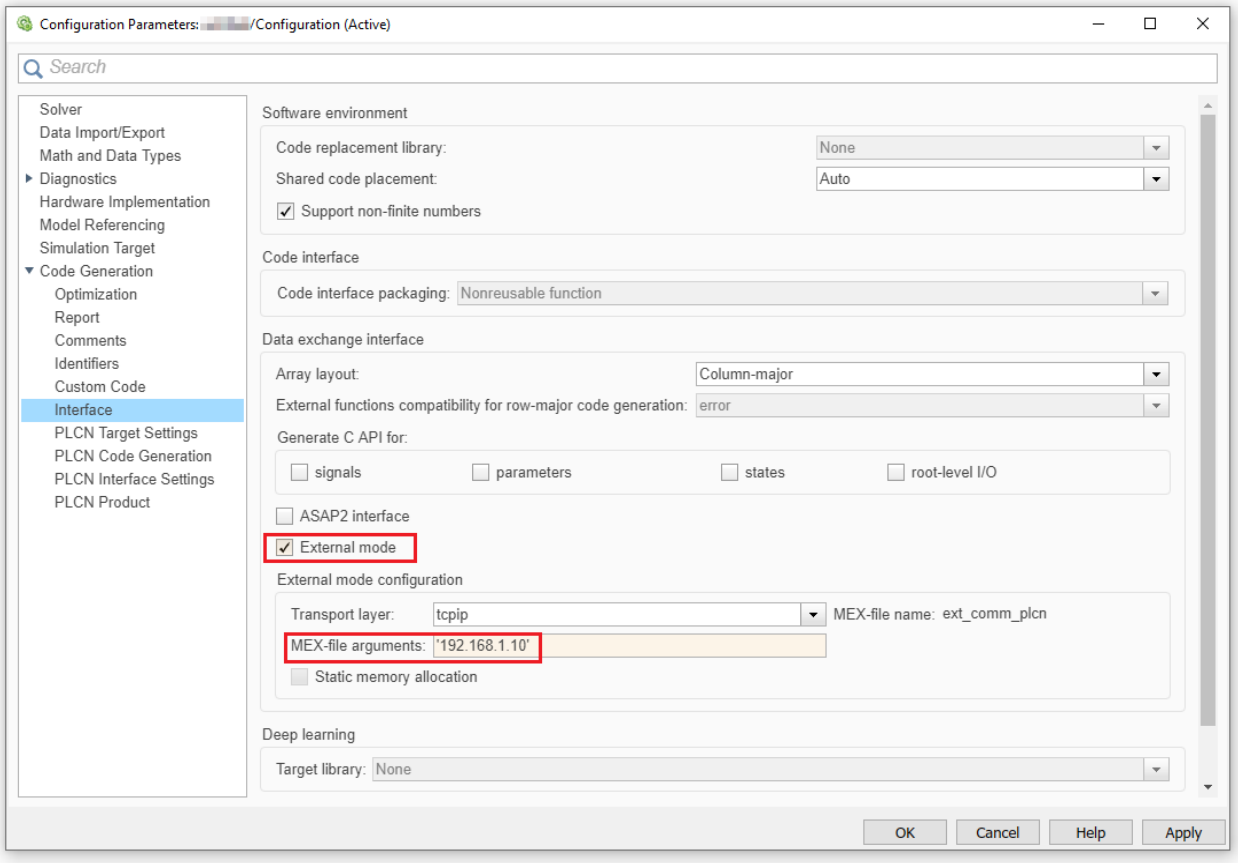


Figure 11-1 “Interface” configuration page

11.2 Controlling the model execution via Simulink

For executing External Mode, you have the option of starting and stopping model execution via the buttons in the MATLAB Simulink interface. To do so, you must enable the “Wait for start package” setting. To do this, proceed as follows:

Enabling the “Wait for start package” setting

In the toolbar, click on the button.

- Click on the “PLC Code Generation” entry in the “Select:” area of the “Configuration Parameters” window (see Section 5.2.2).
- Enable the “Wait for start package” check box.

This prevents controlling the model execution via the “xExecute” port. You can now start and stop model execution exclusively via the buttons in the MATLAB Simulink interface.

11.3 Executing the model in External Mode

Requirements:

– You have selected “External” simulation mode in the MATLAB Simulink toolbar.
– You have configured the model for External Mode (see Section 11.1).
– You have built the model for External Mode, either via the “rtwbuild” command or via the “Build for Monitoring” command in the toolbar.

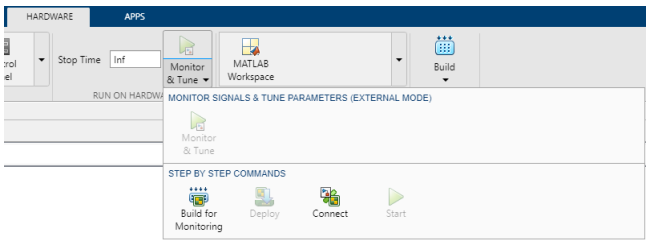


Figure 11-2 External Mode “Build for Monitoring”

Proceed as follows:

- To integrate and execute the model in PLCnext Engineer, proceed as described in Section 6.

⇒ Model access is available via External Mode.

You have the option to establish or disconnect the connection to the controller via the buttons in MATLAB Simulink (see Table 11-1).

Please note:

Apart from the port used, you can still adjust all External Mode settings after compiling. For example, the IP address of the controller and the instance path of the target program.

11.4 Using External Mode

To start or stop model execution in External Mode, proceed as follows:

Starting model execution in MATLAB Simulink

1. Via the buttons in MATLAB Simulink:
   - Use the buttons in MATLAB Simulink as shown in Table 11-1.

The buttons are located on the “Hardware” tab.

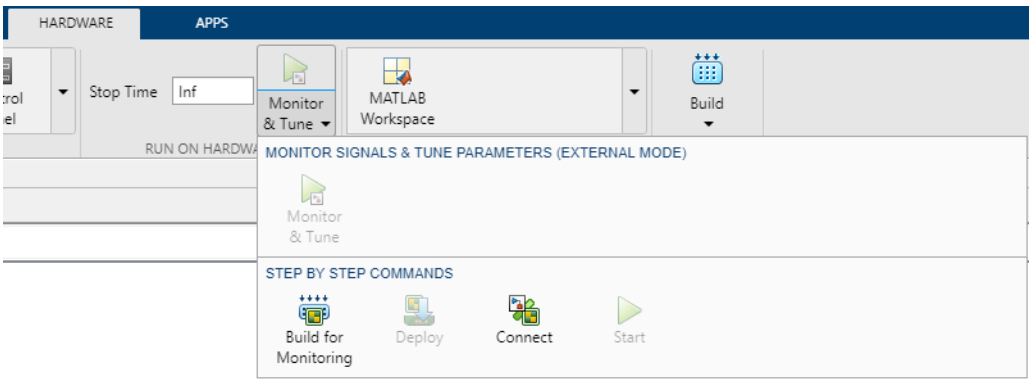


Figure 11-3 External Mode “Monitor & Tune”

Table 11-1 Buttons on the “Hardware - Monitor & Tune” tab in MATLAB Simulink

Button	Function
Connect	Establish connection to the controller. **WARNING: Risk of personal injury and property damage due to altered model behavior** When the connection to the controller is established, the parameters from MATLAB Simulink are applied. If parameter changes and adjustments have been made in PLCnext Engineer, these are lost. This can result in an altered behavior of the model. • Make sure that your system is in a safe state at all times.
Disconnect	Disconnect the connection to the controller. The model remains in the last state it had.
Start	Start model execution.
Stop	Stop model execution. For PLCnext programs, this stops the model. If the connection to the controller is disconnected in Simulink and re-established, you can restart the execution and continue at the same point.

There are two options for parameterization (“Parameter tuning”) in External Mode:

Parameterization in MATLAB Simulink

1. In MATLAB Simulink: via the “External Mode Control Panel” window (see Figure 11-4).
   - To open the window, click on “Code, External Mode Control Panel” in the MATLAB Simulink menu bar.

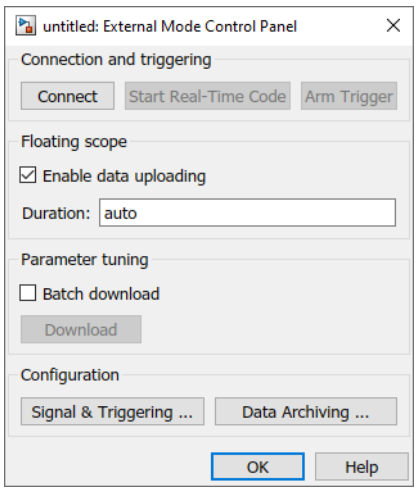


Figure 11-4 “External Mode Control Panel” window in MATLAB Simulink

Please note:
If you have enabled the “Inline Parameters” check box on the “Optimization, Signals and Parameters” configuration page, the input fields in the block masks are grayed out. In this case, proceed as follows:

- Change the parameters via the MATLAB command line.
- To transfer the parameters to the controller, click on the “Download” button in the “External Mode Control Panel” window.

11.4.1 Connecting with multi-instanced models

Generally, you can establish one External Mode connection per port. Even if several models have been configured with the same port, only one model can use External Mode at a time. If a model with multiple instances is running on the controller, you can set the instance with which the connection is required via the instance path parameter when configuring External Mode.

You can check the instance path of the running programs in PLCnext Engineer.

11.4.2 Authentication and data transmission

Using the MATLAB command “PLCN.ExtMode.setExtModePassword(modelName,password)” you can generate a “<model>_ExtMode_Key.bin” file that serves as the key for the “External Mode” connection.

If this file is stored in the model folder during compilation, a key is requested for establishing a connection to the “External Mode” server in the controller. This means that a Simulink connection can only be established if the key file is in the model folder.

For security reasons, make sure that the key is present in the key file as well as in the “ExtModeFunctions.cpp” generated during code generation and that it can also be read from there.

11.4.3 Replacing referenced models with subsystems

External Mode in MATLAB Simulink does not support referenced models. Using the “PLCN.Utils.convertRefMdls2Subsys(modelName,newModelName)” MATLAB command replaces all referenced models with subsystems in the main model. The referenced models can then be converted to new models as subsystems.

12 Controlling the controller remotely

PLCnext Target for Simulink provides functions for automating control via MATLAB.

This relates to starting and stopping, file transfers, shell access as well as reading out variable values and querying messages (notifications).

The central starting point is the creation of a “DeviceConnection”, which can then be used to access the further functions.

12.1 PLCN.Remote.DeviceConnection

The connection to a PLCnext controller is established by generating a DeviceConnection instance.

Example:

```
deviceConnection = PLCN.Remote.DeviceConnection('admin', 'password',
'192.168.1.10');
```

Generating the DeviceConnection does not automatically lead to establishing the connection. If the controller is offline or not available, the connection is not established.

The actual connection is always established on-demand, that is to say when a connection to the controller is needed for accessing a method or property.

If the DeviceConnection object is deleted in MATLAB, the connection is terminated automatically.

12.1.1 Methods

Constructor

```
obj = PLCN.Remote.DeviceConnection(userName, password, ip[, port] [, con-
nectionTimeOut] [, requestTimeOut])
```

Arguments:

userName (char[]): User name for the connection (e.g., “admin”).

password (char[]): Password of the specified user.

ip (char[]): Controller IP address.

port (int): Controller port. If not specified, the default port 41100 is used.

connectionTimeOut (int): TimeOut in seconds for connection establishment. If not specified, 10 seconds are used.

requestTimeOut (int): TimeOut in seconds for the communication. If not specified, 10 seconds are used.

