Simulink® PLC Coder™ 1
User's Guide

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

© COPYRIGHT 2010–2011 by The MathWorks, Inc.

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)

Contents

Getting Started

1

Mapping Simulink Semantics to Structured Text

2

Generating Test Bench Code

3

Working with Tunable Parameters in the Simulink® PLC Coder Environment

4

Controlling Generated Code Partitions

5

8

Configuration Parameters for Simulink® PLC Coder Models

9

Index

1

Getting Started

Product Overview

Introduction

Simulink® PLC Coder™ generates hardware-independent IEC 61131-3 structured text from Simulink® models, Stateflow® charts, and MATLAB® Coder™ 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.

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

- MATLAB Coder 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 correctness of 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

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.

Expected Background

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 1-7 for a list of these platforms.

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 1-6
"Supported Simulink and Stateflow Blocks" on page 1-6
"System Requirements" on page 1-7
"Supported IDE Platforms" on page 1-7

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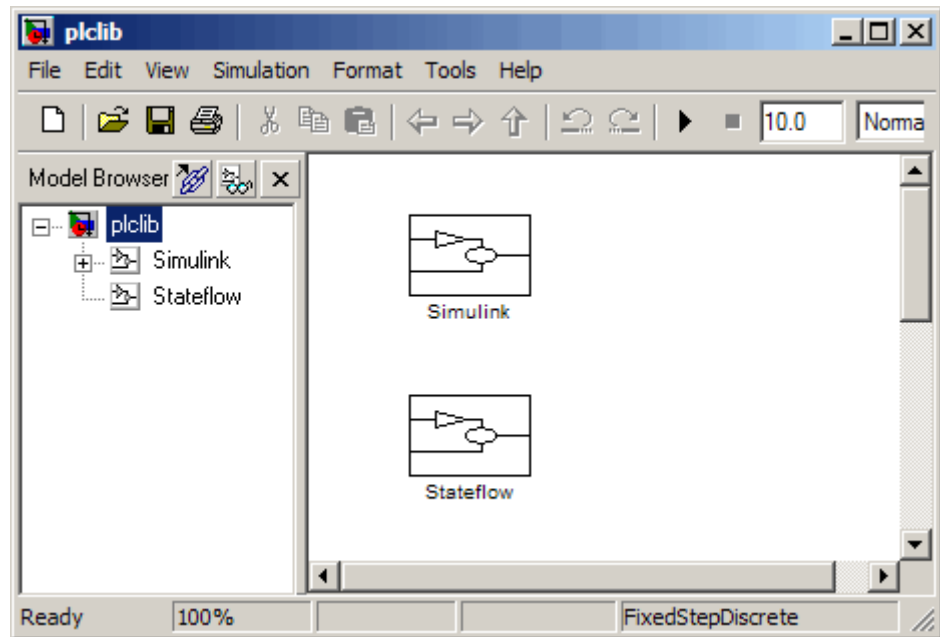
This library contains two sublibraries, Simulink and Stateflow. Each sublibrary contains the blocks that you can include in a Simulink PLC Coder model.

