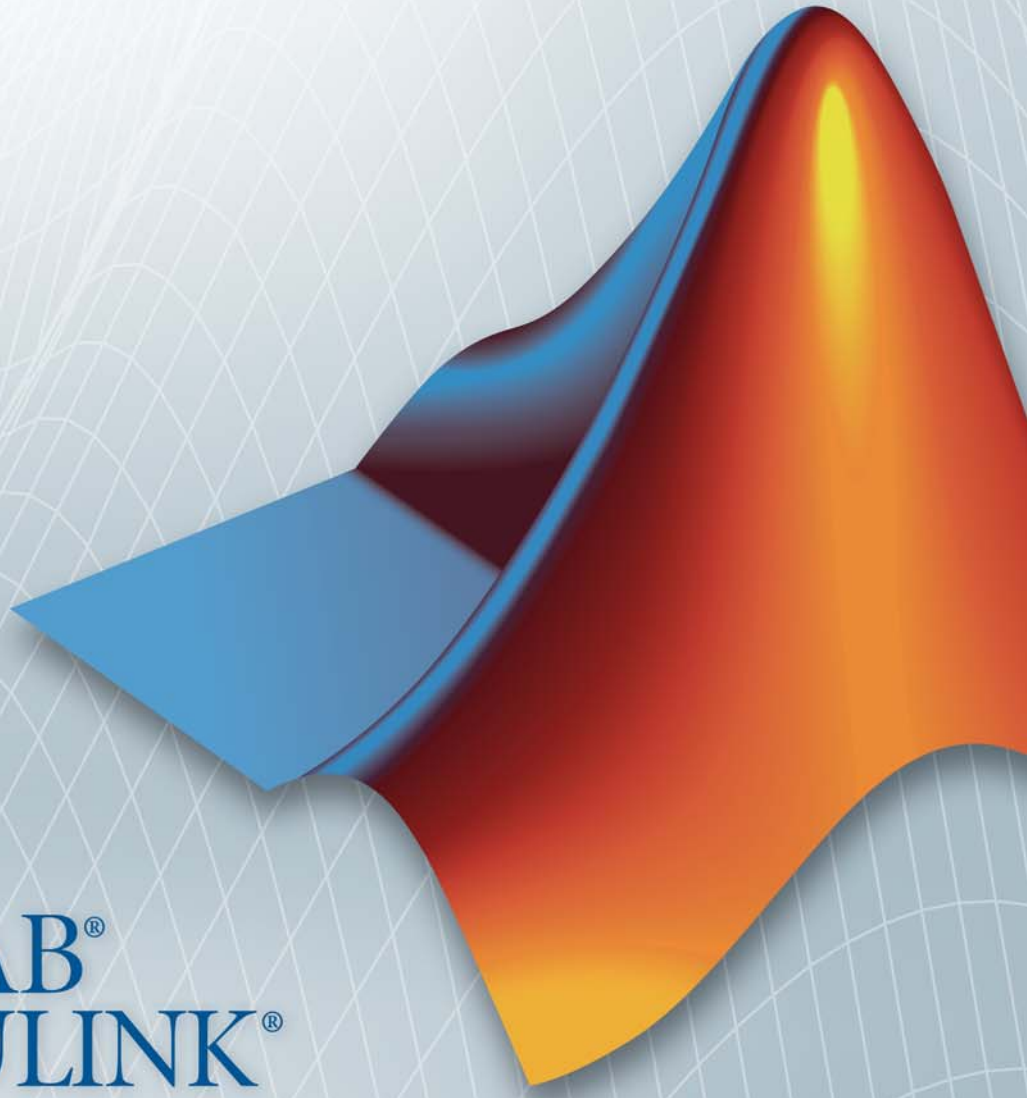
Simulink® PLC Coder™

User's Guide

R2012a



MATLAB®
&SIMULINK®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Simulink® PLC Coder™ User's Guide

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

March 2010	Online only	New for Version 1.0 (Release 2010a)
September 2010	Online only	Revised for Version 1.1 (Release 2010b)
April 2011	Online only	Revised for Version 1.2 (Release 2011a)
September 2011	Online only	Revised for Version 1.2.1 (Release 2011b)
March 2012	Online only	Revised for Version 1.3 (Release 2012a)

Contents

1 Getting Started

Product Description 1-2
Key Features 1-2

PLC Code Generation in the Development Process . . . 1-3

Expected Users 1-4

Glossary 1-5

Accessing Demos 1-6

Related Products 1-7
Requirements for the Simulink® PLC Coder™ Product . . . 1-7
Supported Simulink and Stateflow Blocks 1-7
System Requirements 1-8
Supported IDE Platforms 1-8

Basic Workflow 1-12

Preparing Your Model to Generate Structured Text Code 1-13
Solvers 1-13
Configuring Simulink Models for Structured Text Code Generation 1-13
Checking System Compatibility for Structured Text Code Generation 1-19

Generating and Examining Structured Text Code 1-23
Generating Structured Text Code from the Model Window 1-23
Generating Structured Text Code with the MATLAB Interface 1-30

Generating Structured Text Code and Integrating with Existing Siemens SIMATIC STEP 7 Projects 1-31

Automatically Importing Structured Text Code 1-33
PLC IDEs That Qualify for Importing Code Automatically 1-33
Generating and Automatically Importing Structured Text Code 1-34
Troubleshooting Automatic Import Issues 1-35

2 Mapping Simulink Semantics to Structured Text

How Simple Subsystem Code Maps to Function Blocks 2-2

How Reusable Subsystem Code Maps to Function Blocks 2-4

How Triggered Subsystem Code Maps to Function Blocks 2-6

How Stateflow Subsystem Code Maps to Function Blocks 2-8

How MATLAB® Coder™ Subsystem Code Maps to Function Blocks 2-10

How Alias Data Types Map in Generated Code 2-12

3 Generating Test Bench Code

Working with Generated Structured Text 3-2
How Test Bench Verification Works 3-2
Generated Files 3-2
Integrating Generated Code into Custom Code 3-2

Generate and Manually Import Test Bench Code 3-5

Automatically Importing Structured Text Code 3-9
PLC IDEs that Qualify for Importing Code Automatically 3-9
Automatically Importing to KW-Software MULTIPROG 5.0 and Phoenix Contact PC WORX 6.0 IDEs 3-10
Generating, Automatically Importing, and Verifying Structured Text 3-11

4 Working with Tunable Parameters in the Simulink® PLC Coder™ Environment

Configuring Tunable Parameters for Your Model 4-2
About Tunable Parameters in the Simulink® PLC Coder™ Environment 4-2
Configure Your Model for Tunable Parameters 4-4

Identifying Tunable Parameters 4-7

Tune Parameters Using Simulink.Parameter Objects 4-11
Work Directly with Simulink.Parameter Objects 4-11
Work with Simulink.Parameter Objects Using Model Explorer 4-14

Configure Tunable Parameters Using the Configuration Parameters Dialog Box 4-16

Defining Tunable Parameter Values in the MATLAB Workspace 4-16
Configuring Parameters to Be Tunable 4-18

5 Controlling Generated Code Partitions

Function Block Partitions 5-2
About Function Block Partitions 5-2
Example: One Function Block for Atomic Subsystems 5-2
Example: One Function Block for Virtual Subsystems . . . 5-3
Example: Multiple Function Blocks for Nonvirtual Subsystems 5-4
Controlling Generated Code Using Subsystem Block Parameters 5-5

6 Integrating Externally Defined Symbols

Integrate Externally Defined Symbols 6-2

Integrate Custom Function Block in Generated Code 6-3

7 IDE-Specific Considerations

Introduction 7-2

Considerations for All Target IDEs 7-3

Rockwell Automation RSLogix Considerations 7-4

Add-On Instruction and Function Blocks 7-4
Double-Precision Data Types 7-4
Unsigned Integer Data Types 7-4
Unsigned Fixed-Point Data Types 7-5
Enumerated Data Types 7-5

Siemens SIMATIC STEP 7 Considerations 7-6
Double-Precision Floating-Point Data Types 7-6
int8 and Unsigned Integer Types 7-6
Unsigned Fixed-Point Data Types 7-6
Enumerated Data Types 7-7

8 Supported Simulink and Stateflow Blocks

Simulink Blocks 8-2

Stateflow Blocks 8-12

9 Limitations

Coder Limitations 9-2
Current Limitations 9-2
Fixed-Point Data Type Limitations 9-3
Permanent Limitations 9-5

Block Restrictions 9-6
Simulink Block Support Exceptions 9-6
Stateflow Chart Exceptions 9-6
Reciprocal Sqrt Block 9-7
Lookup Table Blocks 9-7

Functions — Alphabetical List

10

Configuration Parameters for Simulink® PLC Coder™ Models

11

PLC Coder: General **11-2**
PLC Coder: General Tab Overview **11-3**
Target IDE **11-4**
Target IDE Path **11-6**
Code Output directory **11-8**
Generate testbench for subsystem **11-9**

PLC Coder: Comments **11-10**
Comments Overview **11-11**
Include comments **11-11**
Simulink block / Stateflow object comments **11-12**
Show eliminated blocks **11-13**

PLC Coder: Symbols **11-14**
Symbols Overview **11-15**
Maximum identifier length **11-16**
Use the same reserved names as Simulation Target **11-17**
Reserved names **11-18**
Externally Defined Symbols **11-19**

Index

1

Getting Started

- "Product Description" on page 1-2
- "PLC Code Generation in the Development Process" on page 1-3
- "Expected Users" on page 1-4
- "Glossary" on page 1-5
- "Accessing Demos" on page 1-6
- "Related Products" on page 1-7
- "Basic Workflow" on page 1-12
- "Preparing Your Model to Generate Structured Text Code" on page 1-13
- "Generating and Examining Structured Text Code" on page 1-23
- "Automatically Importing Structured Text Code" on page 1-33

Product Description

Generate IEC 61131 structured text for PLCs and PACs

Simulink® PLC Coder™ generates hardware-independent IEC 61131 structured text from Simulink models, Stateflow® charts, and Embedded MATLAB® functions. The structured text is generated in PLCopen and other file formats supported by widely used integrated development environments (IDEs). As a result, you can compile and deploy your application to numerous programmable logic controller (PLC) and programmable automation controller (PAC) devices.

Simulink PLC Coder generates test benches that help you verify the structured text using PLC and PAC IDEs and simulation tools. Support for industry standards is available through IEC Certification Kit (for IEC 61508 and IEC 61511).

Key Features

- Automatic generation of IEC 61131-3 structured text
- Simulink support, including reusable subsystems, PID controller blocks, and lookup tables
- Stateflow support, including graphical functions, truth tables, and state machines
- Embedded MATLAB support, including if-else statements, loop constructs, and math operations
- Support for multiple data types, including Boolean, integer, enumerated, and floating-point, as well as vectors, matrices, buses, and tunable parameters
- IDE support, including B&R Automation Studio®, PLCopen, Rockwell Automation® RSLogix™ 5000, Siemens® SIMATIC® STEP® 7, and Smart Software Solutions CoDeSys
- Test-bench creation

PLC Code Generation in the Development Process

Simulink PLC Coder software lets you generate IEC-61131-3 compliant structured text code from Simulink models. This software brings the Model-Based Design approach into the domain of PLC and PAC development. Using the coder, system architects and designers can spend more time fine-tuning algorithms and models through rapid prototyping and experimentation, and less time on coding PLCs.

Typically, you use a Simulink model to simulate a design for realization in a PLC. Once satisfied that the model meets design requirements, run the Simulink PLC Coder compatibility checker utility. This utility verifies compliance of model semantics and blocks for PLC target IDE code generation compatibility. Next, invoke the Simulink PLC Coder tool, using either the command line or the graphical user interface. The coder generates structured text code that implements the design embodied in the model.

Usually, you also generate a corresponding test bench. You can use the test bench with PLC emulator tools to drive the generated structured text code and evaluate its behavior.

The test bench feature increases confidence in the generated code and saves time spent on test bench implementation. The design and test process are fully iterative. At any point, you can return to the original model, modify it, and regenerate code.

At completion of the design and test phase of the project, you can easily export the generated Structure Text code to your PLC development environment. You can then deploy the code.

Expected Users

The Simulink PLC Coder product is a tool for control and algorithm design and test engineers in the following applications:

- PLC manufacturing
- Machine manufacturing
- Systems integration

You should be familiar with:

- MATLAB® and Simulink software and concepts
- PLCs
- Structured text language

If you want to download generated code to a PLC IDE, you should also be familiar with your chosen PLC IDE platform. See “Supported IDE Platforms” on page 1-8 for a list of these platforms.

Glossary

Term	Definition
PAC	Programmable automation controller.
PLC	Programmable logic controller.
IEC 61131-3	IEC standard that defines PLC coder languages, including the structured text language for which the Simulink PLC Coder software generates code.
PLCopen	Vendor- and product-independent organization that works with the IEC 61131-3 standard. The Simulink PLC Coder product can generate structured text using the PLCopen XML standard format. See `http://www.plcopen.org/pages/tc6_xml/xml_intro/index.htm` for details.
structured text	High-level textual language defined by IEC-61131-3 standard for the programming of PLCs.
function block	Structured text language programming concept that allows the encapsulation and reuse of algorithmic functionality.

Accessing Demos

The Simulink PLC Coder software provides demos in:

```
matlabroot\toolbox\plccoder\plccoderdemos
```

To see a list of available demos, in the MATLAB Command Window, type:

```
plccoderdemos
```

This command displays the Simulink PLC Coder demos page in the MATLAB Help browser. The MATLAB Help browser allows you to access the documentation and demo models for all the MathWorks® products that you have installed. To access any of these demos, select the name on the demo page. Some of the demos included with the product are:

Demo	Description
Generating Structured Text for a Simple Simulink Subsystem	Demonstrates the code generated for a simple subsystem consisting of basic Simulink blocks.
Generating Structured Text for a Hierarchical Simulink Subsystem	Demonstrates the code generated for a hierarchical subsystem consisting of other Simulink subsystems.
Generating Structured Text for a Reusable Simulink Subsystem	Demonstrates the code generated for a reusable subsystem consisting of basic Simulink blocks.
Generating Structured Text for a Stateflow Chart	Demonstrates the code generated for a Stateflow Chart block.
Generating Structured Text for a MATLAB Block	Demonstrates the code generated for a MATLAB Function block implementing tank valve control logic.

Related Products

In this section...

"Requirements for the Simulink® PLC Coder™ Product" on page 1-7
"Supported Simulink and Stateflow Blocks" on page 1-7
"System Requirements" on page 1-8
"Supported IDE Platforms" on page 1-8

Requirements for the Simulink PLC Coder Product

The Simulink PLC Coder product requires current versions of these products:

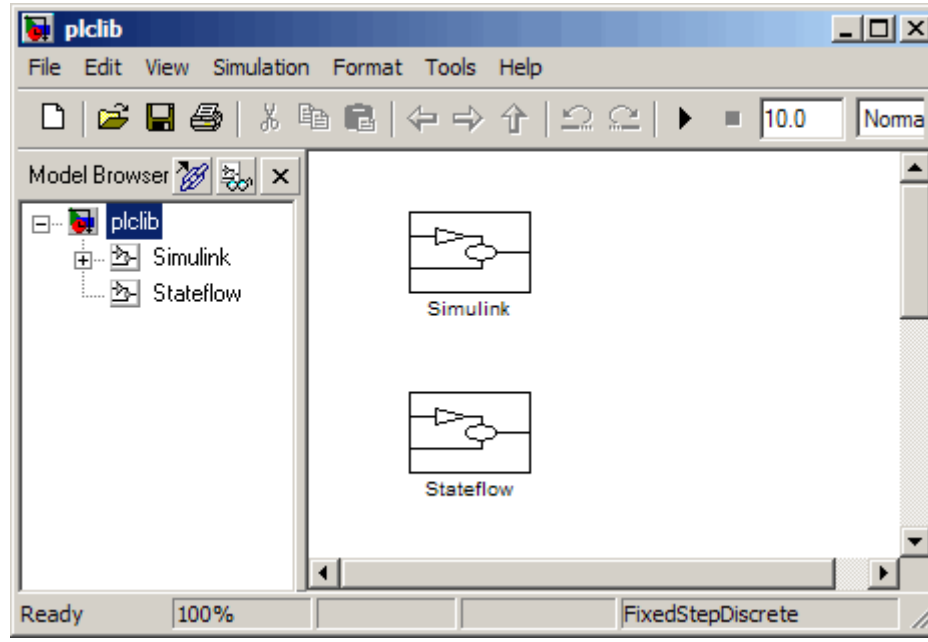
- MATLAB
- Simulink

The Stateflow product is recommended.

See the MathWorks Web site at Related Products for a list of related products.

Supported Simulink and Stateflow Blocks

To access a Simulink library of blocks that the Simulink PLC Coder software supports, type `plclib` in the MATLAB Command Window. The coder can generate structured text code for subsystems that contain these blocks. The library window is displayed.



This library contains two sublibraries, Simulink and Stateflow. Each sublibrary contains the blocks that you can include in a Simulink PLC Coder model.

See Chapter 8, "Supported Simulink and Stateflow Blocks" for a list of the supported blocks. See "Block Restrictions" on page 9-6 for restrictions on using these blocks.

System Requirements

Requirement	Description
32-bit or 64-bit operating system	Windows® platform supported by MathWorks

Supported IDE Platforms

The Simulink PLC Coder product supports the following IDE platforms:

- 3S-Smart Software Solutions CoDeSys Version 2.3 or 3.3
- B&R Automation Studio 3.0
- Beckhoff® TwinCAT® 2.11
- KW-Software MULTIPROG® 5.0

Note The Simulink PLC Coder software supports only the English version of KW-Software MULTIPROG target IDE.

- Phoenix Contact® PC WORX™ 6.0

Note The Simulink PLC Coder software supports only the English version of Phoenix Contact PC WORX target IDE.

- Rockwell Automation RSLogix 5000 Series Version 17 or 18

Note The Simulink PLC Coder software can generate code for Add-On instructions (AOIs) and routine code.

- Siemens SIMATIC STEP 7 Version 5.4

Note The Simulink PLC Coder software assumes that:

- English systems use English S7
- German systems use German S7

- Generic
- PLCopen XML

See the MathWorks Web site at Supported IDEs for a list of supported IDEs and platforms.

3S-Smart Software Solutions CoDeSys Software

To get CoDeSys Version 2.3 or 3.3, see:

```
http://www.3s-software.com/index.shtml?en_download
```

This download page requires you to be a registered user.

1. If you are not yet a registered user, create an account. It might take a few days to receive a password for the account.
2. When you receive a password, use it to access the download page.
3. On the download page, select the CoDeSys software to download.

   You do not need to download the CoDeSys SP RTE demo.
4. Follow CoDeSys download and installation instructions to install the software.

B&R Automation Studio 3.0 Software

To get the B&R Automation Studio product, see:

```
http://www.br-automation.com/cps/rde/xchg/br-productcatalogue-
/hs.xsl/cookies_allowed.htm?caller=products_5309_ENG_HTML.htm/
```

Beckhoff TwinCAT 2.11

To get the Beckhoff TwinCAT 2.11 product, see:

```
http://www.beckhoff.com/english.asp?twincat/default.htm
```

KW-Software MULTIPROG 5.0

To get the KW-Software MULTIPROG 5.0 product, see:

```
http://www.kw-software.com/com/index1024.html
```

Phoenix Contact PC WORX Version 6.0

To get the Phoenix Contact PC WORX Version 6.0 product, see:

http://www.phoenixcontact.com/automation/32131_31906.htm

Rockwell Automation RSLogix 5000 Software

To get the Rockwell Automation RSLogix 5000 product, see:

http://www.rockwellautomation.com/rockwellsoftware/design/-rslogix5000/

The coder can generate code for Rockwell Automation RSLogix 5000 Add-On instructions (AOIs) and routines.

Siemens SIMATIC STEP 7

To get the Siemens SIMATIC STEP 7 Version 5.4 product, see:

http://www.sea.siemens.com/us/Products/Automation/-Engineering-Software/step-7-pro/Pages/step-7-pro.aspx.

Basic Workflow

The basic workflow of Simulink PLC Coder users includes:

1. Define and design a Simulink model from which you want to generate code.
2. Identify the model components for which you want to generate code for importing to a PLC.
3. Place the components in a Subsystem block.
4. Identify your target PLC IDE.
5. Select a solver.
6. Configure the Subsystem block to be atomic.
7. Check that the model is compatible with the Simulink PLC Coder software.
8. Simulate your model.
9. Configure model parameters to generate code for your PLC IDE.
10. Examine the generated code.
11. Import code to your PLC IDE.

Preparing Your Model to Generate Structured Text Code

In this section...

"Solvers" on page 1-13

"Configuring Simulink Models for Structured Text Code Generation" on page 1-13

"Checking System Compatibility for Structured Text Code Generation" on page 1-19

Solvers

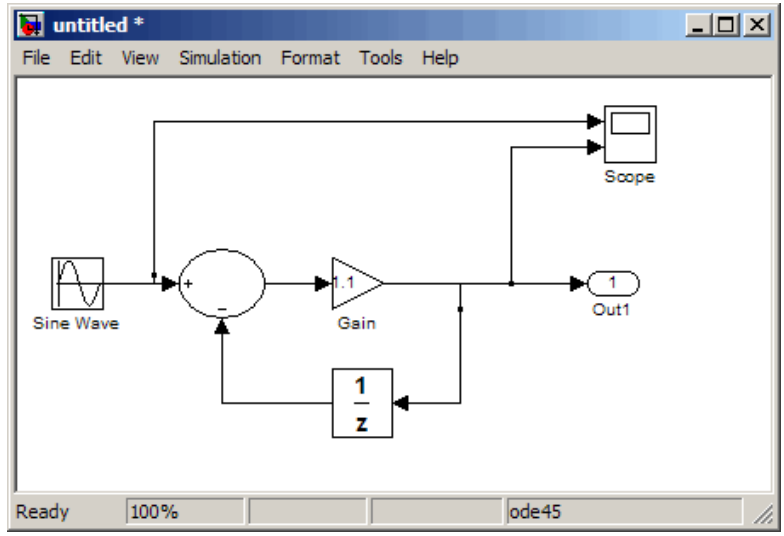
Choose a solver for your Simulink PLC Coder model.

Model	Solver Setting
Continuous	Use a continuous solver and configure a fixed sample time for the subsystem for which you generate code.
Discrete	Discrete fixed-step solver.

Configuring Simulink Models for Structured Text Code Generation

This topic assumes that you have a model for which you want to generate and import code to a PLC IDE. Before you use this model, perform the following steps.

1. In the MATLAB Command Window, open your model. For example:

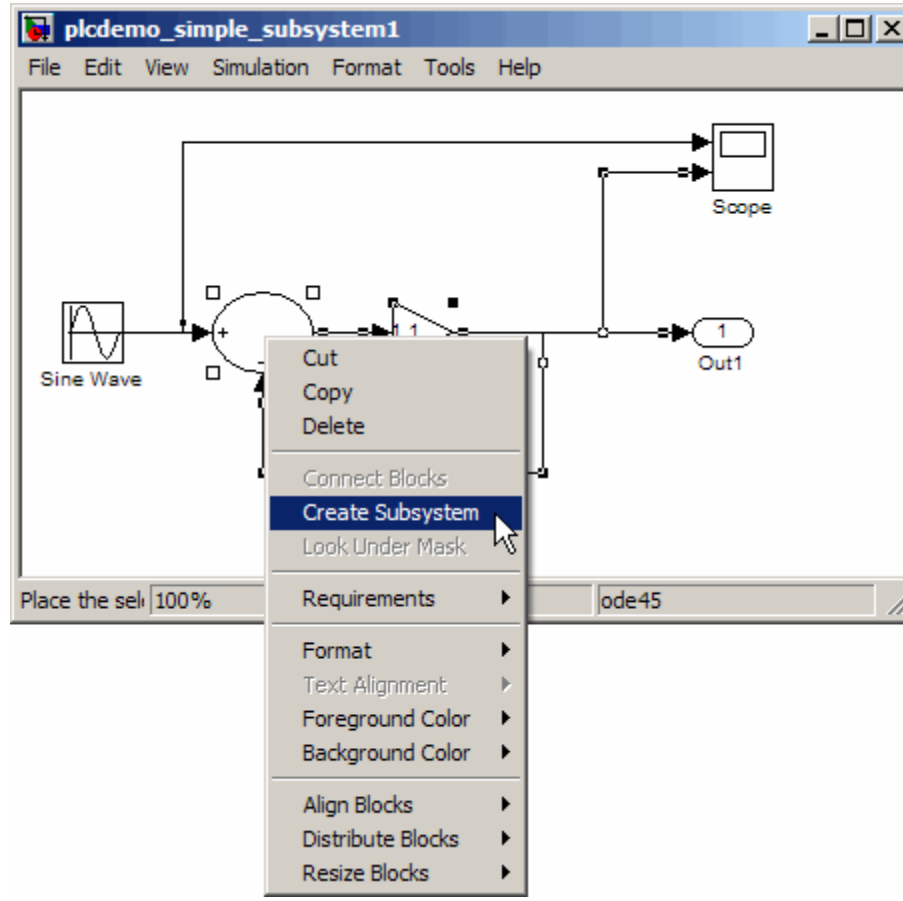


2 Configure the model to use the fixed-step discrete solver. To do this, select **Simulation > Configuration Parameters** and in the Solver pane, set **Type** to `Fixed-step` and **Solver** to `discrete (no continuous states)`.

