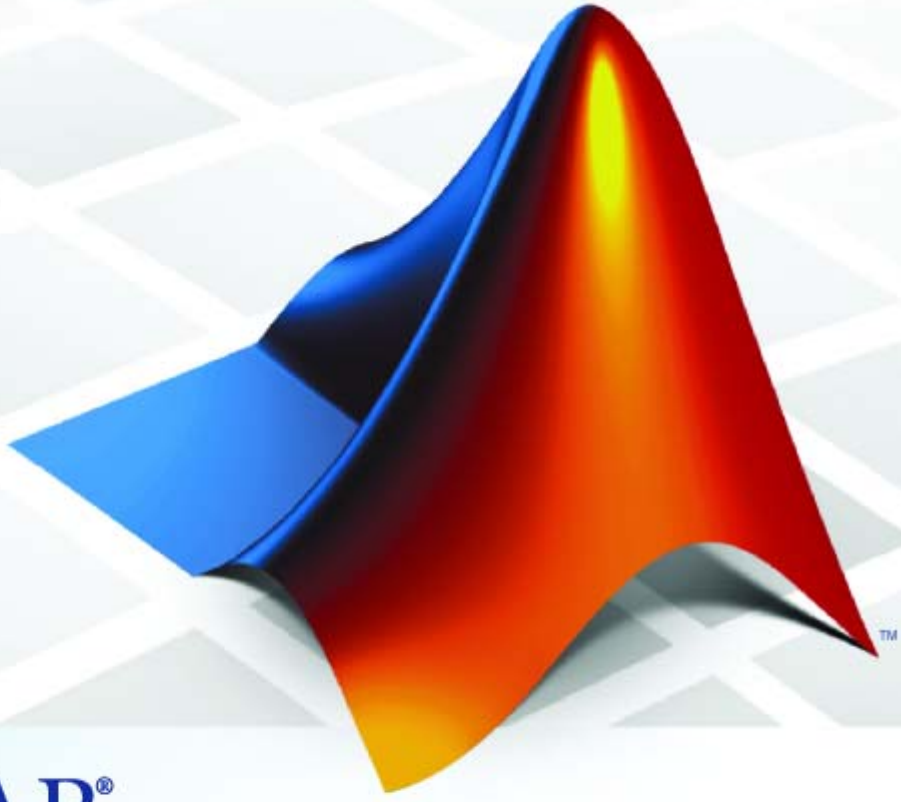
Simulink® PLC Coder™ 1
User's Guide



MATLAB® & SIMULINK®

The MathWorks™
Accelerating the pace of engineering and science

How to Contact The MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Simulink® PLC Coder™ User's Guide

Trademarks

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

Patents

The 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)

Contents

1 Getting Started

Mapping Simulink Semantics to Structured Text

2

Generating Test Bench Code

3

Working with Tunable Parameters in the Simulink® PLC Coder Environment

4

5 Controlling Generated Code Partitions

6 IDE-Specific Considerations

7 Limitations

8 Functions — Alphabetical List

9 Configuration Parameters for Simulink® PLC Coder Models

Index

1

Getting Started

Product Overview

In this section...

Introduction

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

Simulink PLC Coder generates test benches that help you verify the structured text using PLC and PAC IDEs and simulation tools.

Key features:

- Automatic generation of IEC 61131-3 structured text
- Simulink support, including reusable subsystems, PID controller blocks, and lookup tables
- Stateflow support, including graphical functions, truth tables, and state machines
- Embedded MATLAB support, including if-else statements, loop constructs, and math operations

- Support for multiple data types, including Boolean, integer, enumerated, and floating-point, as well as vectors, matrices, buses, and tunable parameters
- IDE support, including B&R Automation Studio®, PLCopen, Rockwell Automation® RSLogix™ 5000, and Smart Software Solutions CoDeSys
- Test-bench creation

PLC Code Generation in the Development Process

Simulink PLC Coder software lets you generate IEC-61131 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 that the Simulink PLC Coder software generates.
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 an Embedded MATLAB Block	Demonstrates the code generated for an Embedded MATLAB Function block implementing tank valve control logic.

Related Products

In this section...