Connect

```
connect()
```

Forces the connection to be established once and triggers an error if this fails. This can be used to determine the point in time for connection establishment. If connect is not called up, the connection is established automatically as soon as a property or method is accessed that needs this connection.

Reconnect

```
reconnect([timeOut])
```

Repeatedly attempts to establish the connection to the controller until it is established or until the timeout elapses.

This is useful, for example, if the controller is restarted and it is unknown when the actual connection can be established.

Arguments:

timeOut (int): Period of time during which establishing a connection is attempted. Default value: 150 s

isConnected

```
[connected, message] = isConnected()
```

Returns the actual connection state.

Return values:

connected (bool): True if the controller is connected.

message (char[]): Additional information on the reason for the actual connection state.

12.1.2 Properties

Device

The Device property enables access to the general properties and methods of the controller.

See also “PLCN.Remote.Device”.

Service

The Service property enables access to the PLCnext service of the controller.

See also “PLCN.Remote.Service”.

FileSystem

The FileSystem property enables access to the file system of the controller.

Shell

The Shell property enables SSH/SCP access to the controller.

See also “PLCN.Remote.Service”.

Notifications

The Notifications property enables access to the messages of the controller.

See also “PLCN.Remote.Notifications”.

GlobalDataSpace

The GlobalDataSpace property enables access to the variable values of the controller.

See also “PLCN.Remote.GlobalDataSpace”.

12.2 PLCN.Remote.Device

Using the device class functions, you can start and stop the controller, and request information on the runtime and hardware.

12.2.1 Methods

stop

```
stop()
```

Stops the controller.

startCold

```
startCold()
```

Loads the settings of the controller and performs a cold restart. All data is reset.

startWarm

```
startWarm()
```

Loads the settings of the controller and performs a warm start. During a warm start, all data is reset and all remanent data restored (system start).

startHot

```
startHot()
```

Performs a hot start. During a hot start, data is neither reset nor restored. All variable values are retained.

restart

```
restart([isAsync],[timeOut])
```

Reboots the device.

Arguments:

isAsync (bool): If true, the further execution of the MATLAB program is continued immediately upon rebooting. If false, the execution is stopped until the controller is rebooted. Default value: false.

timeOut (int): TimeOut in seconds for starting the controller when isAsync is set to false. Default value: 150 s.

getPlcState

```
state = getPlcState()
```

Returns the current runtime state of the controller.

Return values:

Structure with elements:

IsRunning (bool): True if the controller is running.

Description (char[]): Text description of the controller state.

getDeviceState

```
state = getDeviceState()
```

Returns the current state of the hardware.

Return values:

Structure with elements:

boardTemperature (int): Temperature of the main board in °C.

boardHumidity (int): Humidity of the main board as a percentage.

ledRun (bool): True if the “Run” LED is on.

ledFail (bool): True if the “Fail” LED is on.

cpuTotalLoad (int): Current processor load as a percentage.

getDeviceInfo

```
info = getDeviceInfo()
```

Returns information on the hardware and software.

Return values:

Structure with elements:

serialNumber (char[]): Serial number

hardwareVersion (char[]): Hardware version number.

fpgaVersion (char[]): FPGA version number.

firmwareVersion (char[]): Firmware version number.

12.3 PLCN.Remote.Service

Using the service class functions, you can start and stop the PLCnext service on the controller.

12.3.1 Methods

isRunning

```
value = isRunning()
```

Returns information on the hardware and software.

Return values:

bool: True if the PLCnext service on the controller is running.

stop

```
stop()
```

Stops the PLCnext service.

restart

```
restart([isAsync], [timeOut])
```

Restarts the PLCnext service.

Arguments:

isAsync (bool): When true, the further execution of the MATLAB program is continued immediately upon triggering the restart. When false, the execution is stopped until the PLCnext service is rebooted and responds to queries.
Default value: false.

timeOut (int): TimeOut in seconds for restarting the PLCnext service when isAsync is set to false.
Default value: 150 s.

12.4 PLCN.Remote.FileSystem

Using the FileSystem class functions, you can access the file system of the controller and also exchange files between the controller and the local computer.

Please note that the file and directory paths have to be specified correctly for the respective environments.

For paths on the controller in Linux format this means: Directory separator '/'.

12.4.1 Methods

dir

```
entries = dir(path, [pattern], [isRecursive])
```

Lists the files and directories on the controller.

Arguments:

path (char[]): Path on the controller (absolute or relative to the current directory).

pattern (char[]): Search pattern (default value: '*').

isRecursive (bool): True if the subdirectories are to be searched recursively.

Return values:

entries(struct[]): File data system with the following fields:
- **path** (char[]): Complete file path including file names.
- **isdir** (bool): True if it is a directory, false if it is a file.
- **folder** (char[]): Directory that contains the file or directory.
- **name** (char[]): Name of the file or directory.

pwd

```
path = pwd()
```

Returns the current controller path. In the other methods, some relative path specifications can be specified. These are resolved relative to this path.

Return values:

path (char[]): Current controller path.

cd

```
cd(path)
```

Switches the current controller path.

Example:

```
deviceConnection.FileSystem.cd('/opt');
```

Arguments:

path (char[]): Path on the controller (absolute or relative to the current directory).

isfolder

```
result = isfolder(path)
```

Tests the current path.

Arguments:

path (char[]): Path on the controller (absolute or relative to the current directory).

Return values:

result (bool): True if the path specified references a directory, otherwise false.

isfile

```
result = isfile(path)
```

Tests the current path.

Arguments:

path (char[]): Path on the controller (absolute or relative to the current directory).

Return values:

result (bool): True if the path specified references a file, otherwise false.

download

```
download(devicePath, targetPath, [overwrite])
```

Transmits a file or a complete directory from the controller to the local computer.

Arguments:

devicePath (char[]): File or directory path on the controller (absolute or relative to the current controller directory).

targetPath (char[]): File or directory path on the local computer (absolute or relative to the current directory of the current computer).

overwrite (bool): When true, the files existing on the controller are overwritten. When false, interrupted with an error message.

upload

```
upload(sourcePath, devicePath, [overwrite])
```

Transmits a file or a complete directory from the local computer to the controller.

Arguments:

sourcePath (char[]): File or directory path on the local computer (absolute or relative to the current directory of the current computer).

devicePath (char[]): File or directory path on the controller (absolute or relative to the current controller directory).

overwrite (bool): When true, the files existing on the local computer are overwritten. When false, interrupted with an error message.

movefile

```
movefile(source, destination, [overwrite])
```

Moves a file or complete directory within the controller.

Arguments:

source (char[]): Source path (file or directory) on the controller (absolute or relative to the current directory of the current controller).

destination (char[]): Target path (file or directory) on the controller (absolute or relative to the current directory of the controller).

overwrite (bool): When true, the existing files are overwritten. When false, interrupted with an error message.

mkdir

```
mkdir(devicePath)
```

Creates a new directory on the controller.

Arguments:

devicePath (char[]): Directory path on the controller (absolute or relative to the current directory of the current controller).

rmdir

```
rmdir(devicePath)
```

Deletes a directory on the controller.

Arguments:

devicePath (char[]): Directory path on the controller (absolute or relative to the current directory of the current controller).

delete

```
delete(devicePath)
```

Deletes a file on the controller.

Arguments:

devicePath (char[]): File path on the controller (absolute or relative to the current directory of the current controller).

12.4.2 Static methods

fullfile

```
path = fullfile(path1, [path2], ... [pathN])
```

Combines partial paths into a complete path on the controller using the necessary directory separators. The path is generated as a LINUX path, that means the character '/' is used as separator.

Example:

```
path = PLCN.Remote.FileSystem.fullfile('/opt', 'plcnext', 'logs');
```

12.5 PLCN.Remote.Shell

Using the service class functions, you can realize SSH/SCP access to the controller. A configured shell instance is made available via each DeviceConnection via the Shell property.

For other purposes, you can establish an SSH connection to any communication partner via the constructor. Access is realized via PuTTY.

Before the first connection establishment or when the SSH host key of a device changes (e.g., due to a firmware update), you must register this key in the local management system. You can use the updatehostkey method for this purpose.

12.5.1 Methods

Constructor

```
obj = PLCN.Remote.Shell(userName, password, hostname[, port])
```

Arguments:

userName (char[]): User name for the connection (e.g., “admin”).

password (char[]): Password of the specified user.

hostname (char[]): Controller IP address or host name.

port (int): Controller SSH port; if not specified, the default port 22 is used.

execute

```
[status, result] = execute(cmd)
```

Executes a shell command on the controller.

Arguments:

cmd (char[]): Command that is to be executed via SSH on the controller

Return values:

status (bool): ExitCode from PuTTY.

result(char[]): Text output of the command.

getfile

```
[status, result] = getfile(source, destination)
```

Loads a file from the controller to the local computer.

Arguments:

source (char[]): File path on the controller.

destination (char[]): Target path in the local system.

Return values:

status (bool): ExitCode from PuTTY.

result(char[]): Text output of the command.

putfile

```
[status, result] = putfile(source, destination)
```

Loads a file from the local computer to the controller.

Arguments:

source (char[]): File path in the local system.

destination (char[]): Target path on the controller.

Return values:

status (bool): ExitCode from PuTTY.

result(char[]): Text output of the command.

updatehostkey

```
updatehostkey()
```

Updates the management of the known SSH host on the local computer with the SSH host key that is returned by the connection.

This is an interactive command. Upon execution, the user must confirm the returned key. This is an SSH security feature.

12.5.2 Updating the SSH host key

If an SSH connection to the device is established for the first time, the SSH host key must be registered in the local SSH management system. The updatehostkey method of the shell object can be used for this.

In special cases, the host key can be updated automatically in the background. This can be enabled via execution of the following function:

```
PLCN.Remote.Settings.setAutoConfirmSshHostKeys(true);
```

Familiarize yourself with the security risks and only use this option in trusted network environments.

12.6 PLCN.Remote.GlobalDataSpace

Using the GlobalDataSpace class functions, you can read out the content of the variables in the controller.

12.6.1 Methods

createSubscription

```
subscription = createSubscription(variableNames)
```

Generates a GDS subscription that can read out a list of variable values. Access is via “DirectRead”, which means the variables are accessed directly. The consistency is not guaranteed.

Arguments:

variableNames (string[]): List of variable names.

Return value:

subscription (PLCN.Remote.GDSSubscription): Subscription object with which the variables can be accessed.

12.7 PLCN.Remote.GDSSubscription

Using the GDSSubscription class, you can access the variable content of the subscription. The subscription is automatically terminated when the GDSSubscription object is deleted in MATLAB.

12.7.1 Methods

readValues

```
values = readValues(obj)
```

Reads out the values of the subscription variables.

Return value:

values (cell{}): Cell array with all variable values in the same sequence as that specified during generation of the subscription.

12.8 PLCN.Remote.Notifications

With the notification class functions, you gain access to the controller messages.

12.8.1 Methods

Constructor

```
obj = PLCN.Remote.Shell(userName, password, hostname[, port])
```

Arguments:

userName (char[]): User name for the connection (e.g., “admin”).

password (char[]): Password of the specified user.

hostname (char[]): Controller IP address or host name.

port (int): Controller SSH port; if not specified, the default port 22 is used.