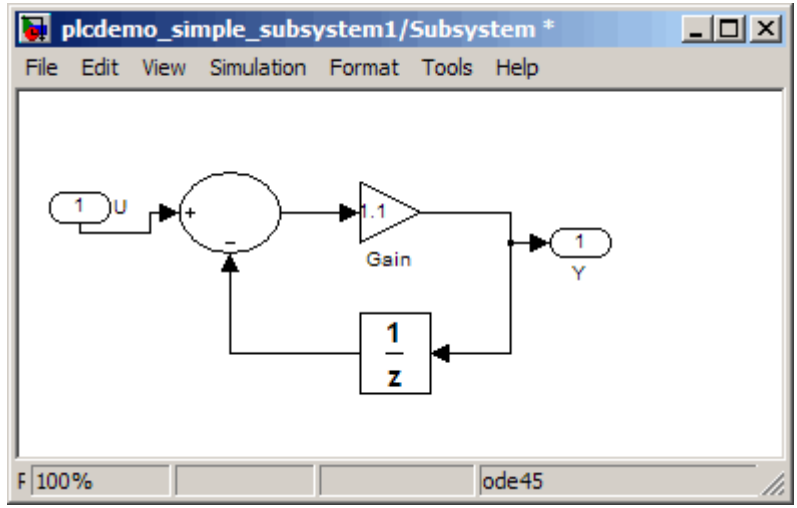
   If your model uses a continuous solver, has a subsystem, configure a fixed sample time for the subsystem for which you generate code.

3 Save this model as `plcdemo_simple_subsystem1.mdl`.

4 Place the components for which you want to generate structured text code in a subsystem. For example:

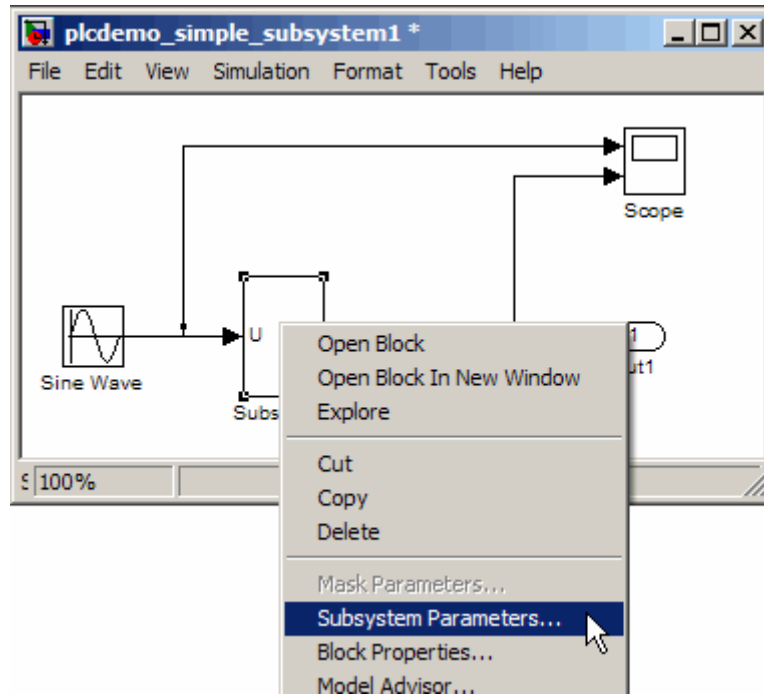


Optionally, rename In1 and Out1 to U and Y respectively. This operation results in a subsystem like the following:

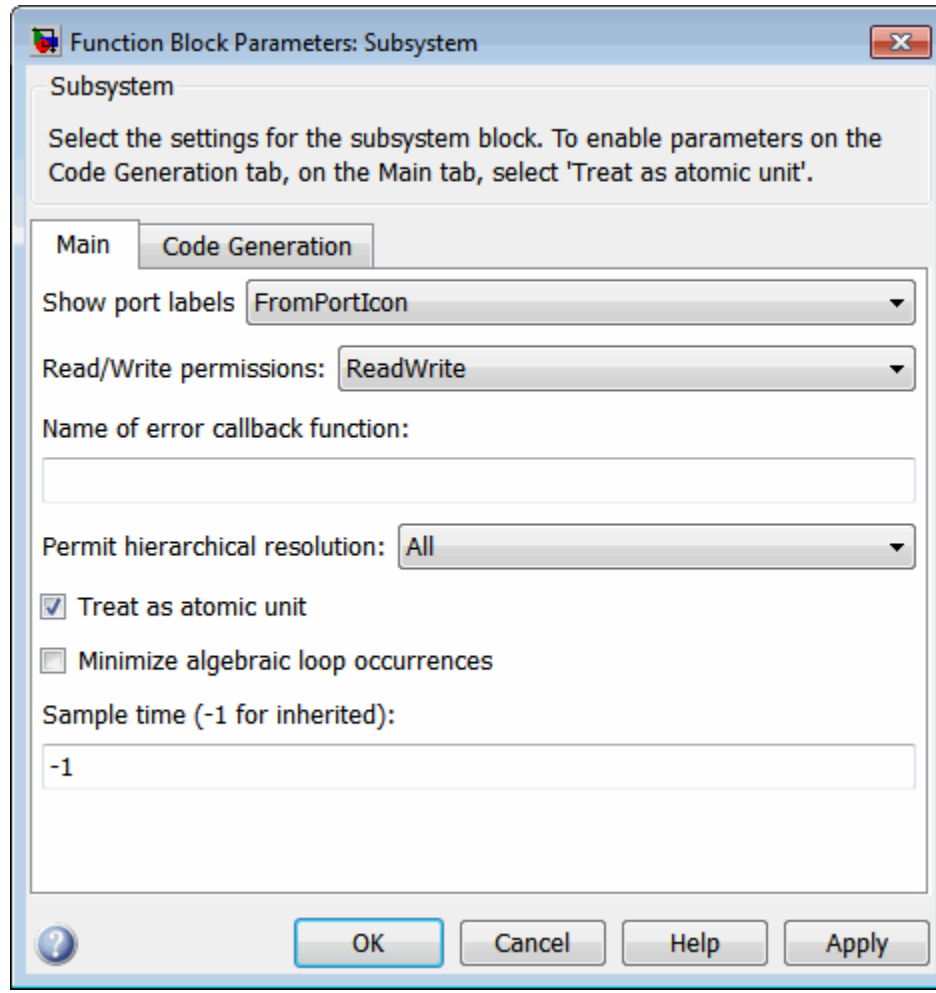


5 Save the subsystem.

6 In the top-level model, right-click the Subsystem block and select **Subsystem Parameters**.



7 In the resulting block dialog box, select **Treat as atomic unit**.



8 Click **OK**.

9 Simulate your model.

10 Save your model. In later procedures, you can use either this model, or the `plcdemo_simple_subsystem.mdl` model that comes with your software.

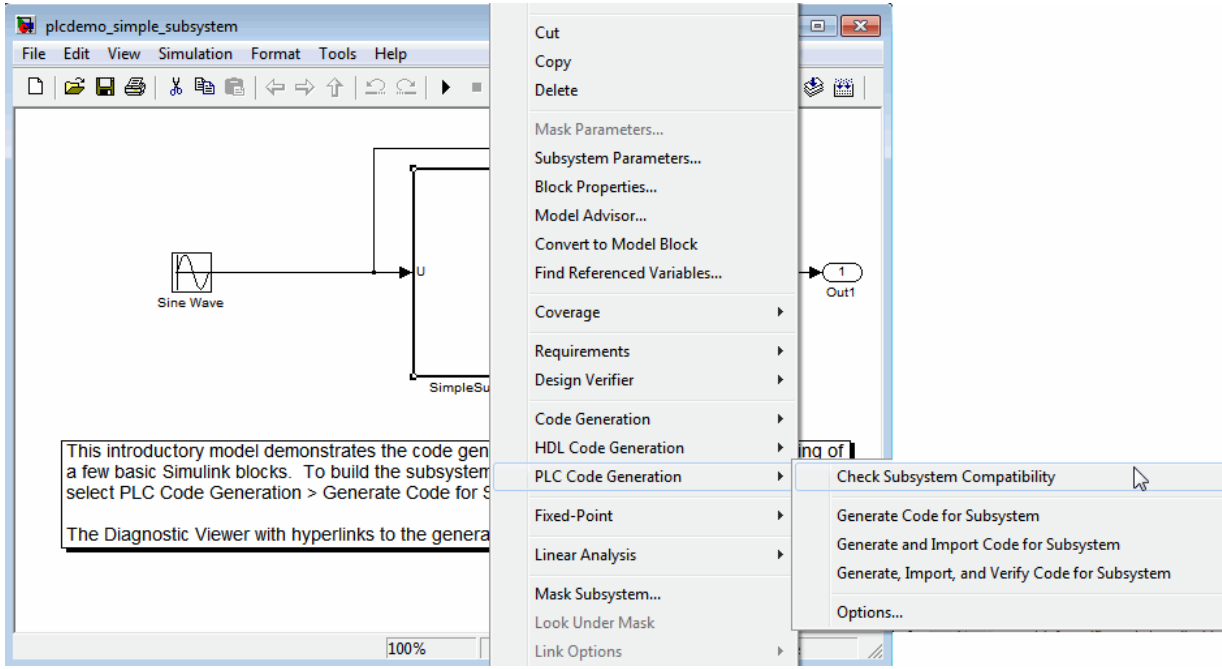
You are now ready to:

- Set up your subsystem to generate structured text code. See “Checking System Compatibility for Structured Text Code Generation” on page 1-19.
- Generate structured text code for your IDE. See “Generating and Examining Structured Text Code” on page 1-23.

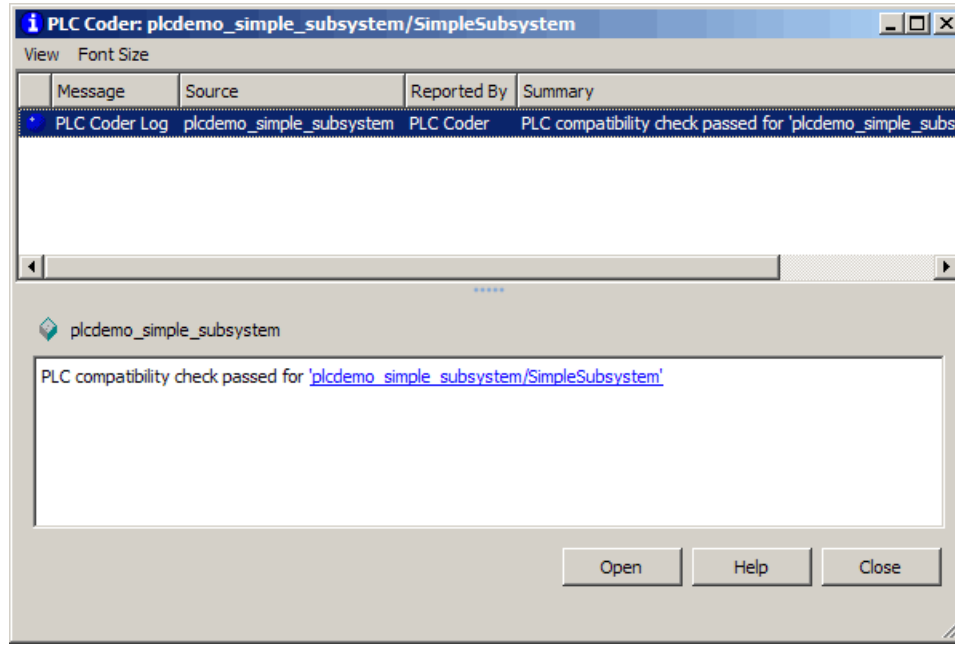
Checking System Compatibility for Structured Text Code Generation

This topic assumes that you have a model that you have configured to work with the Simulink PLC Coder software.

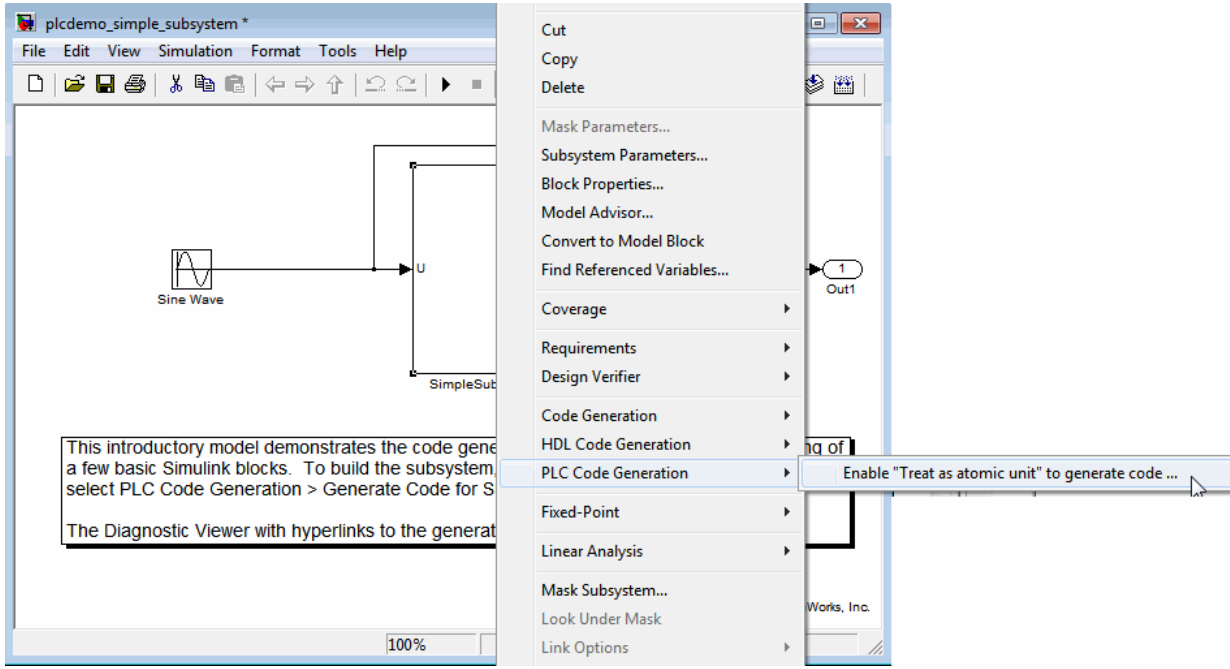
1. In your model, navigate to the subsystem for which you want to generate code.
2. Right-click that Subsystem block and select **PLC Code Generation > Check Subsystem Compatibility**.



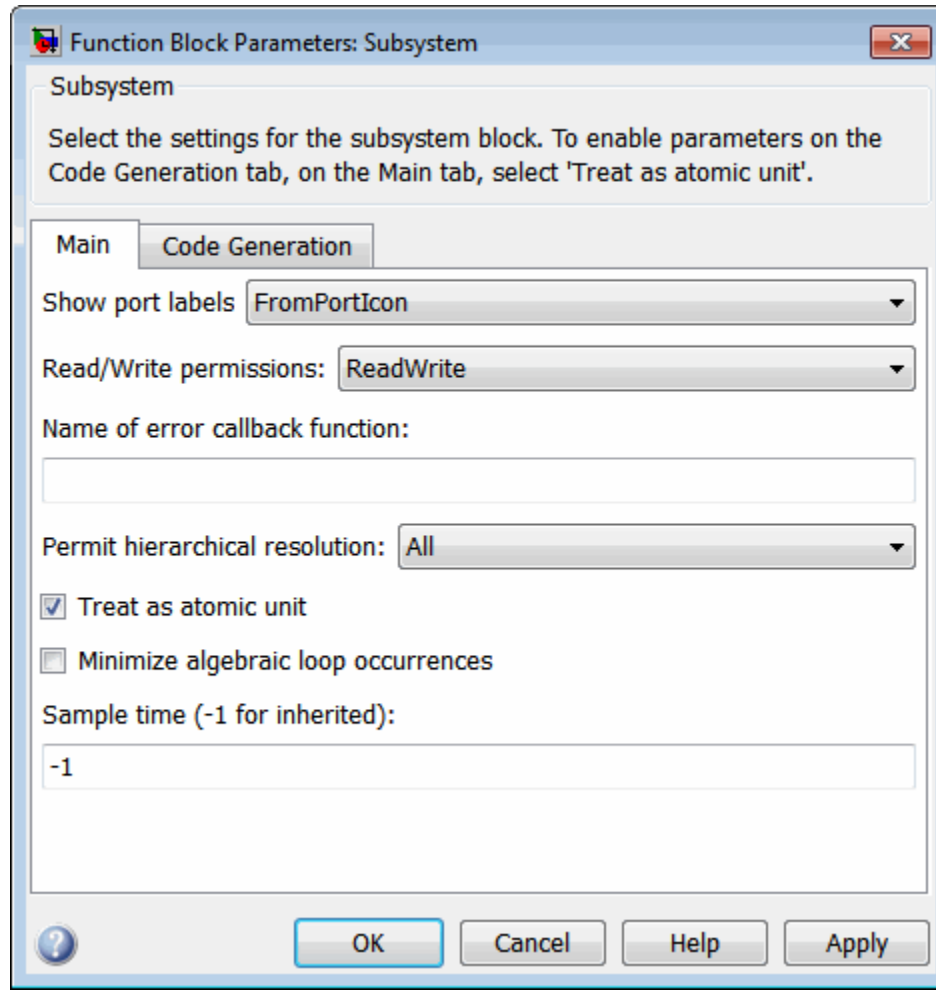
The coder verifies that your model satisfies the Simulink PLC Coder criteria and displays an information window when done.



If the subsystem is not atomic, right-clicking the Subsystem block and selecting **PLC Code Generation** prompts you to select **Enable "Treat as atomic unit" to generate code**.



This command opens the block parameter dialog box so that you can select the **Treat as atomic unit** check box.



You are now ready to generate structured text code for your IDE. See “Generating and Examining Structured Text Code” on page 1-23.

Generating and Examining Structured Text Code

In this section...

"Generating Structured Text Code from the Model Window" on page 1-23

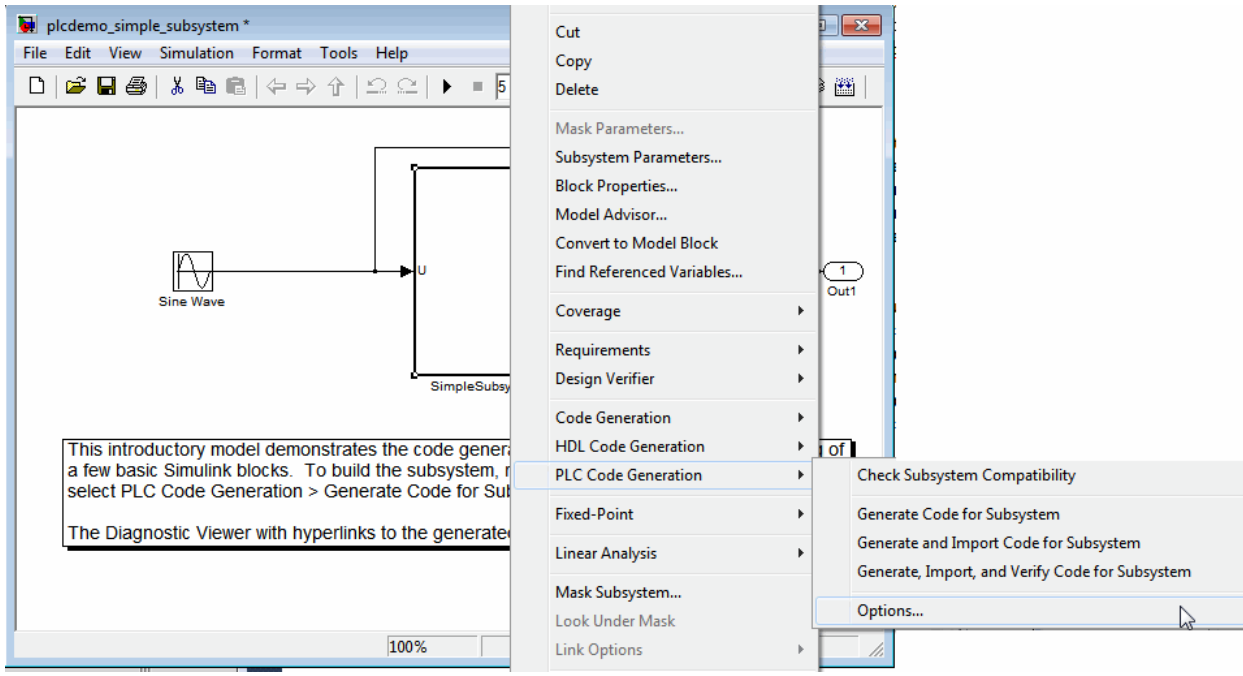
"Generating Structured Text Code with the MATLAB Interface" on page 1-30

"Generating Structured Text Code and Integrating with Existing Siemens SIMATIC STEP 7 Projects" on page 1-31

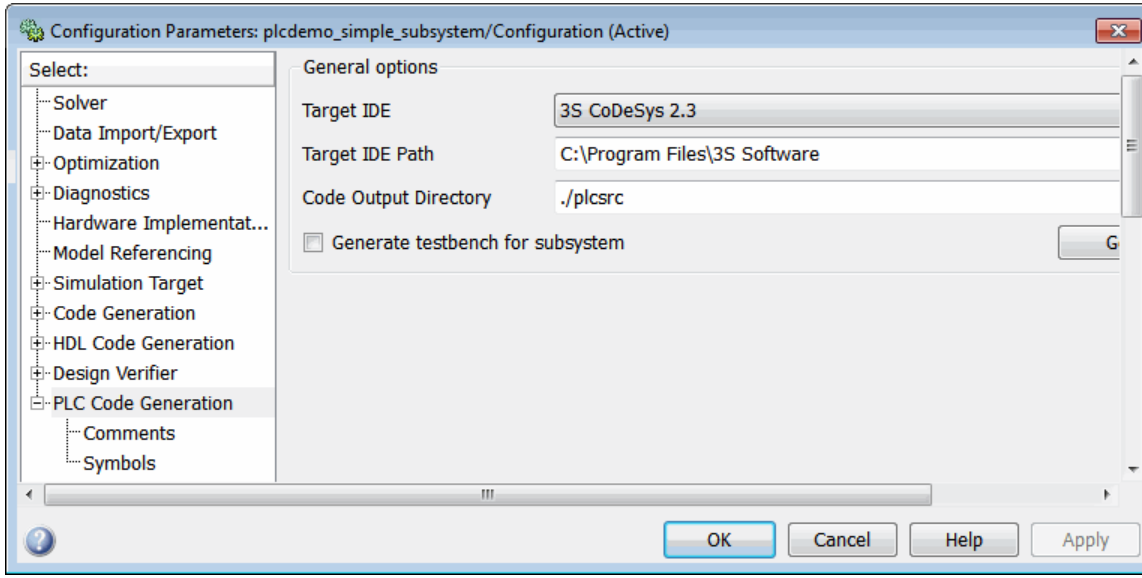
Generating Structured Text Code from the Model Window

This topic assumes that you have set up your environment and Simulink model to use the Simulink PLC Coder software to generate structured text code. If you have not yet done so, see "Preparing Your Model to Generate Structured Text Code" on page 1-13.