Requirements for the Simulink PLC Coder Product

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

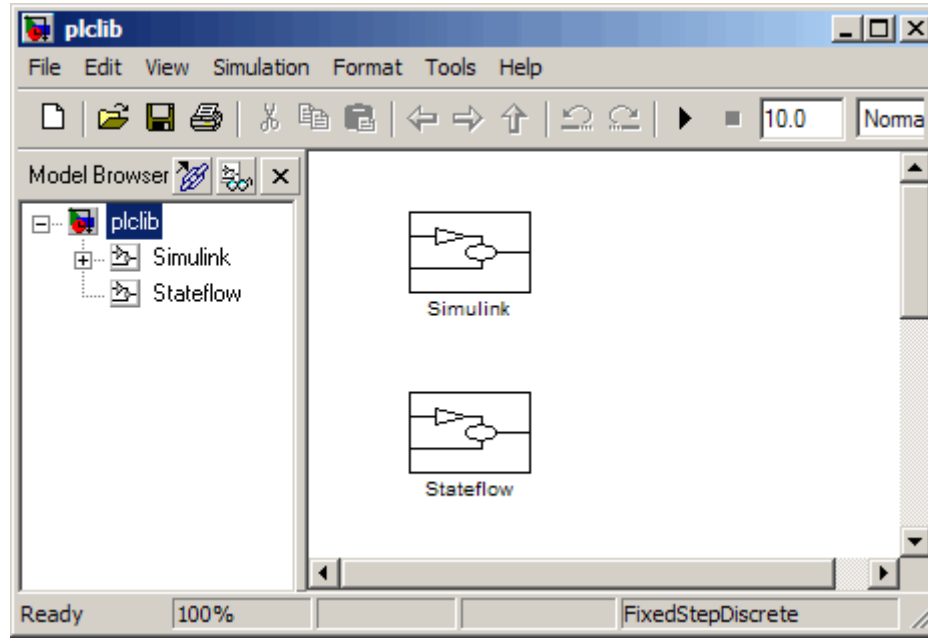
- MATLAB
- Simulink

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 7-4 for restrictions on using these blocks.

System Requirements

Requirement	Description
32-bit operating system	Windows® platform supported by The 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

- Rockwell Automation RSLogix 5000 Series Version 17
- B&R Automation Studio 3.0
- PLCOpen XML
- Beckhoff® TwinCAT® 2.11
- Generic

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.

Rockwell Automation RSLogix 5000 Software

To get the Rockwell Automation RSLogix 5000 product, see:

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

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
```

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.

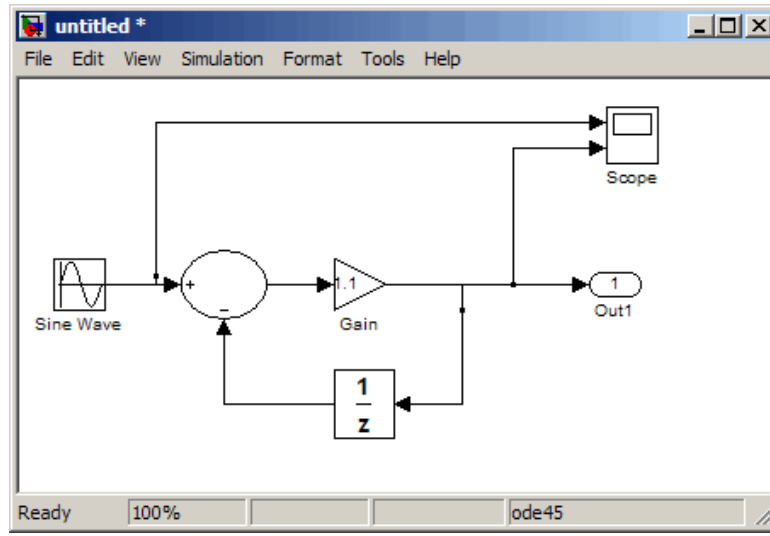
Preparing Your Model to Generate Structured Text Code

In this section...

Configuring Simulink Models for Structured Text Code Generation

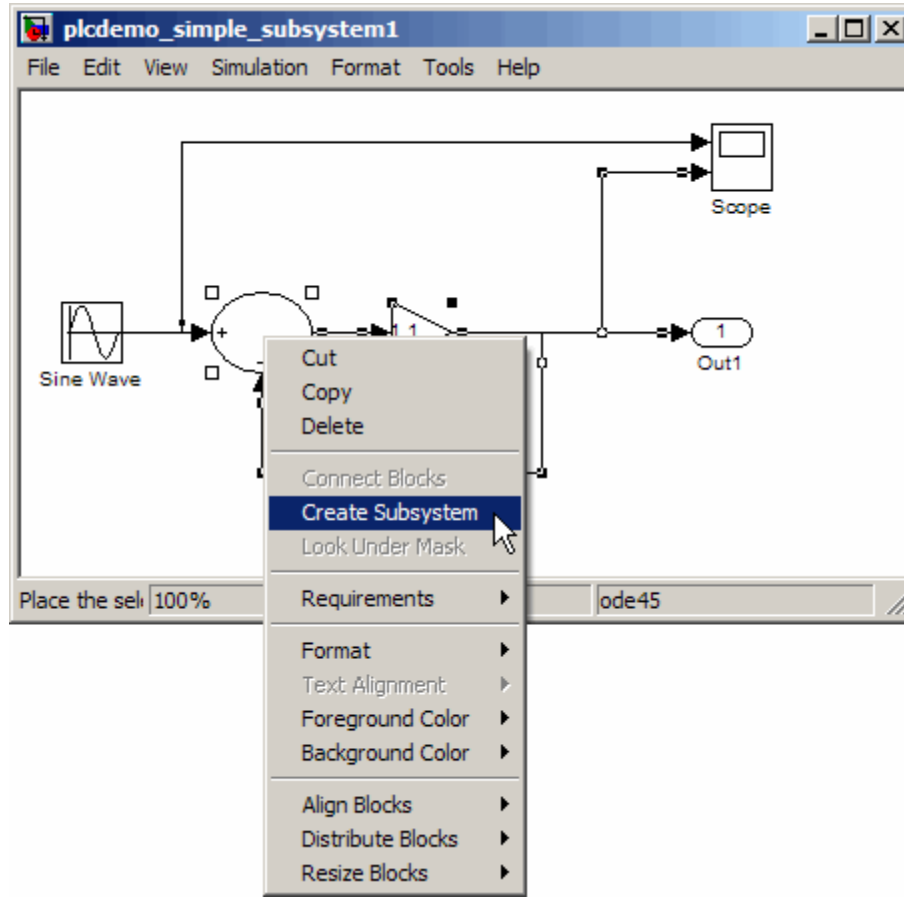
This topic assumes that you have a model for which you want to generate and import code to a PLC. Before you use this model, perform the following.