See "Block Restrictions" on page 7-6 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
- 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/
```

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 downloading to a PLC.

3 Place the components in a Subsystem block.

4 Identify your target PLC IDE.

5 Configure the Subsystem block to be atomic.

6 Check that the model is compatible with the Simulink PLC Coder software.

7 Simulate your model.

8 Configure model parameters to generate code for your PLC IDE.

9 Examine the generated code.

10 Import code to your PLC IDE.

Preparing Your Model to Generate Structured Text Code

In this section...
"Configuring Simulink Models for Structured Text Code Generation" on page 1-12
"Ensuring System Compatibility for Structured Text Code Generation" on page 1-17

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:
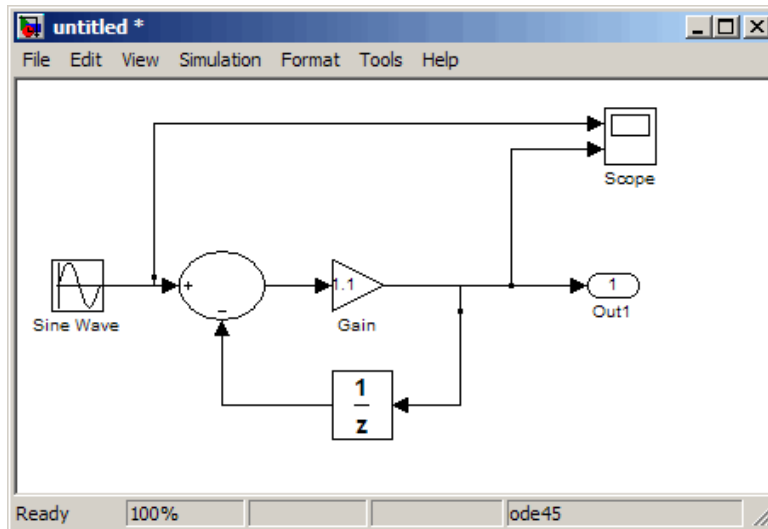
2 Save this model as `plcdemo_simple_subsystem1.mdl`.

3 Place the components for which you want to generate structured text code in a subsystem. For example:
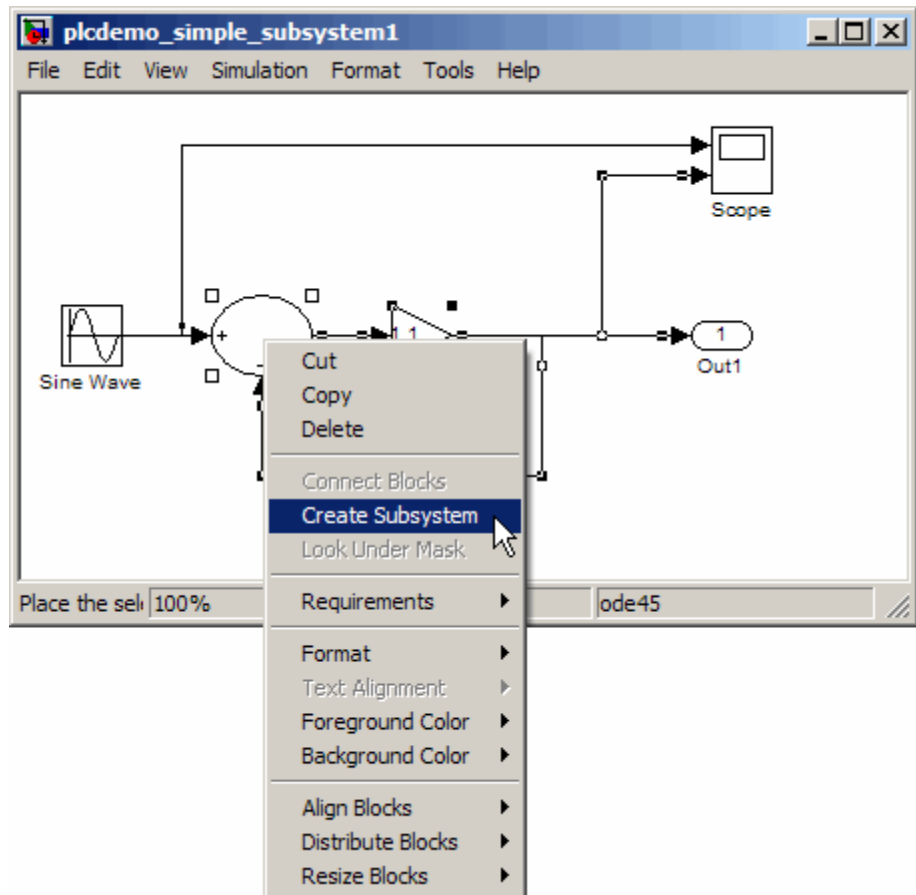
Optionally, rename In1 and Out1 to U and Y respectively. This operation results in a subsystem like the following:
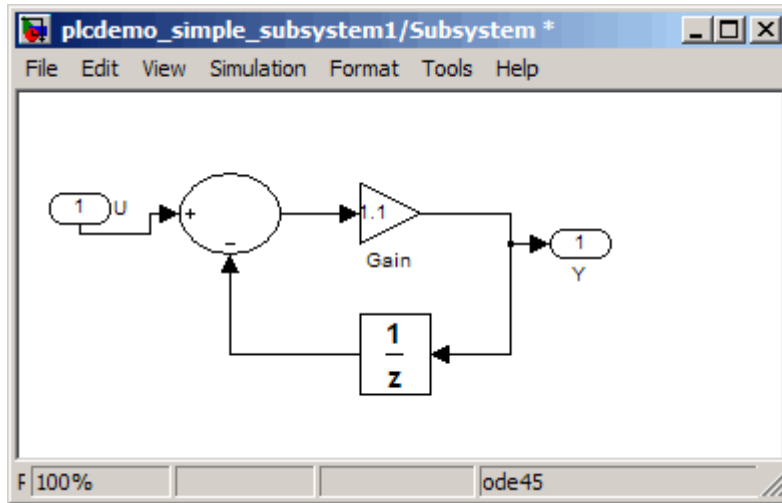
4 Save the subsystem.

5 In the top-level model, right-click the Subsystem block and select **Subsystem Parameters**.
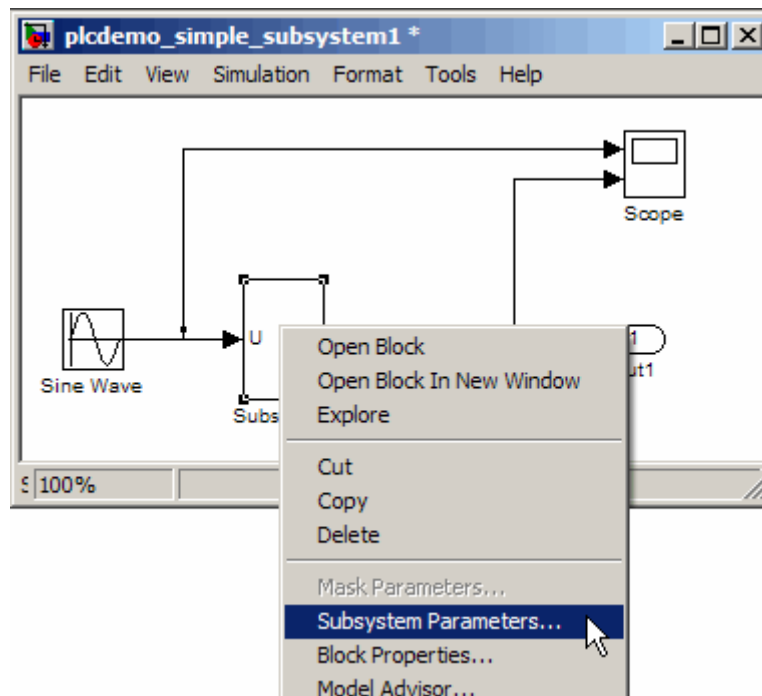
6 In the resulting block dialog box, select **Treat as atomic unit**.
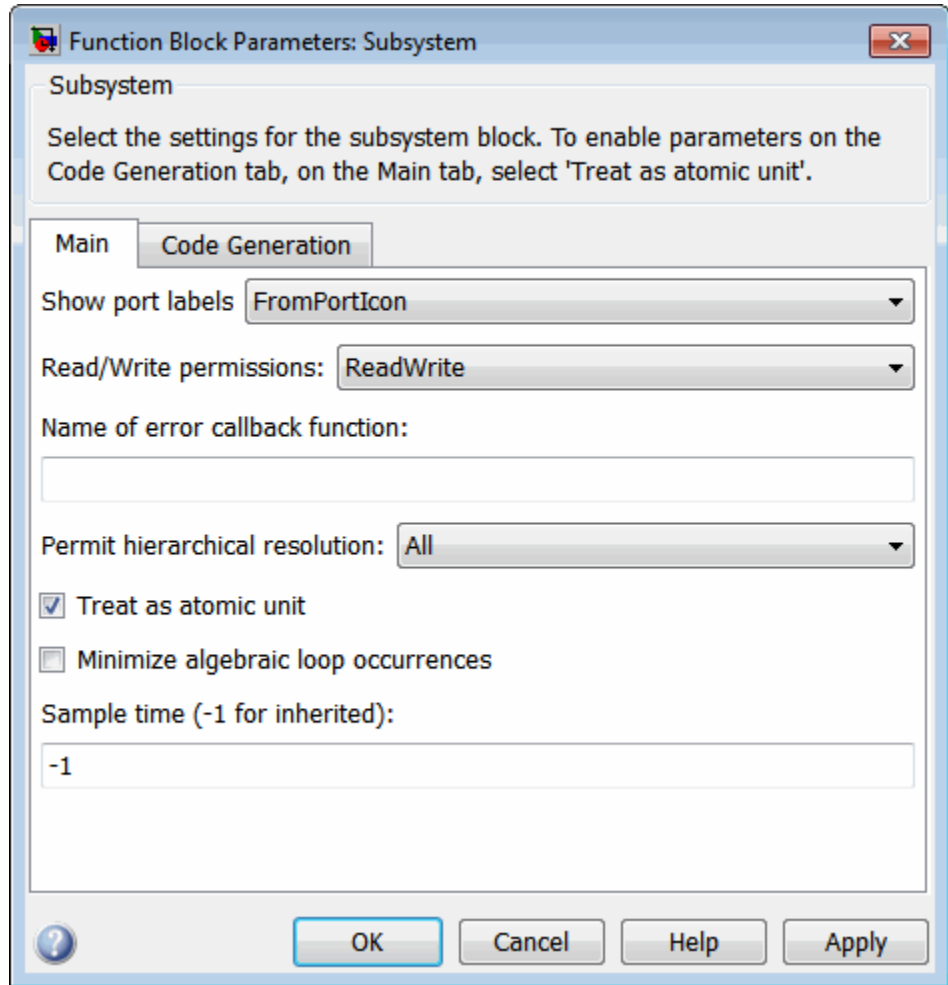
7 Click **OK**.

8 Simulate your model.

9 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:

- Ensure that you have properly set up your subsystem to generate structured text code. See "Ensuring System Compatibility for Structured Text Code Generation" on page 1-17.

- Generate structured text code for your IDE. See "Generating and Examining Structured Text Code" on page 1-21.

Ensuring 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**.
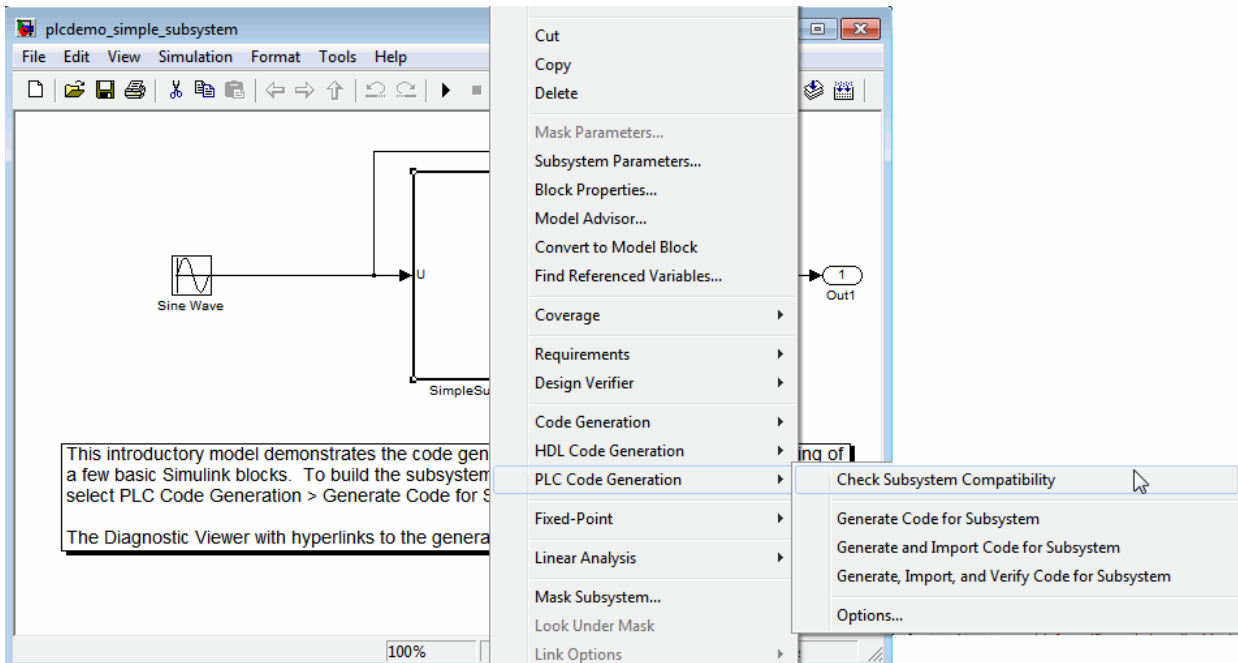
The coder verifies that your model satisfies the Simulink PLC Coder criteria and displays an information window when done.
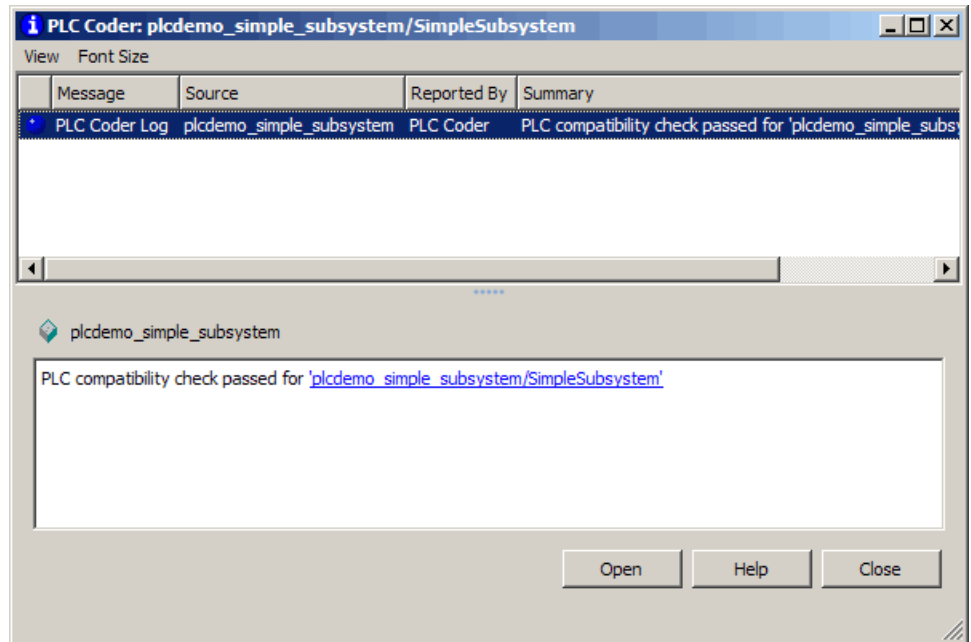
If the subsystem is not atomic, right-clicking the Subsystem block and selecting **PLC Code Generation** prompts you to select **Enable "Treat as atomic unit" to generate code**.
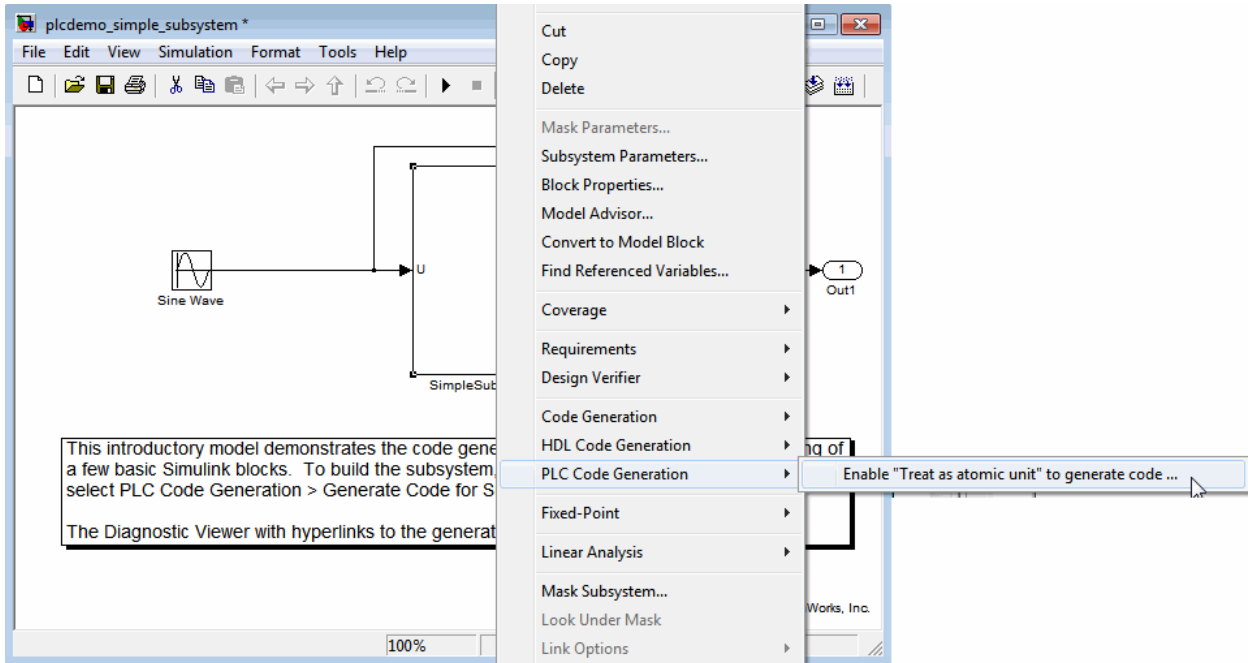
This command opens the block parameter dialog box so that you can select the **Treat as atomic unit** check box.
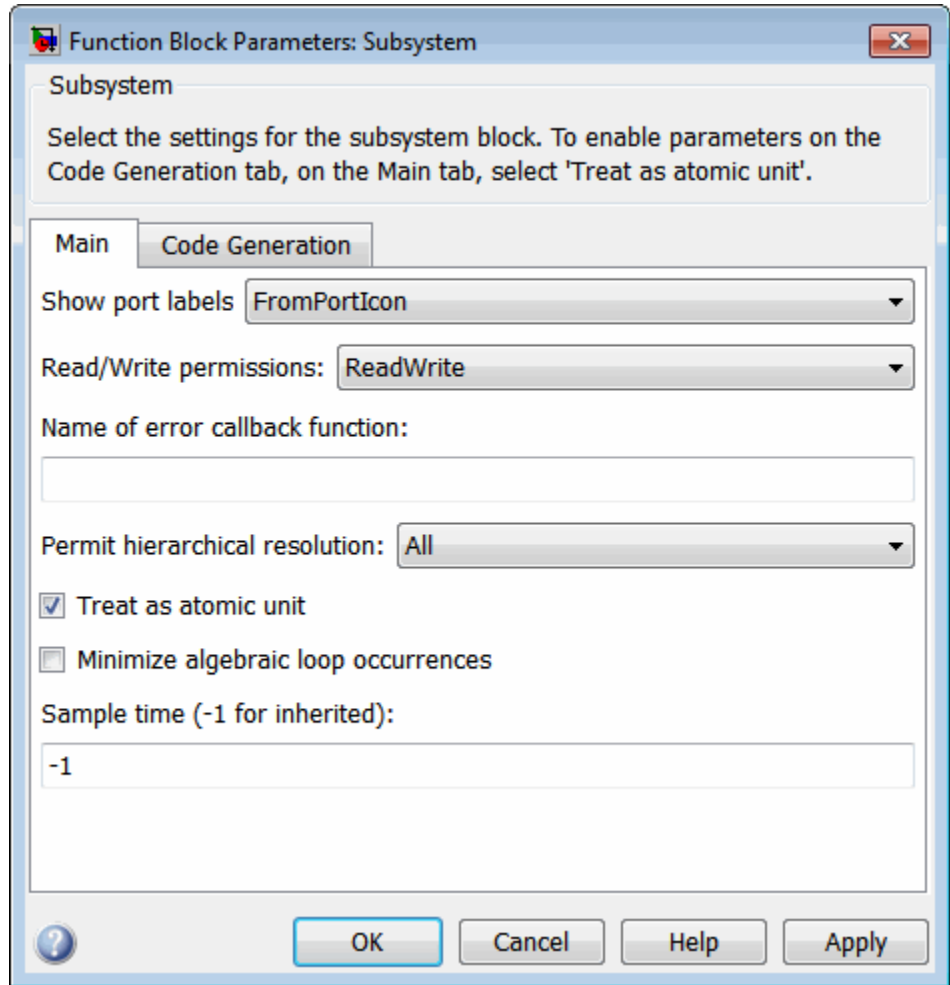
You are now ready to generate structured text code for your IDE. See "Generating and Examining Structured Text Code" on page 1-21.

Generating and Examining Structured Text Code

In this section...

"Generating Structured Text Code from the Model Window" on page 1-21

"Generating Structured Text Code with the MATLAB Interface" on page 1-28

"Generating Structured Text Code and Integrating with Existing Siemens SIMATIC STEP 7 Projects" on page 1-29

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 1-12.

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**.
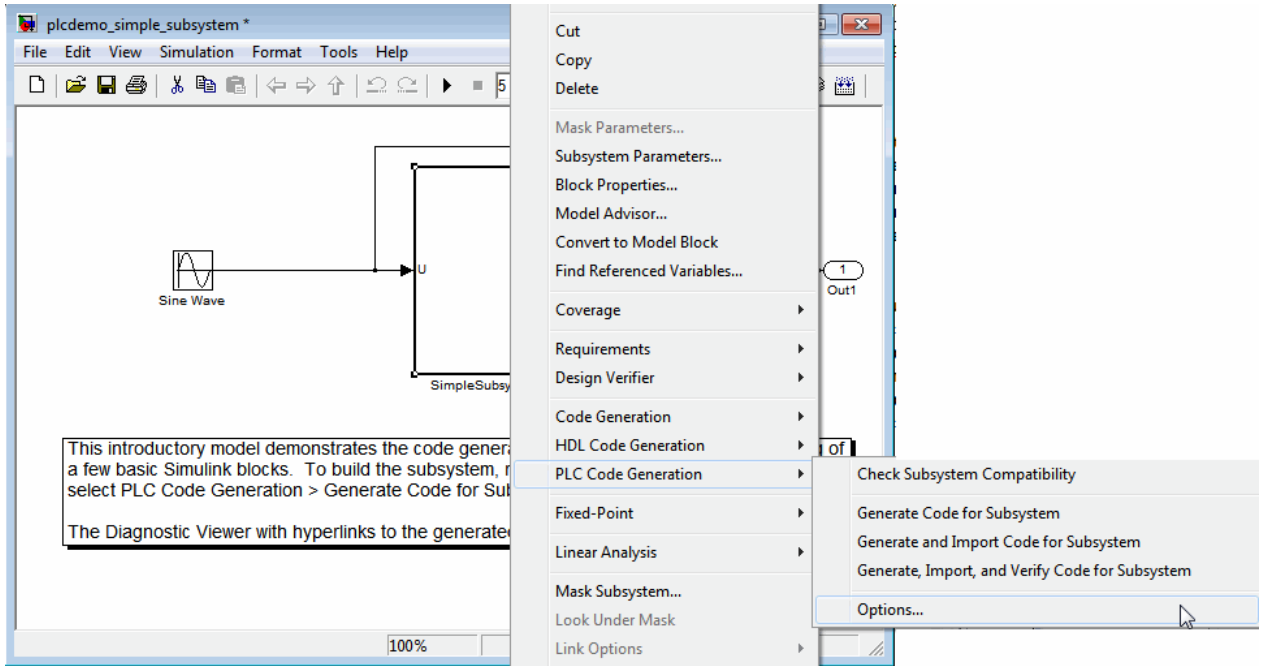
The Configuration Parameters dialog box is displayed.
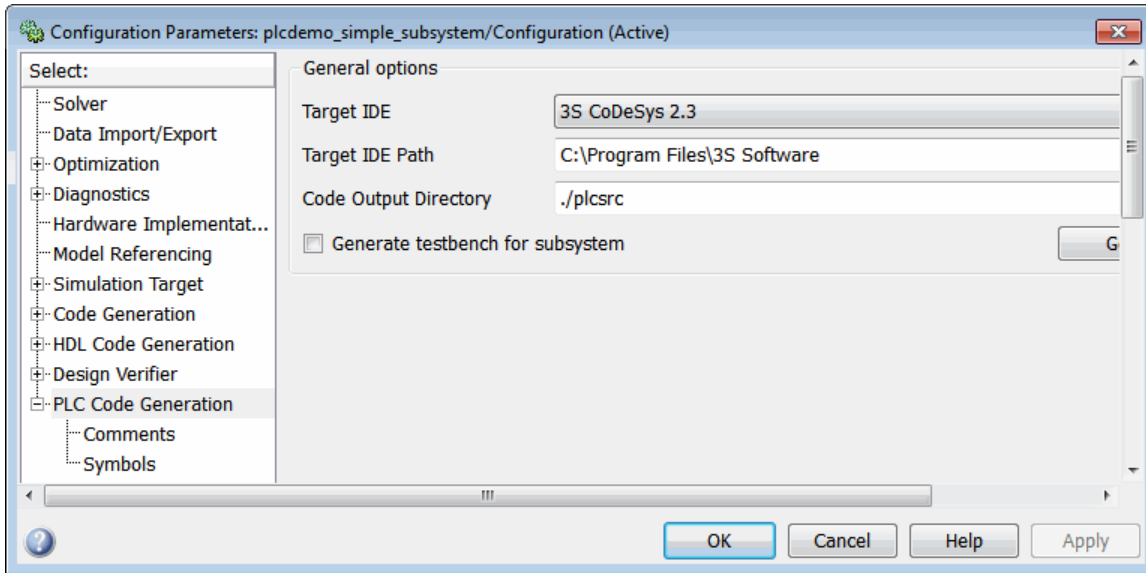
3 In **PLC Code Generation > General options > Target IDE**, select a target IDE. For example, select CoDeSys 2.3.

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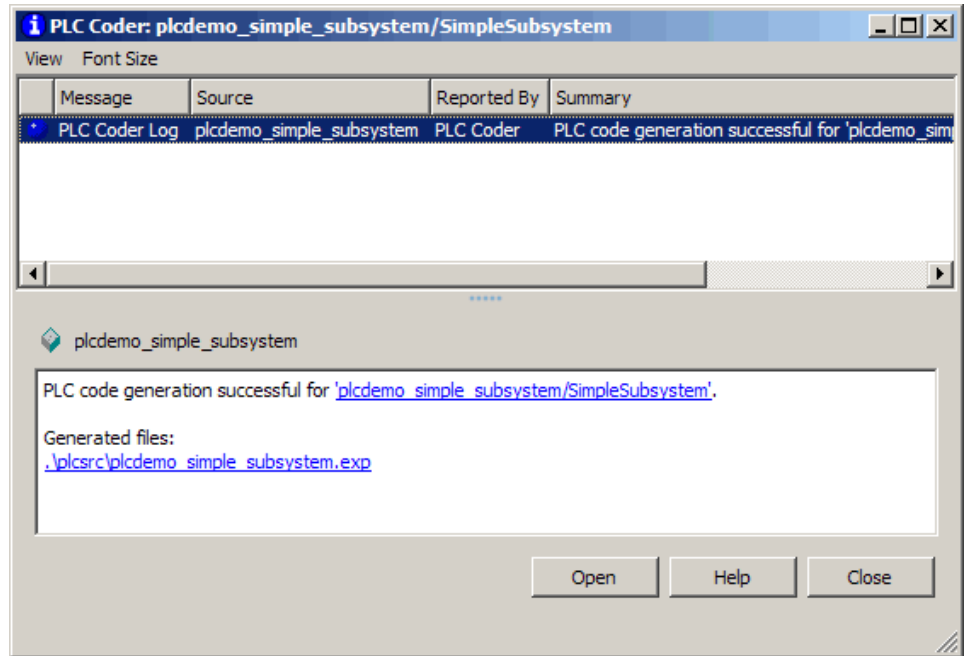
This window has links that you can click to open the associated files.

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 1-27.

Platform	Generated Files
3S-Smart Software Solutions CoDeSys IDE	*current_folder*\plcsrc\model_name.exp — Structured text file appropriate for downloading to the target IDE.
B&R Automation Studio IDE	The following files in *current_folder*\plcsrc\model_name — Files appropriate for downloading 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.

Platform	Generated Files
	In the subsystem folder: • `IEC.lby` — Function blocks library definition file in XML format. • `subsystem_block_name.fun` — Text file. Function block interface definitions file. The coder generates one file per project. • `subsystem_block_name.st` — Text file. Structured text function body files. The coder generates one file per function block in the generated code. • `subsystem_block_name.typ` — Text file. Structure and enumerated type definitions file. The coder generates one file per project. • `subsystem_block_name.var` — Text file. Global constant definitions file. The coder generates one file per project. • `TestBench.st` — (If test bench is generated) Text file. Structured text file for generated test bench code. 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 appropriate for downloading to the target IDE.
KW-Software MULTIPROG 5.0	*current_folder*\plcsrc\model_name.xml — Structured text file, in XML format, appropriate for downloading to the target IDE.
Phoenix Contact PC WORX 6.0	*current_folder*\plcsrc\model_name.xml — Structured text file, in XML format, appropriate for downloading to the target IDE.

Platform	Generated Files
Rockwell Automation RSLogix 5000 IDE	*current_folder*\plcsrc\model_name.L5X . — Structured text file appropriate for downloading to the target IDE. This file is in XML format and contains the generated structured text code for your model.
Siemens SIMATIC STEP 7 IDE	*current_folder*\plcsrc\model_name.scl — Structured text file appropriate for downloading to the target IDE. *current_folder*\plcsrc\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 downloading 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 downloading 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.