GetAllKnownNotificationNames

```
names = GetAllKnownNotificationNames()
```

Return value:

names (cell{}): Cell array with the names of all known notifications.

queryNotifications

```
notifications = queryNotifications(numberOfMessages, [filter], [ascending])
```

Reads out all messages that meet the specified criteria.

Arguments:

numberOfMessages (int): Maximum number of messages to be read out.

filter (struct): Filter settings with the supported properties:
- **notificationNameRegExp** (char[]): Regular output with which the name of the message can be filtered.
- **timestampAfter** (DateTime): If set, only messages that are older than the specified time stamp are read out.
- **timestampBefore** (DateTime): If set, only messages that are more recent than the specified time stamp are read out.

If several filter properties are set, these are linked with AND logic.

ascending (bool): Determines the order of the messages read out. If true, the oldest messages are returned first; if false, the most recent.

Default value: false

Return value:

notifications (struct[]): Messages read out in a struct array with the following properties:
- **id** (char[]): Message ID
- **name** (char[]): Message name
- **severity** (char[]): Severity ('INTERNAL','INFO','WARNING','ERROR','CRITICAL','FATAL').
- **timestamp** (DateTime): Message time stamp
- **payload** (cell{}): Cell array with the payload data of the respective message (depending on message type)

createSubscription

```
subscription = createSubscription([filter])
```

Generates a message subscription that simplifies polling access to the messages of the controller.

The subscription is created and can then be polled periodically. It then returns all messages that have accumulated since the time it was created and that match the corresponding filter properties.

Arguments:

filter (struct): Filter settings with the supported properties:
notificationNameRegExp (char[]): Regular output with which the name of the message can be filtered.

Return value:

subscription (PLCN.Remote.NotificationSubscription): Subscription object with which the messages can be accessed.

12.9 PLCN.Remote.NotificationSubscription

With the NotificationSubscription class functions, you have polling access to the controller messages.

12.9.1 Methods

getLastNotification

```
notification = getLastNotification()
```

Returns the latest not-yet-returned message that matches the filter set when generating the subscription.

Return value:

notification (Struct):
Message Struct with the contents described in PLCN.Remote.Notification.queryNotifications. Empty if no matching messages exist.

getNextNotifications

```
notification = getNextNotifications(count)
```

Delivers the next count messages that have not yet been read out and that match the filter set when generating the subscription.

Arguments:

count (int): Number of requested messages. If, at the time of the query, there are fewer not-yet-queried messages, these are returned. If there are more messages, getNextNotifications can be called up again to return the next messages.

Return value:

notification (Struct[]):
Struct array with the contents described in PLCN.Remote.Notification.queryNotifications. The oldest matching messages are returned first.

13 Diagnosing errors in the model

13.1 Debugging via the Viewer for Simulink add-on

The Viewer for Simulink add-on allows you to view Simulink models in PLCnext Engineer together with the display of live data for the model signals present in the GDS. If a model is built accordingly in Simulink, the user of the finished PCWLX library can use the feature without a MATLAB license.

To be able to use the “Viewer for Simulink” add-on in PLCnext Engineer, you must export the model data using the “Model Exporter” during the build. You can enable the export with the “PLCN_ExportDiagram” model setting.

In order to view signals in the “Viewer for Simulink”, you must also enable test points on the ports. To enable the test points in the desired part of the model, you can use the following command:

```
PLCN.Config.activateTestpoints('<modelName>’,true,
find_system(bdroot,'FindAll', 'on','selected','on'))
```

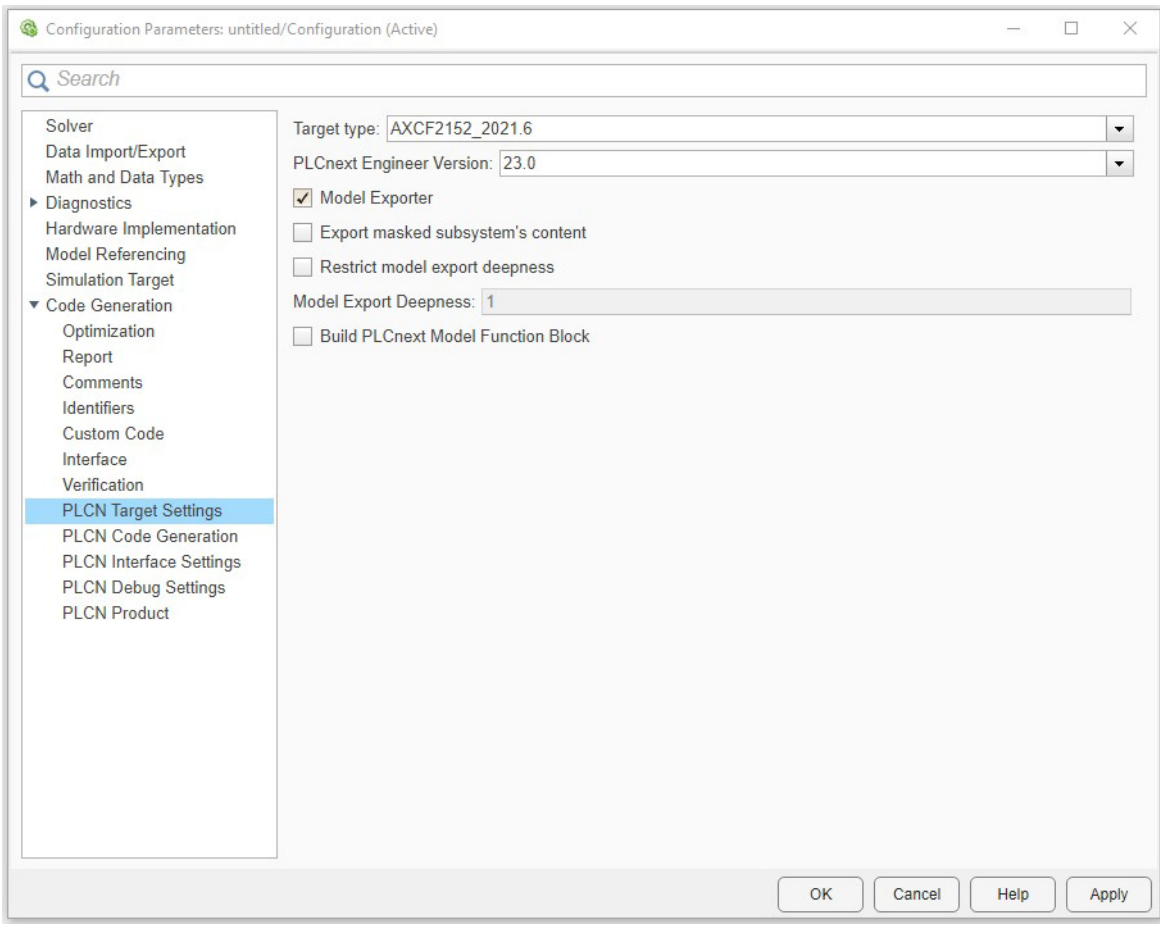


Figure 13-1 PLCN Target Settings - Model Exporter configuration page

Moreover, you can specify whether the content of masked subsystems is also to be exported via the setting “PLCN_ExportMasked”. You can limit the number of exported subsystem levels via the setting “PLCN_ExportDeepness”.

You can only display signals that have been generated in the Simulink Coder and have also been provided as a GDS signal.

To generate parameters, either the “DefaultParameterBehavior” model setting has to be set to “Tunable” or the storage class of the parameter has to be set to “Model default”. You can set the “OptimizeBlockIOStorage” setting to “off” to maximize the number of BlockIO signals that can be generated.

However, you should be aware that the explicit generation of signals comes at the expense of efficiency.

In contrast, you can specify the provision of the generated signals as a GDS signal via the “PLCN Interface Settings” in the model settings and via the “Signal interface settings” via the context menu in the model. The setting should be set to at least ‘GDS’.

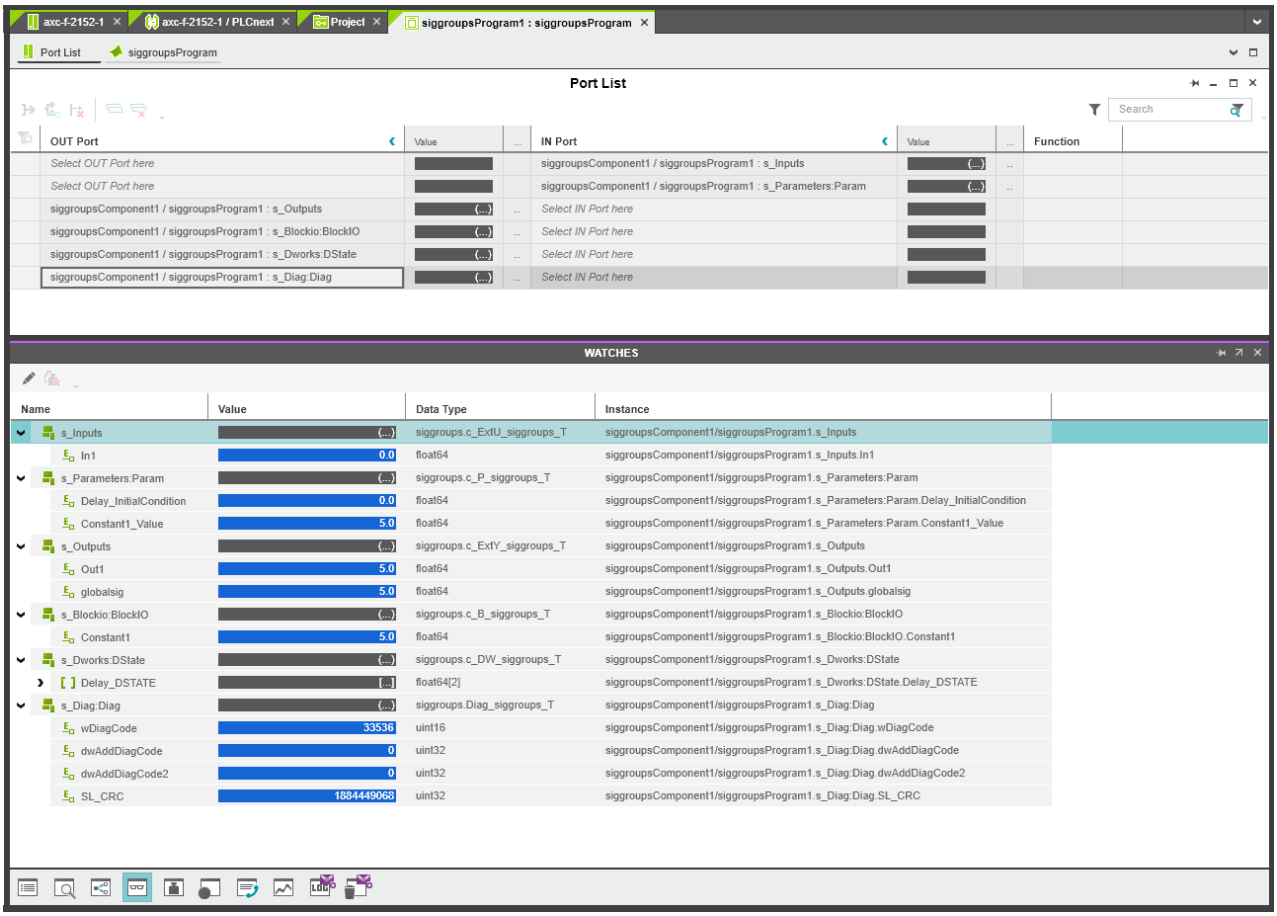


Figure 13-2 Example: structure exported from a Simulink model

The signals are automatically displayed in the correct position in the PLCnext Engineer “Model View”. Here, you can use them – as well as other GDS signals – together with the “Watch window”, “data logger”, etc.