1. If you do not have the `plcdemo_simple_subsystem` model open, open it now.
2. Right-click the Subsystem block and select **PLC Code Generation > Options**.



The Configuration Parameters dialog box is displayed.



3 In **PLC Code Generation > General options > Target IDE**, select a target IDE. For example, select `CoDeSys 2.3`.

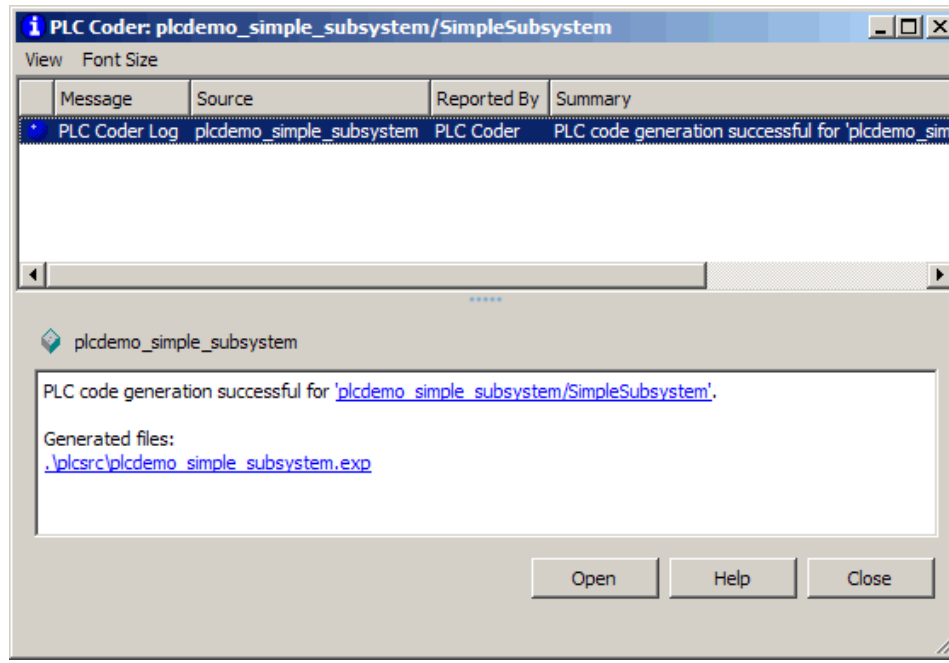
4 Click **Apply**.

5 Click the **Generate code** button.

This button:

- Generates structured text code (same as the **PLC Code Generation > Generate Code for Subsystem** option)
- Stores generated code in *`model_name`*`.exp` (for example, `plcdemo_simple_subsystem.exp`)

When code generation is complete, an information window is displayed.



This window has links that you can click to open the associated files.

The Simulink PLC Coder software generates structured text code and stores it according to the target IDE platform. These platform-specific paths are default locations for the generated code. To customize generated file names, see "Specifying Custom Names for Generated Files" on page 1-30.

Platform	Generated Files
3S-Smart Software Solutions CoDeSys 2.3	*current_folder*\plcsrc\model_name.exp — Structured text file for importing to the target IDE.
3S-Smart Software Solutions CoDeSys 3.3	*current_folder*\plcsrc\model_name.xml — Structured text file for importing to the target IDE.

Platform	Generated Files
B&R Automation Studio IDE	The following files in *current_folder*\plcsrc\model_name — Files for importing to the target IDE: • Package.pkg — (If test bench is generated) Top-level package file for function blocks library and test bench main program in XML format. In the main folder (if test bench is generated): • IEC.prg — Test bench main program definition file in XML format. • mainInit.st — Text file. Test bench init program file in structured text. • mainCyclic.st — Text file. Test bench cyclic program file in structured text. • mainExit.st — Text file. Test bench exit program file in structured text. • main.typ — Text file. Main program type definitions file in structured text. • main.var — Text file. Main program variable definitions file in structured text.
Beckhoff TwinCAT 2.11	*current_folder*\plcsrc\model_name.exp — Structured text file for importing to the target IDE.
KW-Software MULTIPROG 5.0	*current_folder*\plcsrc\model_name.xml — Structured text file, in XML format, for importing to the target IDE.
Phoenix Contact PC WORX 6.0	*current_folder*\plcsrc\model_name.xml — Structured text file, in XML format, for importing to the target IDE.
Rockwell Automation RSLogix 5000 IDE: AOI	*current_folder*\plcsrc\model_name.L5X — (If test bench is generated) Structured text file for importing to the target IDE using Add-On Instruction (AOI) constructs. This file is in XML format and contains the generated structured text code for your model.

Platform	Generated Files
Rockwell Automation RSLogix 5000 IDE: Routine	*current_folder*\plcsrc\model_name.L5X — (If test bench is generated) Structured text file for importing to the target IDE using routine constructs. This file is in XML format and contains the generated structured text code for your model. In *current_folder*\plcsrc\model_name (if test bench is not generated), the following files are generated: • *subsystem_block_name*.L5X — Structured text file in XML format. Contains program tag and UDT type definitions and the routine code for the top-level subsystem block. • *routine_name*.L5X — Structured text files in XML format. Contains routine code for other subsystem blocks.
Siemens SIMATIC STEP 7 IDE	*current_folder*\plcsrc\model_name\model_name.scl — Structured text file for importing to the target IDE. *current_folder*\plcsrc\model_name\model_name.asc — (If test bench is generated) Text file. Structured text file and symbol table for generated test bench code.
Generic	*current_folder*\plcsrc\model_name.st — Pure structured text file. If your target IDE is not available for the Simulink PLC Coder product, consider generating and importing a generic structured text file.
PLCopen XML	*current_folder*\plcsrc\model_name.xml — Structured text file formatted using the PLCopen XML standard. If your target IDE is not available for the Simulink PLC Coder product, but uses a format like this standard, consider generating and importing a PLCopen XML structured text file.

The example in this topic illustrates generated code for the CoDeSys Version 2.3 PLC IDE. Generated code for other platforms, such as Rockwell Automation RSLogix 5000, is in XML or other format and looks different.

```
FUNCTION_BLOCK SimpleSubsystem
VAR_INPUT
    ssMethodType: SINT;
    U: LREAL;
END_VAR
VAR_OUTPUT
    Y: LREAL;
END_VAR
VAR
    UnitDelay_DSTATE: LREAL;
END_VAR
VAR_TEMP
    rtb_Gain: LREAL;
END_VAR
CASE ssMethodType OF
    SS_INITIALIZE:
        (* InitializeConditions for UnitDelay: '<S1>/Unit Delay'  *)
        UnitDelay_DSTATE := 0;

    SS_OUTPUT:
        (* Gain: '<S1>/Gain' incorporates:
         *  Inport: '<Root>/U'
         *  Sum: '<S1>/Sum'
         *  UnitDelay: '<S1>/Unit Delay'
          *)
        rtb_Gain := (U - UnitDelay_DSTATE) * 0.5;

        (* Outport: '<Root>/Y'  *)
        Y := rtb_Gain;

        (* Update for UnitDelay: '<S1>/Unit Delay'  *)
        UnitDelay_DSTATE := rtb_Gain;

END_CASE;
END_FUNCTION_BLOCK
```

After generating structured text code, examine it. If your model has author names, creation dates, and model descriptions, the generated code contains these items in the header comments. The header also lists fundamental sample times for the model and the subsystem block for which you generate code.

For a description of how the generated code for the Simulink components map to structured text components, see Chapter 2, “Mapping Simulink Semantics to Structured Text”.

If you are confident that the generated structured text is good, optionally change your workflow to automatically generate and import code to the target IDE. For more information, see “Automatically Importing Structured Text Code” on page 1-33.

Specifying Custom Names for Generated Files

To specify a different name for the generated files, set the **Function name options** parameter in the Subsystem block:

1. Right-click the Subsystem block for which you want to generate code and select `Subsystem Parameters`.
2. In the Main tab, select the **Treat as atomic unit** check box.
3. Click the **Code Generation** tab.
4. From the **Function Packaging** parameter list, select either `Function` or `Reusable Function`.

   These options enable the **Function name options** and **File name options** parameters.
5. Select the option that you want to use for generating the file name:

Function name options	Generated File Name
`Auto`	Default. Uses the model name, as listed in "Preparing Your Model to Generate Structured Text Code" on page 1-13, for example, `plcdemo_simple_subsystem`.
`Use subsystem name`	Uses the subsystem name, for example, `SimpleSubsystem`.
`User specified`	Uses the custom name that you specify in the **Function name** parameter, for example, `SimpleSubsystem`.

Generating Structured Text Code with the MATLAB Interface

You can generate structured text code for a subsystem from the MATLAB Command Window with the `plcgeneratecode` function. The function assumes that you have configured the parameters for the model, or that you

want to use the default settings. For example, to open the Configuration Parameters dialog box for the subsystem, type:

```
plcopenconfigset('plcdemo_simple_subsystem/Simple_Subsystem')
```

Configure the subsystem as described in "Generating Structured Text Code from the Model Window" on page 1-23.

To generate the code for the subsystem, type:

```
generatedfiles = plcgeneratecode('plcdemo_simple_subsystem/Simple_Subsystem')
```

Generating Structured Text Code and Integrating with Existing Siemens SIMATIC STEP 7 Projects

This topic describes a workflow to integrate generated code into an existing Siemens SIMATIC STEP 7 project.

This topic assumes that:

- You have generated code for the Siemens SIMATIC STEP 7 target IDE. If you have not yet done so, see "Generating Structured Text Code from the Model Window" on page 1-23.
- You have a Siemens SIMATIC STEP 7 project into which you want to integrate the generated code.

1. In the Siemens SIMATIC STEP 7 project, right-click **Sources** and select **Insert New Object > External Source**.

   A browser window is displayed.

2. In the browser window, navigate to the folder that contains the Simulink PLC Coder generated code you want to integrate.

3. In this folder, select *`model_name`*`.scl`, then click **OK**.

   A new entry named *`model_name`* appears in the **Sources** folder.

4. In the **Sources** folder, double-click *`model_name`*.

The generated code is listed in the SCL editor window.

5 In the SCL editor window, select **Options > Customize**.

The customize window is displayed.

6 In the customize window, select **Create block numbers automatically**.

7 Click **OK**.

This action enables the software to generate automatically the symbol addresses for Subsystem blocks.

8 In the SCL editor window, compile the *model_name*.scl file for the Subsystem block.

The new Function Block is now integrated and available for use with the existing Siemens SIMATIC STEP 7 project.

Automatically Importing Structured Text Code

In this section...

"PLC IDEs That Qualify for Importing Code Automatically" on page 1-33

"Generating and Automatically Importing Structured Text Code" on page 1-34

"Troubleshooting Automatic Import Issues" on page 1-35

PLC IDEs That Qualify for Importing Code Automatically

If you are confident that your model produces structured text that does not require visual examination, you can generate and automatically import structured text code to one of the following target PLC IDEs:

- CoDeSys Version 2.3
- KW-Software MULTIPROG Version 5.0
- Phoenix Contact PC WORX Version 6.0
- Rockwell Automation RSLogix 5000 Version 17 or 18
- Siemens SIMATIC STEP 7 Version 5.4 only for the following versions:
  - Siemens SIMATIC Manager: Version V5.4+SP5+HF1, Revision K5.4.5.1
  - S7-SCL: Version V5.3+SP5, Revision K5.3.5.0
  - S7-PLCSIM: Version V5.4+SP3, Revision K5.4.3.0

This topic describes how to work with the default CoDeSys Version 2.3 IDE. The procedure should work without additional changes for the other supported PLC IDEs, with the exception of the KW-Software MULTIPROG 5.0 and Phoenix Contact PC WORX 6.0 IDE. For notes on how to automatically import structured text code to these IDEs, see "Automatically Importing to KW-Software MULTIPROG 5.0 and Phoenix Contact PC WORX 6.0 IDEs" on page 3-10.

Generating and Automatically Importing Structured Text Code

You can generate and automatically import structured text code. Before you start:

- In the target IDE, save any current project.
- Close all open projects.
- Close the target IDE and all target IDE-related windows.

Note While the automatic import process is in progress, do not touch your mouse or keyboard. Doing so might disrupt the automatic import process. You can resume normal operations when the process completes.

The following procedure assumes that you have installed your target PLC IDE in a default location and uses the CoDeSys V2.3 IDE. If you installed the target PLC IDE in a nondefault location, open the Configuration Parameters dialog box. In the PLC Coder node, set the **Target IDE Path** parameter to the installation folder of your PLC IDE. See "Target IDE Path" on page 11-6 for more details.

1. If it is not already started, start the MATLAB Command Window.
2. Open the `plcdemo_simple_subsystem` model.
3. Right-click the Subsystem block and select **PLC Code Generation > Generate and Import Code for Subsystem**.

   The software then:

   a. Generates the code.

   b. Starts the target IDE interface.

   c. Creates a new project.

   d. Imports the generated code to the target IDE.

If you want to generate, import, and run the structured text code, see "Automatically Importing Structured Text Code" on page 3-9.

Troubleshooting Automatic Import Issues

This topic describes guidelines, hints, and tips for questions or issues you might have while using the automatic import capability of the Simulink PLC Coder product.

Supported Target IDEs

The Simulink PLC Coder software supports only the following versions of target IDEs for automatic import and verification:

- 3S-Smart Software Solutions CoDeSys Version 2.3
- KW-Software MULTIPROG 5.0 (English)
- Phoenix Contact PC WORX 6.0 (English)
- Rockwell Automation RSLogix 5000 Series Version 17 or 18 (English)

  For the Rockwell Automation RSLogix routine format, you must generate testbench code for automatic import and verification.
- Siemens SIMATIC STEP 7 Version 5.4 (English and German)

Unsupported Target IDEs

The following target IDEs currently do not support automatic import. For these target IDEs, the automatic import menu items (**Generate and Import Code for Subsystem** and **Generate, Import, and Verify Code for Subsystem**) are disabled.

- 3S-Smart Software Solutions CoDeSys Version 3.3
- B&R Automation Studio IDE
- Beckhoff TwinCAT 2.11
- Generic
- PLCopen

Possible Automatic Import Issues

When the Simulink PLC Coder software fails to finish automatically importing for the target IDE, it reports an issue in a message dialog box. To remedy issue, try the following actions:

- Check that the coder supports the target IDE version and language setting combination.
- Check that you have specified the target IDE path in the subsystem Configuration Parameters dialog box.
- Close any currently open projects in the target IDE, close the target IDE completely, and try again.
- Some target IDEs can have issues supporting the large data sets the coder test bench generates. In these cases, try to shorten the simulation cycles to reduce the data set size, then try the automatic import again.
- Other applications can interfere with automatic importing to a target IDE. Try to close other unrelated applications on the system and try the automatic import again.

2

Mapping Simulink Semantics to Structured Text

When you examine generated code, you evaluate how well the Simulink PLC Coder software has generated code from your model. The following topics describe how the coder maps Simulink subsystem semantics to function block semantics in structured text. As examples, the topics describe the mapping in the context of the different subsystem types that Simulink supports. The examples assume that you have already generated code (see "Generating Structured Text Code from the Model Window" on page 1-23). These topics use code generated with CoDeSys Version 2.3. All demos are located in the *matlabroot*`\toolbox\plccoder\plccoderdemos` folder.

- "How Simple Subsystem Code Maps to Function Blocks" on page 2-2
- "How Reusable Subsystem Code Maps to Function Blocks" on page 2-4
- "How Triggered Subsystem Code Maps to Function Blocks" on page 2-6
- "How Stateflow Subsystem Code Maps to Function Blocks" on page 2-8
- "How MATLAB® Coder™ Subsystem Code Maps to Function Blocks" on page 2-10
- "How Alias Data Types Map in Generated Code" on page 2-12

How Simple Subsystem Code Maps to Function Blocks

This topic assumes that you have generated structured text code from a Simulink model. If you have not yet done so, see "Generating Structured Text Code from the Model Window" on page 1-23.

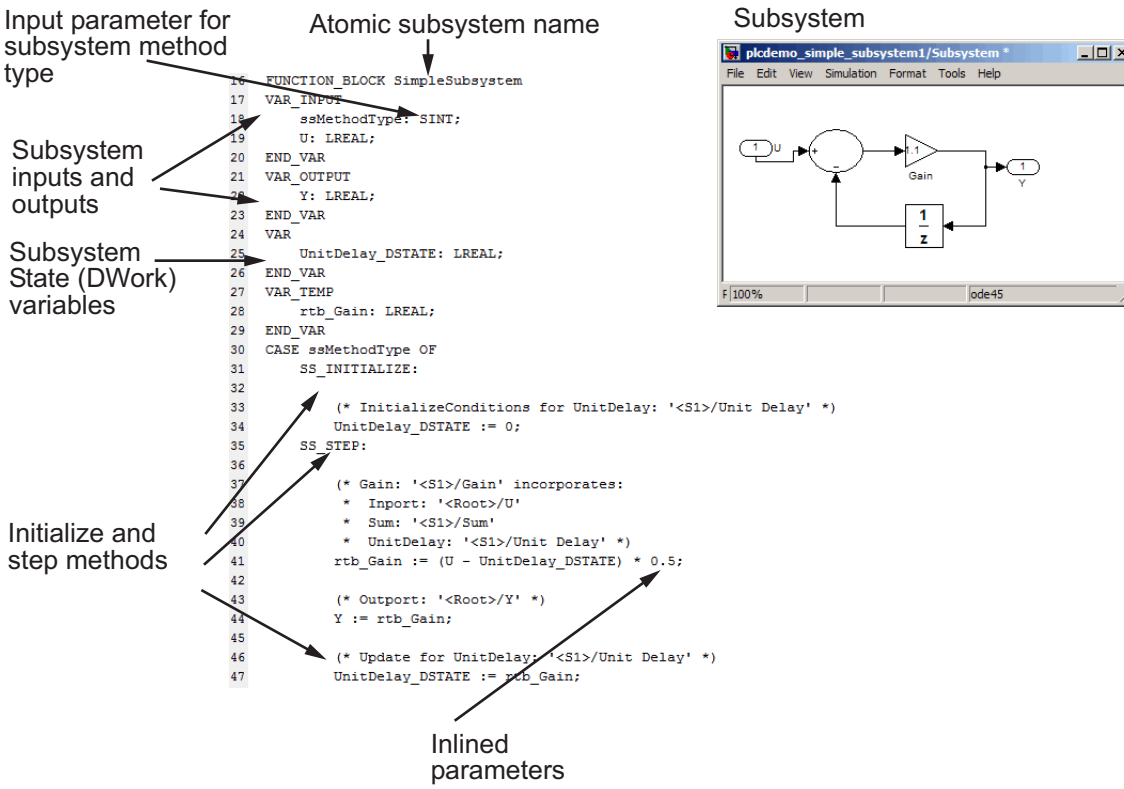
The example in this topic shows generated code for the CoDeSys Version 2.3 IDE. Generated code for other IDE platforms looks different.

1. If you do not have the `plcdemo_simple_subsystem.exp` file open, open it in the MATLAB editor. In the folder that contains the file, type:

   ```
   edit plcdemo_simple_subsystem.exp
   ```

   A file like the following is displayed.

   The following figure illustrates the mapping of the generated code to structured text components for a simple Simulink subsystem. The Simulink subsystem corresponds to the structured text function block, `Subsystem`.



2 Inspect this code as you ordinarily do for PLC code. Check the generated code.

How Reusable Subsystem Code Maps to Function Blocks

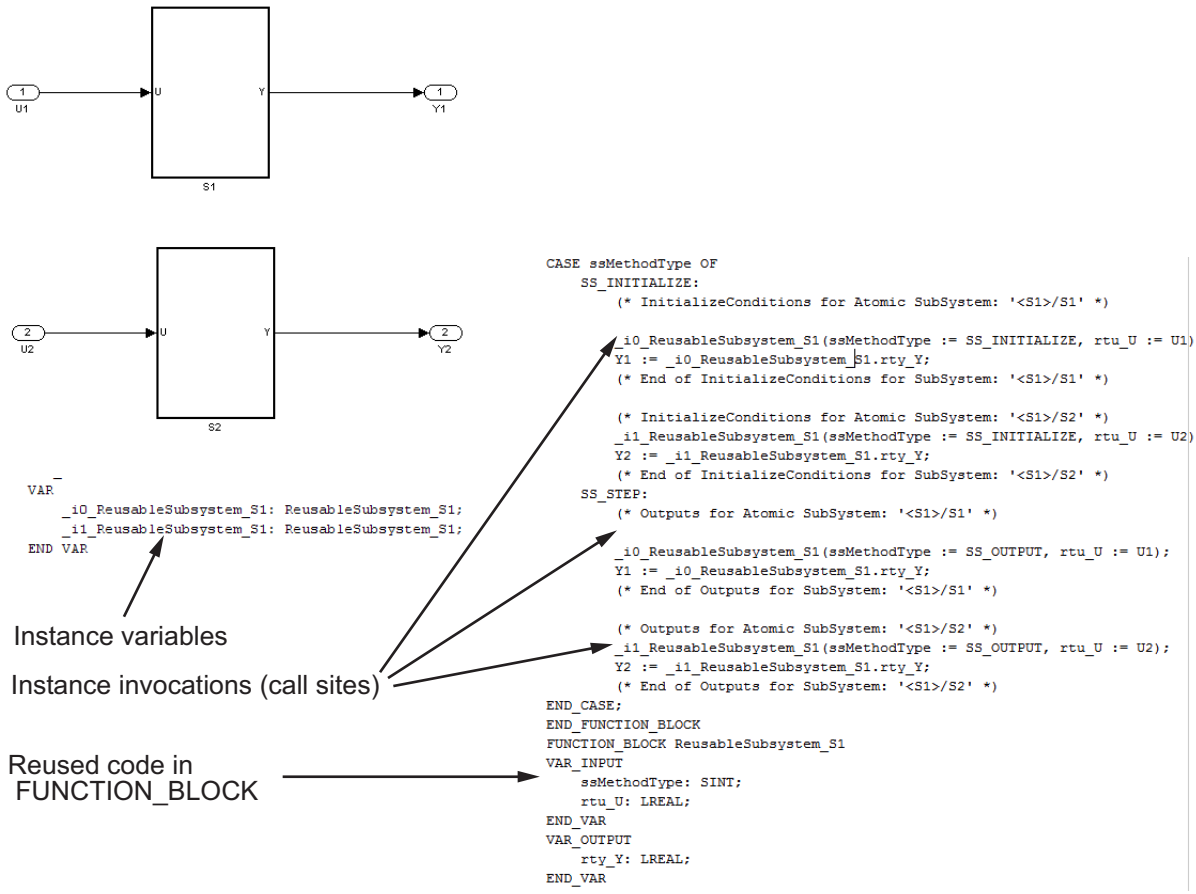
This topic assumes that you have generated structured text code from a Simulink model. If you have not yet done so, see "Generating Structured Text Code from the Model Window" on page 1-23.

The example in this topic shows generated code for the CoDeSys Version 2.3 IDE. Generated code for other IDE platforms looks different.

1. Open the `plcdemo_reusable_subsystem` model.
2. Right-click the Subsystem block and select **PLC Coder > Generate Code for Subsystem**.

   The Simulink PLC Coder software generates structured text code and places it in *`current_folder`*`/plcsrc/plcdemo_reusable_subsystem.exp`.
3. If you do not have the `plcdemo_reusable_subsystem.exp` file open, open it in the MATLAB editor.