```
15   FUNCTION_BLOCK SimpleSubsystem
16   VAR_INPUT
17       ssMethodType: SINT;
18       U: LREAL;
19   END_VAR
20   VAR_OUTPUT
21       Y: LREAL;
22   END_VAR
23   VAR
24       UnitDelay_DSTATE: LREAL;
25   END_VAR
26   VAR_TEMP
27       rtb_Gain: LREAL;
28   END_VAR
29   CASE ssMethodType OF
30       SS_INITIALIZE:
31           (* InitializeConditions for UnitDelay: '<S1>/Unit Delay'  *)
32           UnitDelay_DSTATE := 0;
33
34       SS_OUTPUT:
35           (* Gain: '<S1>/Gain' incorporates:
36            *  Inport: '<Root>/U'
37            *  Sum: '<S1>/Sum'
38            *  UnitDelay: '<S1>/Unit Delay'
39             *)
40           rtb_Gain := (U - UnitDelay_DSTATE) * 0.5;
41
42           (* Outport: '<Root>/Y'  *)
43           Y := rtb_Gain;
44
45           (* Update for UnitDelay: '<S1>/Unit Delay'  *)
46           UnitDelay_DSTATE := rtb_Gain;
47
48   END_CASE;
49   END_FUNCTION_BLOCK
```

After generating structured text code, examine it to ensure that it is what you expect. See Chapter 2, "Mapping Simulink Semantics to Structured Text" for a description of how the generated code for the Simulink components map to structured text components.

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 1-31.

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 1-12, 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 1-21.

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 1-21.

• 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 1-31
"Generating and Automatically Importing Structured Text Code" on page 1-32
"Troubleshooting Automatic Import Issues" on page 1-33

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 3-7.

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 9-6 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**.

Note When automatically importing code, the software creates and downloads a test bench regardless of the check box setting **PLC Code Generation > Generate testbench for subsystem** in the Configuration Parameters dialog box.

The software then:

a Generates the code and test bench.

    b Starts the target IDE interface.

    c Creates a new project.

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

If you want to generate, import, and run the structured text code, see "Automatically Importing Structured Text Code" on page 3-6.

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)

- 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 error in a message dialog box. To remedy issue, try the following actions:

- Ensure that the coder supports the target IDE version and language setting combination.

- Ensure that you have correctly 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.

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 1-21). All demos are located in the *matlabroot*\toolbox\plccoder\plccoderdemos folder.

These topics use code generated with CoDeSys Version 2.3.

- "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 1-21.

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 now in the MATLAB editor. Type the following in the folder that contains the file:

```
edit plcdemo_simple_subsystem.exp
```

A file like the following is displayed.

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

Input parameter for subsystem method type

Atomic subsystem name

Subsystem

Subsystem inputs and outputs

Subsystem State (DWork) variables

Initialize, output, and update methods

```
15  FUNCTION_BLOCK SimpleSubsystem
16  VAR_INPUT
17      ssMethodType: SINT;
18      U: LREAL;
19  END_VAR
20  VAR_OUTPUT
21      Y: LREAL;
22  END_VAR
23  VAR
24      UnitDelay_DSTATE: LREAL;
25  END_VAR
26  VAR_TEMP
27      rtb_Gain: LREAL;
28  END_VAR
29  CASE ssMethodType OF
30      SS_INITIALIZE:
31          (* InitializeConditions for UnitDelay: '<S1>/Unit Delay'  *)
32          UnitDelay_DSTATE := 0;
33
34      SS_OUTPUT:
35          (* Gain: '<S1>/Gain' incorporates:
36          *   Inport: '<Root>/U'
37          *   Sum: '<S1>/Sum'
38          *   UnitDelay: '<S1>/Unit Delay'
39          *)
40          rtb_Gain := (U - UnitDelay_DSTATE) * 0.5;
41
42          (* Outport: '<Root>/Y'  *)
43          Y := rtb_Gain;
44
45          (* Update for UnitDelay: '<S1>/Unit Delay'  *)
46          UnitDelay_DSTATE := rtb_Gain;
```

Inlined parameters

2 Inspect this code as you ordinarily do for PLC code. Ensure that the generated code is as you expect.

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 1-21.

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 now in the MATLAB editor.

  This figure illustrates the mapping of the generated code for a reusable Simulink subsystem to structured text components. 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).

```
                                              CASE ssMethodType OF
                                                  SS_INITIALIZE:
                                                      (* InitializeConditions for atomic SubSystem: '<S1>/S1'  *)
                                                      _i0_ReusableSubsystem_S1(ssMethodType := SS_INITIALIZE, rtu_U := U1);
                                                      (* end of InitializeConditions for SubSystem: '<S1>/S1'  *)
                                                      (* InitializeConditions for atomic SubSystem: '<S1>/S2'  *)
                                                      _i1_ReusableSubsystem_S1(ssMethodType := SS_INITIALIZE, rtu_U := U2);
                                                      (* end of InitializeConditions for SubSystem: '<S1>/S2'  *)
                                                  SS_OUTPUT:
                                                      (* Outputs for atomic SubSystem: '<S1>/S1'  *)
                                                      _i0_ReusableSubsystem_S1(ssMethodType := SS_OUTPUT, rtu_U := U1);
                                                      rtb_Gain_f := _i0_ReusableSubsystem_S1.rty_Y;
                                                      (* end of Outputs for SubSystem: '<S1>/S1'  *)
                                                      (* Outport: '<Root>/Y1'  *)
                                                      Y1 := rtb_Gain_f;

                                                      (* Outputs for atomic SubSystem: '<S1>/S2'  *)
                                                      _i1_ReusableSubsystem_S1(ssMethodType := SS_OUTPUT, rtu_U := U2);
                                                      rtb_Gain := _i1_ReusableSubsystem_S1.rty_Y;
                                                      (* end of Outputs for SubSystem: '<S1>/S2'  *)
                                                      (* Outport: '<Root>/Y2'  *)
                                                      Y2 := rtb_Gain;

VAR                                           END_CASE;
    _i0_ReusableSubsystem_S1: ReusableSubsystem_S1;   END_FUNCTION_BLOCK
    _i1_ReusableSubsystem_S1: ReusableSubsystem_S1;   FUNCTION_BLOCK ReusableSubsystem_S1
END_VAR                                       VAR_INPUT
                                                  ssMethodType: SINT;
                                                  rtu_U: LREAL;
                                              END_VAR
                                              VAR_OUTPUT
                                                  rty_Y: LREAL;
                                              END_VAR
```
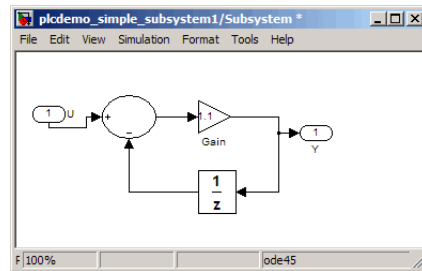
Instance variables

Instance invocations (call sites)

Reused code in
FUNCTION_BLOCK

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

5 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 1-21.
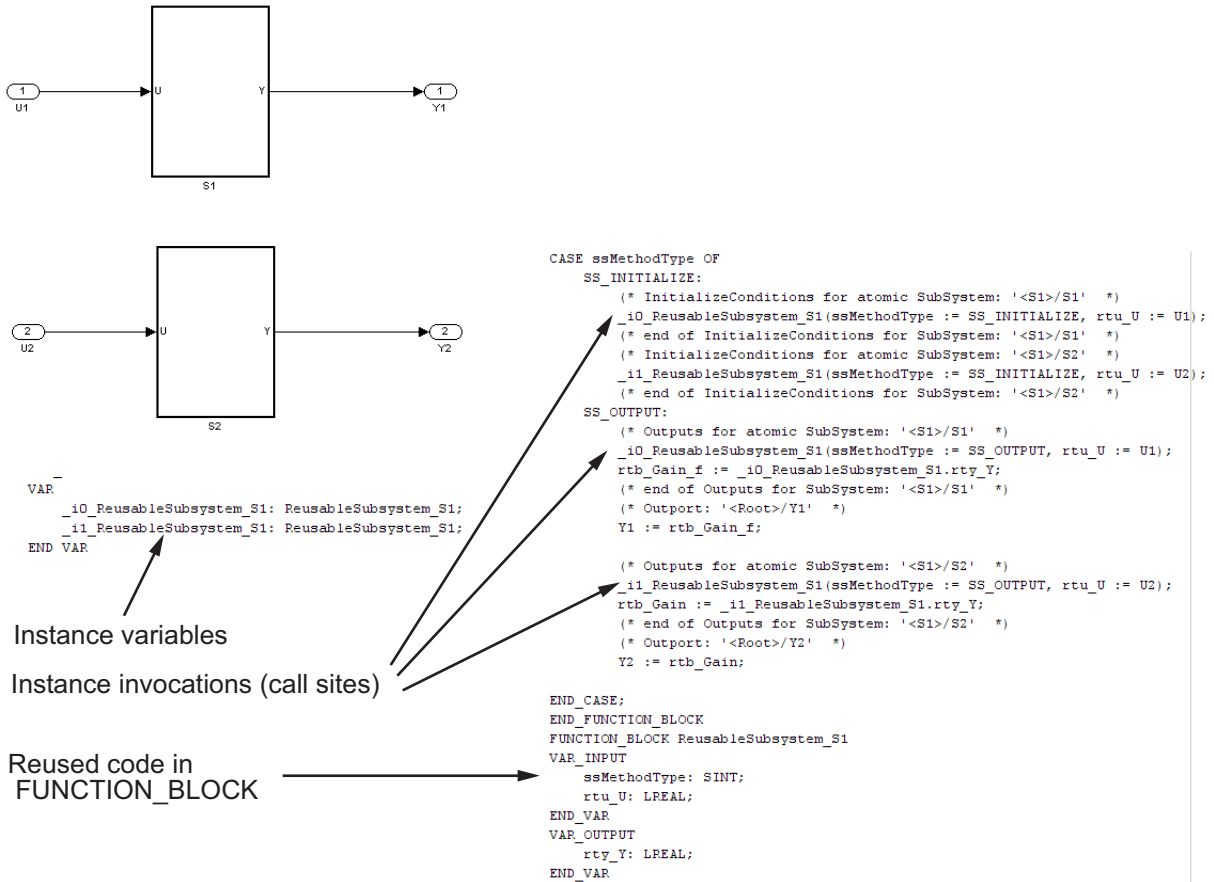
The example in this topic shows generated code for the CoDeSys Version 2.3 PLC IDE. Generated code for other IDE platforms looks different.

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 now in the MATLAB editor.

This figure illustrates the mapping of the generated code for a triggered Simulink subsystem to structured text components. The first part of the figure shows the Controller subsystem and the triggered Stateflow chart 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.
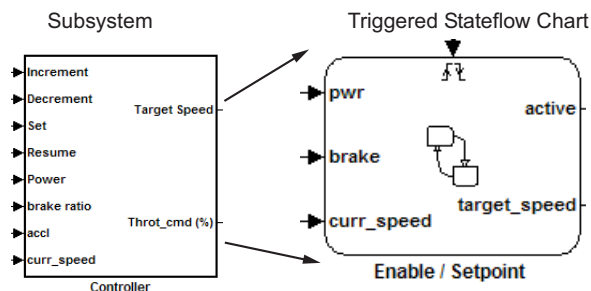
Generated code