13.2 Debugging via Eclipse

You can debug the code running on the controller via the Eclipse IDE with the help of gdbserver. This provides you with the following options:

– Observing the behavior of the model on the controller directly via the generated C code.
– Debugging the model on the controller directly via the generated C code.
– Searching for and finding specific errors during execution.

13.2.1 Model requirements

The following requirements must be met so that you can debug the model:

The model has been built with “Build Optimization (PLCN_OptFlags)”, “Debug”, or “Release, with debug infos” (default setting). The appropriate settings for exception handling should be set for investigating errors (see Section 5.2.2).

13.2.2 Preparing and performing debugging

For each build, a corresponding Eclipse project is automatically generated which can be started with the load_Eclipse_Project.bat file that is also generated in the build directory. If the build directory was created on a different system, you must first adjust the corresponding options and paths in the batch file.

You can debug the loaded Eclipse project with GDB on the controller. You can also adapt and rebuild the model in Eclipse, e.g., by inserting log outputs, etc. Note that the PLCnext Eclipse Toolchain is not used here. This means that it is not possible to easily adapt GDS ports or to change the type of controller that the model is being built for. This adaptation option is mainly intended for minor adaptations to support the debugging process.

After loading the project, you may have to add a path mapping to the selected debug configuration. Under “Source”, create the mapping from the build folder on the original system to the new build folder (paths including the “<model>_PLCN” folder).

After the preparation work has been performed in Eclipse, preparation work needs to be performed in PLCnext Engineer. Depending on the application, it can be useful or necessary to disable (value = 0) the watchdog in the task whose program you want to debug. If necessary, also the watchdogs of all tasks. Otherwise, if a program is stopped in the debugger, this usually triggers the watchdog. This may also happen when connecting to the debugger.

It can also be useful to stop the controller and only restart it after the debugger is connected.

After both Eclipse and PLCnext Engineer have been prepared for debugging, you can start the gdbserver on the controller.

- To do so, connect to the controller via SSH and execute the following command:

```
sudo gdbserver :2345 --attach $(pgrep -of Arp.System.Application)
```

Here, port 2345 is the port set previously in the debug configuration. After the command has been executed, the connection to PLCnext Engineer is lost. You can only reconnect after debugging has been started in Eclipse or after the running gdbserver in the SSH terminal has been ended via “Ctrl+C”.

You can then start the debug process in Eclipse. To do so, select the correct configuration in the “Run, Debug Configurations” menu and start the process via the “Debug” button.

The time for the debugger to connect to the gdbserver on the controller depends on the program size, the utilization of the local system, the controller, and also the connection to the controller. You can also follow this process in the “Debugger Console” window in Eclipse. You can see that the connection has been completed when all threads are listed in the debug window in Eclipse.

If a watchdog on the controller was triggered while the connection was being established, you can use the “F5” key (Write and Start Project) to restart the project.

If the error in the model is triggered again and the appropriate Exception Handling model setting is set (see Section 8.4, “Exception handling”), Eclipse jumps directly to the corresponding point and stops program execution there. At this point, you can utilize the versatile Eclipse debugger options. In particular the stack trace to determine the position within the model step function, as well as the view of the defined variables and their values.

Alternatively, you can also set breakpoints independently of errors and examine and follow the program execution.

13.2.3 Errors

If breakpoints do not work, the model was not built with one of the “Debug Build Optimizations” or the sources/library cannot be found or assigned. See Section 13.2.2 for information on how to adjust the imported settings.

13.3 Debugging with GDB

As an alternative to remote debugging with the gdbserver, you can use the GDB debugger command lines directly on the controller via SSH. If you can do without the graphic display in Eclipse, the whole process can be carried out much faster.

14 Hardware-in-the-Loop (HIL) tests

The basic idea behind HIL tests for a model is to compare the runtime behavior in the local simulation with the behavior on the controller. If there are no relevant deviations, the test is successful. If there are deviations, you can analyze these in detail and remedy them.

The versions described here function in such a way that the model input data (at the root inports) are recorded or transmitted from Simulink. The model simulation is performed with the input data on the controller and then the model output data (at the root outports) is again verified in Simulink. So far, only single-task models are supported.

14.1 Test Manager

The MATLAB class “PLCN.TestManager” created for this purpose is used for this method. First, a simulation of the test model is performed locally in Simulink, whereby both the input data and the output data is recorded and saved in a file.

The test model is then loaded onto the controller and executed together with a PLCnext Test Harness component. During each cycle, the Test Harness component resets the model inputs to the values previously recorded in Simulink and also stores the output data.

The two output data sets are then compared again in Simulink. Deviations for certain signals can be displayed graphically and further investigated.

14.1.1 Preparing the model and input data

In preparation for the test, the Simulink model to be tested is required first. Furthermore, you must feed the model with input data. Two methods for doing so are described below.

The simplest method is to generate a Test Harness for the model using Simulink Test; the respective desired inputs for the actual model are then set in the Test Harness. The step size of the model should be the same as that of the main model. Moreover, the stop time should be the duration that is to be tested for.

Simulink Test is not required for the second method. In this method, input ports are fed with data via standard Simulink Toolbox functions during the simulation. Using the MATLAB function “createInputDataset”, you can create a data set with data for every input port. You can then adapt this as desired and link it with the model (see MATLAB documentation on “createInputDataset”). Alternatively, you can also link a corresponding signal for each relevant input port via the “Connect Input” function in the block parameters of the port. Instead of linking the ports directly, you can also work with a Test Harness model that is independent from Simulink Test and in which the model to be tested is integrated as a reference.

With both methods, the inports of the model to be tested are either linked with data directly (and the model itself is simulated) or supplied with data from a superordinate model (this is then simulated for recording the data). The superordinate model is then designated as the parent model. If a parent model does not exist, the test model itself is the parent model.

For later execution, the test model is built as usual on the controller as a PLCnext library.

14.1.2 Performing tests

There are two options for performing the HIL tests.

The first option is used if the Simulink Test add-on has not been installed. For this option, the Test Harness is executed via the "PLCN.TestHarness.TestManager.testSimVsPlcnController(modelName, harnessName, harness-IPAddr, harnessPassword, testHarnessFolder, runOnce, capInByte)" MATLAB command.

If the Simulink Test add-on has been installed on the system, a test file for the test is prepared on the PLCnext target via the "PLCN.TestHarness.TestManager.prepareTestCasesForPlcnUse(modelName, testFilePath, TestFileName)" command. If no previous test file is available, using this command generates a new test file with a test case. If a previous test file is available, the test cases for the test are prepared on the PLCnext target and then executed on the controller.

If you use the "Configure Targets" button on the "PLCnext" tab, the "Target IP" and the "Target Admin Password for Test Harness" are taken from the model settings. See Section 17, "Using the PLCnext tab" and Section 5.2.1, ""PLCN Target Settings" configuration page".

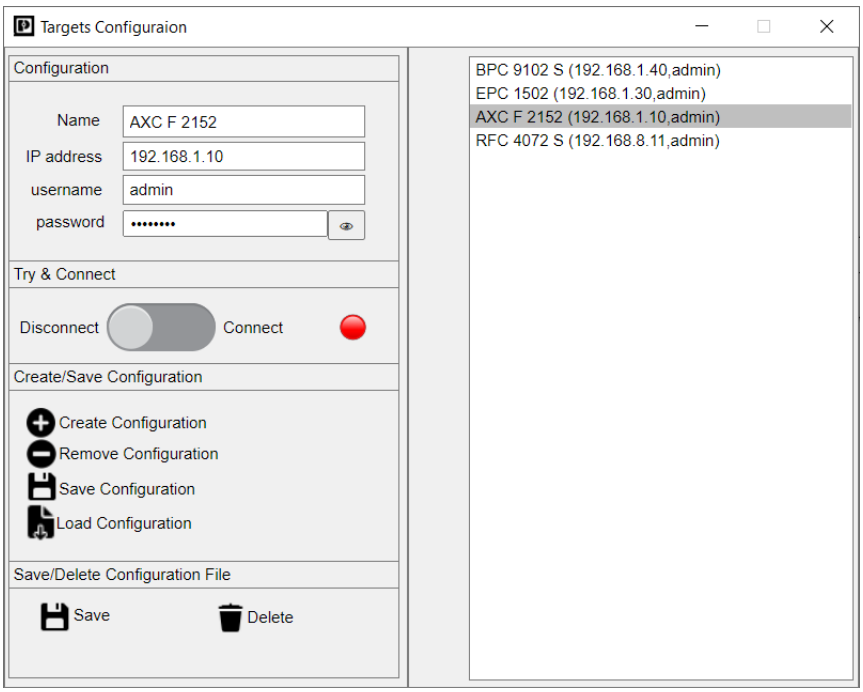


Figure 14-1 "Target Configuration" dialog

In addition, when the abovementioned command is used, the test model that has been created in Simulink and the PLCnext Test Harness component are loaded to the controller.

In order to generate the Test Harness component, you need to use the "PLCN.TestManager.buildComponent(TargetType)" command.

The "TargetType" argument corresponds to the "PLCN_TargetType" model parameter. This command generates a non-model-specific but SDK-specific PLCnext "Test Harness Library" (file name "TestHarness.pcwlx") in the current folder via the PLCnCLI tool. This "Test Harness" component contains two programs: "InputDataProgram" and "OutputDataProgram".

You need to integrate both these program libraries into a PLCnext Engineer project. If the programs are instantiated in a task with a defined cycle time, you have to ensure the following sequence:

“InputDataProgram”, “<Model>Program”, and then “OutputDataProgram”.

The project is then loaded to the controller and started. When the model is started, the test case is not yet performed, as the corresponding test files from Simulink are not yet available. At the start of the test, all test data is read from the configuration files to the memory. The test results are only written at the end, when you replace the “runOnce” option with 1 via the “PLCN.TestHarness.TestManager.testSimVsPlcnController” MATLAB command.

For large data volumes, which exceed the memory capacity of the PLCnext controller SD card, “runOnce” is replaced with 0. This means that all test data is read from the configuration files of the input parameter file to the memory, partially based on “capInByte” (default value of 10000 bytes), and the test results are partially written to the output parameter file.

14.1.3 Functions of the PLCN.TestManager class

This class provides the following functions:

```
function testSimVsPlcnController(modelName,harnessName,IPAddr,Passwd)
```

This MATLAB command calls up the following functions:

- addBlockportsToLoggroup
- executeSimulinkRun
- setDataSet
- buildComponent
- writeProjects
- executeControllerRun
- loadControllerRunLog
- clearBlockportsFromLoggroup
- compareRuns

```
function prepareTestCasesForPlcnUse(modelName,testFilePath,testFileName)
```

If a test file is available in the Simulink Test add-on in MATLAB, the test cases in the test file are prepared for execution on the PLCnext target via this MATLAB command. Subsequently, test cases are automatically executed. You must add a “Simulation Output Signal Set” to each test case.