   The following figure illustrates the mapping of the generated code to structured text components for a reusable Simulink subsystem . This graphic contains a copy of the hierarchical subsystem, ReusableSubsystem. This subsystem contains two identical subsystems, S1 and S2. This configuration enables code reuse between the two instances (look for the `ReusableSubsystem` string in the code).



4 Examine the generated structured text code. The code defines `FUNCTION_BLOCK ReusableSubsystem_S1` once.

Look for two instance variables that correspond to the two instances declared inside the parent `FUNCTION_BLOCK ReusableSubsystem` (`_instance_ReusableSubsystem_S1_1: ReusableSubsystem_S1` and `_instance_ReusableSubsystem_S1_0: ReusableSubsystem_S1`). The code invokes these two instances separately by passing in different inputs. The code invokes the outputs per the Simulink execution semantics.

How Triggered Subsystem Code Maps to Function Blocks

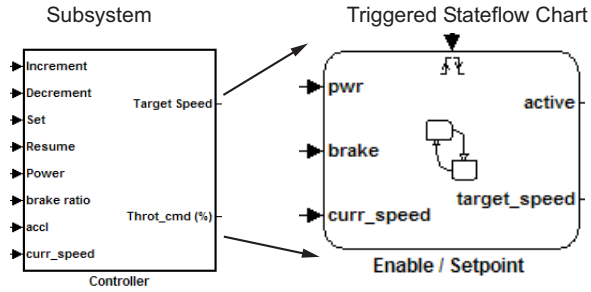
This topic assumes that you have generated structured text code from a Simulink model. If you have not yet done so, see “Generating Structured Text Code from the Model Window” on page 1-23.

The example in this topic shows generated code for the CoDeSys Version 2.3 PLC IDE. Generated code for other IDE platforms looks different.

1. Open the `plcdemo_cruise_control` model.
2. Right-click the Controller subsystem block and select **PLC Coder > Generate Code for Subsystem**.

   The Simulink PLC Coder software generates structured text code and places it in *`current_folder`*`/plcsrc/plcdemo_cruise_control.exp`.
3. If you do not have the `plcdemo_cruise_control.exp` file open, open it in the MATLAB editor.

   The following figure illustrates the mapping of the generated code to structured text components for a triggered Simulink subsystem . The first part of the figure shows the Controller subsystem and the triggered Stateflow chart that it contains. The second part of the figure shows excerpts of the generated code. Notice the zero-crossing functions that implement the triggered subsystem semantics.



Generated code

```
    EnableSetpoint_Trig_ZCE: ARRAY [0..6] OF USINT := 3,3,3,3,3,3,3;
    _i0ZZANY_ZERO_CROSSING: _ANY_ZERO_CROSSING;
END_VAR
...
...
...
    SS_STEP:

        (* DiscretePulseGenerator: '<S1>/Pulse Generator' *)
        IF (clockTickCounter < 1) AND (clockTickCounter >= 0) THEN
            temp1 := 1;
        ELSE
            temp1 := 0;
        END_IF;
        rtb_PulseGenerator := temp1;
        IF clockTickCounter >= 1 THEN
            clockTickCounter := 0;
        ELSE
            clockTickCounter := clockTickCounter + 1;
        END_IF;
        (* End of DiscretePulseGenerator: '<S1>/Pulse Generator' *)

        (* Chart: '<S1>/Enable // Setpoint ' incorporates:
         *  TriggerPort: '<S2>/ input events ' *)
        _i0ZZANY_ZERO_CROSSING(inState := USINT_TO_DINT(EnableSetpoint_Trig_ZCE[0]), i
        lEventVar := _i0ZZANY_ZERO_CROSSING.outEvent;
        EnableSetpoint_Trig_ZCE[0] := DINT_TO_USINT(_i0ZZANY_ZERO_CROSSING.outState);
        zcEvent[0] := lEventVar;
        (* Inport: '<Root>/Increment' *)
...
...
...
FUNCTION_BLOCK _ANY_ZERO_CROSSING
...
...
...
END_FUNCTION_BLOCK
```

Triggered subsystem semantics

How Stateflow Subsystem Code Maps to Function Blocks

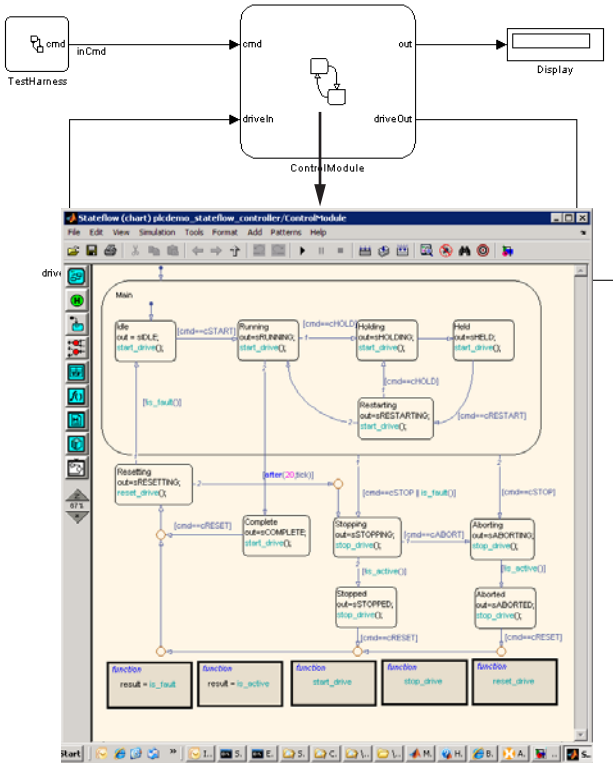
This topic assumes that you have generated structured text code from a Simulink model. If you have not yet done so, see “Generating Structured Text Code from the Model Window” on page 1-23.

The example in this topic shows generated code for the CoDeSys Version 2.3 PLC IDE. Generated code for other IDE platforms looks different.

1. Open the `plcdemo_stateflow_controller` model.
2. Right-click the ControlModule chart and select **PLC Coder > Generate Code for Subsystem**.

   The Simulink PLC Coder software generates structured text code and places it in *`current_folder`*`/plcsrc/plcdemo_stateflow_controller.exp`.
3. If you do not have the `plcdemo_stateflow_controller.exp` file open, open it in the MATLAB editor.

   The following figure illustrates the mapping of the generated code to structured text components for a Simulink Subsystem block that contains a Stateflow chart.



Inlined code for Stateflow chart

```
CASE is_c2_ControlModule OF
    ControlModule_IN_Aborted:
        (* During 'Aborted': '<S1>:12' *)
        IF inCmd = cRESET THEN
            (* Transition: '<S1>:40' *)
            (* Transition: '<S1>:41' *)
            (* Transition: '<S1>:38' *)
            (* Transition: '<S1>:35' *)
            (* Transition: '<S1>:33' *)
            is_c2_ControlModule := ControlModule_IN_Resetting;
            temporalCounter_i1 := 0;
            (* Entry 'Resetting': '<S1>:7' *)
            b_out := sRESETTING;
            (* Graphical Function 'reset_drive': '<S1>:92' *)
            (* Transition: '<S1>:94' *)
            b_driveOut.Start := FALSE;
            b_driveOut.Stop := FALSE;
            b_driveOut.Reset := TRUE;
        END_IF;
    ControlModule_IN_Aborting:
...
...
...
        END_IF;
    ControlModule_IN_Complete:
        (* During 'Complete': '<S1>:9' *)
        IF inCmd = cRESET THEN
            (* Transition: '<S1>:23' *)
            (* Transition: '<S1>:33' *)
            is_c2_ControlModule := ControlModule_IN_Resetting;
            temporalCounter_i1 := 0;
            (* Entry 'Resetting': '<S1>:7' *)
            b_out := sRESETTING;
            (* Graphical Function 'reset_drive': '<S1>:92' *)
            (* Transition: '<S1>:94' *)
            b_driveOut.Start := FALSE;
            b_driveOut.Stop := FALSE;
            b_driveOut.Reset := TRUE;
        END_IF;
    ControlModule_IN_Main:
```

4 Examine the generated structured text code.

The Simulink PLC Coder software aggressively inlines the generated code for the Stateflow chart. The coder performs this inlining because different functions from Stateflow charts share some global state data. However, function blocks in structured text code do not share state data. As a result, the coder software cannot map these functions onto separate function blocks. Instead, it must inline these functions.

How MATLAB Coder Subsystem Code Maps to Function Blocks

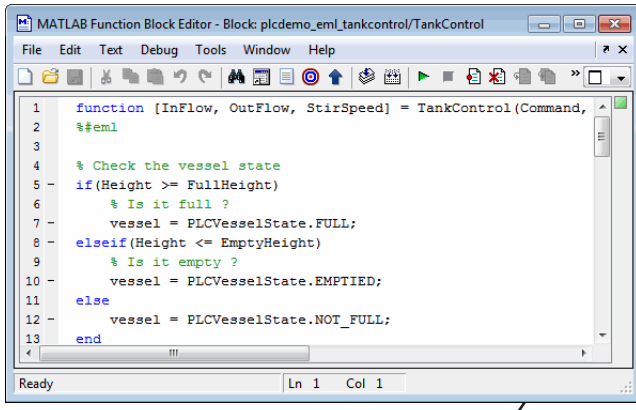
This topic assumes that you have generated structured text code from a Simulink model. If you have not yet done so, see "Generating Structured Text Code from the Model Window" on page 1-23.

The example in this topic shows generated code for the CoDeSys Version 2.3 IDE. Generated code for other IDE platforms looks different.

1. Open the `plcdemo_eml_tankcontrol` model.
2. Right-click the TankControl block and select **PLC Code Generation > Generate Code for Subsystem**.

   The Simulink PLC Coder software generates structured text code and places it in *`current_folder`*`/plcsrc/plcdemo_eml_tankcontrol.exp`.
3. If you do not have the `plcdemo_eml_tankcontrol.exp` file open, open it in the MATLAB editor.

   The following figure illustrates the mapping of the generated code to structured text components for a Simulink Subsystem block that contains a MATLAB Function block. The coder tries to perform inline optimization on the generated code for MATLAB subfunctions. If the coder determines that it is more efficient to leave the subfunction as is, it places the generated code in a structured text construct called `FUNCTION`.
4. Examine the generated structured text code.



MATLAB code

Generated code for MATLAB subfunctions

```
FUNCTION_BLOCK TankControl
VAR_INPUT
    Command: PLCCommandState;
    Height: LREAL;
END_VAR
VAR_OUTPUT
    InFlow: LREAL;
    OutFlow: LREAL;
    StirSpeed: LREAL;
END_VAR
VAR
END_VAR
VAR_TEMP
    vessel: PLCVesselState;
    EmptyValve: PLCValveState;
    FillValve: PLCValveState;
END_VAR
(* Check the vessel state *)
IF Height >= 10 THEN
    (* Is it full ? *)
    vessel := FULL;
ELSIF Height <= 2 THEN
    (* Is it empty ?  *)
    vessel := EMPTIED;
ELSE
    vessel := NOT_FULL;
END_IF;
(* Process the command mode *)
CASE Command OF
    FILL:
        (* Fill Tank *)
        EmptyValve := SHUT;
        IF vessel = FULL THEN
            FillValve := SHUT;
        ELSE
            FillValve := OPEN;
        END_IF;
    HOLD:
        (* Hold Contents *)
        EmptyValve := SHUT;
        FillValve := SHUT;
    EMPTY:
        (* Empty Tank *)
        FillValve := SHUT;
        IF vessel = EMPTIED THEN
            EmptyValve := SHUT;
        ELSE
            EmptyValve := OPEN;
        END_IF;
    ELSE
        EmptyValve := SHUT;
        FillValve := SHUT;
END_CASE;
(* compute inflow and outflow *)
```

How Alias Data Types Map in Generated Code

The coder maps alias data types to the base data type in the generated code.

3

Generating Test Bench Code

- "Working with Generated Structured Text" on page 3-2
- "Generate and Manually Import Test Bench Code" on page 3-5
- "Automatically Importing Structured Text Code" on page 3-9

Working with Generated Structured Text

In this section...
"How Test Bench Verification Works" on page 3-2
"Generated Files" on page 3-2
"Integrating Generated Code into Custom Code" on page 3-2

How Test Bench Verification Works

The Simulink PLC Coder software simulates your model and automatically captures the input and output signals for the subsystem that contains your algorithm. This set of input and output signal data is the test bench data. The coder also automatically generates a test bench (test harness) using the text bench data. The test bench runs the generated code to verify that the output is functionally and numerically equivalent to the output from the execution of a Simulink model. To perform this verification, import the generated structured text and the test bench data into your target IDE.

You can import test bench code:

- Manually, as described in "Generate and Manually Import Test Bench Code" on page 3-5.
- Automatically, including running the test bench, as described in "Automatically Importing Structured Text Code" on page 3-9

Generated Files

Depending on the target IDE platform, the Simulink PLC Coder software generates code into one or more files. See "Generating Structured Text Code from the Model Window" on page 1-23 for list of the target IDE platforms and the possible generated files.

Integrating Generated Code into Custom Code

For the top-level subsystem that has internal state, the generated `FUNCTION_BLOCK` code has `ssMethodType`. `ssMethodType` is a special input argument that the coder adds to the input variables section of the `FUNCTION_BLOCK` section during code generation. `ssMethodType` enables you

to execute code for Simulink Subsystem block methods such as initialization and computation steps. The generated code executes the associated CASE statement based on the value passed in for this argument.

To use `ssMethodType` with a `FUNCTION_BLOCK` for your model, in the generated code, the top-level subsystem function block prototype has one of the following formats:

Has Internal State	ssMethodType Contains...
Yes	The generated function block for the block will have an extra first parameter `ssMethodType` of integer type. This extra parameter is in addition to the function block I/O parameters mapped from Simulink block I/O ports. To use the function block, first initialize the block by calling the function block with `ssMethodType` set to integer constant `SS_INITIALIZE`. If the IDE does not support symbolic constants, set `ssMethodType` to integer value 0. For each follow-up invocation, call the function block with `ssMethodType` set to constant `SS_STEP`. If the IDE does not support symbolic constants, set `ssMethodType` to integer value 1. These settings cause the function block to initialize or compute and return output for each time step.
No	The function block interface only has parameters mapped from Simulink block I/O ports. There is no `ssMethodType` parameter. To use the function block in this case, call the function block with I/O arguments.

For non top-level subsystems, in the generated code, the subsystem function block prototype has one of the following formats:

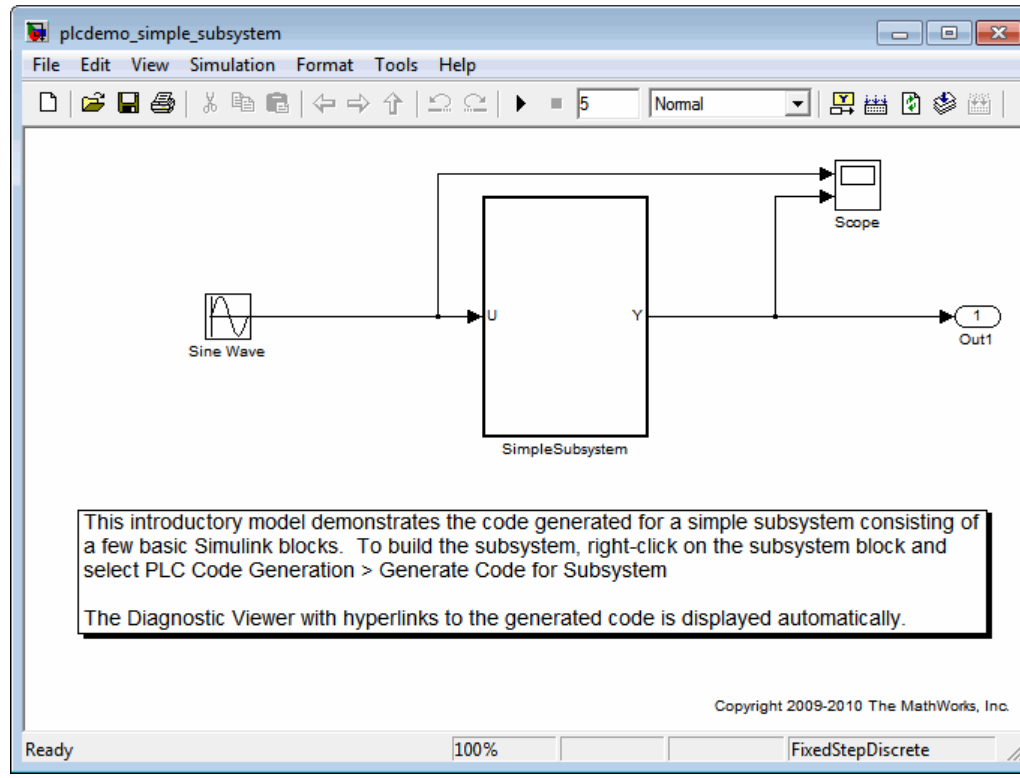
Has Internal State	ssMethodType Contains...
Yes	The function block interface has the `ssMethodType` parameter. The generated code might have `SS_INITIALIZE`, `SS_OUTPUT`, or other `ssMethodType` constants to implement Simulink semantics.
No	The function block interface only has parameters mapped from Simulink block I/O ports. There is no `ssMethodType` parameter.

Generate and Manually Import Test Bench Code

This example shows how to generate test bench code. It uses the CoDeSys V2.3 IDE as an example target IDE.

This example assumes that you have an appropriately configured model from which to generate structured text. If you have not yet done this procedure, see "Preparing Your Model to Generate Structured Text Code" on page 1-13. All demos are located in the *matlabroot*`\toolbox\plccoder\plccoderdemos` folder.

1. If you do not have the `plcdemo_simple_subsystem` model open, open it now.
2. Check that you have connected the inputs and outputs of the subsystem for which you want to generate the test bench. You can import this test bench with the generated code to the target IDE to verify that the output is functionally and numerically equivalent to the output from the execution of a Simulink model. For example:



3 Right-click the Subsystem block and select **PLC Code Generation > Options**.

The Configuration Parameters dialog box is displayed.

4 In **PLC Code Generation > General options > Target IDE**, select your target IDE, for example, `CoDeSys 2.3`.

5 Select the **Generate testbench for subsystem** check box.

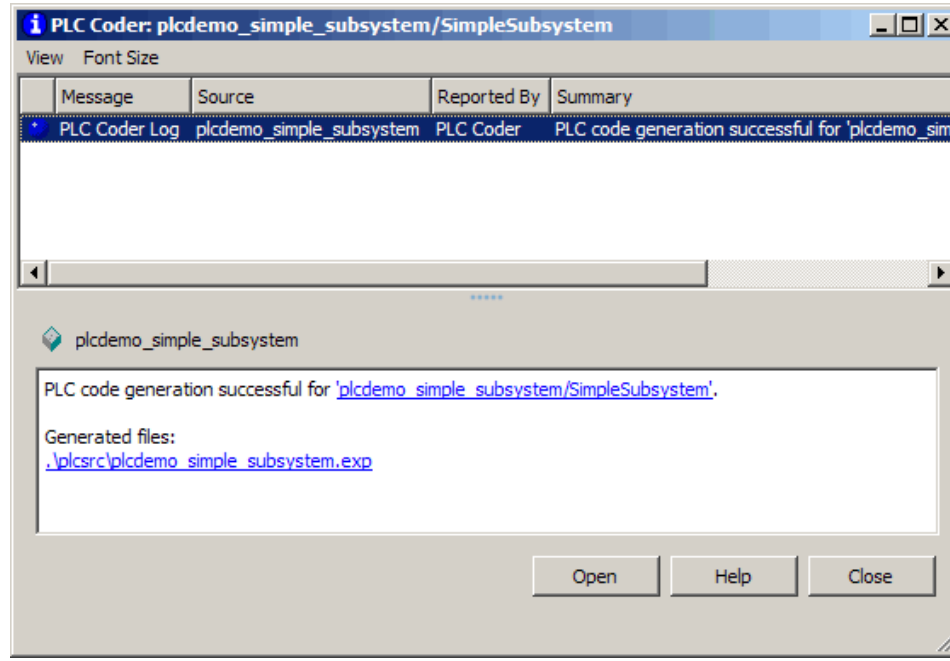
6 Click **Apply**.

7 Click the **Generate code** button.

This button:

- Generates structured text code (same as the **PLC Code Generation > Generate Code for Subsystem** option)
- Generates the test bench for code through Simulink simulation
- Combines the generated code and test bench into *`model_name`*`.exp` (for example, `plcdemo_simple_subsystem.exp`)

  When the code generation is complete, an information window is displayed.



8 Click **OK**.

The Simulink PLC Coder software generates structured text code and writes it to *`current_folder`*`/plcsrc/plcdemo_simple_subsystem.exp`. Depending on the target IDE, the coder might generate additional supporting files.

9 Close the model.

```
bdclose(sys)
```

See the user manual for your target IDE for information on how to import generated code into the target IDE.

Automatically Importing Structured Text Code

In this section...

"PLC IDEs that Qualify for Importing Code Automatically" on page 3-9

"Automatically Importing to KW-Software MULTIPROG 5.0 and Phoenix Contact PC WORX 6.0 IDEs" on page 3-10

"Generating, Automatically Importing, and Verifying Structured Text" on page 3-11

PLC IDEs that Qualify for Importing Code Automatically

This topic assumes that you have read "Automatically Importing Structured Text Code" on page 1-33. If you have not yet done so, read that topic first. It also assumes that you are confident that your model produces structured text that does not require visual examination.

You can generate, automatically import, and optionally verify structured text code for one of the following target PLC IDEs:

- 3S-Smart Software Solutions CoDeSys Version 2.3
- KW-Software MULTIPROG 5.0
- Phoenix Contact PC WORX Version 6.0
- Rockwell Automation RSLogix 5000

  For the Rockwell Automation RSLogix routine format, you must generate testbench code for automatic import and verification.
- Siemens SIMATIC STEP 7 Version 5.4 only for the following versions:
  - Siemens SIMATIC Manager: Version V5.4+SP5+HF1, Revision K5.4.5.1
  - S7-SCL: Version V5.3+SP5, Revision K5.3.5.0
  - S7-PLCSIM: Version V5.4+SP3, Revision K5.4.3.0

If you do not want to run and verify the generated code and want only to import it, see "Generating and Automatically Importing Structured Text Code" on page 1-34.

Automatically Importing to KW-Software MULTIPROG 5.0 and Phoenix Contact PC WORX 6.0 IDEs

Before you can automatically import generated code to this IDE, create an `Empty` template. This topic assumes that you have already set your target IDE to KW-Software MULTIPROG 5.0 or Phoenix Contact PC WORX 6.0.

1. Start the KW-Software MULTIPROG 5.0 or Phoenix Contact PC WORX 6.0 IDE.
2. Select **File > Delete Template** and search for and delete any template named `Empty`. Click **OK** when done.
3. Select **File > New Project**, select `Project Wizard`, then click **OK**.

   The Project Wizard starts.

   a. In the **Project Name** field, type `Empty`,
   b. In the **Project Path** field, type or select a path to which you have write privileges.
   c. Click **Next**.
   d. In the remaining wizard dialog boxes, click **Next** to leave the default selections. At the end of the wizard, click **Finish**.

   The IDE updates with the new `Empty` project tree.
4. In the project, delete everything under the following nodes:
   - Logical POUs
   - Physical Hardware
5. Check that the project tree has only top-level nodes for `Libraries`, `Data Types`, `Logical POUs`, and `Physical Hardware`. There should be no subtree nodes.
6. In the IDE, select **File > Save As Template**.
7. In **Template Name**, type `Empty`.
8. Click **OK**.

9 Close the IDE interface.

When you are ready, open your model, right-click the Subsystem block, and select one of the following:

- **PLC Code Generation > Generate and Import Code for Subsystem**
- **PLC Code Generation > Generate, Import, and Verify Code for Subsystem**

The coder:

1 Generates the code and test bench.

2 Starts the IDE.

3 Creates a new, empty project using your `Empty` template.

4 Imports the generated code and test bench in XML file to the IDE.

5 If you selected **PLC Code Generation > Generate, Import, and Verify Code for Subsystem**, the IDE also runs the generated code to verify it.