```
    EnableSetpoint_Trig_ZCE: ARRAY [0..6] OF USINT := 3,3,3,3,3,3,3;
    _iOZZANY_ZERO_CROSSING: _ANY_ZERO_CROSSING;
END_VAR
...
...
...
    SS_OUTPUT:
        (* DiscretePulseGenerator: '<S1>/Pulse Generator'  *)
        rtb_PulseGenerator := SEL((clockTickCounter < 1) AND (clockTickCounter >= 0), 0, 1)
        IF clockTickCounter >= 1 THEN
            clockTickCounter := 0;
        ELSE
            clockTickCounter := clockTickCounter + 1;
        END_IF;


        _iOZZANY_ZERO_CROSSING(inState := USINT_TO_DINT(EnableSetpoint_Trig_ZCE[0]), inInpu
        lEventVar := _iOZZANY_ZERO_CROSSING.outEvent;
        EnableSetpoint_Trig_ZCE[0] := DINT_TO_USINT(_iOZZANY_ZERO_CROSSING.outState);
        zcEvent[0] := lEventVar;
...
...
...
FUNCTION_BLOCK _ANY_ZERO_CROSSING
VAR_INPUT
    inState: DINT;
    inInput: LREAL;
END_VAR
VAR_OUTPUT
    outEvent: DINT;
    outState: DINT;
END_VAR
VAR
END_VAR
VAR_TEMP
END_VAR
outState := SEL(inInput > 0, SEL(inInput < 0, ZERO_ZCSIG, NEG_ZCSIG), POS_ZCSIG);
outEvent := SEL((((inState = NEG_ZCSIG) AND ((outState = ZERO_ZCSIG) OR (outState = POS_ZCS
outState := SEL((inState = NEG_ZCSIG) AND (outState = ZERO_ZCSIG), SEL((inState = POS_ZCSIG
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 1-21.

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 now in the MATLAB editor.

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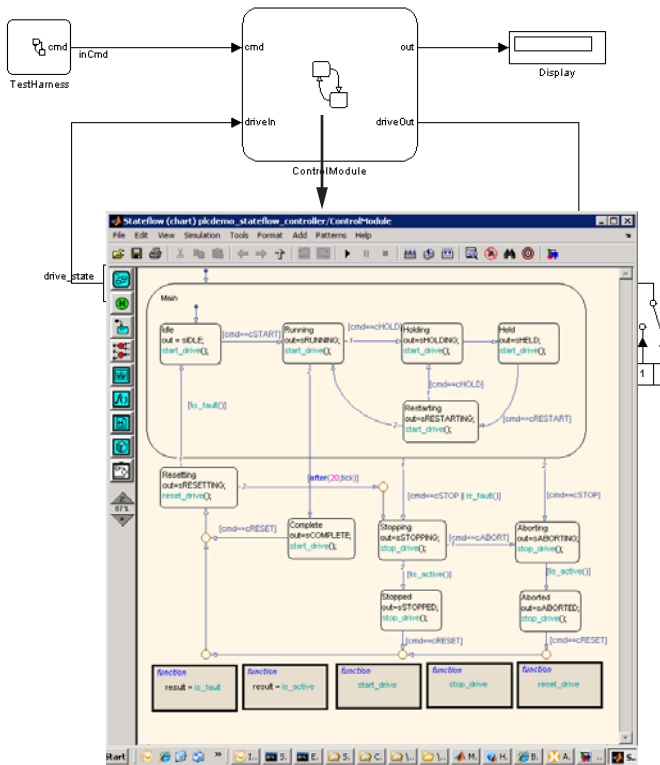
Inlined code for Stateflow chart

```
CASE is_c2_ControlModule OF
    ControlModule_IN_Aborted:
        IF inCmd = cRESET THEN
            is_c2_ControlModule := ControlModule_IN_Resetting;
            temporalCounter_i1 := 0;
            b_out := sRESETTING;
            b_driveOut.Start := FALSE;
            b_driveOut.Stop := FALSE;
            b_driveOut.Reset := TRUE;
        END_IF;
    ControlModule_IN_Aborting:
        IF  NOT drive_state.Active THEN
            is_c2_ControlModule := ControlModule_IN_Aborted;
            b_out := sABORTED;
            b_driveOut.Start := FALSE;
            b_driveOut.Stop := TRUE;
            b_driveOut.Reset := FALSE;
        END_IF;
    ControlModule_IN_Complete:
        IF inCmd = cRESET THEN
            is_c2_ControlModule := ControlModule_IN_Resetting;
            temporalCounter_i1 := 0;
            b_out := sRESETTING;
            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 1-21.

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 now in the MATLAB editor.

This figure illustrates the mapping of the generated code for a Simulink Subsystem block that contains a MATLAB Function block to structured text components. 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`.
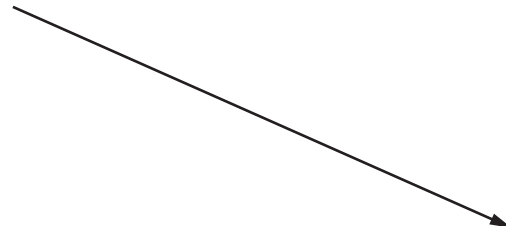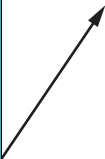
4 Examine the generated structured text code.

MATLAB code

Generated code
for MATLAB
subfunctions

```
FUNCTION_BLOCK TankControl
VAR_INPUT
    ssMethodType: SINT;
    Command: PLCCommandState;
    Height: LREAL;
END_VAR
VAR_OUTPUT
    InFlow: LREAL;
    OutFlow: LREAL;
    StirSpeed: LREAL;
END_VAR
VAR
END_VAR
VAR_TEMP
    eml_vessel: PLCVesselState;
    eml_EmptyValve: PLCValveState;
    eml_FillValve: PLCValveState;
END_VAR
CASE ssMethodType OF
    SS_OUTPUT:

        (* Check the vessel state *)
        IF Height >= 10 THEN
            (* Is it full ? *)
            eml_vessel := FULL;
        ELSIF Height <= 2 THEN
            (* Is it empty ?  *)
            eml_vessel := EMPTIED;
        ELSE
            eml_vessel := NOT_FULL;
        END_IF;
        (* Process the command mode *)
        CASE Command OF
            FILL:
                (* Fill Tank *)
                eml_EmptyValve := SHUT;
                IF eml_vessel = FULL THEN
                    eml_FillValve := SHUT;
                ELSE
                    eml_FillValve := OPEN;
                END_IF;
            HOLD:
                (* Hold Contents *)
                eml_EmptyValve := SHUT;
                eml_FillValve := SHUT;
            EMPTY:
                (* Empty Tank *)
                eml_FillValve := SHUT;
                IF eml_vessel = EMPTIED THEN
                    eml_EmptyValve := SHUT;
                ELSE
                    eml_EmptyValve := OPEN;
                END_IF;
            ELSE
                eml_EmptyValve := SHUT;
                eml_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.

Generating Test Bench Code

Working with Generated Structured Text

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 "Generating and Manually Importing Test Bench Code" on page 3-2

- Automatically, including running the test bench, as described in "Automatically Importing Structured Text Code" on page 3-6

Generating and Manually Importing Test Bench Code

This topic describes the basic procedure for how to generate test bench code. It uses the CoDeSys V2.3 IDE as an example target IDE.

This topic 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 1-12. 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 Ensure 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:
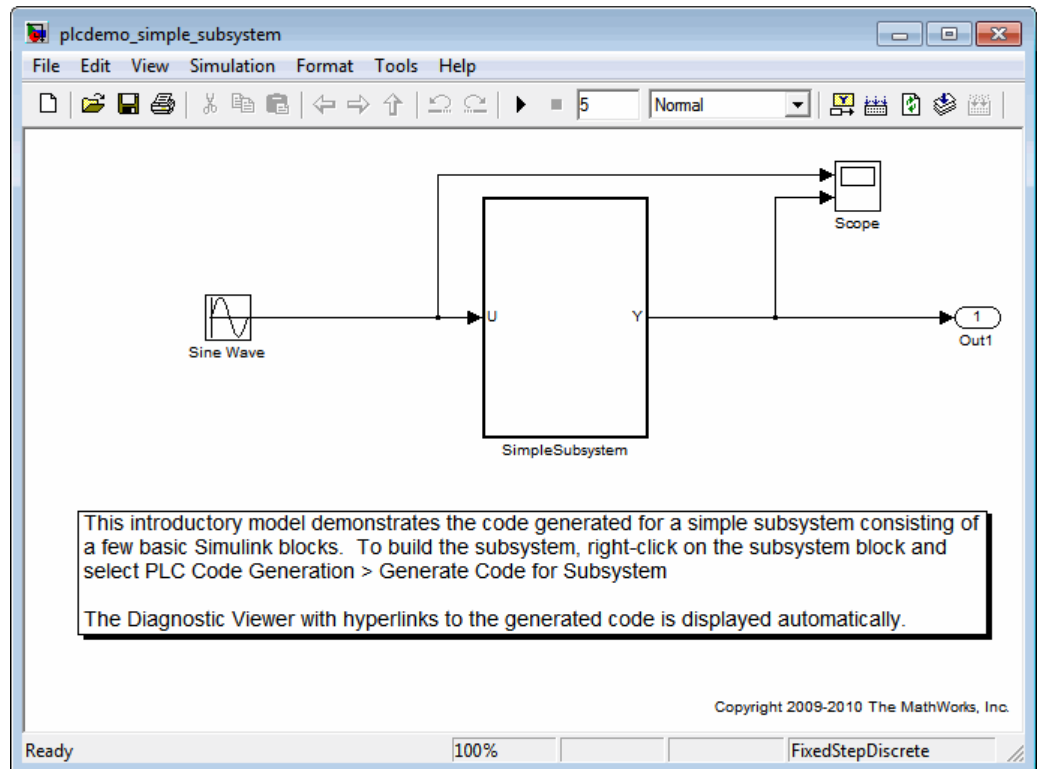
3 Right-click the Subsystem block and select **PLC Code Generation > Options**.

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.
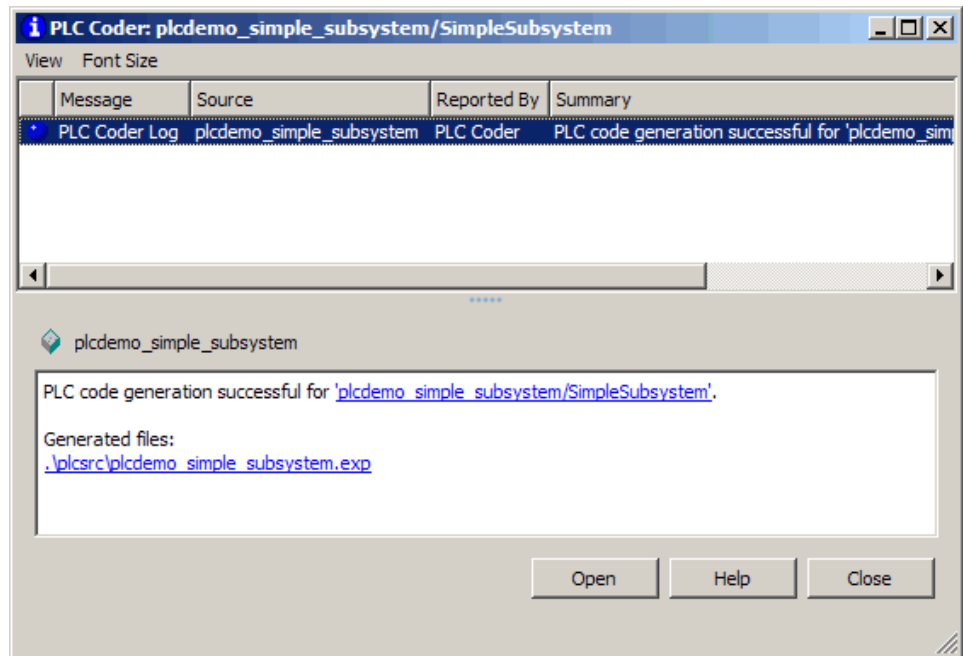
8 Click **OK**.