The following steps are performed via the MATLAB command:

– prepareTestCases
– addBlockportsToLoggroup
– executeSimulinkRun
– setDataSet
– buildComponent
– writeProjects
– executeControllerRun
– loadControllerRunLog
– clearBlockportsFromLoggroup
– compareRuns in Simulink Test App

```
function addBlockportsToLoggroup(obj,modelBlock)
```

All root port signals in the model are marked as logging points. Only the ports marked are used for the test.

```
function clearBlockportsFromLoggroup(obj,modelBlock)
```

All root port signals in the model have their logging point status removed.

```
function executeSimulinkRun(obj)
```

The model is simulated in Simulink and IO data is recorded.

```
function loadSimulinkRunLog(obj,matpath,varname)
```

The I/O data from an earlier simulation is loaded.

```
function writeProject(obj)
```

With this command, the libraries are integrated in a PLCnext Engineer project, and instantiate the programs in a task with an appropriately set cycle time in the following sequence: “InputDataProgram”, “<Model>Program”, and “OutputDataProgram”. Then, the project is loaded to the controller.

```
function executeControllerRun(obj)
```

– writeRundataIn
– uploadRundataInToController
– restartControllerRunExecution
– waitforControllerRunFinished
– downloadRundataOutFromController

```
function loadControllerRunLog(obj);
```

The output data from an earlier simulation are loaded into the class.

```
function compareRuns(obj);
```

The controller and simulation run are compared.

```
function writeRundataIn(obj)
```

The configuration files needed by the Test Harness PLCnext component are created.

```
function uploadRundataInToController(obj)
```

The configuration files are loaded onto the controller. Old files are deleted.

```
function restartControllerRunExecution(obj)
```

The controller is restarted.

```
function waitforControllerRunFinished(obj)
```

The process waits until the output data is available (check every 10 s).

```
function downloadRundataOutFromController(obj)
```

The output data is downloaded and loaded into the class in Simulink.

```
function writeSimulinkRunLog(obj,matpath,varname)
```

Counterpart to “loadSimulinkRunLog”. The current simulation data is saved in a MAT file.

The end result of the comparison is either a message stating that all signals are the same or that there are differences. In both cases, there is a link to the graphic display in Simulink.

The comparison is performed by default with a relative tolerance of 1*eps (“single”) so that with floating point calculations, normal differences are not counted as errors. This tolerance can be adjusted.

14.1.4 Limitations

Currently, only single-task models can be tested. Moreover, the possible duration for the tests is limited by the memory needed for the input and output signals, and therefore by the controller RAM. To achieve the highest possible efficiency and so as not to lengthen the execution time of the model task, the input data is read in completely at the start of the test and the output data is only written to a file at the end of the test.

The “ISubscriptionService” and the “IDataAccessService” are used to supply the model on the controller with data and to read it out again from the outputs. This means that the performance is not the same as when the model is connected directly to other programs via its GDS ports. Accordingly, executing the model task with the Test Harness component takes longer.

14.2 Co-simulation

WARNING: Personal injury and property damage

As real-time behavior cannot be ensured during co-simulation, personal injury and property damage to the system may occur.

- Only use co-simulation in a laboratory setting.

Co-simulation enables you to partially execute a model on a target system (controller) and on a host system (MATLAB Simulink). This means you can compare the model behavior on the controller and in the MATLAB Simulink simulation with one another. The duration of this test is not limited because the model data is exchanged cyclically between Simulink and the controller.

The function is available exclusively for referenced models which are integrated into a parent model. For co-simulation, a client block is generated based on the referenced model.

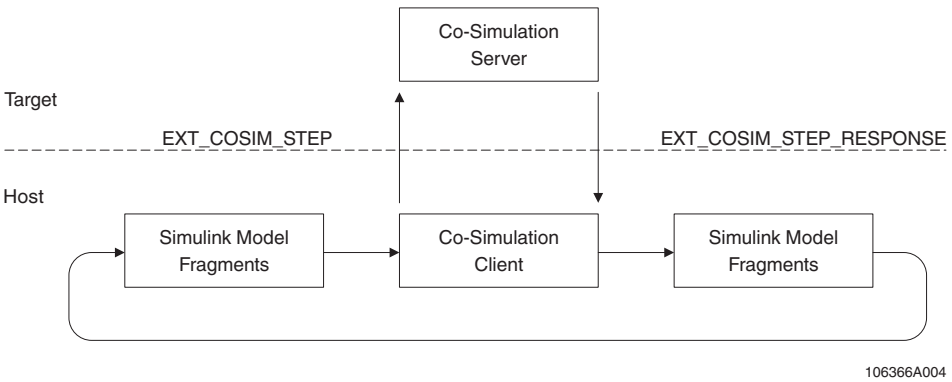


Figure 14-2 Method of operation of co-simulation

Key:

Target:	Controller
Host:	MATLAB Simulink
EXT_COSIM_STEP:	Trigger for processing a model step
EXT_COSIM_STEP_RESPONSE:	Result of processing a model step

14.2.1 Setting up a model for co-simulation in MATLAB Simulink

- Integrate the model you wish to execute in co-simulation as a referenced model into a parent model.

Target model

The referenced model corresponds to the model component executed on the target system (referred to as the target model below).

Host model

The parent model corresponds to the model component executed on the host system (referred to as the host model below).

Setting up the target model

- Make the following settings for the target model:
  - Define the dimensions and data types for the inputs and outputs.

Please note:
You cannot use target models with generic inputs and outputs.

  - On the “Code Generation” configuration page, select the “pxc_plcn.tlc” file (see Figure 5-1).
  - Configure the target model for External Mode (see Section 11.1).
  - Enable the function “Wait for start package”.

Setting up the host model

- Make the following settings for the host model:
  - On the “Code Generation” configuration page, enter the interval (see Figure 5-2).
- Configure the host model for External Mode (see Section 11.1).
- Connect the inputs and outputs of the target model to the desired simulation signals.

Generating a client block

- Right-click on the block for the target model (in Figure 14-3: “Model” block).
- To generate a client block for co-simulation, select “Co-Simulation, Create Interface” in the context menu (see Figure 14-3).

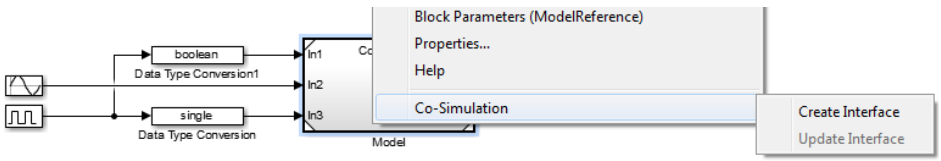


Figure 14-3 Generating a client block

- Save the host model.

⇒ The client block is automatically synchronized with all relevant properties of the target model (e.g., IP address, data types of the inputs and outputs).

Please note:
If you then make changes to the target model, you must carry out synchronization manually at the client block. To do so, proceed as follows:
- Right-click on the client block (in Figure 14-4: “Model_CoSim”).
- Make the same settings via the context menu that you made at the target model.

- Connect the client block inputs to the same simulation signals as the target model inputs (see Figure 14-4).

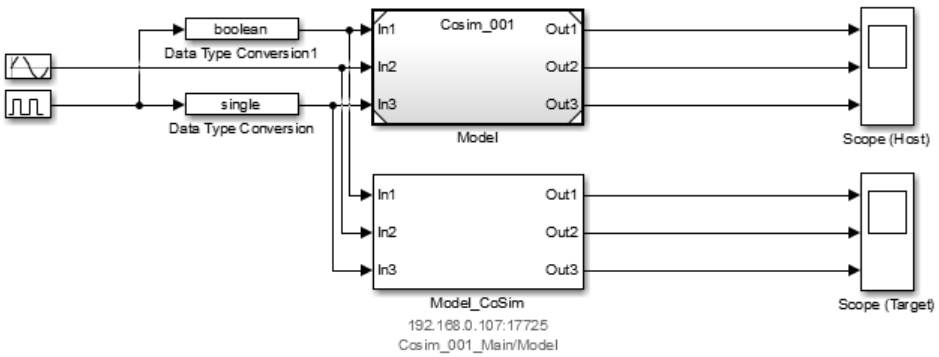


Figure 14-4 Example: connecting the inputs and outputs of the “Model_CoSim” client block

- Connect the outputs to the desired function blocks.

To compare the model behavior, you can automatically compare the output signals of both model blocks.

Instead of comparing both blocks directly, you can also test the outputs of the target blocks against requirements using Simulink Test. Other options are also possible.

Changing the path to the referenced model

If you make changes during model development and wish to change the path to the referenced target model, proceed as follows:

- Double-click on the client block.
- Click on the “CoSimulation” tab in the “Block Parameters” window.
- Enter the new path in the “Block Reference” input field (see Figure 14-5).

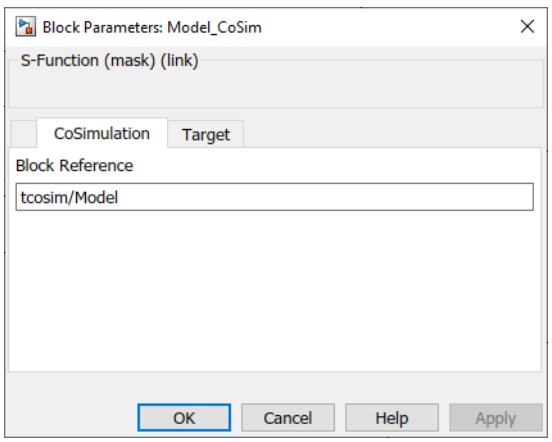


Figure 14-5 Example: changing the path to the referenced model

- Click “OK”.

Changing the connection parameters

If you wish to change the parameters for the connection to the controller, proceed as follows:

- Double-click on the client block.
- Click on the “Target” tab in the “Block Parameters” window.

- Change the parameters in the “IP address”, “Port number”, and “Timeout [s]” input fields (see Figure 14-6).

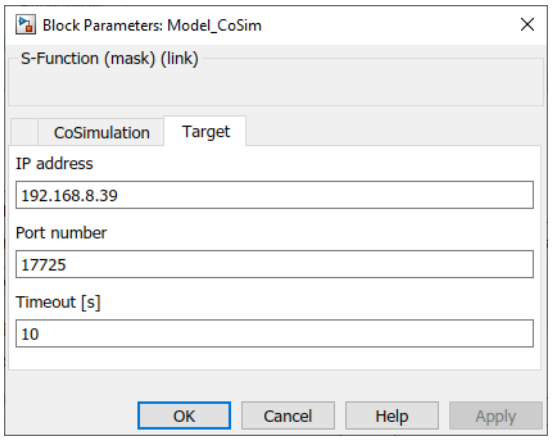


Figure 14-6 Example: changing the connection parameters

- Click “OK”.

14.3 Executing the model in co-simulation

Compiling the target model and loading it onto the controller

- Compile the target model in MATLAB Simulink as described in Section 5.6.
- If the model has changed during build (recognizable via the asterisk alongside the model name), you should execute the function PLCN.refreshTarget.

This way, any potential outdated model settings are adapted.

- You must then save and rebuild the model.
- Load the target model onto the controller. Depending on the model settings, you still have to initialize the target model via the *xInitialize* port.

⇒ Now, the model should be paused and wait for a connection from Simulink due to the setting “Wait for start Package” (*wDiagCode* = 0x8200, *dwAddDiagCode* = 1).

Executing the host model in MATLAB Simulink

- Prior to the simulation, perform an interface update for the co-simulation block. To do so, click on the “Co-Simulation, Update Interface” entry in the context menu.
- To start simulation of the host model in MATLAB Simulink, click on the  button.