Generating, Automatically Importing, and Verifying Structured Text

You can generate, automatically import, and run and verify structured text code. If you want only to generate and automatically import structured text code, see “Automatically Importing Structured Text Code” on page 1-33 instead.

The following procedure assumes that you have installed your target PLC IDE in a default location. If you installed the target PLC IDE in a nondefault location, open the Configuration Parameters dialog box. In the PLC Coder node, set the **Target IDE Path** parameter to the installation folder of your PLC IDE. See “Target IDE Path” on page 11-6 for more details.

Note While the automatic import and verification process is in progress, do not touch your mouse or keyboard. Doing so might disrupt the automatic import or verification process. You can resume normal operations when the process completes.

If you are working with the KW-Software MULTIPROG 5.0 or Phoenix Contact PC WORX 6.0 IDE, see "Automatically Importing to KW-Software MULTIPROG 5.0 and Phoenix Contact PC WORX 6.0 IDEs" on page 3-10.

1. If you do not have the `plcdemo_simple_subsystem` model open, open it now.
2. Right-click the Subsystem block and select **PLC Code Generation > Generate, Import, and Verify Code for Subsystem**.

   The coder then:

   a. Generates the code and test bench.

   b. Starts the target IDE.

   c. Creates a new project.

   d. Imports the generated code and test bench to the new project in the target IDE.

   e. On the target IDE, runs the generated code to verify it.

4

Working with Tunable Parameters in the Simulink PLC Coder Environment

- "Configuring Tunable Parameters for Your Model" on page 4-2
- "Identifying Tunable Parameters" on page 4-7
- "Tune Parameters Using Simulink.Parameter Objects" on page 4-11
- "Configure Tunable Parameters Using the Configuration Parameters Dialog Box" on page 4-16

Configuring Tunable Parameters for Your Model

In this section...

"About Tunable Parameters in the Simulink® PLC Coder™ Environment" on page 4-2

"Configure Your Model for Tunable Parameters" on page 4-4

About Tunable Parameters in the Simulink PLC Coder Environment

Block parameters can be either tunable or nontunable. A tunable parameter is a parameter that you can change while a simulation is running. With the Simulink PLC Coder software, you can tune parameters:

- From the MATLAB workspace, while the model simulation is running.
- In the IDE, while the generated code is running.

The coder maps tunable parameters in the generated code as listed in the following table:

Target IDE	Parameter Storage Class			
	`SimulinkGlobal`	`ExportedGlobal`	`ImportedExtern`	`Imported-ExternPointer`
CoDeSys 2.3	Local function block variables.	Global variable.	Variable definition is skipped.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
CoDeSys 3.3	Local function block variables.	Global variable.	Variable definition is skipped.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.

Target IDE	Parameter Storage Class			
	`SimulinkGlobal`	`ExportedGlobal`	`ImportedExtern`	`Imported-ExternPointer`
B&R Automation Studio 3.0	Local function block variable.	Local function block variable	Local function block variable.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
Beckhoff TwinCAT 2.11	Local function block variable.	Global variable.	Variable definition is skipped.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
KW-Software MULTIPROG 5.0	Local function block variable.	Local function block variable.	Local function block variable.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
Phoenix Contact PC WORX 6.0	Local function block variable.	Global variable.	Variable definition is skipped.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
RSLogix 5000 17, 18: AOI	AOI local tags.	AOI input tags.	AOI input tags.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.

Target IDE	Parameter Storage Class			
	`SimulinkGlobal`	`ExportedGlobal`	`ImportedExtern`	`Imported-ExternPointer`
RSLogix 5000 17, 18: Routine	Instance fields of program UDT tags.	Program tags.	Variable definition is skipped.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
Siemens SIMATIC STEP 7 5.4	Local function block variable.	Local function block variable.	Local function block variable.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
Generic	Local function block variable.	Global variable.	Variable definition is skipped.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
PLCOpen	Local function block variable.	Global variable.	Variable definition is skipped.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.

Configure Your Model for Tunable Parameters

Simulink PLC Coder parameters are inlined and not tunable by default.

To configure a model to enable tunable parameters is:

1 Identify the model parameters that you want to be tunable.

2 Define these parameters in the MATLAB workspace in one of the following ways:

- Create a `Simulink.Parameter` object and use either the Simulink.Parameter command-line interface or Model Explorer to configure parameters.

  Simulink stores `Simulink.Parameter` objects outside the model. This action enables you to share `Simulink.Parameter` objects between multiple models and work with referenced models.

- Use the Configuration Parameters dialog box to define parameters, then configure tunable parameters in the **Configuration Parameters > Optimization > Signals and Parameters > Model Parameter Configuration** dialog box.

  Simulink stores specified global tunable parameters using the Configuration Parameters dialog box with the model. You specify these parameter values in the MATLAB base workspace. You cannot share these parameters between multiple models.

This table lists the possible tunable parameters and how you can set them using either `Simulink.Parameter` or the Configuration Parameters dialog box. When using `Simulink.Parameter`, enter the entire command without hyphens.

Mapping of Tunable Parameters in Generated Code	Simulink.Parameter	Configuration Parameter Dialog Box
Local variables in function block	`Simulink.Parameter.-StorageClass = `SimulinkGlobal'`	Set **Configuration Parameters > Optimization-> Signals and Parameters > Model Parameter Configuration > Storage class** to `SimulinkGlobal (Auto)`
Global variables	`Simulink.Parameter.-StorageClass = 'ExportedGlobal'`	Set **Configuration Parameters > Optimization-> Signals and Parameters**

Mapping of Tunable Parameters in Generated Code	Simulink.Parameter	Configuration Parameter Dialog Box
	`Simulink.Parameter.-CoderInfo.CustomStorageClass = 'Default'`	**> Model Parameter Configuration > Storage class** to `ExportedGlobal` Set **Configuration Parameters > Optimization-> Signals and Parameters > Model Parameter Configuration > Storage type qualifier** to empty field (default)

Global constants	`Simulink.Parameter.Storage-Class = 'ExportedGlobal'`
	
	`Simulink.Parameter.-CoderInfo.CustomStorageClass = 'Const'`
		Set **Configuration Parameters > Optimization-> Signals and Parameters > Model Parameter Configuration > Storage class** to `ExportedGlobal`
		
		Set **Configuration Parameters > Optimization-> Signals and Parameters > Model Parameter Configuration > Storage type qualifier** to `const`

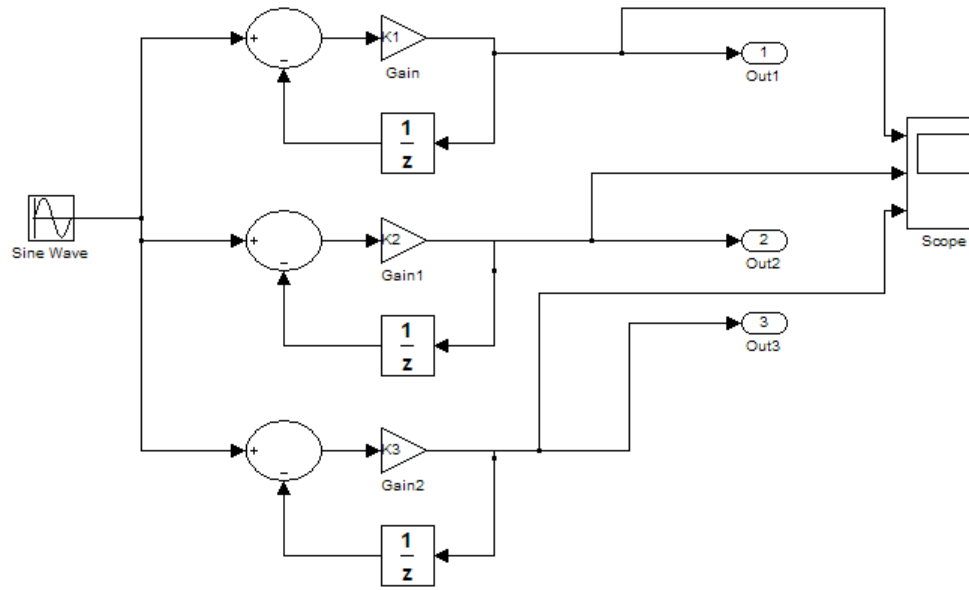
Externally defined variable	`Simulink.Parameter.-StorageClass = 'ImportedExtern'`	Set **Configuration Parameters > Optimization-> Signals and Parameters > Model Parameter Configuration > Storage class** to `ImportedExtern`

Identifying Tunable Parameters

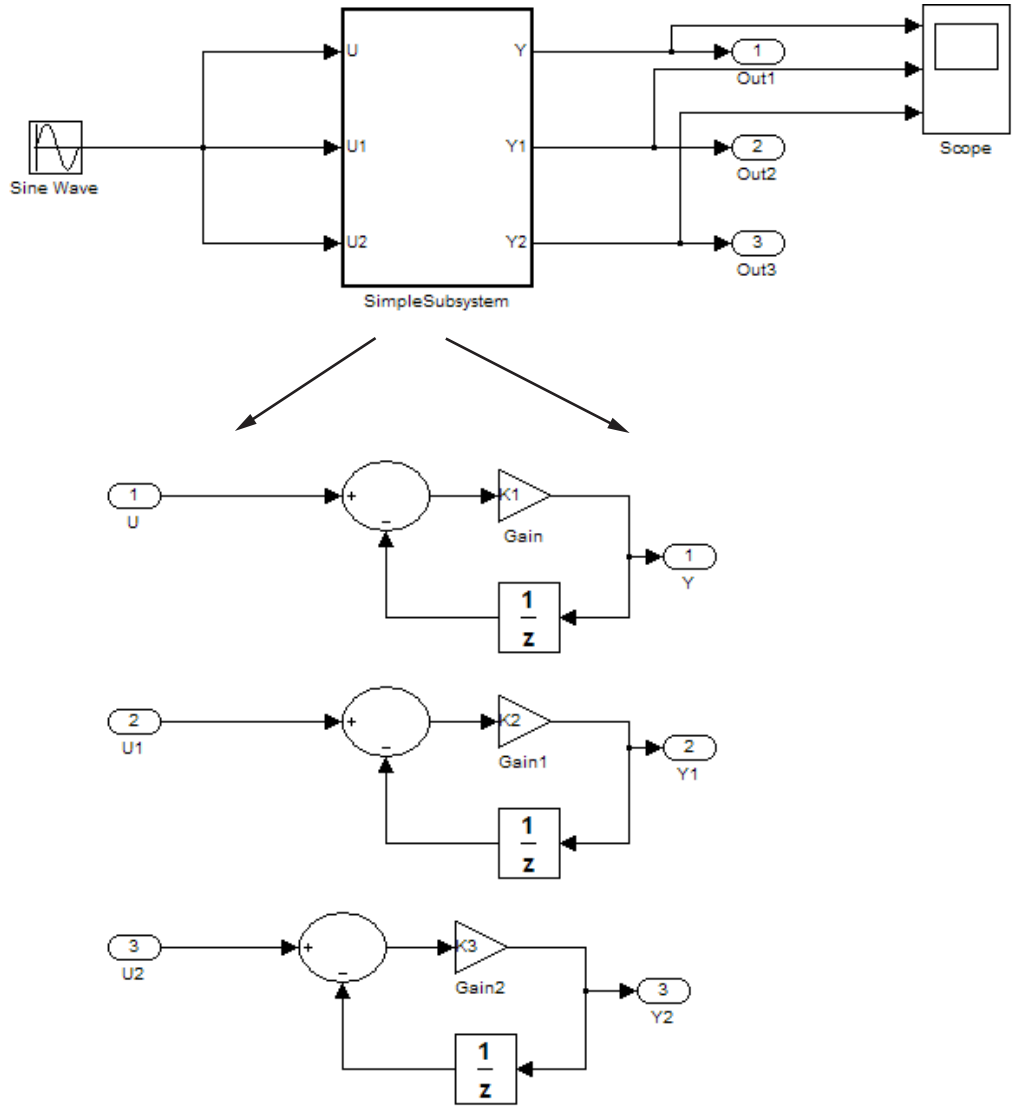
The model `my_plcdemo_tunable_params` shows how to configure tunable parameters. This model is the same as the `plcdemo_tunable_params` and `plcdemo_tunable_params_slparamobj` example models. The difference is that the example model already has the tunable parameters configured.

Note The coder does not support tuning parameters of bus data type.

1. In the MATLAB Command Window, create a model to look like the following.

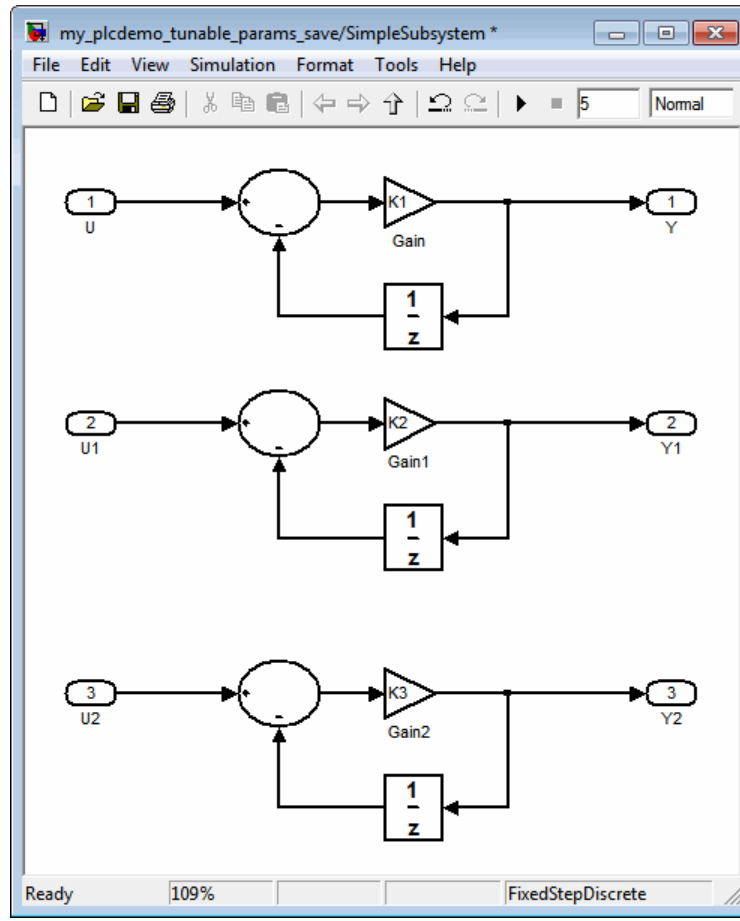


2. Select the Sum, Gain, and Unit Delay blocks and create an atomic subsystem with inputs U, U1, and U2 and outputs Y, Y1, and Y2. Rename the Subsystem block as SimpleSubystem. When you are finished, the top model and atomic subsystem model look like the following model:



3 Save this subsystem as `my_plcdemo_tunable_params.mdl`.

4 Double-click SimpleSubsystem.



5 The three Gain blocks have the constants that you want to make tunable: *K1*, *K2*, and *K3*.

Next, define these parameters in the MATLAB workspace.

- If you want to use `Simulink.Parameter` objects, and use either the Simulink.Parameter command-line interface or Model Explorer to configure parameters, see "Tune Parameters Using Simulink.Parameter Objects" on page 4-11.

- If you want to use the Configuration Parameters dialog box, see "Defining Tunable Parameter Values in the MATLAB Workspace" on page 4-16. Use the Configuration Parameters dialog box to define parameters, then configure tunable parameters in the **Configuration Parameters > Optimization > Signals and Parameters > Model Parameter Configuration** dialog box.

Tune Parameters Using Simulink.Parameter Objects

In this section...

"Work Directly with Simulink.Parameter Objects" on page 4-11

"Work with Simulink.Parameter Objects Using Model Explorer" on page 4-14

Work Directly with Simulink.Parameter Objects

This topic describes how to define tunable parameters in the MATLAB workspace using a MATLAB script that works with `Simulink.Parameter` objects.

Note Alternatively, you can use the Model Explorer to create constants as Simulink.Parameter and define them. For more information, see "Work with Simulink.Parameter Objects Using Model Explorer" on page 4-14.

You must have already created the `my_plcdemo_tunable_params` model or opened `plcdemo_tunable_params` or `plcdemo_tunable_params_slparamobj` and identified the parameters for tuning. If you have not yet done so, see "Identifying Tunable Parameters" on page 4-7.

1. In the MATLAB base workspace, create a script that defines tunable parameters. In the MATLAB Command Window, create a MATLAB file, such as `setup_tunable_params.m`, that contains the following code. This script creates the constants `K1`, `K2`, and `K3` as `Simulink.Parameter` objects, assigns values, and sets the storage class for these constants.

```
% define tunable parameters in base workspace as
% Simulink.Parameter objects

% tunable parameter mapped to local variable
K1 = Simulink.Parameter;
K1.Value = 0.1;
K1.StorageClass = 'SimulinkGlobal';
```

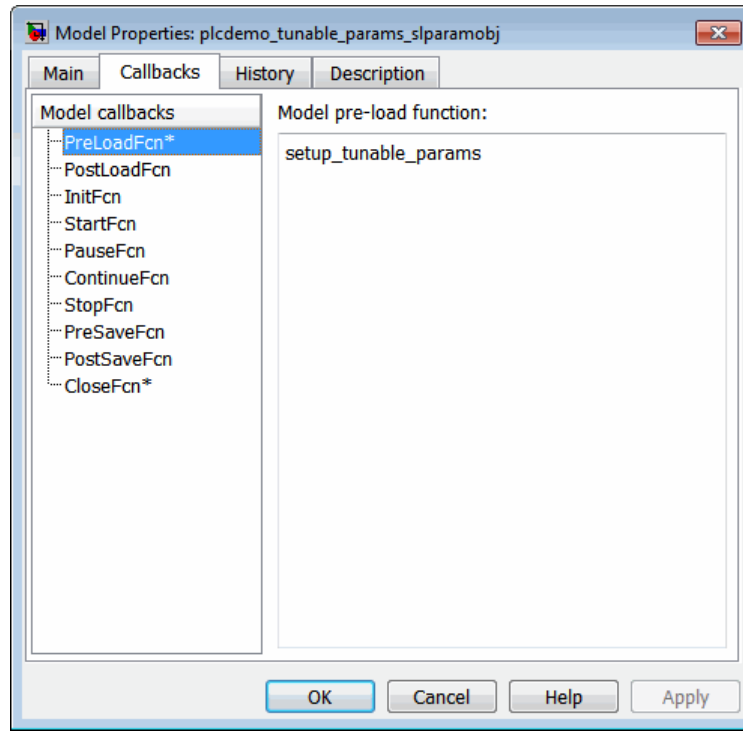
```
% tunable parameter mapped to global variable
K2 = Simulink.Parameter;
K2.Value = 0.2;
K2.StorageClass = 'ExportedGlobal';
K2.CoderInfo.CustomStorageClass = 'Default';

% tunable parameter mapped to global const
K3 = Simulink.Parameter;
K3.Value = 0.3;
K3.StorageClass = 'ExportedGlobal';
K3.CoderInfo.CustomStorageClass = 'Const';
```

2. In the `my_plcdemo_tunable_params` model, select **File > Model Properties**.

3. In the Model Properties dialog box, on the **Callbacks** pane, select `PreLoadFcn`.

4. To run the script at model load time, enter the name of the script that you created, for example, `setup_tunable_params`.

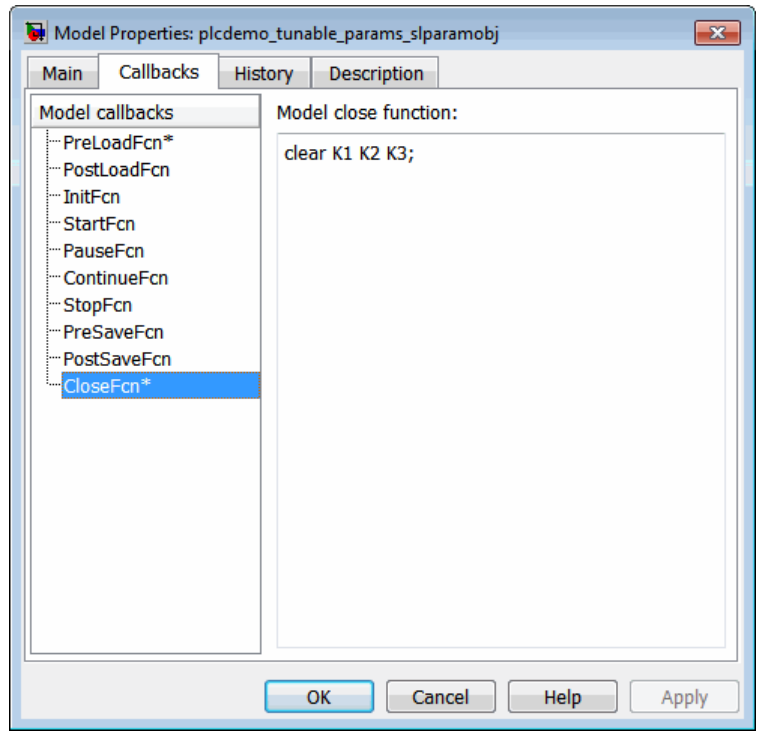


5 Click **Apply**.

6 In the **Callbacks** pane, select `CloseFcn`.

7 In the **Model close function** pane, enter the `clear` command to clear these constants. For example:

```
clear K1 K2 K3;
```



When you close the model, this command clears these constants from the MATLAB workspace.

8 Click **Apply**, then **OK**.

9 In the MATLAB Command Window, you can change the parameter values using the `Simulink.Parameter.Value` field.

10 When you are done, save the model, and generate and inspect the code.

Work with Simulink.Parameter Objects Using Model Explorer

This topic describes how to define tunable parameters in the MATLAB workspace using the Model Explorer. Within the Model Explore, you can create constants as `Simulink.Parameter` objects and tune the parameters.

Note Alternatively, you can use the `Simulink.Parameter` object command-line interface to create constants and define them. For more information, see "Work Directly with Simulink.Parameter Objects" on page 4-11.

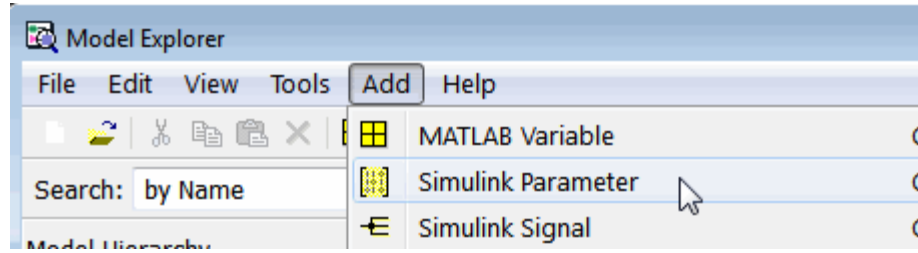
1. In the Simulink editor window, select **View > Model Explorer**.

   In the MATLAB Command Window, create a MATLAB file, such as `setup_tunable_params.m`, that contains the following code. This script creates the constants `K1`, `K2`, and `K3` as `Simulink.Parameter` objects, assigns values, and sets the storage class for these constants.

2. In the **Model Hierarchy** pane, select **Base Workspace**.

   The **Contents** pane is updated with the contents of the MATLAB base workspace.

3. To create a new Simulink.Parameter object, select **Add > Simulink Parameter**.



4. In the **Dialog** pane for the `Simulink.Parameter` object, edit the **Storage class** and **Value** parameters. Click **Apply** to save changes.

5. When you are done, save the model, and generate and inspect the code.

For more information about using the Model Explorer to work with data objects, see "Using the Model Explorer to Create Data Objects" in the *Simulink User's Guide*.

Configure Tunable Parameters Using the Configuration Parameters Dialog Box

In this section...