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

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 1-21 for list of the target IDE platforms and the possible generated files.

Integrating Generated Code into Custom Code

The ssMethodType is a special input argument that the coder adds to each subsystem FUNCTION_BLOCK section during code generation. ssMethodType enables you to selectively execute code for Simulink Subsystem block methods such as initialization and computation output. 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:

1 For the first time only, call FUNCTION_BLOCK with ssMethodType = SS_INITIALIZE (with other inputs). This argument performs initialization.

2 Multiple times as needed, call FUNCTION_BLOCK with ssMethodType = SS_OUTPUT together with ssMethodType = SS_UPDATE (with other inputs). This argument sequence performs computation output.

Automatically Importing Structured Text Code

In this section...

"PLC IDEs that Qualify for Importing Code Automatically" on page 3-6

"Automatically Importing to KW-Software MULTIPROG 5.0 and Phoenix Contact PC WORX 6.0 IDEs" on page 3-7

"Generating, Automatically Importing, and Verifying Structured Text" on page 3-8

PLC IDEs that Qualify for Importing Code Automatically

This topic assumes that you have read "Automatically Importing Structured Text Code" on page 1-31. 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

- 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 1-32.

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 Ensure 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 1-31 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 9-6 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 3-7.

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.

Working with Tunable Parameters in the Simulink PLC Coder Environment

Configuring Tunable Parameters for Your Model

In this section...

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 the simulation is running. The Simulink PLC Coder software allows you to tune parameters:

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

Workflow Overview

This topic describes how to configure your model to enable tunable parameters. By default, Simulink PLC Coder parameters are inlined and not tunable.

The general workflow for configuring a model to enable tunable parameters is:

1 Identify the model parameters you want to be tunable.

2 Define these parameters in the MATLAB workspace.

3 Configure tunable parameters in the **Configuration Parameters** > **Optimization** > **Signals and Parameters** > **Model Parameter Configuration** dialog box.

Identifying Tunable Parameters

This topic creates the model my_plcdemo_tunable_params to show how to configure tunable parameters. This model is the same as the plcdemo_tunable_params model. The difference is that the demo model already has the tunable parameters configured, while this topic guides you in configuring the tunable parameters.

**1** In the MATLAB Command Window, create a model to look like the following:
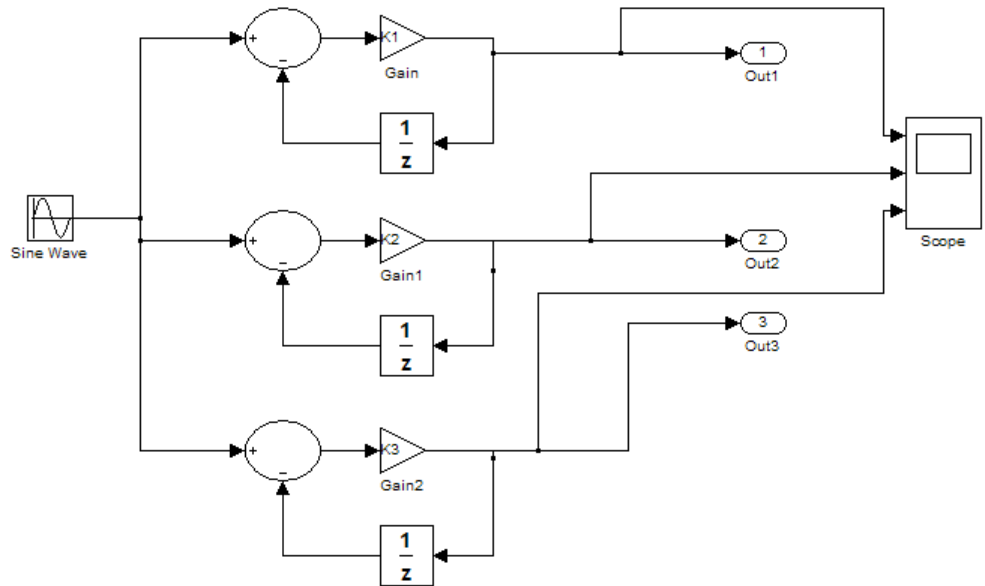
2 Select the Sum, Gain, and Unit Delay blocks and create an atomic subsystem with inputs U, U1, and U2 and outputs Y, Y1, and Y2. Rename the Subsystem block as SimpleSubsystem. When you are finished, the top model and atomic subsystem model should look like the following:
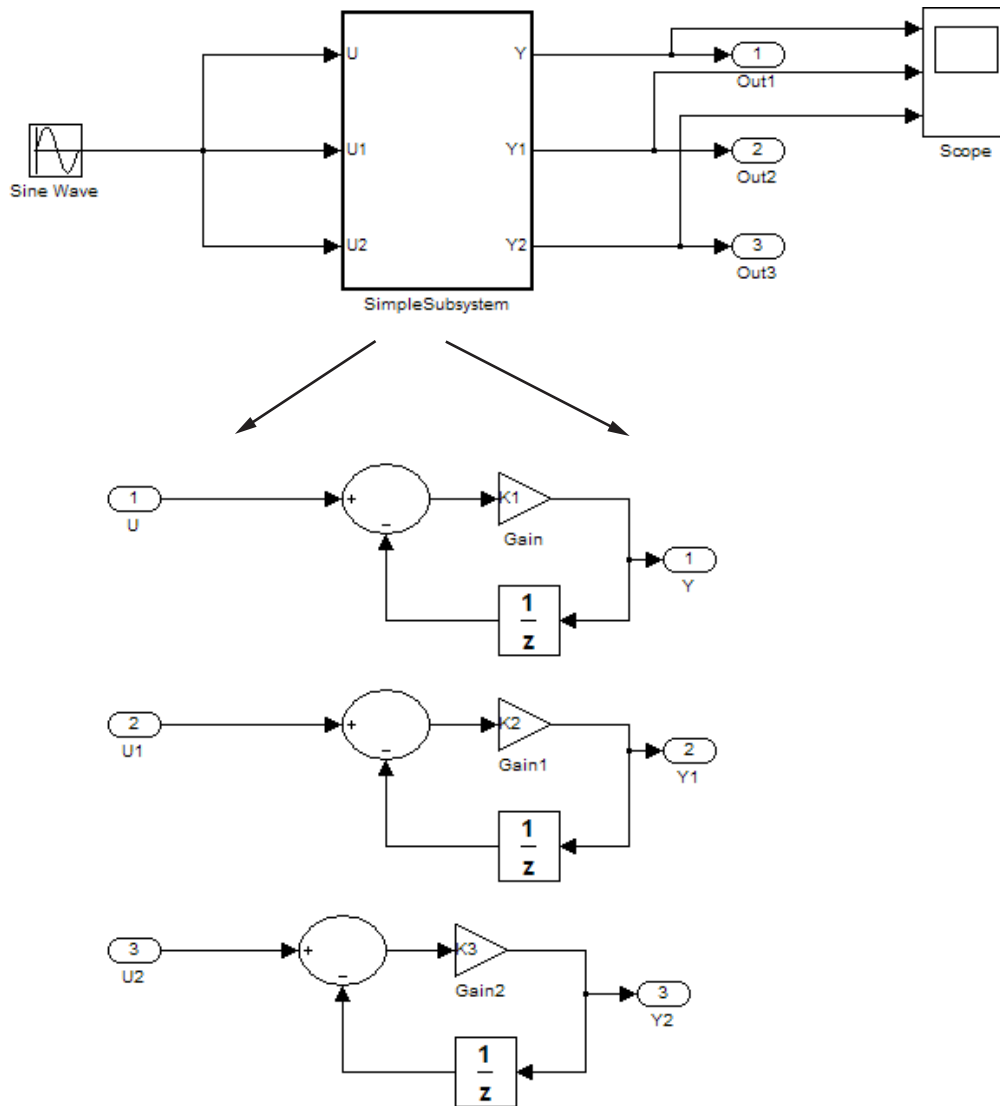
3 Save this subsystem as my_plcdemo_tunable_params.mdl.

4 Double-click SimpleSubsystem.
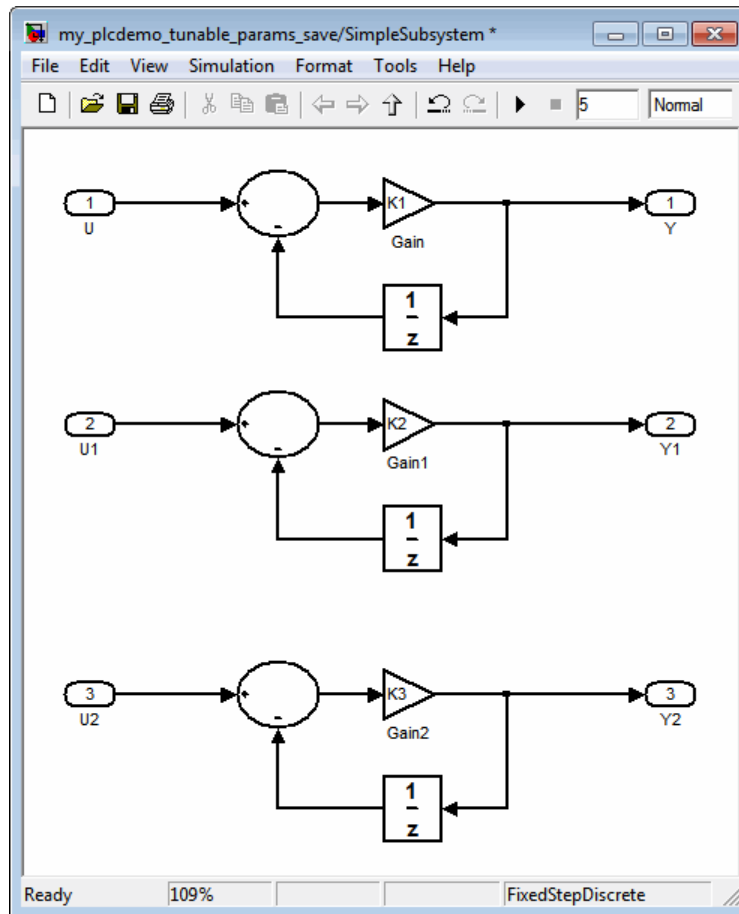
5 Note the three Gain blocks that have the constants that you want to make tunable: *K1*, *K2*, and *K3*.

Next, define these parameters in the MATLAB workspace. See "Defining Tunable Parameters in the MATLAB Workspace" on page 4-6.