⇒ Simulation of the host model is started and connected to the controller. Execution of the target model therefore starts automatically.

Please note:

If you stop the host model simulation in MATLAB Simulink, the target model is paused. To avoid asynchronous data, in this case you must restart the program execution before you re-establish a connection.

14.3.1 Limitations

Currently, only single-task models can be tested. The data types of the host model ports are limited to the types double, single, int8, uint8, int16, uint16, int32, uint32, and boolean. Complex types and Struct types are not possible, but array types are.

Here, the model signals are transmitted cyclically, whereby the duration of the simulation execution takes a maximum amount of time. During execution on the controller, the model steps are only executed when the old output data has been sent to Simulink and new input data is available.

15 Multirate

15.1 Versions

Multirate, single task

Here, the model is executed within a task that is running at the fastest task rate available in the model. You can use this version by disabling the “EnableMultitasking” model setting.

A scheduler that works via task counters is used for processing the different rates in a task. As is the case in the “single rate, single task” version, the real-time scheduler of the controller is used for processing the task.

The disadvantage of “multi-rate, single task” is the low utilization of computing time across the temporal program flow, or the artificial increase of the minimum cycle time that can be used. The computing time of all task rates together constitutes the minimum possible cycle time of the entire program. This cycle time is always fully utilized when – but only when – all task rates coincide. For example: in the case of a 10 ms, a 20 ms, and a 50 ms rate, this only occurs every tenth time.

Multirate, multitask

Here, the model is executed within several tasks that each run at one of the task rates available in the model. You can use this version by enabling the “EnableMultitasking” model setting.

The real-time scheduler of the controller is used to process the various tasks.

When the code is being generated, an appropriate program is generated for each task rate. After integration of the PCWLX model library into PLCnext Engineer, the programs are distributed between the respective tasks. Keep in mind that the following assumptions are made during the code generation in the Simulink Coder and that you have to take them into account during the integration into PLCnext Engineer:

– The faster a model rate, the higher the priority of the associated task. Tasks with higher priority can interrupt tasks with lower priority in favor of their own execution.
  Tasks with a low interval in PLCnext Engineer require a higher priority (numeric value has to be lower) than tasks with a higher interval.
– All model programs are executed on the same processor core or only one task rate is executed at a time.
  All tasks belonging to a model have to be allocated on the same processor core or on the same ESM.
– Execution of the individual tasks is performed with exact timing and is limited to the respective task interval.
  This is safeguarded by the real-time scheduler of the controller. Via the watchdog, a range with precise start and end points are safeguarded for each cycle. The execution time of the task has to be within this range.
  The task intervals of the model tasks have to correspond to the values set in the model – the ratio of the task rates in relation to each other has to be maintained.

For further information, refer to the MathWorks documentation on “Handle Rate Transitions” at https://de.mathworks.com/help/rtw/ug/handle-rate-transitions.html.

Multirate, multicore

Here, the model is executed within several tasks that each run at one of the task rates available in the model. You can use this version by enabling the “ConcurrentTasks” model setting.

To use the concurrent execution feature, a special model configuration is required (see the Mathworks documentation “Configure Your Model for Concurrent Execution”).

During code generation, several programs are generated, analogous to the “multirate, multitask” variant. However, in the assumptions made during code generation, number 1 and 2 are omitted. This way, distribution of the generated programs to tasks in various process cores is allowed.

For further information, refer to the MathWorks documentation on “Model Concurrent Execution for Symmetric Multicore CPU Platforms” at https://de.mathworks.com/help/ecoder/ug/model-concurrent-execution-for-symmetric-multicore-cpu-platforms.html.

15.2 Data consistency

Due to the parallel processing or, in the case of multicore programs, even simultaneous processing of various program parts, special attention has to be paid to the data exchange between these program parts. In terms of the behavior, the two points “integrity” (if the data arrives correctly) and “determinism” (if the data arrives at clearly defined points in time) of the data are important. This can be adjusted individually for each exchange of data between the program parts.

Multirate, multitask

In Simulink, signals between model parts with various task rates are transmitted in “Rate Transition” blocks. You can either explicitly insert “Rate Transition” blocks into the model (recommended method) or implicitly use them automatically when the “AutoInsertRateTranBlk” setting is active. For both methods, there are various options available for the data transfer:

– “Ensure data integrity during data transfer” enabled
  This option is only configurable for explicit “Rate Transition” blocks. Without block implicitly active.
  Ensures the content consistency of data. In the generated code, this is achieved by means of buffers.
– “Ensure deterministic data transfer (maximum delay)“ enabled/“Deterministic Data Transfer: Always“
  Data transfer between fast and slow task, when both tasks coincide.
  Slow -> Fast: As a result of the prioritization of the fast task, the fast task reads values of the slow task that are outdated by one cycle.
  Fast -> Slow: As a result of the prioritization of the fast task, the slow task always reads the current values of the fast task.
– “Ensure deterministic data transfer (maximum delay)” disabled/“Deterministic Data Transfer: Never (minimum delay)”
  Slow -> Fast: Slow task uses double buffer, cyclically writes the current value and shifts the index. This way, the fast task can almost always read the current value from the slow task.
  Slow -> Fast: Slow task uses semaphores during the reading process, so that the value cannot be changed by the fast task while this is in progress. The current value can always be read.
  Since there is no temporal synchronization during data exchange, varying task runtimes can lead to the reading program randomly reading outdated values twice or skipping current values.

Behavior corresponds to that of the multicore programs. However, it has the advantage that no model changes such as those for “Concurrent Execution” need to be made necessary. But it also has the disadvantage that it is not possible to use multiple processor cores, which means that the computing power required for executing the program cannot be distributed.

Multirate, multicore

The transmission of signals between model parts with different task rates is safeguarded where necessary via mutexes. Note that the content consistency of data is safeguarded, but not the temporal consistency. Multicore programs, just like the processor cores, run asynchronously in terms of time. As a result, it is possible that the reading program randomly reads outdated values twice or skips current values during data exchanges.

15.3 Behavior in relation to control ports

During start and stop of the module as well as during loading and unloading, it is ensured that the individual program parts always start or stop running at the point in time when they would be executed together (common sample hit). This means that you always have a deterministic start and intermediate state for the model execution.

If the “PLCN_ParamMultirateHandling” setting “Buffer” is used, the buffers for GDS parameter changes are also only transferred to the model at the time they are addressed.

16 Adapting code generation for PLCnext programs

To be able to integrate your own initialization steps that cannot be executed in the step function of the model for reasons of execution time, you have the option of adapting the code generation for the PLCnext component.

To do this, you can adapt any inlined S function of the model and add instructions for code generation to its TLC file.

16.1 Adapting component code generation

The TLC functions addToComponentSourceSection and addToComponentHeaderSection are available for generating code in the PLCnext components.

addToComponentSourceSection(sectionName,buffer_in)

Arguments:

sectionName: Includes, Declarations, Initialize, LoadSettings, SetupSettings, SubscribeServices, LoadConfig, SetupConfig, ResetConfig, PublishServices, Dispose, PowerDown, Start, Stop

buffer_in: TLC-Buffer

Example:

```
%% Add code to the component.SetupConfig implementation
%openfile buffer
Log::Info("Component.SetupConfig");
%closefile buffer
%<addToComponentSourceSection("SetupConfig", buffer)>
```

The section names Initialize, LoadSettings, SetupSettings, SubscribeServices, LoadConfig, SetupConfig, ResetConfig, PublishServices, Dispose, PowerDown, Start, Stop reference the corresponding sections in the Component.cpp.
In relation to the above example, this means:

```
void TestSimComponent::SetupConfig(void)
{
    (code wird hier geschrieben)
}
```

The section names Includes and Declarations allow you to add the corresponding code in the Component.cpp outside of the component class.

addToComponentHeaderSection(sectionName,buffer_in)

Arguments:

sectionName: Includes, Declarations

buffer_in: TLC-Buffer

Enables corresponding code to be added in the Component.hpp outside of the component class.

The following example adds the Include line in the Component.hpp:

```
%openfile buffer
#include "Arp/System/Commons/Logging.h"
using namespace Arp::System::Commons::Diagnostics::Logging;
%closefile buffer
%<addToComponentHeaderSection("Includes", buffer)>
```

```
#pragma once
#include "Arp/System/Core/Arp.h"
```

...

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

...

16.2 Triggering model code when the component state changes

The functions addToOnComponentStart, addToOnComponentStop, and addToModelPreRun enable the execution of individual code in the model when the component has been stopped or started. If the model is instanced multiple times, meaning it is present several times as a program in the ESM, the respective code is executed per instance.

addToOnComponentStart(buffer_in)

Arguments:

buffer_in: TLC-Buffer

Enables code to be added in the Model.cpp. The code is executed as soon as the component is started (Component.Start is executed).

Example:

```
%% Add code to the model.cpp that will be executed each time
%% when component.Start was triggered
%openfile buffer
Log::Info("Model called from Component.Start");
%closefile buffer
%<addToOnComponentStart(buffer)>
```

addToOnComponentStop(buffer_in)

Arguments:

buffer_in: TLC-Buffer

Enables code to be added in the Model.cpp. The code is executed as soon as the component is stopped (Component.Stop is executed).

Example:

```
%% Add code to the model.cpp that will be executed each time
%% when component.Stop was triggered
%openfile buffer
Log::Info("Model called from Component.Stop");
%closefile buffer
%<addToOnComponentStop(buffer)>
%<addToModelPreRun(buffer_in)>
```

Arguments:

buffer_in: TLC-Buffer

Enables code to be added in the Model.cpp. The code is executed once for the model instance when the component is started. If the PLC is started via “cold start” or “warm start”, the behavior is the same as with addToOnComponentStart. With a “hot start”, even though Component.Start is executed, the addtoModelPreRun code is not re-executed.

Example:

```
%openfile buffer
Log::Info("ModelPreRun: Executed after model was instantiated and the
component started.");
%closefile buffer
%<addToModelPreRun(buffer)>
```

16.3 General behavior

Multiple calls to the addTo methods do not overwrite the existing content, but attach the transferred buffers to the end of the corresponding area.

17 Using the PLCnext tab

On the “PLCnext” tab, you can execute various functions and options of the PLCnext Target for Simulink by means of a button in MATLAB Simulink.

Please note:
The “PLCnext” tab is available for MATLAB Simulink version ≥R2021b.

If the “PLCnext” tab is not available after installing PLCnext Target for Simulink, use the “sl_refresh_customizations” command to enable it.

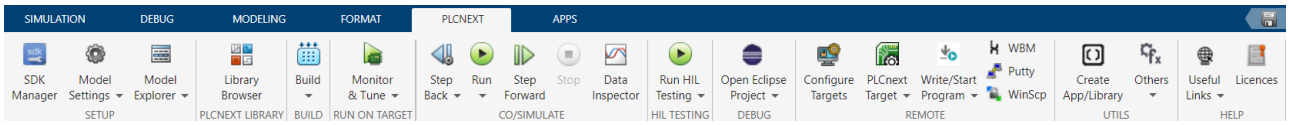


Figure 17-1 PLCnext tab

Table 17-1 Buttons on the PLCnext tab in MATLAB Simulink

Button	Function
SDK Manager	SDK Manager for managing the SDK.
Model Settings	Parameters for model configuration:
	Edit model configuration parameters.
	Call up model properties.
	Model configuration parameters of the currently referenced model instance.
	Model properties of the currently referenced model instance.
Model Explorer	Table view of several models:
	Table view of several models for viewing, editing, and adding elements.
	Table view of several models for viewing, editing, and adding data.