"Defining Tunable Parameter Values in the MATLAB Workspace" on page 4-16

"Configuring Parameters to Be Tunable" on page 4-18

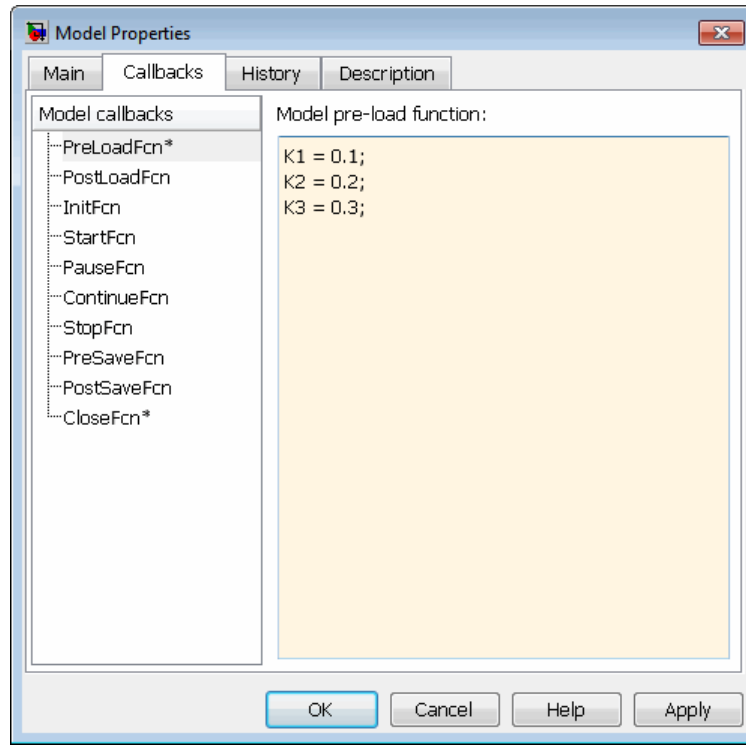
Defining Tunable Parameter Values in the MATLAB Workspace

This topic describes how to define tunable parameter values in the MATLAB workspace using the Simulink Model Properties dialog box. Defining tunable parameters in this way enables the model to automatically define parameters each time that you open the model.

You must have already created the `my_plcdemo_tunable_params` model or opened `plcdemo_tunable_params` and identified the parameters for tuning. If you have not yet done so, see "Identifying Tunable Parameters" on page 4-7.

1. In the `my_plcdemo_tunable_params` model, select **File > Model Properties**.
2. In the Model Properties dialog box, on the **Callbacks** pane, select `PreLoadFcn`.
3. In the **Model pre-load function** pane, enter the three constants *K1*, *K2*, and *K3*. Assign initial values to them. For example:

```
K1 = 0.1;
K2 = 0.2;
K3 = 0.3;
```

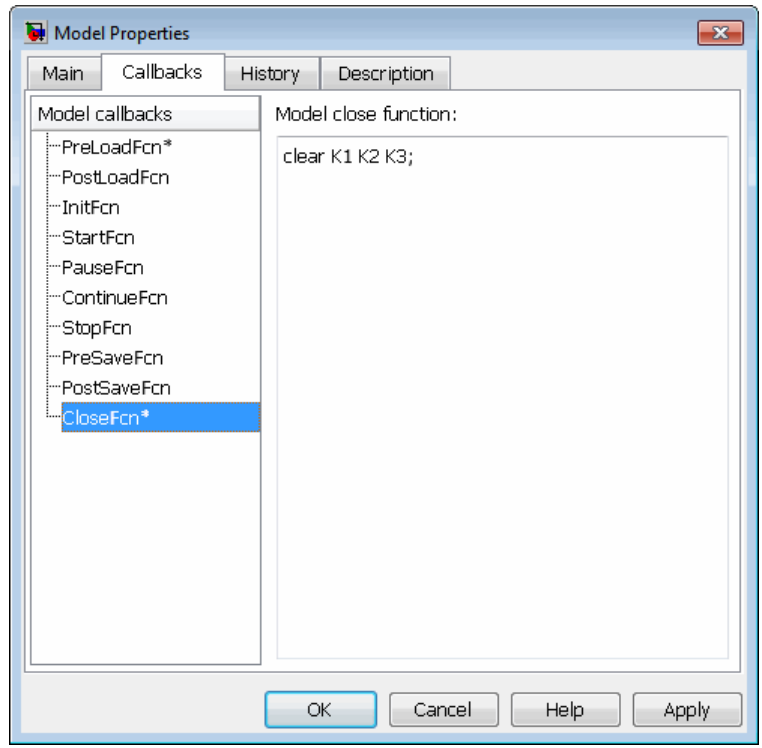


4 Click **Apply**.

5 In the **Callbacks** pane, select `CloseFcn`.

6 In the **Model close function** pane, enter the `clear` command to clear these constants. For example:

```
clear K1 K2 K3;
```



When you close the model, this command clears these constants from the MATLAB workspace.

7 Click **Apply**, then **OK**.

Your next task is to configure these parameters to be tunable. See "Configuring Parameters to Be Tunable" on page 4-18.

Configuring Parameters to Be Tunable

This topic describes how to configure parameters to be tunable using the Simulink Configuration Parameters dialog box.

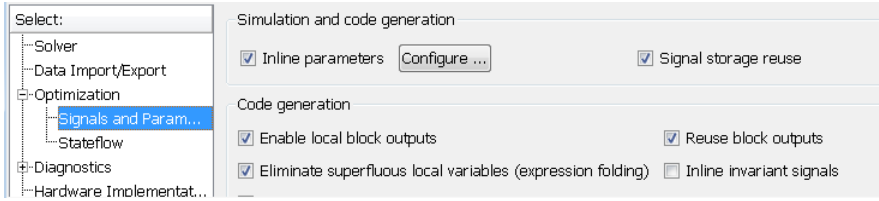
You must have already created the `my_plcdemo_tunable_params` model or opened `plcdemo_tunable_params` and defined the parameters for tuning. If

you have not yet done so, see “Defining Tunable Parameter Values in the MATLAB Workspace” on page 4-16.

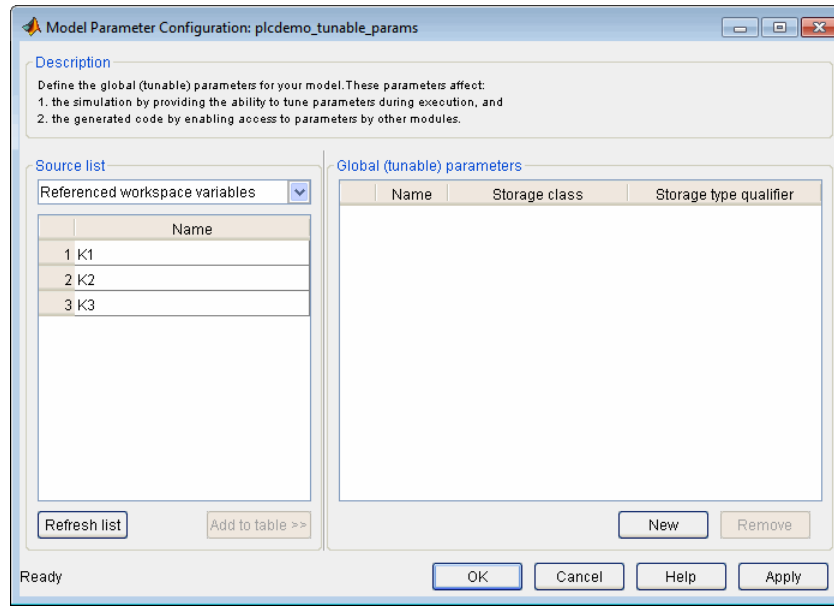
You must already be familiar with the tunable parameter properties on the **Global (tunable) parameters** pane. For more information, see Setting Tunable Parameter Properties in the Simulink Coder™ documentation.

This example uses code generated with CoDeSys Version 2.3.

1. In the model, right-click SimpleSubsystem and select **PLC Code Generation > Options**.
2. Navigate to **Optimization > Signals and Parameters**.
3. In the **Simulation and code generation** section, select the **Inline parameters** check box. (This check box is cleared by default.)

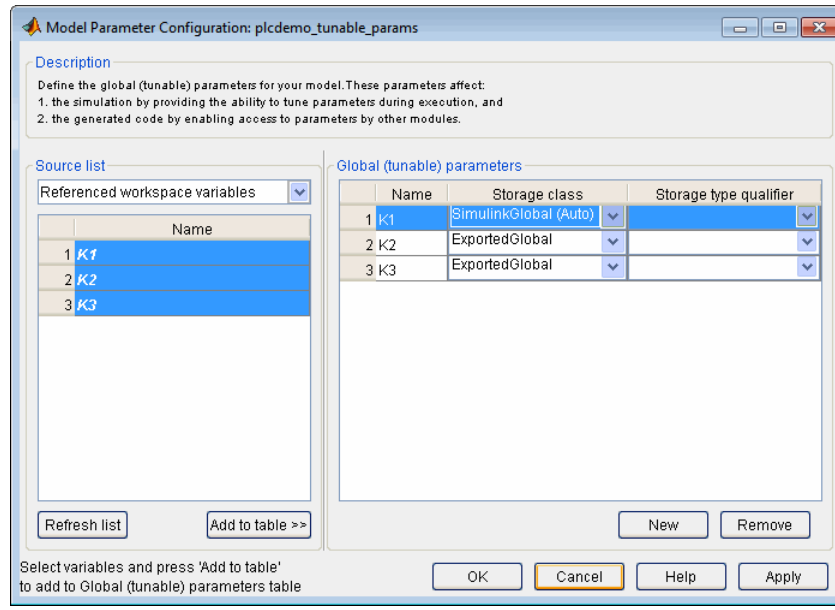


4. Click **Configure**.



5 In the Model Parameter Configuration dialog box, from the **Source list**, select `Referenced workspace variables`.

6 Use the **Ctrl** key to select all the parameters and click **Add to table >>** to add them to the **Global (tunable) parameters** table.



By default, this dialog box sets all parameters to the `SimulinkGlobal (Auto)` storage class. This setting generates code with the tunable parameters set at the local level. In this case, these parameters appear at the function block level in each function block that uses the parameter.

You can also optionally set the storage type qualifier for a parameter to `const`.

7 Click **Apply** and **OK**.

8 In the Configuration Parameters dialog box, navigate to **PLC Code Generation > General options**.

9 Select the **Target IDE** and **Output Directory** settings, then click **Generate code**.

10 Observe that the `VAR` section of `Function Block SimpleSubsystem` defines *K1*, *K2*, and *K3*.

```
15  FUNCTION_BLOCK SimpleSubsystem
16  VAR_INPUT
17      ssMethodType: SINT;
18      U: LREAL;
19      U1: LREAL;
20      U2: LREAL;
21  END_VAR
22  VAR_OUTPUT
23      Y: LREAL;
24      Y1: LREAL;
25      Y2: LREAL;
26  END_VAR
27  VAR
28      K1: LREAL := 0.1;
29      K2: LREAL := 0.2;
30      K3: LREAL := 0.3;
31      UnitDelay_DSTATE: LREAL;
32      UnitDelay1_DSTATE: LREAL;
33      UnitDelay2_DSTATE: LREAL;
34  END_VAR
```

11 To configure a parameter to be a global variable in the generated code, set the parameter storage class of *K2* to `ExportedGlobal`. Leave the storage type qualifier unset.

Some target IDEs do not support the access of global variables. In this case, the Simulink PLC Coder software uses `SimulinkGlobal` as the automatic storage class.

To configure a parameter to be a global constant in the generated code, set the parameter storage class of *K3* to `ExportedGlobal`. Set storage type qualifier to `const`.

12 Click **Apply** and **OK**, then rebuild the code.

13 Observe that *K2* is now in the `VAR_GLOBAL` section. *K3* is in the `VAR_GLOBAL_CONSTANT` section.

```
91  VAR_GLOBAL CONSTANT
92      SS_INITIALIZE: SINT := 2;
93      SS_OUTPUT: SINT := 3;
94      K3: LREAL := 0.3;
95  END_VAR
96  VAR_GLOBAL
97      K2: LREAL := 0.2;
98  END_VAR
```

14 To configure a parameter so that you or somebody else can provide it through external structured text, set the parameter storage class of *K1* to `ImportedExtern`. The coder does not generate a variable declaration for the parameter in the code. Leave the storage type qualifier unset.

15 Click **Apply** and **OK**, then rebuild the code.

16 Observe that *K1* no longer appears in the `VAR` section of the generated code. (Compare to Step 10.)

```
27  VAR
28      UnitDelay_DSTATE: LREAL;
29      UnitDelay1_DSTATE: LREAL;
30      UnitDelay2_DSTATE: LREAL;
31  END_VAR
```

Note The Simulink PLC Coder software does not support setting the parameter storage class to `ImportedExternPointer`. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.

5

Controlling Generated Code Partitions

Function Block Partitions

In this section...

"About Function Block Partitions" on page 5-2

"Example: One Function Block for Atomic Subsystems" on page 5-2

"Example: One Function Block for Virtual Subsystems" on page 5-3

"Example: Multiple Function Blocks for Nonvirtual Subsystems" on page 5-4

"Controlling Generated Code Using Subsystem Block Parameters" on page 5-5

About Function Block Partitions

The Simulink PLC Coder software converts subsystems to function block units according to the following rules:

- Generates a function block for the top-level atomic subsystem for which you generate code.
- Generates a function block for an atomic subsystem whose **Function packaging** parameter is set to `Function` or `Reusable function`.
- Inlines generated code from atomic subsystems, whose **Function packaging** parameter is set to `Inline`, into the function block that corresponds to the nearest ancestor subsystem. This nearest ancestor cannot be inlined.

These topics use code generated with CoDeSys Version 2.3.

Example: One Function Block for Atomic Subsystems

The code for the `plcdemo_simple_subsystem` demo is an example of generating code with one function block. The atomic subsystem for which you generate code does not contain any other subsystems.

```
FUNCTION_BLOCK SimpleSubsystem
VAR_INPUT
    ssMethodType: SINT;
    U: LREAL;
END_VAR
VAR_OUTPUT
    Y: LREAL;
END_VAR
VAR
    UnitDelay_DSTATE: LREAL;
END_VAR
VAR_TEMP
    rtb_Gain: LREAL;
END_VAR
CASE ssMethodType OF
    SS_INITIALIZE:
        (* InitializeConditions for UnitDelay: '<S1>/Unit Delay'  *)
        UnitDelay_DSTATE := 0;

    SS_OUTPUT:
        (* Gain: '<S1>/Gain' incorporates:
         *  Inport: '<Root>/U'
         *  Sum: '<S1>/Sum'
         *  UnitDelay: '<S1>/Unit Delay'
          *)
        rtb_Gain := (U - UnitDelay_DSTATE) * 0.5;

        (* Outport: '<Root>/Y'  *)
        Y := rtb_Gain;

        (* Update for UnitDelay: '<S1>/Unit Delay'  *)
        UnitDelay_DSTATE := rtb_Gain;

END_CASE;
END_FUNCTION_BLOCK
```

Example: One Function Block for Virtual Subsystems

The `plcdemo_hierarchical_virtual_subsystem` demo contains an atomic subsystem that has two virtual subsystems, S1 and S2, inlined. A virtual subsystem does not have the **Treat as atomic unit** parameter selected. When you generate code for the hierarchical subsystem, the code contains only the `FUNCTION_BLOCK HierarchicalSubsystem` component. There are no additional function blocks for the S1 and S2 subsystems.

```
FUNCTION_BLOCK HierarchicalSubsystem
VAR_INPUT
    ssMethodType: SINT;
    In1: LREAL;
    In2: LREAL;
    In3: UINT;
    In4: LREAL;
END_VAR
VAR_OUTPUT
    Out1: LREAL;
    Out2: LREAL;
END_VAR
VAR
    UnitDelay_DSTATE: LREAL;
    UnitDelay1_DSTATE: LREAL;
    UnitDelay_DSTATE_f: LREAL;
    UnitDelay_DSTATE_a: LREAL;
END_VAR
VAR_TEMP
    rtb_Gain: LREAL;
    rtb_Gain_m: LREAL;
END_VAR
CASE ssMethodType OF
    SS_INITIALIZE:
        (* InitializeConditions for U
```

Example: Multiple Function Blocks for Nonvirtual Subsystems

The `plcdemo_hierarchical_subsystem` demo contains an atomic subsystem that has two nonvirtual subsystems, S1 and S2. Virtual subsystems have the **Treat as atomic unit** parameter selected. When you generate code for the hierarchical subsystem, that code contains the `FUNCTION_BLOCK HierarchicalSubsystem`, `FUNCTION_BLOCK HierarchicalSubsystem_S1`, and `FUNCTION_BLOCK HierarchicalSubsystem_S2` components.

Function Block for Hierarchical Subsystem

```
 7  FUNCTION_BLOCK HierarchicalSubsystem
 8  VAR_INPUT
 9      ssMethodType: SINT;
10      In1: LREAL;
11      In2: LREAL;
12      In3: UINT;
13      In4: LREAL;
14  END_VAR
```

Function Block for Hierarchical S1

```
114  FUNCTION_BLOCK HierarchicalSubsystem_S1
115  VAR_INPUT
116      ssMethodType: SINT;
117      rtu_U: LREAL;
```

Function Block for Hierarchical S2

```
84  FUNCTION_BLOCK HierarchicalSubsystem_S2
85  VAR_INPUT
86      ssMethodType: SINT;
87      rtu_U: LREAL;
88  END_VAR
```

Controlling Generated Code Using Subsystem Block Parameters

You can partition generated code using the following Subsystem block parameters on the Code Generation tab. See the Subsystem block documentation for details.

- **Function packaging**
- **Function name options**

Leave the **File name options** set to the default, `Auto`.

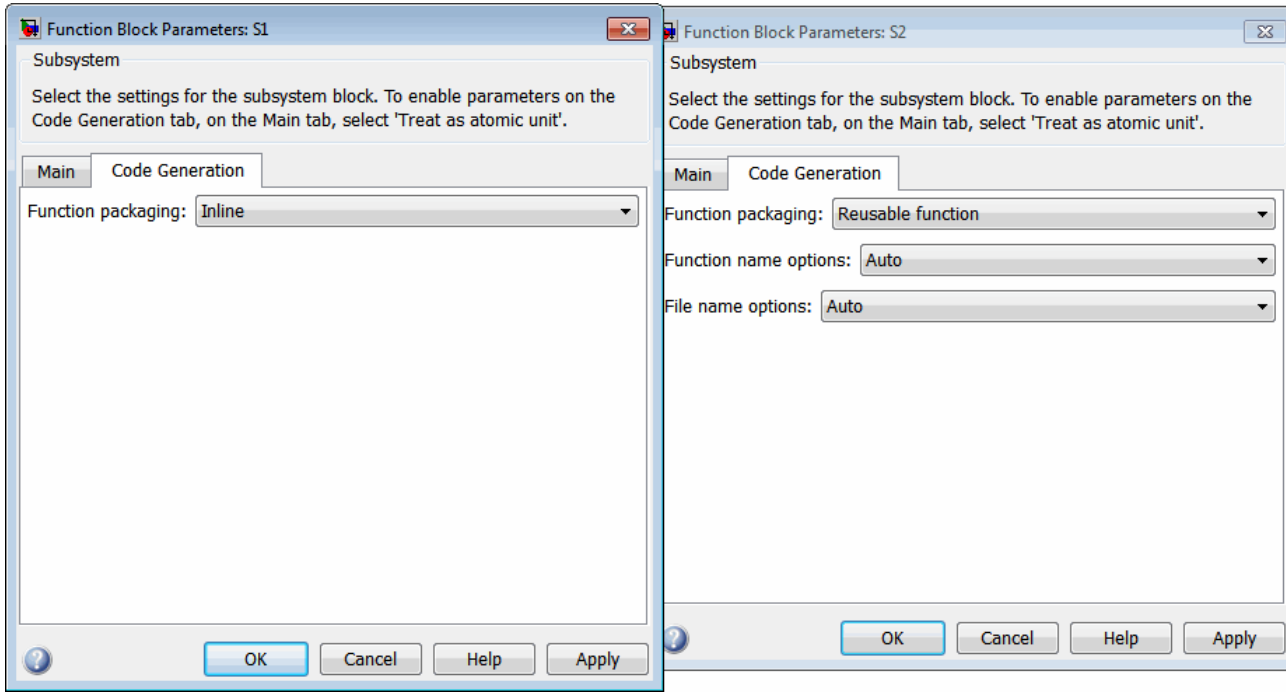
Generating Separate Partitions and Inlining Subsystem Code

Use the **Function packaging** parameter to specify the code format to generate for an atomic (nonvirtual) subsystem. The Simulink PLC Coder software interprets this parameter depending on the setting that you choose:

Setting	Coder Interpretation
`Auto`	Uses the optimal format based on the type and number of subsystem instances in the model.
`Reusable function`, `Function`	Generates a function with arguments that allows the subsystem code to be shared by other instances of it in the model.
`Inline`	Inlines the subsystem unconditionally.

For example, in the `plcdemo_hierarchical_virtual_subsystem`, you can:

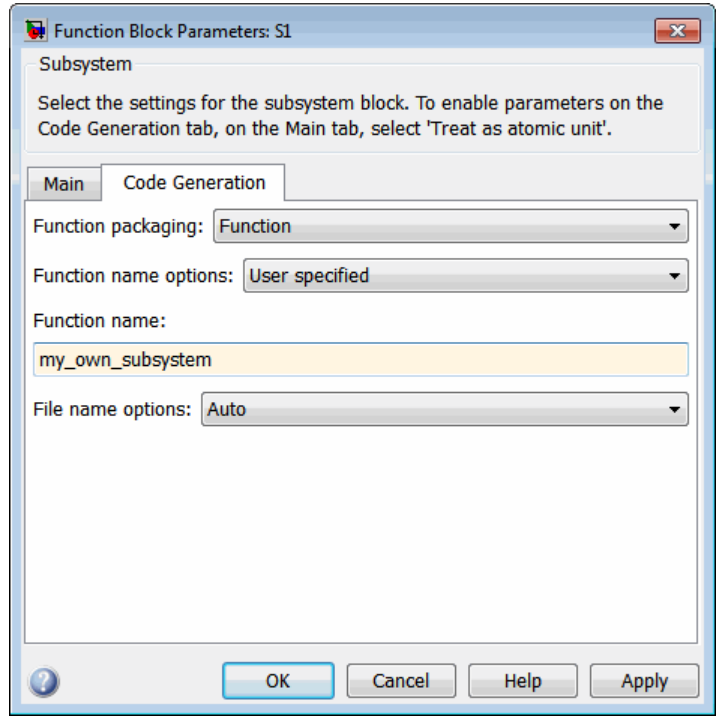
- Inline the S1 subsystem code by setting **Function packaging** to `Inline`. This setting creates one function block for the parent with the S1 subsystem inlined.
- Create a function block for the S2 subsystem by setting **Function packaging** to `Reusable function`, `Auto`, or `Function`. This setting creates two function blocks, one for the parent, one for S2.



Changing the Name of a Subsystem

You can use the **Function name options** parameter to change the name of a subsystem from the one on the block label. When the Simulink PLC Coder generates software, it uses the string you specify for this parameter as the subsystem name. For example, in the `plcdemo_hierarchical_virtual_subsystem` demo:

1. Open the S1 subsystem block parameter dialog box.
2. Click the Code Generation tab.
3. Set **Function packaging** to `Function`.
4. Set **Function name options** to `User specified`.
5. In the **Function name** field, specify a custom name. For example, type `my_own_subsystem`.