Defining Tunable Parameters in the MATLAB Workspace

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

It assumes that you 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 4-3.
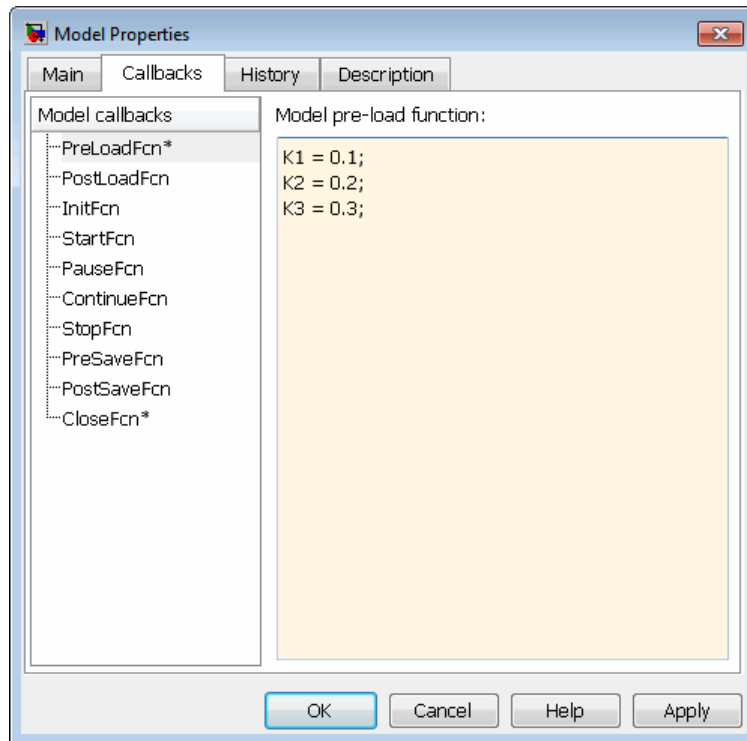
1 In the my_plcdemo_tunable_params model, select **File > Model Properties**.

   The Model Properties dialog box is displayed.
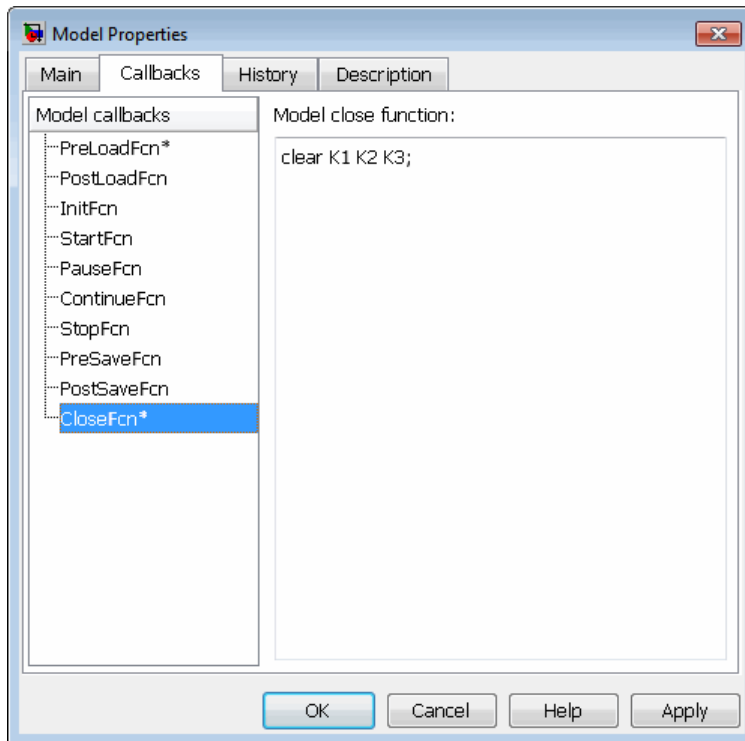
2 In the **Callbacks** pane, select PreLoadFcn.

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;
```

This command ensures that you clear these constants from the MATLAB workspace when you close the model.

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

Your next task is to configure these parameters to be tunable. See "Configuring Parameters to Be Tunable" on page 4-8.

Configuring Parameters to Be Tunable

This topic describes how to configure parameters to be tunable using the Simulink Configuration Parameters dialog box.
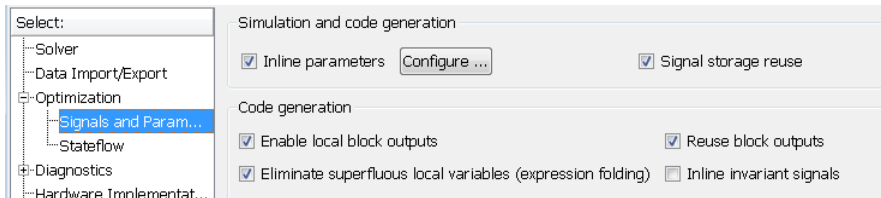
It assumes that you created the my_plcdemo_tunable_params model or opened plcdemo_tunable_params and defined the parameters for tuning. If

you have not yet done so, see "Defining Tunable Parameters in the MATLAB Workspace" on page 4-6.

This topic assumes that you are familiar with the tunable parameter properties on the **Global (tunable) parameters** pane. See Setting Tunable Parameter Properties in the Simulink® Coder™ documentation for further details.

This topic 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.)

Select:	Simulation and code generation	
Solver	☑ Inline parameters Configure ...	☑ Signal storage reuse
Data Import/Export		
Optimization	Code generation	
Signals and Param...	☑ Enable local block outputs	☑ Reuse block outputs
Stateflow	☑ Eliminate superfluous local variables (expression folding)	☐ Inline invariant signals
Diagnostics		
Hardware Implementat...		

The **Configure** button is enabled.

4 Click **Configure**.

The Model Parameter Configuration dialog box is displayed.
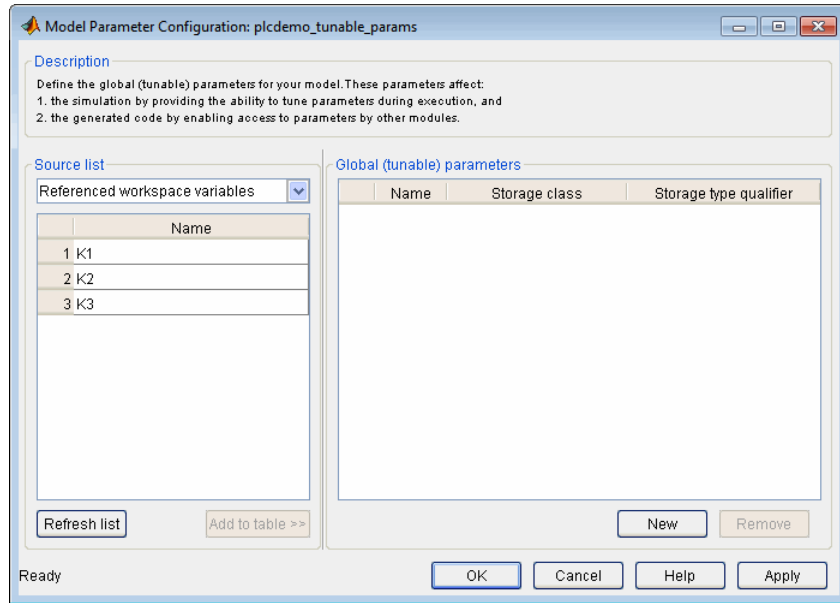
5 From **Source list**, select `Referenced workspace variables`.

6 Use the **Ctrl** key to select all the parameters and click **Add to table >>** to add them to the **Global (tunable) parameters** table.
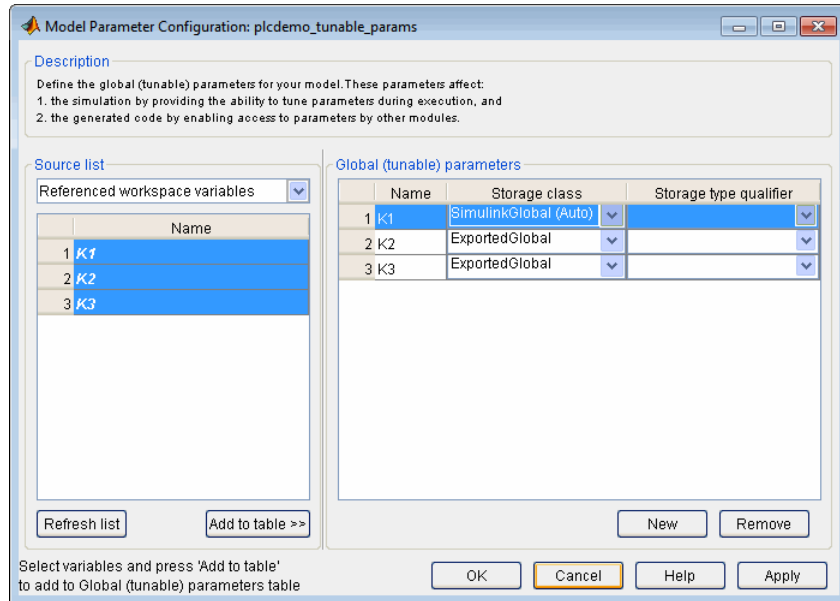
By default, this dialog box sets all parameters to the `SimulinkGlobal (Auto)` storage class. This setting generates code with the tunable parameters set at the local level. In this case, these parameters appear at the function block level in each function block that uses the parameter.

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 Ensure that **Target IDE** and **Output Directory** settings are appropriate, 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, such as the Rockwell Automation RSLogix 5000 IDE, 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 the setting of the parameter storage class to `ImportedExternPointer`. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.

Tunable Parameters Considerations

When tuning parameters, the coder does not support:

- Specifying parameters using `Simulink.Parameter` object. Use global tunable parameters instead.

- Tuning parameters of bus data type.

4-13

Controlling Generated Code Partitions

Function Block Partitions

About Function Block Partitions

The Simulink PLC Coder software converts subsystems to function block units, one subsystem per function block. You control generated code partitioning by the number and types of Subsystem blocks that you have in your model. The coder generates structured text function blocks as follows:

- Generates one function block for an atomic subsystem that contains no other subsystems.

- Generates one function block for an atomic subsystem that contains only virtual subsystems. For virtual subsystems, the Simulink PLC Coder software generates code that is indistinguishable from the rest of the contents of the atomic subsystem. It generates code that is flattened.

- Generates a function block for each nonvirtual subsystem contained in an atomic subsystem. Nonvirtual subsystems can be atomic, fcn-call, or enabled. You can customize this partitioning with the **Function packaging** parameter of the Subsystem block.

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.