Table 17-1 Buttons on the PLCnext tab in MATLAB Simulink

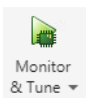
Button	Function
	Shows the available PLCnext libraries.
	Generate code and build the model:
	Generate code and build the model.
	Generate code only. The makefile is not executed.
	Set up model for monitoring and optimization:
	Set up and run the model for External Mode with a PLCnext controller, as well as options for monitoring signals and optimizing parameters.
	Build model for External Mode.
	Integrate model on the PLCnext controller.
	Establish a connection to the PLCnext controller.
	Start model execution.
	Stop model execution.
	Disconnect from the PLCnext controller. The model remains in the last state it had.
	Execution of hardware-in-the-loop tests:
	Perform all steps together.
	Generate model for HIL testing on a PLCnext controller.
	Generate test harness code for PLCnext controllers.

Table 17-1 Buttons on the PLCnext tab in MATLAB Simulink

Button	Function
	Run test in Simulink.
	Integrate code and other data on the PLCnext controller for the HIL test.
	Run tests on the PLCnext controller.
	Compare test results in Simulink with the PLCnext controller.
	Open the project in Eclipse and start the gdbserver:
	Open the project in Eclipse.
	Start the gdbserver on the PLCnext controller.
	Management and control configuration of the PLCnext controller. See also Section 17.1.1, “Creating a configuration profile for a PLCnext controller”.
	Remote control options for the PLCnext controller: See also Section 17.1.2, “Controlling the PLCnext controller remotely”.
	Start the controller in “Cold” state.
	Start the PLCnext controller in “Warm” state.
	Start the PLCnext controller in “Warm” state.
	Stop the PLCnext controller.
	Restart the PLCnext controller.
	Display the processing status of the model on the PLCnext controller.
	Display diagnostic information of the PLCnext controller.

Table 17-1 Buttons on the PLCnext tab in MATLAB Simulink

Button	Function
	Display hardware information of the PLCnext controller used.
	Write the project to the PLCnext controller and start it.
	Write the project to the PLCnext controller and start it.
	Write the project change without downtime to the PLCnext controller and start it.
	Update the functionalities of the project on the PLCnext controller.
	Update function changes on the PLCnext controller.
	Establish connection to web-based management of the PLCnext controller.
	Establish a connection to the PLCnext controller with PuTTY.
	Establish a connection to the PLCnext controller with WinSCP.
	Create a PLCnext application for the PLCnext Store. See also Section 17.2, “Creating a PLCnext app/library for the PLCnext Store”.
	Additional options for the PLCnext Target for Simulink:
	Retrieve model information.
	Create and merge PCWLX libraries for different architectures (32 bit, 64 bit).
	Convert reference models to subsystems, for use in External Mode.
	Hyperlinks to websites in the PLCnext ecosystem:

Table 17-1 Buttons on the PLCnext tab in MATLAB Simulink

Button	Function
	Open the PLCnext Community website.
	Open the PLCnext GitHub website.
	Open the PLCnext Info Center website.
	Open the PLCnext Target for Simulink website.
	Open PLCnext Target for Simulink Matlab live script. The Matlab live script is an additional documentation with live templates and controllers, images, gifs, etc. that supports you when working with PLCnext Target for Simulink.
	Open the PLCnext Store website.
	Open the PLCnext Youtube channel.
	License information for third-party components used.

17.1 Configuring and operating the PLCnext controller

17.1.1 Creating a configuration profile for a PLCnext controller

With the “Configure Targets” button, you can create different configuration profiles for a PLCnext controller. You can then use the configuration profiles created for remote control of the PLCnext controller. Further setting and operating elements are available for this purpose (see also Section 17.1.2, “Controlling the PLCnext controller remotely”).

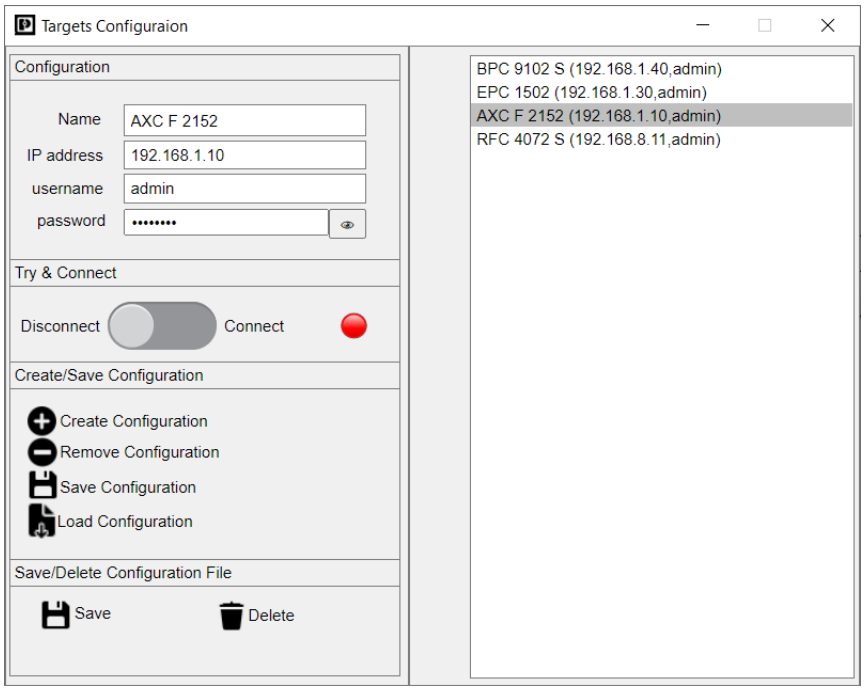


Figure 17-2 “Targets Configuration” dialog

Table 17-2 Description of the settings in the “Targets Configuration” dialog

Setting	Description
Name	Name of the controller.
IP address	IP address of the controller.
Username	User name for logging in to the PLCnext controller.
Password	Password for logging in to the PLCnext controller.
Connect	Establish connection test to the PLCnext controller.
Disconnect	Disconnect from the PLCnext controller.
Create Confiuration	Configure and create a new PLCnext controller.
Remove Configuration	Remove a configured PLCnext controller from the list.
Save Configuration	Save a configured PLCnext controller.
Load Configuration	Load and activate a configured PLCnext controller.

Table 17-2 Description of the settings in the “Targets Configuration” dialog

Setting	Description
Save	Save a configured PLCnext controller file.
Delete	Delete a configured PLCnext controller file.

17.1.2 Controlling the PLCnext controller remotely

With the “Target” button, you can use various operating elements to remotely control the PLCnext controller. Once the configuration is complete and a connection to the PLCnext controller has been established, you can use the “Target Information” tab to retrieve additional status information from the PLCnext controller.

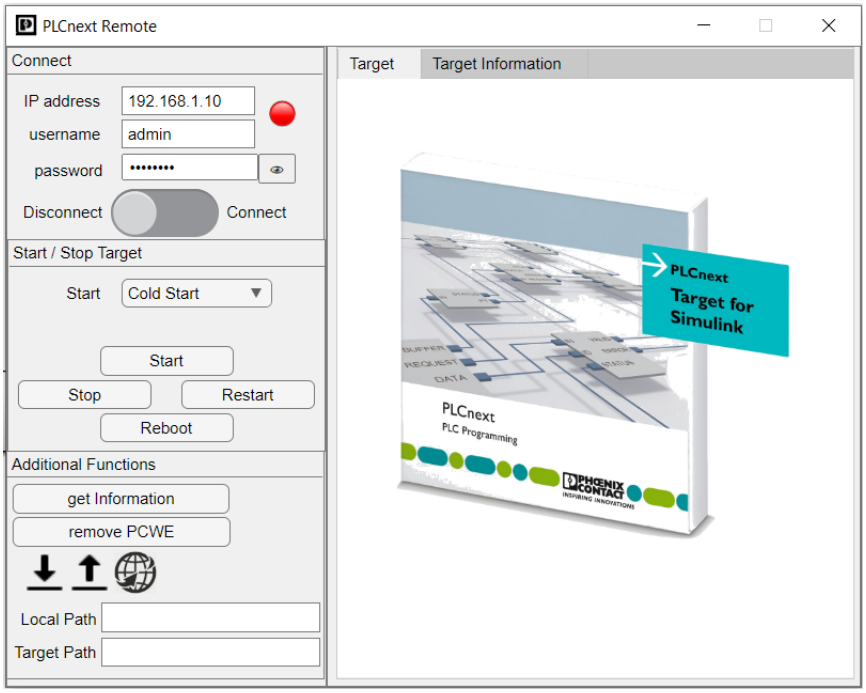


Figure 17-3 “PLCnext Remote” configuration dialog

Table 17-3 Description of the settings in the “PLCnext Remote” configuration dialog

Setting	Description
IP address	IP address of the controller.
Username	User name for logging in to the PLCnext controller.
Password	Password for logging in to the PLCnext controller.
Connect	Establish a connection to the PLCnext controller.
Disconnect	Disconnect from the PLCnext controller.
Start	Start the PLCnext controller.
Stop	Stop the PLCnext controller.
Restart	Restart the PLCnext controller.

Table 17-3 Description of the settings in the “PLCnext Remote” configuration dialog

Setting	Description
Reboot	Reboot the PLCnext controller.
Get Information	Call up additional diagnostic information for the PLCnext controller.
Remove PCWE	Delete the PCWE path on the PLCnext controller.
Download	Download data from the “Target Path” of the PLCnext controller to the local path.
Upload	Upload data from the local path to the “Target Path” of the PLCnext controller.
WBM	Open web-based management of the PLCnext controller.
Local Path	Path to the local directory.
Target Path	Path to the target directory.

17.2 Creating a PLCnext app/library for the PLCnext Store

With the “Create App/Library” button, you can convert a model into a PLCnext app or library using an HTML application to make it available via the PLCnext Store.

17.2.1 Creating a PLCnext app

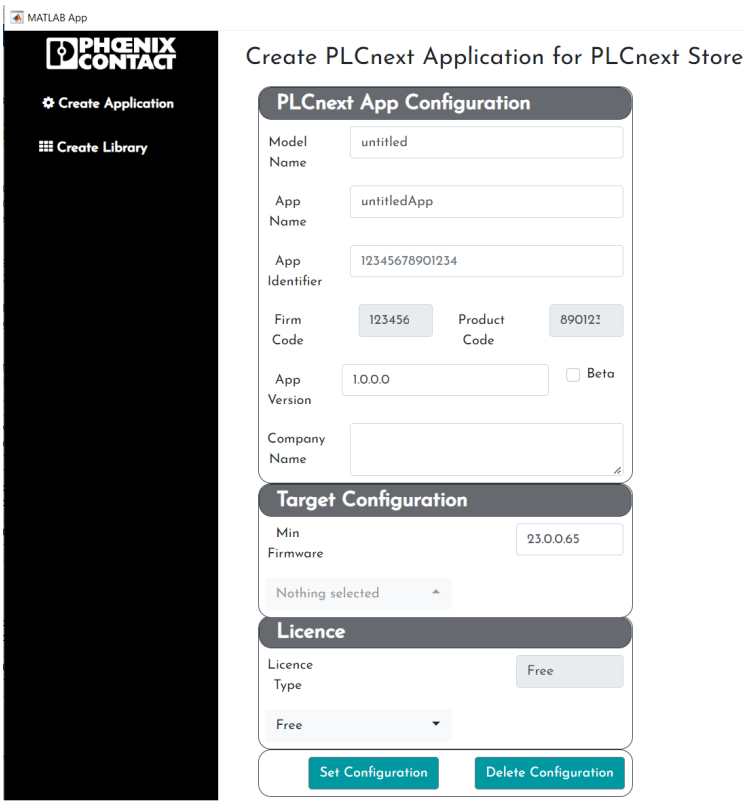


Figure 17-4 “PLCnext App Configuration” dialog

Table 17-4 Description of the settings in the “PLCnext App Configuration” dialog

Setting	Description
Model Name	Name of the model
App Name	Name of the PLCnext application
App Identifier	A unique app identifier (combination of “Firm Code /“Product Code”) assigned by the PLCnext Store during app creation. This is used to create unique folder names. The app ID must be a string of 14 numbers.