6 Save the new settings.

7 Generate code for the parent subsystem.

8 Observe the renamed function block.

```
114  FUNCTION_BLOCK my_own_subsystem
115  VAR_INPUT
116      ssMethodType: SINT;
117      rtu_U: LREAL;
118  END_VAR
```

6

Integrating Externally Defined Symbols

- "Integrate Externally Defined Symbols" on page 6-2
- "Integrate Custom Function Block in Generated Code" on page 6-3

Integrate Externally Defined Symbols

The coder allows you to suppress symbol definitions in the generated code. This suppression allows you to integrate a custom element, such as user defined function blocks, function blocks, data types, and named global variable and constants, in place of one generated from a Simulink subsystem. You must then provide these definitions when importing the code into the target IDE. You must:

- Define the custom element in the subsystem for which you want to generate code.
- Name the custom element.
- Add the name of the custom element to **PLC Code Generation > Symbols > Externally Defined Symbols** in the Configuration Parameters dialog box.
- Generate code.

For a description of how to integrate a custom function block, see "Integrate Custom Function Block in Generated Code" on page 6-3. For a description of the **Externally Defined Symbols** parameter, see "Externally Defined Symbols" on page 11-19.

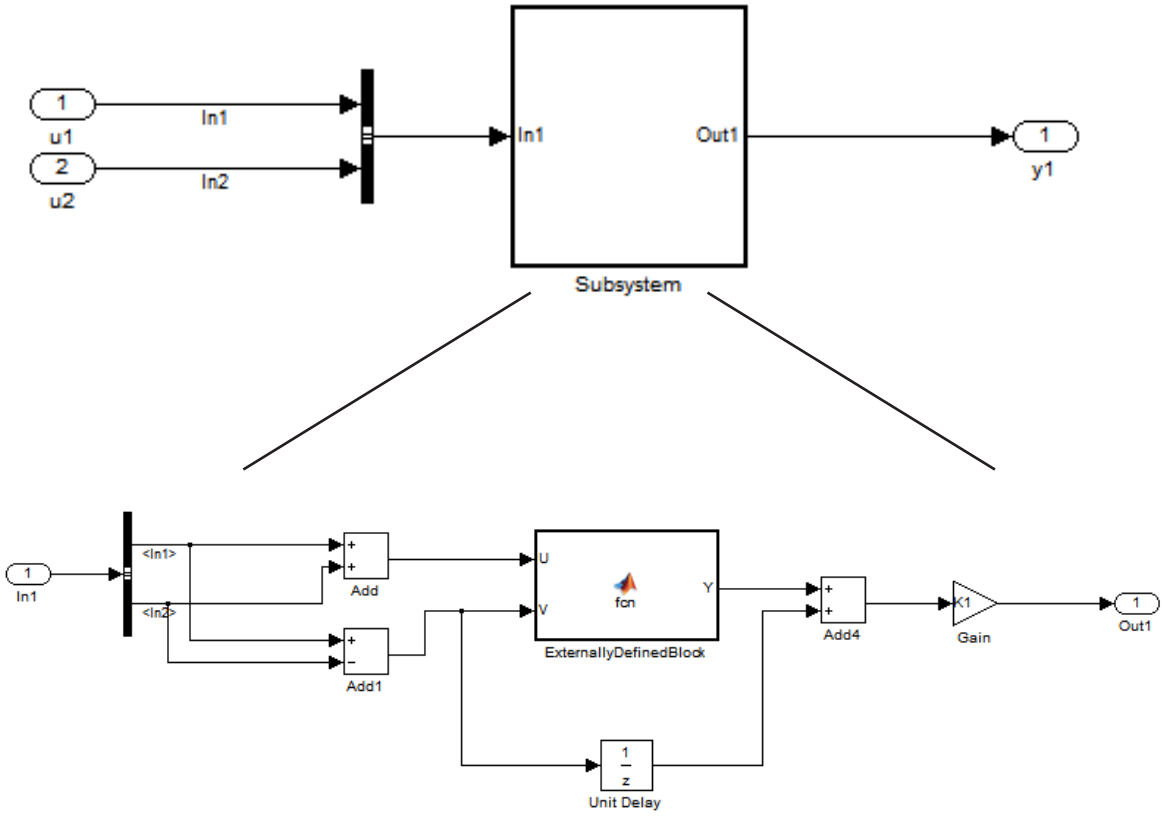
Integrate Custom Function Block in Generated Code

To integrate a custom function block, ExternallyDefinedBlock, this procedure uses the example `plcdemo_external_symbols`.

1. In a Simulink model, add a MATLAB Function block.
2. Double-click the MATLAB Function block.
3. In the MATLAB editor, minimally define inputs, outputs, and stubs. For example:

   ```
   function Y = fcn(U,V)
   % Stub behavior for simulation. This block
   % is replaced during code generation
   Y = U + V;
   ```

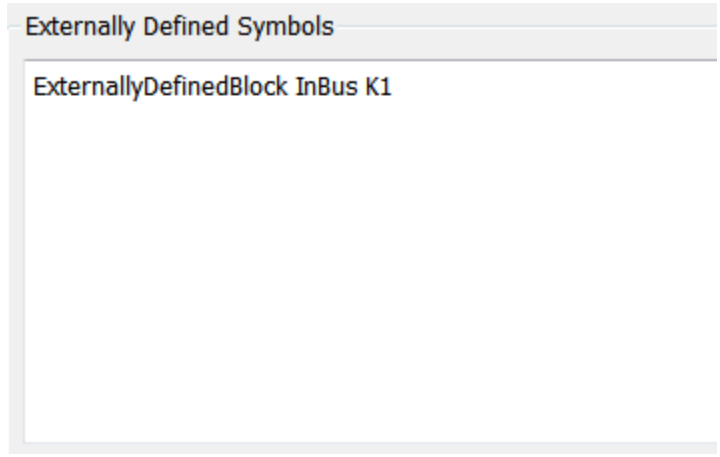
4. Change the MATLAB Function block name to ExternallyDefinedBlock.
5. Create a subsystem from this MATLAB Function block.
6. Complete the model to look like `plcdemo_external_symbols`.



7 Open the Configuration Parameters dialog box for the model.

8 Add `ExternallyDefinedBlock` to **PLC Code Generation > Symbols > Externally Defined Symbols**.

9 The `plcdemo_external_symbols` model also suppresses `K1` and `InBus`. Add these symbol names to the **Externally Defined Symbols** field, separated by spaces. For other settings, see the `plcdemo_external_symbols` model.



10 Save and close your new model. For example, save it as `plcdemo_external_symbols_mine`.

11 Generate code for the model.

12 In the generated code, look for instances of `ExternallyDefinedBlock`.

The reference of `ExternallyDefinedBlock` is:

```
VAR
    UnitDelay_DSTATE: LREAL;
    _i0_ExternallyDefinedBlock: ExternallyDefinedBlock;
END_VAR
```

The omission of `ExternallyDefinedBlock` is:

```
(* MATLAB Function: '<S1>/ExternallyDefinedBlock' *)
_i0_ExternallyDefinedBlock(U := rtb_Add, V := rtb_Add1);
rtb_Y := _i0_ExternallyDefinedBlock.Y;
```

7

IDE-Specific Considerations

- "Introduction" on page 7-2
- "Considerations for All Target IDEs" on page 7-3
- "Rockwell Automation RSLogix Considerations" on page 7-4
- "Siemens SIMATIC STEP 7 Considerations" on page 7-6

Introduction

This chapter describes IDE-specific considerations you should be aware of when generating and downloading code.

Considerations for All Target IDEs

The coder converts matrix data types to single-dimensional vectors (column-major) in the generated structured text.

Rockwell Automation RSLogix Considerations

This topic describes the considerations to remember for this target IDE platform.

Add-On Instruction and Function Blocks

The structured text concept of function block exists for Rockwell Automation RSLogix target IDEs as an Add-On instruction (AOI). The Simulink PLC Coder software generates AOIs for Add-On instruction format, not FUNCTION_BLOCK.

Double-Precision Data Types

The Rockwell Automation RSLogix target IDE does not support double-precision data types. At code generation, the Simulink PLC Coder converts this data type to single-precision data types in generated code.

Note Design your model to use single-precision data type (single) as much as possible instead of double-precision data type (double). If you must use doubles in your model, note that the numerical results produced by the generated structured text might differ from Simulink results. This difference is due to double-single conversion in the generated code.

Unsigned Integer Data Types

The Rockwell Automation RSLogix target IDE does not support unsigned integer data types. At code generation, the Simulink PLC Coder converts this data type to signed integer data types in generated code.

Note Design your model to use signed integer data types (int8, int16, int32) as much as possible instead of unsigned integer data types (uint8, uint16, uint32). Doing so avoids overflow issues that unsigned-to-signed integer conversions can cause in the generated code.

Unsigned Fixed-Point Data Types

In the generated code, Simulink PLC Coder converts fixed-point data types to target IDE integer data types. Because the Rockwell Automation RSLogix target IDE does not support unsigned integer data types, do not use unsigned fixed-point data types in the model. See “Fixed-Point Data Type Limitations” on page 9-3 for coder limitations for fixed-point data type support.

Enumerated Data Types

The Rockwell Automation RSLogix target IDE does not support enumerated data types. At code generation, the Simulink PLC Coder converts this data type to 32–bit signed integer data type in generated code.

Siemens SIMATIC STEP 7 Considerations

This topic describes the considerations to remember for this target IDE platform.

Double-Precision Floating-Point Data Types

The Siemens SIMATIC STEP 7 target IDE does not support double-precision floating-point data types. At code generation, the Simulink PLC Coder converts this data type to single-precision real data types in generated code.

Note Design your model to use single-precision floating-point data type (single) as much as possible instead of double-precision floating-point data type (double). If you must use double-precision floating-point data types in your model, the numerical results produced by the generated structured text might differ from Simulink results. Design your model so that the possible precision loss of numerical results of the generated code does not affect the expected semantics of the model.

---

int8 and Unsigned Integer Types

The Siemens SIMATIC STEP 7 SCL language does not support int8 and unsigned integer data types. At code generation, the Simulink PLC Coder converts int8 and unsigned integer data types to int16 or int32 in generated code.

Note Design your model to use int16 and int32 data types as much as possible instead of int8 or unsigned integer data types. If you must use int8 or unsigned integers, the numerical results produced by the generated structured text might differ from Simulink results. Design your model so that effects of integer data type conversion of the generated code do not affect the expected semantics of the model.

Unsigned Fixed-Point Data Types

In the generated code, Simulink PLC Coder converts fixed-point data types to target IDE integer data types. Because the Siemens SIMATIC STEP 7

target IDE does not support unsigned integer data types, do not use unsigned fixed-point data types in the model. See “Fixed-Point Data Type Limitations” on page 9-3 for coder limitations for fixed-point data type support.

Enumerated Data Types

The Siemens SIMATIC STEP 7 target IDE does not support enumerated data types. At code generation, the Siemens SIMATIC STEP 7 converts this data type to 16–bit signed integer data type in generated code.

8

Supported Simulink and Stateflow Blocks

- "Simulink Blocks" on page 8-2
- "Stateflow Blocks" on page 8-12

Simulink Blocks

To access a Simulink library of blocks that the Simulink PLC Coder software supports, in the MATLAB Command Window, type `plclib`.

Additional Math & Discrete/Additional Discrete

Transfer Fcn Direct Form II

Transfer Fcn Direct Form II Time Varying

Unit Delay Enabled

Unit Delay Enabled External IC

Unit Delay Enabled Resettable

Unit Delay Enabled Resettable External IC

Unit Delay External IC

Unit Delay Resettable

Unit Delay Resettable External IC

Unit Delay With Preview Enabled

Unit Delay With Preview Enabled Resettable

Unit Delay With Preview Enabled Resettable External RV

Unit Delay With Preview Resettable

Unit Delay With Preview Resettable External RV

Commonly Used Blocks

Inport

Bus Creator

Bus Selector

Constant

Data Type Conversion

Demux

Discrete-Time Integrator

Gain

Ground

Logical Operator

Mux

Product

Relational Operator

Saturation

Scope

Subsystem

Inport

Outport

Sum

Switch

Terminator

Unit Delay

Discontinuities

Coulomb and Viscous Friction

Dead Zone Dynamic

Rate Limiter

Rate Limiter Dynamic

Relay

Saturation

Saturation Dynamic

Wrap To Zero

Discrete

Difference

Discrete Transfer Fcn

Discrete Derivative

Discrete FIR Filter

Discrete Filter

PID Controller

PID Controller (2 DOF)

Discrete State-Space

Discrete-Time Integrator

Integer Delay

Memory

Tapped Delay

Transfer Fcn First Order

Transfer Fcn Lead or Lag

Transfer Fcn Real Zero

Unit Delay

Zero-Order Hold

Logic and Bit Operations

Bit Clear

Bit Set

Bitwise Operator

Compare To Constant

Compare To Zero

Detect Change

Detect Decrease

Detect Increase

Detect Fall Negative

Detect Fall Nonpositive

Detect Rise Nonnegative

Detect Rise Positive

Extract Bits

Interval Test

Interval Test Dynamic

Logical Operator

Shift Arithmetic

Lookup Tables

Dynamic-Lookup

Interpolation Using Prelookup

PreLookup

n-D Lookup Table

Math Operations

Abs

Add

Assignment

Bias

Divide

Dot Product

Gain

Math Function

Matrix Concatenate

MinMax

MinMax Running Resettable

Permute Dimensions

Polynomial

Product

Product of Elements

Reciprocal Sqrt

Reshape

Rounding Function

Sign

Slider Gain

Sqrt

Squeeze

Subtract

Sum

Sum of Elements

Trigonometric Function

Unary Minus

Vector Concatenate

Model Verification

Assertion

Check Discrete Gradient

Check Dynamic Gap

Check Dynamic Range

Check Static Gap

Check Static Range

Check Dynamic Lower Bound

Check Dynamic Upper Bound

Check Input Resolution

Check Static Lower Bound

Check Static Upper Bound

Model-Wide Utilities

DocBlock

Model Info

Ports & Subsystems

Atomic Subsystem

CodeReuse Subsystem

Enabled Subsystem

Enable

Function-Call Subsystem

Subsystem

Inport

Outport

Signal Attributes

Data Type Conversion

Data Type Duplicate

Signal Conversion

Signal Routing

Bus Assignment

Bus Creator

Bus Selector

Demux

From

Goto

Goto Tag Visibility

Index Vector

Multiport Switch

Mux

Selector

Sinks

Display

Floating Scope

Scope

Stop Simulation

Terminator

To File

To Workspace

XY Graph

Sources

Constant

Counter Free-Running

Counter Limited

Enumerated Constant

Ground

Pulse Generator

Repeating Sequence Interpolated

Repeating Sequence Stair

User-Defined Functions

MATLAB Function

Fcn

Stateflow Blocks

The code supports the following Stateflow blocks.

Stateflow

Stateflow Chart

Truth Table

9

Limitations

- "Coder Limitations" on page 9-2
- "Block Restrictions" on page 9-6

Coder Limitations

In this section...
"Current Limitations" on page 9-2
"Fixed-Point Data Type Limitations" on page 9-3
"Permanent Limitations" on page 9-5

Current Limitations

The Simulink PLC Coder software does not support the following Simulink semantics:

- Complex data types
- Model reference
- Global data store memory (DSM)
- Absolute time temporal logic in Stateflow charts

Note Absolute time temporal logic is supported for only the Rockwell Automation RSLogix 5000 IDE.

- Stateflow machine-parented data and events
- Exported graphical functions in Stateflow charts
- Limited support for math functions. The coder does not support the following functions: `tanh`, `cosh`, `sinh`, `rand`.
- Merge block
- Multi-rate models
- Signal and state storage classes
- Virtual buses at the input ports of the top-level Atomic Subsystem block.
- For Each Subsystem block
- Variable-size signals

- Nonfinite data, for example NaN or Inf.

Fixed-Point Data Type Limitations

Simulink PLC Coder software supports the fixed-point data type. To generate code for fixed-point data types, configure block and model parameters as described in this topic.

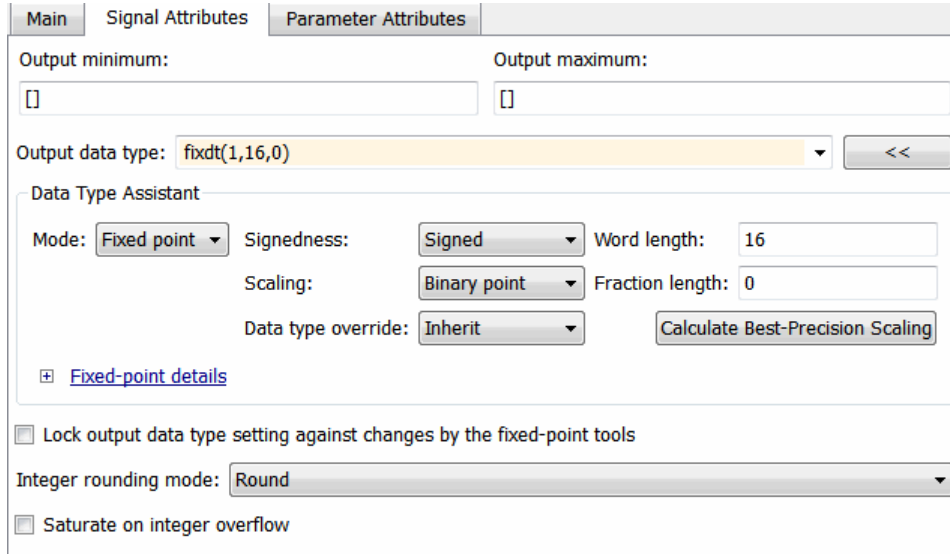
Note If you do not configure the blocks and models as directed, the generated structured text might:

- Not compile.
- Compile, but return results that differ from the simulation results.

Block Parameters

Properly configure block parameters:

1. If the block in the subsystem has a **Signal Attributes** tab, navigate to that tab.
2. For the **Integer rounding mode** parameter, select `Round`.
3. Clear the **Saturate on integer overflow** check box.
4. For the **Output data type** parameter, select a fixed-point data type.
5. Click the **Data Type Assistant** button.
6. For the Word length parameter, enter `8`, `16`, or `32`.
7. For the **Mode** parameter, select `Fixed point`.
8. For the **Scaling** parameter, select `Binary point`.



9 Click **OK**.

Be sure to edit the model configuration parameters (see "Model Configuration Parameters" on page 9-4).

Model Configuration Parameters

Properly configure model configuration parameters:

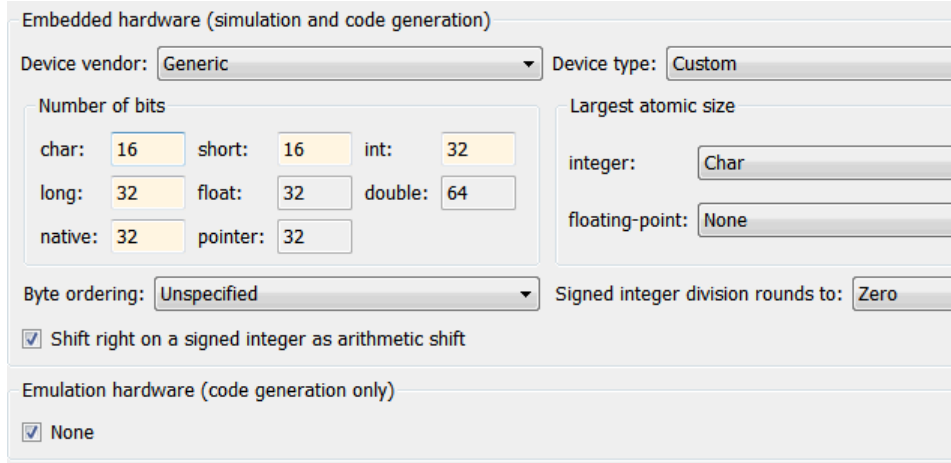
1 In model Configuration Parameters dialog box, click the `Hardware Implementation` node.

2 For the **Device vendor** parameter, select `Generic`.

3 For the **Device type**, select `Custom`.

4 For the **Signed integer division rounds to**, select `Zero`.

5 For the **Number of bits**, set **char** to `16`.



Permanent Limitations

The structured text language has inherent restrictions. As a result, the Simulink PLC Coder software has the following restrictions:

- The Simulink PLC Coder software supports generating code only for atomic subsystems.
- No blocks that require continuous time semantics. This restriction includes continuous integrators, zero-crossing blocks, physical modeling blocks, and so on.
- No pointer data types.
- No recursion (including recursive events).

Block Restrictions

In this section...

"Simulink Block Support Exceptions" on page 9-6

"Stateflow Chart Exceptions" on page 9-6

"Reciprocal Sqrt Block" on page 9-7

"Lookup Table Blocks" on page 9-7

Simulink Block Support Exceptions

The Simulink PLC Coder software supports the `plclib` blocks with the following exceptions. Also, see Chapter 9, "Limitations" for a list of limitations of the software.

If you get unsupported fixed-point type messages during code generation, update the block parameter. Open the block parameter dialog box. Navigate to the **Signal Attributes** and **Parameter Attributes** tabs. Check that the **Output data type** and **Parameter data type** parameters are not `Inherit: Inherit via internal rule`. Set these parameters to either `Inherit: Same as input` or a desired non-fixed-point data type, such as `double` or `int8`.

Stateflow Chart Exceptions

If you receive a message about consistency between the original subsystem and the S-function generated from the subsystem build, and the model contains a Stateflow chart that contains one or more Simulink functions, use the following procedure to address the issue:

1. Open the model and double-click the Stateflow chart that causes the issue.

   The chart Stateflow Editor dialog box is displayed.

2. Right-click in this dialog box.

3. In the context-sensitive menu, select **Properties**.

   The Chart dialog box is displayed.

4. In the Chart dialog box, navigate to the **States When Enabling** parameter and select `Held`.

5. Click **Apply** and **OK** and save the model.

Reciprocal Sqrt Block

The Simulink PLC Coder software does not support the Simulink Reciprocal Sqrt block `signedSqrt` and `rSqrt` functions.

Lookup Table Blocks

Simulink PLC Coder has limited support for lookup table blocks. The coder does not support:

- Number of dimensions greater than 2
- Cubic spline interpolation method
- Begin index search using a previous index mode
- Cubic spline extrapolation method

Note The Simulink PLC Coder software does not support the Simulink Lookup Table Dynamic block. For your convenience, the plclib/Simulink/Lookup Tables library contains an implementation of a dynamic table lookup block using the Prelookup and Interpolation Using Prelookup blocks.

10

Functions — Alphabetical List

plccoderdemos

Purpose

Product demos

Syntax

```
plccoderdemos
```

Description

`plccoderdemos` displays the Simulink PLC Coder demos in the MATLAB Help browser.

Examples

Display demos in the MATLAB Help browser.

```
plccoderdemos
```

See Also

`plcopenconfigset`

Purpose

Manage user preferences

Syntax

```
plccoderpref
plccoderpref('plctargetide')
plccoderpref('plctargetide', preference_value)
plccoderpref('plctargetide', 'default')
plccoderpref('plctargetidepaths')
plccoderpref('plctargetidepaths','default')
```

Description

`plccoderpref` displays the current set of user preferences, including the default target IDE.

`plccoderpref('plctargetide')` returns the current default target IDE. This default can be the target IDE set previously, or the factory default. The factory default is `'codesys23'`.

`plccoderpref('plctargetide', preference_value)` sets the default target IDE to the one that you specify in *`preference_value`*. This command sets the *`preference_value`* to persist as the default target IDE for all future MATLAB sessions.

`plccoderpref('plctargetide', 'default')` sets the default target IDE to the factory default target IDE (`'codesys23'`).

`plccoderpref('plctargetidepaths')` returns a 1-by-1 structure of the installation paths of all supported target IDEs.

`plccoderpref('plctargetidepaths','default')` sets the contents of the 1-by-1 structure of the installation paths to the default values.

Tips

Use the Simulink Configuration Parameters dialog box to change the installation path of a target IDE (**Target IDE Path**).

Input Arguments

`plctargetide`

String directive that specifies the default target IDE.