```
15    FUNCTION_BLOCK SimpleSubsystem
16    VAR_INPUT
17        ssMethodType: SINT;
18        U: LREAL;
19    END_VAR
20    VAR_OUTPUT
21        Y: LREAL;
22    END_VAR
23    VAR
24        UnitDelay_DSTATE: LREAL;
25    END_VAR
26    VAR_TEMP
27        rtb_Gain: LREAL;
28    END_VAR
29    CASE ssMethodType OF
30        SS_INITIALIZE:
31            (* InitializeConditions for UnitDelay: '<S1>/Unit Delay'  *)
32            UnitDelay_DSTATE := 0;
33
34        SS_OUTPUT:
35            (* Gain: '<S1>/Gain' incorporates:
36             *   Inport: '<Root>/U'
37             *   Sum: '<S1>/Sum'
38             *   UnitDelay: '<S1>/Unit Delay'
39             *)
40            rtb_Gain := (U - UnitDelay_DSTATE) * 0.5;
41
42            (* Outport: '<Root>/Y'  *)
43            Y := rtb_Gain;
44
45            (* Update for UnitDelay: '<S1>/Unit Delay'  *)
46            UnitDelay_DSTATE := rtb_Gain;
47
48    END_CASE;
49    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.

```
 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
15   VAR_OUTPUT
16       Out1: LREAL;
17       Out2: LREAL;
18   END_VAR
19   VAR
20       UnitDelay_DSTATE: LREAL;
21       UnitDelay1_DSTATE: LREAL;
22       UnitDelay_DSTATE_f: LREAL;
23       UnitDelay_DSTATE_a: LREAL;
24   END_VAR
25   VAR_TEMP
26       rtb_Gain: LREAL;
27       rtb_Gain_m: LREAL;
28   END_VAR
29   CASE ssMethodType OF
30       SS_INITIALIZE:
31           /* 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.
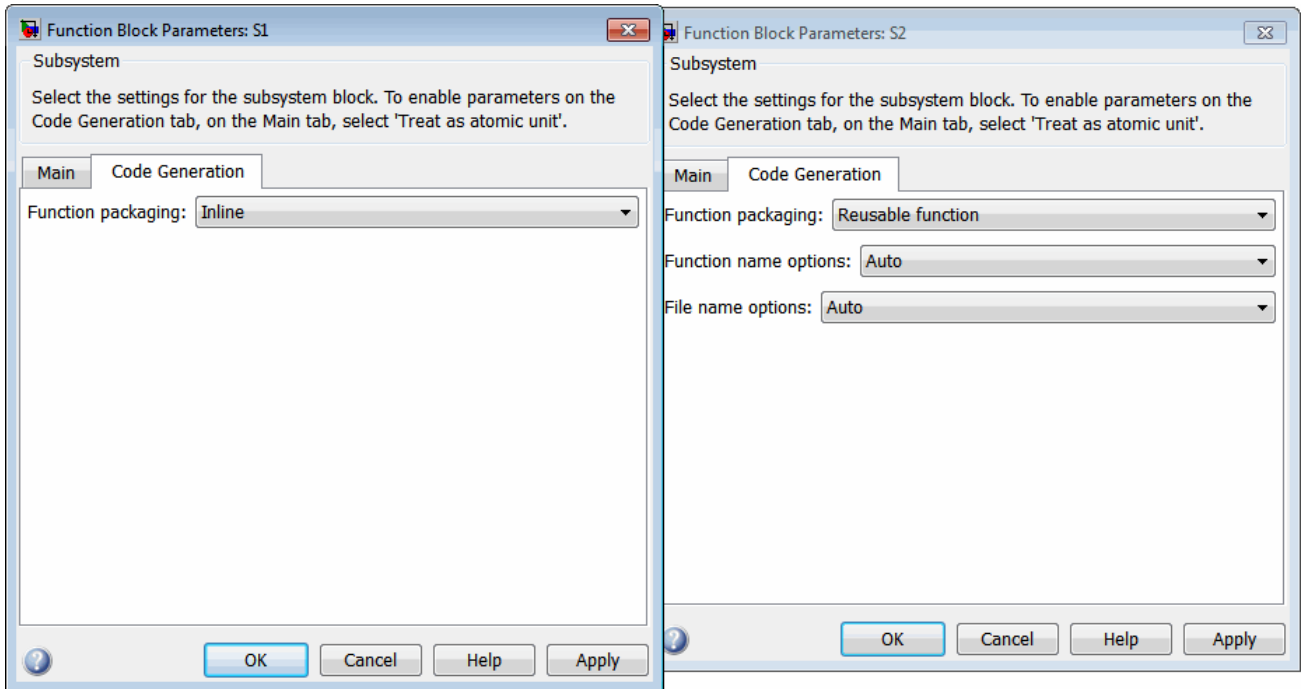
Changing the Name of a Subsystem

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.
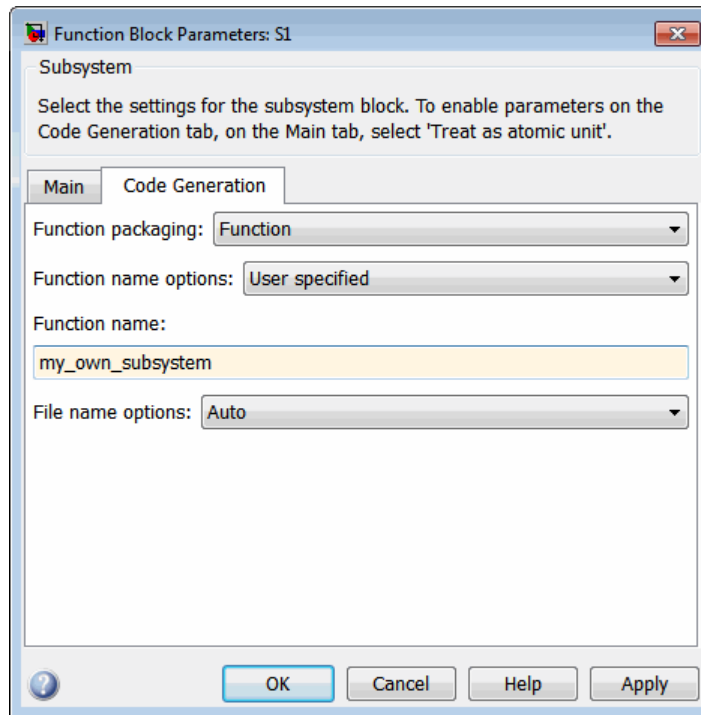
6 Save the new settings.

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

IDE-Specific Considerations

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 this target, 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 7-2 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 to ensure 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 to ensure 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 7-2 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.

Limitations

- "Coder Limitations" on page 7-2
- "Block Restrictions" on page 7-6

Coder Limitations

In this section...
"Current Limitations" on page 7-2
"Fixed-Point Data Type Limitations" on page 7-2
"Permanent Limitations" on page 7-5

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

- 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, atan2, rand.

- Merge block

- Multi-rate models

- Signal and state storage classes

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 properly configure the blocks and models, 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 7-4).

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.

- The solver type for the Simulink model must be fixed-step and discrete.

- 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

The Simulink PLC Coder software supports the `plclib` blocks with the following exceptions. Also, see Chapter 7, "Limitations" for a list of limitations of the software.

If you get unsupported fixed-point type error messages during code generation, update the block parameter. Open the block parameter dialog box. Navigate to the **Signal Attributes** and **Parameter Attributes** tabs. Ensure 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 an appropriate non-fixed-point data type, such as `double` or `int8`.

Stateflow Chart Exceptions

If you receive an error like the following during code generation and the model contains a Stateflow chart that contains one or more Simulink functions, use the following procedure to address the issue:

```
The 'State when enabling' parameter of the Trigger Port
block inside the Function-Call subsystem
'msimpleslFcn/Subsystem/Chart/slFcn' is set to 'inherit'.
Simulink cannot ensure consistency between the original
subsystem 'msimpleslFcn/Subsystem' and the S-function
generated from subsystem build. The 'State when enabling'
parameter must be set to 'reset' or 'held'.
```

1 Open the model and double-click the Stateflow chart that is causing 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.

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 ensures that *preference_value* persists 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
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'
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.

```
plccoderpref('plctargetidepaths')

ans =

        codesys23: 'C:\Program Files\3S Software'
        codesys33: 'C:\Program Files\3S CoDeSys'
      rslogix5000: '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'
     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.

Examples   Generate code for the subsystem,
                  plcdemo_simple_subsystem/Simple_Subsystem.

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

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/Simple_Subsystem`.

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

See Also   `plcgeneratecode`

9

Configuration Parameters for Simulink PLC Coder Models

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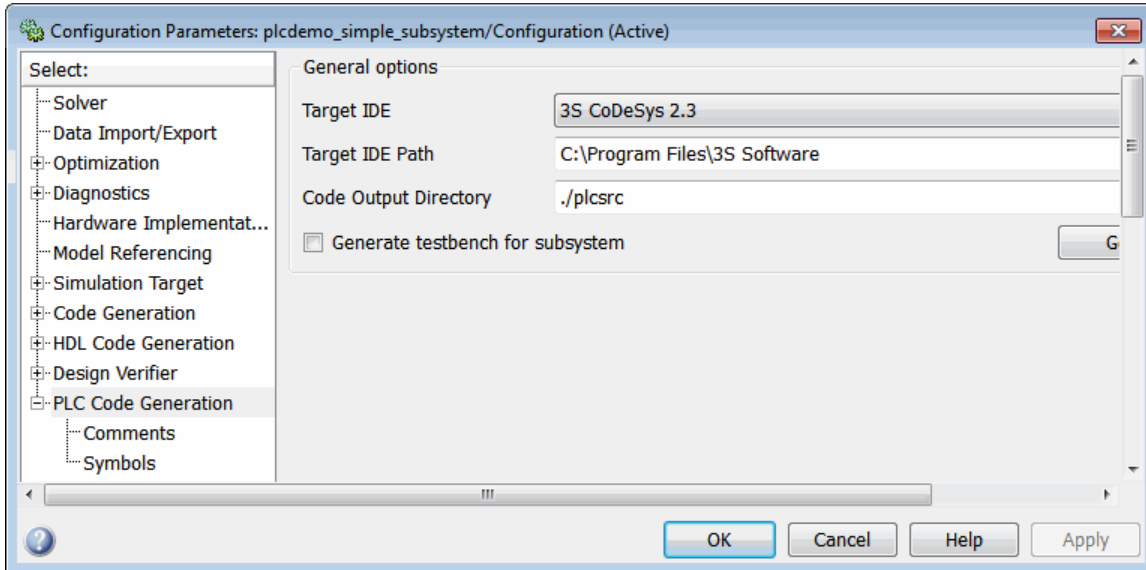
In this section...
"PLC Coder: General Tab Overview" on page 9-3
"Target IDE" on page 9-4
"Target IDE Path" on page 9-6
"Code Output directory " on page 9-8
"Generate testbench for subsystem" on page 9-9

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 1-12

"Generating Structured Text Code from the Model Window" on page 1-21

Target IDE

Select the target IDE in which to generate code.

Settings

Default: Structured Text (CoDeSys)

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
> Generates structured text code for Rockwell Automation RSLogix 5000.

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 your desired target IDE 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

- Separate the reserved names using commas or spaces.

Command-Line Information

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

See Also

"Generating Structured Text Code from the Model Window" on page 1-21

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 1-31

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 1-21

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.

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 1-21

PLC Coder: Comments

In this section...
"Comments Overview" on page 9-11
"Include comments" on page 9-11
"Simulink block / Stateflow object comments " on page 9-12
"Show eliminated blocks" on page 9-13

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 1-21

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 1-21

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 1-21

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 1-21

PLC Coder: Symbols

Auto-generated identifier naming rules

Maximum identifier length: 31

Reserved names

☐ Use the same reserved names as Simulation Target

Reserved names:

<table>
<tr><td>**In this section...**</td></tr>
<tr><td>"Symbols Overview" on page 9-15</td></tr>
<tr><td>"Maximum identifier length" on page 9-16</td></tr>
<tr><td>"Use the same reserved names as Simulation Target" on page 9-17</td></tr>
<tr><td>"Reserved names" on page 9-18</td></tr>
</table>

Symbols Overview

Select the automatically generated identifier naming rules.

See Also

"Generating Structured Text Code from the Model Window" on page 1-21

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 1-21

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 1-21

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 to prevent error messages.

- 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 1-21

Index