Table 17-4 Description of the settings in the “PLCnext App Configuration” dialog

Setting	Description
App Version	This version string identifies the current version of the app. You must observe the following notation: Full (external) version format: <major>. <minor>.<patch>.<build>.<state> Limitation of the numerical values: Major, Minor, and Patch: unsigned 8 bit Build: unsigned 32 bit
Beta	Set the beta flag.
Company Name	Manufacturer name
Targets	Controller on which this app is to be run. This string must be identical to the representation in WBM of the controller (“Information, Type” item) in order to be recognized automatically. The target can contain a single entry or multiple entries (as of PLCnext firmware version 2021.0) of supported targets.
Minimum FW-Version	Minimum firmware version required to run the app. This value is automatically generated from the configured “Target”.
Licence Type	The app license type is required for app management. This is necessary to specify the required action when an app does not have a valid license. – Free: The app is license-free and may be launched without a valid license file. – Limited: The app requires a valid license to run with full functionality. Without this, the app can only start with “limited” functionality. – Full: The app requires a valid license. Without a valid license, the app will stop at system startup. For example, the app must not be started after an offline installation via WBM.
Set Configuration	Replace configurations of the PLCnext application and generate the app.
Delete Configuration	Reset the configuration of the PLCnext application.

17.2.1.1 Creating and installing the PLCnext application

To create a PLCnext application for the model, proceed as follows:

- Register with the PLCnext Store.
  See Log in or register in the PLCnext Store Info Center.
- Become a developer.
  See “Become developer” in the PLCnext Store Info Center.
- Manage your developer company.
  See “Manage registered company” in the PLCnext Store Info Center.
- Configure a PLCnext application in the “Create Application” configuration dialog and generate it with the “Set Configuration” button.
- Publish the PLCnext application.
  See “Publishing a Solution App” in the PLCnext Store Info Center.
- Purchase a license.
  See “Purchasing a license” in the PLCnext Store Info Center.

- Install the license.
  See “Installing a license” in the PLCnext Store Info Center.
- Connect the PLCnext controller to the PLCnext Store.
  See “Connecting a PLCnext Control with the PLCnext Store” in the PLCnext Store Info Center.
- Install the PLCnext application.
  See “Online installing a free app”, “Online installing a chargeable app”, “Offline installing a free app”, or “Offline installing a chargeable app” in the PLCnext Store Info Center.

17.2.1.2 Uninstalling the license of the application from a device

To uninstall the license from a device, proceed as follows:

- Uninstall the license from your device list.
  See “Uninstalling a license” in the PLCnext Store Info Center.

17.2.1.3 Support for the application

The technical and commercial support is to be ensured by the contributor. The “Contact Developer” button allows users to contact you to ask questions and contribute ideas. The user’s request is sent to the email address you set as the support email address in the app. A response within one business day is recommended. The request must be responded to within 3 business days at the latest.

17.2.1.4 Publishing new versions of the application

If you want to publish a new version of your application, proceed as follows:

- Extend the functionality of your app.
- Fix bugs in your app.
- Check the functionality of your app when Phoenix Contact releases a new firmware.
- Correct or revise the app description page using the “Edit” button, if necessary.

17.2.2 Creating a PLCnext library

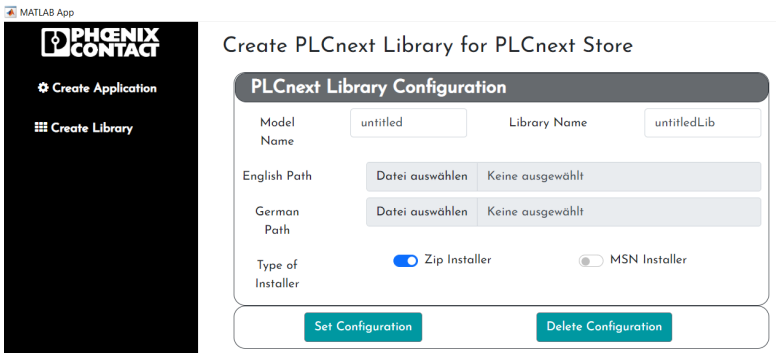


Figure 17-5 “PLCnext Library Configuration” dialog

Tabelle 17-5 Beschreibung der Einstellungen im Konfigurationsdialog „PLCnext Library Configuration“

Setting	Description
Model Name	Name of the model
Library Name	Name of the PLCnext library
English Path	Path where the English manual is located
German Path	Path where the German manual is located
Type of Installer	Selection of whether the PLCnext library should be created as a Zip or MSN installer
Set Configuration	Apply the configuration of the PLCnext library and create the library
Delete Configuration	Reset the configuration of the PLCnext library.

18 Known restrictions

WARNING: Personal injury and property damage

Some of the restrictions listed below can cause unexpected model behavior and consequently result in personal injury and property damage.

- Make sure that your system is in a safe state at all times.

Limitations for PLCnext programs

– If signals of the “Parameter” group are output as structure (PLCN_ParamGroupStruct), parameter tuning via External Mode is not possible with these parameters.
Workaround: If there are parameters that nevertheless are to be tuned via External Mode, assign the “GDS” interface setting to each of the blocks via the context menu. As a result, these parameters are not included in the structure.

Restrictions when using PLCnext Engineer

– If a <model>.pcwlx library is integrated into PLCnext Engineer and PLCnext Engineer is in debug mode, the “EngineeringLibraryBuilder” may crash when the model is recompiled.

Restrictions during installation

If the Windows account that is used to perform the installation (requires administrator rights) deviates from the Windows account under which MATLAB has been licensed, the installation cannot be performed.

- In this case, start “setup.exe” with the following argument:

```
/ComponentArgs “Product“:“TOOLBOX_CHECKUP=“false“
```

- Additionally, you have to manually register the toolbox in each MATLAB installation in which it is to be used, by navigating within the installation folder and subsequently executing the MATLAB commands.

Example:

```
C:\Program Files (x86)\Phoenix Contact\PC WORX Target for Simulink Vers
ion 2.0\pxc_plcn
PLCN.uninstall();
PLCN.install();
```

If you have installed PLCnext Target for Simulink on a target system where a PLCnCLI 2022.0 tool is already installed, we recommend the following steps:
After installation of PLCnext Target for Simulink, deinstall PLCnCLI 2022.0 and then reinstall it again.

Restrictions during the build

– S functions with spaces in the path lead to build errors.
Occurs as soon as self-written non-inlined S functions (not TLC) with spaces in the path are used or if such MATLAB S functions contain spaces.
Workaround: Do not use spaces in the respective path.

– The PLCN subsystem function block library build fails with certain model settings.
Workaround: To avoid errors, the ModelReferenceVersionMismatchMessage and ModelReferenceIOMismatchMessage settings should be set to “none” in the original model.
In addition, each subsystem must be able to determine a SampleTime for itself. Even if this is not necessary for the actual build.
To achieve this, you can either connect the subsystems to ports in the original model (with a defined SampleTime) or write the SampleTimes to one of the IN ports or to one of the signal sources in the subsystems.

– For enumeration fields, some names are not permitted, as the build process may then not be completed successfully: “Arp”, “CoSimConnection”, “COLD”, “PxcPreFilter”.

Restrictions on the conversion of referenced models to subsystems

– The conversion does not support a co-simulation block in the main model. You may only use this option if you do not set up a co-simulation block in the main model.

Restrictions on the PLCnext tab

– The “PLCnext” tab is available for MATLAB Simulink version ≥R2021b.

Restriction when creating function blocks for subsystems

Before building function blocks for subsystems, make sure there is no referenced model in the main model. If there is, start by converting the referenced model into a subsystem.

A Appendix

A 1 Demo mode

The PLCnext Target for Simulink software add-on is installed in demo mode.

The runtime of a model compiled in demo mode is limited to one hour. The model is then stopped without further warning. The wDiagCode diagnostic output switches to the final value 0xC900.

A 2 Supported data types (PLCnext controllers)

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

Table A-1 Supported data type combinations between C++, MATLAB Simulink, and PLCnext Engineer programs

C++	MATLAB Simulink	PLCnext Engineer	Used in data type Array	Used in data type Struct
bool	Boolean	BOOL	-*	x
int8_t	int8	SINT	x	x
uint8_t	uint8	USINT	x	x
int16_t	int16	INT	x	x
uint16_T	uint16	UINT	x	x
int32_t	int32	DINT	x	x
uint32_t	uint32	UDINT	x	x
int64_t	int64	LINT	x	x
uint64_t	uint64	ULINT	x	x
uint8_t	uint8	BYTE	x	x
uint16_t	uint16	WORD	x	x
uint32 _t	uint32	DWORD	x	x
uint64_t	uint64	LWORD	x	x
float	single	REAL	x	x
double	double	LREAL	x	x
char[]	string	StaticString	-	-
Array of primitive data types				x

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

B Appendix for document lists

B 1 List of figures

Section 4

Section 5

Section 6

Section 9

Section 11

Section 13

Section 14

Section 17

B 2 List of tables

Section 2

Section 5

Section 8

Section 9

Section 11

Section 17

Appendix A

B 3 Index

P

R

S

T

U

W

X

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 general Terms and Conditions of Phoenix Contact apply exclusively, in particular as concerns any warranty liability.

Phoenix Contact reserves the right to register its own intellectual property rights for the product identifications of Phoenix Contact products that are used here. Registration of such intellectual property rights by third parties is prohibited.

Other product identifications may be afforded legal protection, even where they may not be indicated as such.

How to contact us

Internet

Up-to-date information on Phoenix Contact products and our Terms and Conditions can be found on the Internet at:
phoenixcontact.com

Make sure you always use the latest documentation.
It can be downloaded at:
phoenixcontact.net/products

Subsidiaries

If there are any problems that cannot be solved using the documentation, please contact your Phoenix Contact subsidiary.
Subsidiary contact information is available at phoenixcontact.com.

Published by

PHOENIX CONTACT GmbH & Co. KG
Flachsmarktstraße 8
32825 Blomberg
GERMANY

PHOENIX CONTACT Development and Manufacturing, Inc.
586 Fulling Mill Road
Middletown, PA 17057
USA

Should you have any suggestions or recommendations for improvement of the contents and layout of our manuals, please send your comments to:
tecdoc@phoenixcontact.com

PHOENIX CONTACT GmbH & Co. KG
Flachsmarktstraße 8
32825 Blomberg, Germany
Phone: +49 5235 3-00
Fax: +49 5235 3-41200
E-mail: info@phoenixcontact.com
phoenixcontact.com

106366_en_16

INSPIRING INNOVATIONS