Value	Description
codesys23	3S-Smart Software Solutions CoDeSys Version 2.3 (default) target IDE
codesys33	3S-Smart Software Solutions CoDeSys Version 3.3 target IDE
brautomation30	B&R Automation Studio 3.0 target IDE
twincat211	Beckhoff TwinCAT 2.11 target IDE
multiprog50	KW-Software MULTIPROG 5.0 target IDE
pcworx60	Phoenix Contact PC WORX 6.0
rslogix5000	Rockwell Automation RSLogix 5000 Series target IDE for AOI format
rslogix5000_routine	Rockwell Automation RSLogix 5000 Series target IDE for routine format
step7	Siemens SIMATIC STEP 7 Version 5 target IDE
plcopen	PLCopen XML target IDE
generic	Generic target IDE

Default: codesys23

`plctargetidepaths`

String that specifies the target IDE installation path. Contains a 1-by-1 structure of the installation paths of all supported target IDEs.

```
codesys23: 'C:\Program Files\3S Software'
codesys33: 'C:\Program Files\3S CoDeSys'
rslogix5000: 'C:\Program Files\Rockwell Software'
rslogix5000_routine: 'C:\Program Files\Rockwell Software'
brautomation30: 'C:\Program Files\BrAutomation'
```

```
multiprog50: 'C:\Program Files\KW-Software\MULTIPROG 5.0'
pcworx60: 'C:\Program Files\Phoenix Contact\Software Suite 150'
step7: 'C:\Program Files\Siemens'
plcopen: ''
twincat211: 'C:\TwinCAT'
generic: ''
```

`default`

String that sets your preferences to the factory default.

Examples

Return the current default target IDE.

```
plccoderpref('plctargetide')
```

Set `rslogix5000` as the new default target IDE.

```
plccoderpref('plctargetide', 'rslogix5000')
```

Assume that you have previously changed the installation path of the CoDeSys 2.3 target IDE. Return the current target IDE installation paths.

```
          codesys23: 'C:\Program Files2\3S-Software\CoDeSys\v2.3
          codesys33: 'C:\Program Files\3S CoDeSys'
        rslogix5000: 'C:\Program Files\Rockwell Software'

rslogix5000_routine: 'C:\Program Files\Rockwell Software'
     brautomation30: 'C:\Program Files\BrAutomation'
        multiprog50: 'C:\Program Files\KW-Software\MULTIPROG 5.0'
           pcworx60: 'C:\Program Files\Phoenix Contact\Software Suite 150'
              step7: 'C:\Program Files\Siemens'
            plcopen: ''
         twincat211: 'C:\TwinCAT'
            generic: ''
```

Set the installation path of all the target IDEs, including CoDeSys 2.3, to factory default.

```
» plccoderpref('plctargetidepaths','default')

ans =

          codesys23: 'C:\Program Files\3S Software'
          codesys33: 'C:\Program Files\3S CoDeSys'
        rslogix5000: 'C:\Program Files\Rockwell Software'
rslogix5000_routine: 'C:\Program Files\Rockwell Software'
     brautomation30: 'C:\Program Files\BrAutomation'
        multiprog50: 'C:\Program Files\KW-Software\MULTIPROG 5.0'
           pcworx60: 'C:\Program Files\Phoenix Contact\Software Suite 150'
              step7: 'C:\Program Files\Siemens'
            plcopen: ''
         twincat211: 'C:\TwinCAT'
            generic: ''
```

Purpose

Generate structured text for subsystem

Syntax

```
generatedfiles = plcgeneratecode(subsystem)
```

Description

`generatedfiles = plcgeneratecode(subsystem)` generates structured text for the specified atomic subsystem in a model. *`subsystem`* is the fully qualified path name of the atomic subsystem. *`generatedfiles`* is a cell array of the generated file names. You must first load or start the model.

Examples

Generate code for the subsystem, `plcdemo_simple_subsystem/SimpleSubsystem`.

```
plcdemo_simple_subsystem
generatedfiles = plcgeneratecode('plcdemo_simple_subsystem/SimpleSubsystem')
```

See Also

`plcopenconfigset`

plcopenconfigset

Purpose

Open Configuration Parameters dialog box for subsystem

Syntax

```
plcopenconfigset(subsystem)
```

Description

`plcopenconfigset(`*`subsystem`*`)` opens the Configuration Parameters dialog box for the specified atomic subsystem in the model. *subsystem* is the fully qualified path name of the atomic subsystem.

Examples

Open the Configuration Parameters dialog box for the subsystem, `plcdemo_simple_subsystem/SimpleSubsystem`.

```
plcdemo_simple_subsystem
plcopenconfigset('plcdemo_simple_subsystem/SimpleSubsystem')
```

See Also

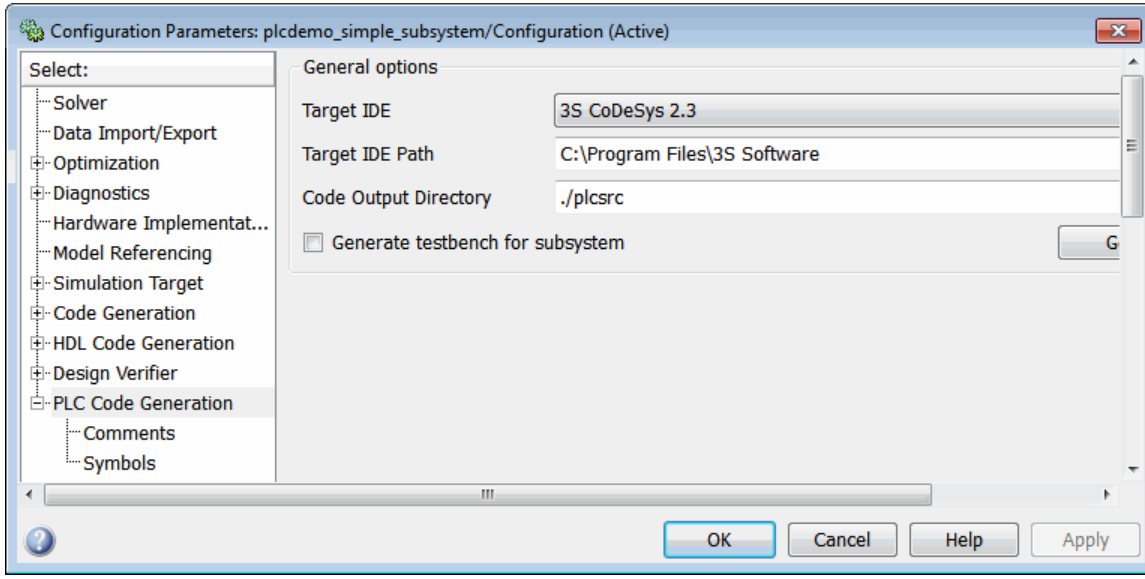
`plcgeneratecode`

11

Configuration Parameters for Simulink PLC Coder Models

- "PLC Coder: General" on page 11-2
- "PLC Coder: Comments" on page 11-10
- "PLC Coder: Symbols" on page 11-14

PLC Coder: General



In this section...
“PLC Coder: General Tab Overview” on page 11-3
“Target IDE” on page 11-4
“Target IDE Path” on page 11-6
“Code Output directory ” on page 11-8
“Generate testbench for subsystem” on page 11-9

PLC Coder: General Tab Overview

Set up general information about generating structured text code to download to target PLC IDEs.

Configuration

To enable the Simulink PLC Coder options pane, you must:

1. Create a model.
2. Add either an Atomic Subsystem block, or a Subsystem block for which you have selected the **Treat as atomic unit** check box.
3. Right-click the subsystem block and select **PLC Code Generation > Options**.

Tip

In addition to configuring parameters for the Simulink PLC Coder model, you can also use this dialog box to generate structured text code and test bench code for the Subsystem block.

See Also

"Preparing Your Model to Generate Structured Text Code" on page 1-13

"Generating Structured Text Code from the Model Window" on page 1-23

Target IDE

Select the target IDE in which to generate code.

Settings

Default: `3S CoDeSys 2.3`

`3S CoDeSys 2.3`
: Generates structured text (IEC 61131) code for 3S-Smart Software Solutions CoDeSys Version 2.3.

`3S CoDeSys 3.3`
: Generates structured text code in PLCopen XML for 3S-Smart Software Solutions CoDeSys Version 3.3.

`B&R Automation Studio 3.0`
: Generates structured text code for B&R Automation Studio 3.0.

`Beckhoff TwinCAT 2.11`
: Generates structured text code for Beckhoff TwinCAT 2.11 software.

`KW-Software MULTIPROG 5.0`
: Generates structured text code in PLCopen XML for KW-Software MULTIPROG® 5.0.

`Phoenix Contact PC WORX 6.0`
: Generates structured text code in PLCopen XML for Phoenix Contact PC WORX 6.0.

`Rockwell RSLogix 5000 17, 18:  AOI`
: Generates structured text code for Rockwell Automation RSLogix 5000 using Add-On Instruction (AOI) constructs.

`Rockwell RSLogix 5000 17, 18:  Routine`
: Generates structured text code for Rockwell Automation RSLogix 5000 routine constructs.

`Siemens SIMATIC Step 7 5.4`
: Generates structured text code for Siemens SIMATIC STEP 7 5.4.

`Generic`
: Generates a pure structured text file. If the target IDE that you want is not available for the Simulink PLC Coder product, consider generating and downloading a generic structured text file.

`PLCopen XML`
: Generates structured text code formatted using PLCopen XML standard.

Tip

- Rockwell Automation RSLogix 5000 routines represent the model hierarchy using hierarchical user-defined types (UDTs). UDT types preserve model hierarchy in the generated code.
- The coder generates code for reusable subsystems as separate routine instances. These subsystems access instance data in program tag fields.

Command-Line Information

Parameter: `PLC_TargetIDE`
Type: string
Value: `'codesys23' | 'codesys33' | 'rslogix5000' | 'rslogix5000_routine' | 'brautomation30' | 'multiprog50' | 'pcworx60' | 'step7' | 'plcopen' | 'twincat211' | 'generic'`
Default: `'codesys23'`

See Also

"Generating Structured Text Code from the Model Window" on page 1-23

Target IDE Path

Enter target IDE installation path. The listed path is the factory default for the **Target IDE** entry.

Settings

Default: `C:\Program Files\3S Software`

`C:\Program Files\3S Software`
: Factory default installation path for 3S-Smart Software Solutions CoDeSys software Version 2.3.

`C:\Program Files\3S CoDeSys`
: Factory default installation path for 3S-Smart Software Solutions CoDeSys software Version 3.3..

`C:\Program Files\BrAutomation`
: Factory default installation path for B&R Automation Studio 3.0 software.

`C:\TwinCAT`
: Factory default installation path for Beckhoff TwinCAT 2.11 software.

`C:\Program Files\KW-Software\MULTIPROG 5.0`
: Factory default installation path for KW-Software MULTIPROG 5.0 software.

`C:\Program Files\Phoenix Contact\Software Suite 150`
: Factory default installation path for Phoenix Contact PC WORX 6.0 software.

`C:\Program Files\Rockwell Software`
: Factory default installation path for Rockwell Automation RSLogix 5000 software.

`C:\Program Files\Siemens`
: Factory default installation path for Siemens SIMATIC STEP 7 5.4 software.

Tip

- The value of this parameter changes when you change the **Target IDE** value.

- If you right-click the Subsystem block, the **PLC Code Generation > Generate and Import Code for Subsystem** command uses this value to import generated code.
- If your target IDE installation is standard, do not edit this parameter. Leave it as the default value.
- If your target IDE installation is nonstandard, edit this value to specify the actual installation path.
- If you change the path and click **Apply**, the changed path remains in effect for that target IDE for other models and between MATLAB sessions. To reinstate the factory default, use the command:

  ```
  plccoderpref('plctargetidepaths','default')
  ```

Command-Line Information

See `plccoderpref`.

See Also

"Automatically Importing Structured Text Code" on page 1-33

Code Output directory

Enter a path to the target folder into which code is generated.

Settings

Default: `plcsrc` subfolder in your working folder

Command-Line Information

Parameter: `PLC_OutputDir`
Type: string
Value: `'./plcsrc'`
Default: `'./plcsrc'`

See Also

"Generating Structured Text Code from the Model Window" on page 1-23

Generate testbench for subsystem

Specify the generation of test bench code for the subsystem.

Settings

Default: off

 On

Enables generation of test bench code for subsystem.

Disables generation of test bench code for subsystems.

Tips

If you right-click the Subsystem block and choose **PLC Code Generation > Generate and Import Code for Subsystem**, the software also generates the test bench for the subsystem, regardless of the setting of the **Generate testbench for subsystem** check box.

Dependency

This parameter is disabled if your model has absolute time temporal logic.

Note The Simulink PLC Coder software supports absolute time temporal logic in Stateflow charts for the Rockwell Automation RSLogix 5000 IDE.

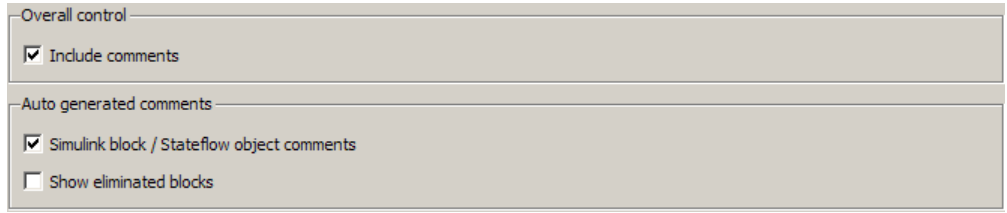
Command-Line Information

Parameter: `PLC_GenerateTestbench`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

See Also

"Generating Structured Text Code from the Model Window" on page 1-23

PLC Coder: Comments



In this section...

"Comments Overview" on page 11-11

"Include comments" on page 11-11

"Simulink block / Stateflow object comments " on page 11-12

"Show eliminated blocks" on page 11-13

Comments Overview

Control the comments that the Simulink PLC Coder software automatically creates and inserts into the generated code.

See Also

"Generating Structured Text Code from the Model Window" on page 1-23

Include comments

Specify which comments are in generated files.

Settings

Default: on

 On

Places comments in the generated files based on the selections in the **Auto generated comments** pane.

 Off

Omits comments from the generated files.

Command-Line Information

Parameter: `PLC_RTWGenerateComments`
Type: string
Value: `'on' | 'off'`
Default: `'on'`

See Also

"Generating Structured Text Code from the Model Window" on page 1-23

Simulink block / Stateflow object comments

Specify whether to insert Simulink block and Stateflow object comments.

Settings

Default: on

 On

Inserts automatically generated comments that describe block code and objects. The comments precede that code in the generated file.

 Off

Suppresses comments.

Command-Line Information

Parameter: `PLC_RTWSimulinkBlockComments`
Type: string
Value: `'on' | 'off'`
Default: `'on'`

See Also

“Generating Structured Text Code from the Model Window” on page 1-23

Show eliminated blocks

Specify whether to insert eliminated block comments.

Settings

Default: off

 On

Inserts statements in the generated code from blocks eliminated as the result of optimizations (such as parameter inlining).

 Off

Suppresses statements.

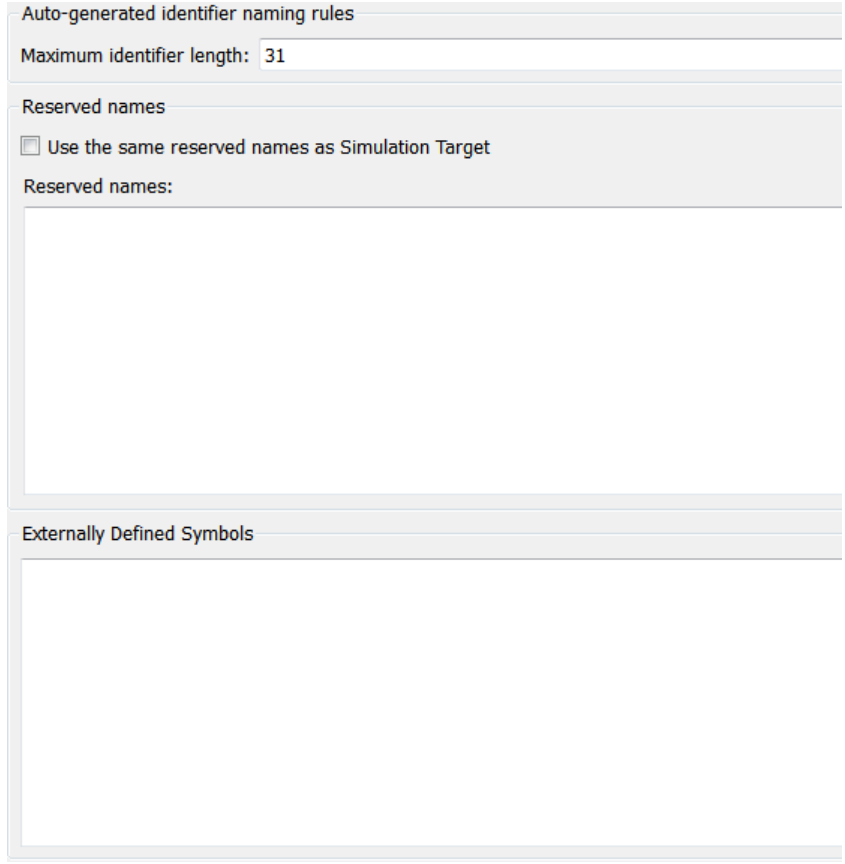
Command-Line Information

Parameter: `PLC_RTWShowEliminatedStatement`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

See Also

“Generating Structured Text Code from the Model Window” on page 1-23

PLC Coder: Symbols



In this section...
“Symbols Overview” on page 11-15
“Maximum identifier length” on page 11-16
“Use the same reserved names as Simulation Target” on page 11-17
“Reserved names” on page 11-18
“Externally Defined Symbols” on page 11-19

Symbols Overview

Select the automatically generated identifier naming rules.

See Also

"Generating Structured Text Code from the Model Window" on page 1-23

Maximum identifier length

Specify the maximum number of characters in generated function, type definition, and variable names.

Settings

Default: 31

Minimum: 31

Maximum: 256

You can use this parameter to limit the number of characters in function, type definition, and variable names. Many target IDEs have their own restrictions. The Simulink PLC Coder software complies with target IDE limitations.

Command-Line Information

Parameter: PLC_RTWMaxIdLength
Type: int
Value: 31 to 256
Default: 31

See Also

"Generating Structured Text Code from the Model Window" on page 1-23

Use the same reserved names as Simulation Target

Specify whether to use the same reserved names as those specified in the **Simulation Target > Symbols pane**.

Settings

Default: off

 On

Enables using the same reserved names as those specified in the **Simulation Target > Symbols pane** pane.

 Off

Disables using the same reserved names as those specified in the **Simulation Target > Symbols pane** pane.

Command-Line Information

Parameter: `PLC_RTWUseSimReservedNames`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

See Also

“Generating Structured Text Code from the Model Window” on page 1-23

Reserved names

Enter the names of variables or functions in the generated code that you do not want to be used.

Settings

Default: `( )`

This action changes the names of variables or functions in the generated code to avoid name conflicts with identifiers in custom code. Reserved names must be shorter than 256 characters.

Tips

- Start each reserved name with a letter or an underscore.
- Each reserved name must contain only letters, numbers, or underscores.
- Separate the reserved names using commas or spaces.

Command-Line Information

Parameter: `PLC_RTWReservedNames`
Type: string
Value: `string`
Default: `''`

See Also

"Generating Structured Text Code from the Model Window" on page 1-23

Externally Defined Symbols

Specify the names of identifiers for which you want to suppress definitions.

Settings

Default: `( )`

This action suppresses the definition of identifiers, such as those for function blocks, variables, constants, and user types in the generated code. This suppression allows the generated code to refer to these identifiers. When you import the generated code into the PLC IDE, you must provide these definitions.

Tips

- Start each name with a letter or an underscore.
- Each name must contain only letters, numbers, or underscores.
- Separate the names using spaces.

Command-Line Information

Parameter: `PLC_ExternalDefinedNames`
Type: string
Value: `string`
Default: `''`

See Also

- “Generating Structured Text Code from the Model Window” on page 1-23
- Chapter 6, “Integrating Externally Defined Symbols”
- `Integrating User Defined Function Blocks, Data Types, and Global Variables into Generated Structured Text`

Index

A

accessing demos 1-6
accessing supported blocks 1-7
atomic subsystems 1-13

B

B&R Automation Studio® 1-8
  accessing 1-10
basic workflow 1-12
Beckhoff® TwinCAT® 1-8
  accessing 1-10
before you start 1-13
block parameters 5-5

C

changing name of a subsystem 5-7
Code generation
  development process 1-3
CoDeSys 1-8
  importing 1-10
compatibility 1-19
configuration parameters
  pane 11-3 11-11 11-15
    Externally Defined Symbols 11-19
    Include comments 11-11
    Maximum identifier length: 11-16
    Output directory 11-8 to 11-9
    Reserved names: 11-18
    Show eliminated blocks 11-13
    Simulink block / Stateflow object comments 11-12
    Target IDE 11-4
    Target IDE Path 11-6
    Use the same reserved names as Simulation Target 11-17
  PLC Coder: General 11-2
  reference 11-1
configuring parameters to be tunable 4-18
configuring Simulink® models 1-13
configuring tunable parameters 4-2

D

defining tunable parameters in the MATLAB® workspace 4-11 4-16

E

expected background 1-4
expected users 1-4

F

function block 1-5
  generating multiple for nonvirtual subsystems 5-4
  generating one for atomic subsystems 5-2
  generating one for virtual subsystems 5-3
  partitions 5-2
functions
  `plccoderdemos` 10-2
  `plccoderpref` 10-3
  `plcgeneratecode` 10-7
  `plcopenconfigset` 10-8

G

generated code
  controlling with block parameters 5-5
  mapping MATLAB® Coder™ subsystems to function blocks 2-10
  mapping reusable code to function blocks 2-4
  mapping Stateflow® enabled and triggered subsystems to function blocks 2-6
  mapping Stateflow® subsystems to function blocks 2-8
  mapping to function blocks 2-2
  partitioning 5-1

generating and examining structured text code 1-23
generating and importing structured text 3-5
generating code 1-23
MATLAB® interface 1-30
generating separate partitions and inlining subsystem code 5-6
generating structured text code 1-23
generating test bench code 3-1
glossary 1-5

I

IDE platforms
supported 1-8
identifying tunable parameters 4-7
IEC 61131-3 1-5

K

KW-Software MULTIPROG®
accessing 1-10

L

limitations 9-1

M

mapping Simulink® semantics 2-1

N

nonvirtual subsystems 5-4

P

partitioning
controlling generated code with subsystem block parameters 5-5
generated code 5-1
multiple function block for nonvirtual subsystems 5-4
one function block for atomic subsystems 5-2
one function block for virtual subsystems 5-3
platforms 1-8
PLC Coder: General
configuration parameters 11-2
`plccoderdemos` function 10-2
`plccoderpref` function 10-3
plcgeneratecode 1-30
`plcgeneratecode` function 10-7
PLCopen 1-5
plcopenconfigset 1-30
`plcopenconfigset` function 10-8

R

related products 1-7
requirements 1-7
Rockwell Automation® RSLogix™ 1-8
accessing 1-11
considerations 7-4
double-precision data types 7-4
enumerated integer data types 7-5
unsigned fixed-point data types 7-5
unsigned integer data types 7-4

S

Siemens® SIMATIC® STEP® 7
accessing 1-11
considerations 7-6
double-precision floating-point data types 7-6
enumerated integer data types 7-7
int8 and unsigned integer data types 7-6
Siemens®SIMATIC®STEP® 7
unsigned fixed-point data types 7-6
Simulink®
semantics 2-1

Simulink® PLC Coder™
limitations 9-1
structured text 1-5
subsystem block parameters 5-5
subsystems
atomic 1-13
supported blocks
accessing 1-7
supported IDE platforms 1-8
system requirements 1-8

T

target platforms 1-8
test bench code
generating 3-1
Treat as atomic unit 1-13
tunable parameters
about 4-2
configuring 4-2
identifying 4-7
MATLAB® workspace 4-11 4-16

U

useful terms 1-5

V

virtual subsystems 5-3

W

working with generated structured text 3-2