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

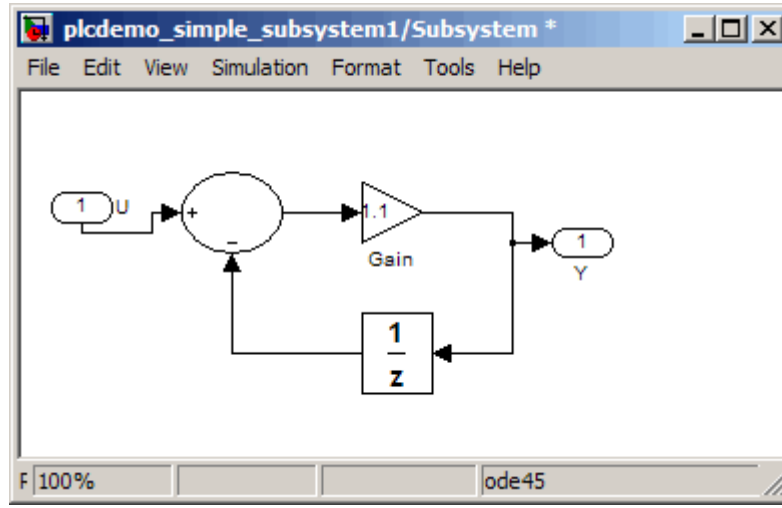


2 Save this model as `plcdemo_simple_subsystem1.mdl`.

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

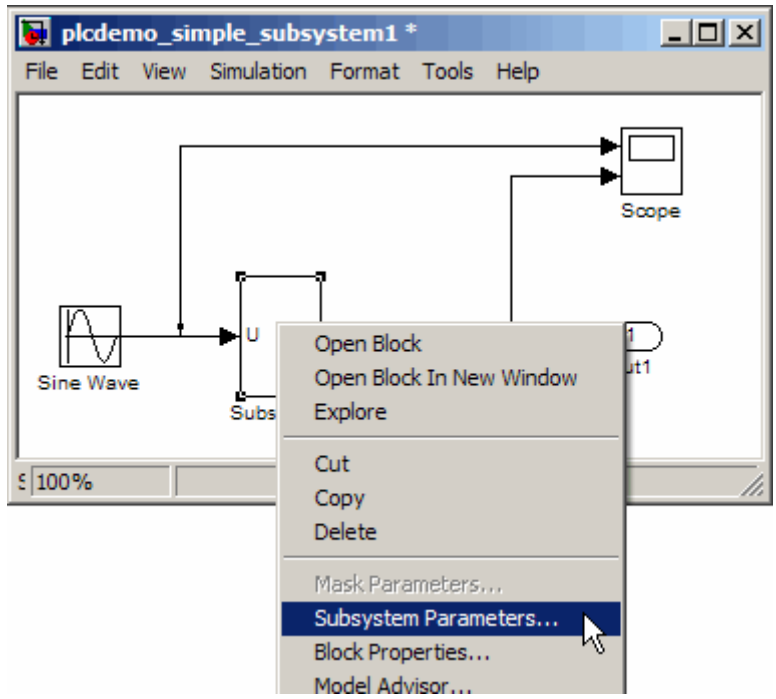


Optionally, rename In1 and Out1 to U and Y respectively. This operation results in a subsystem like the following:

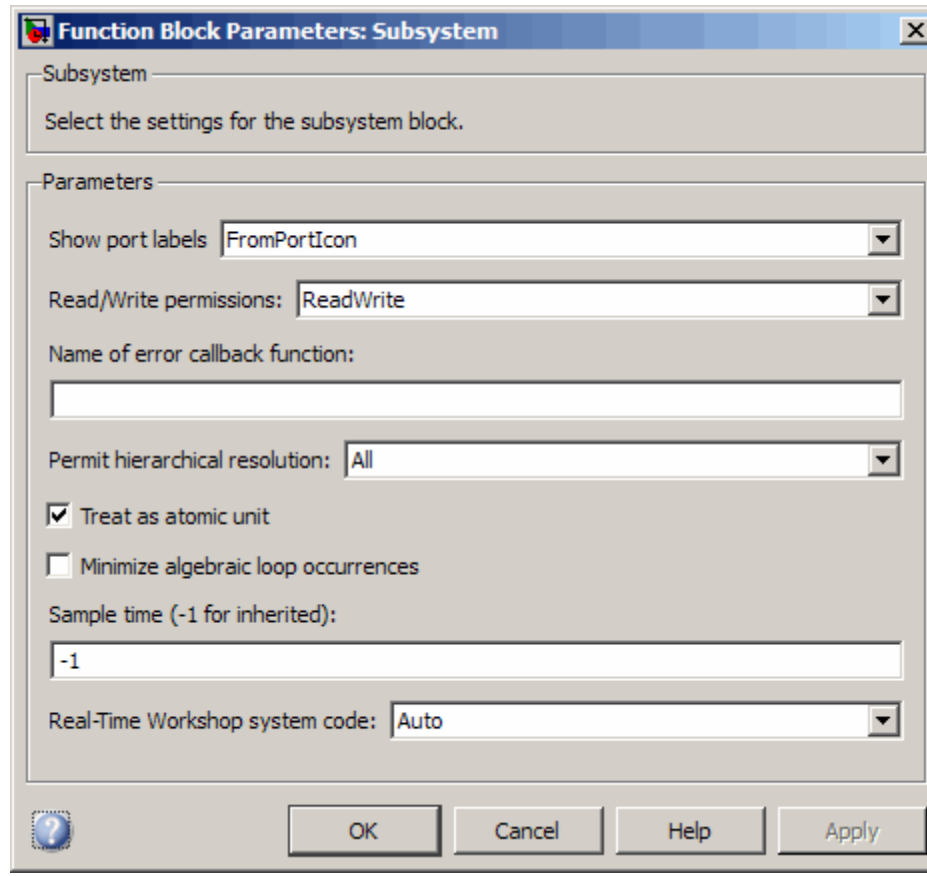


4 Save the subsystem.

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.

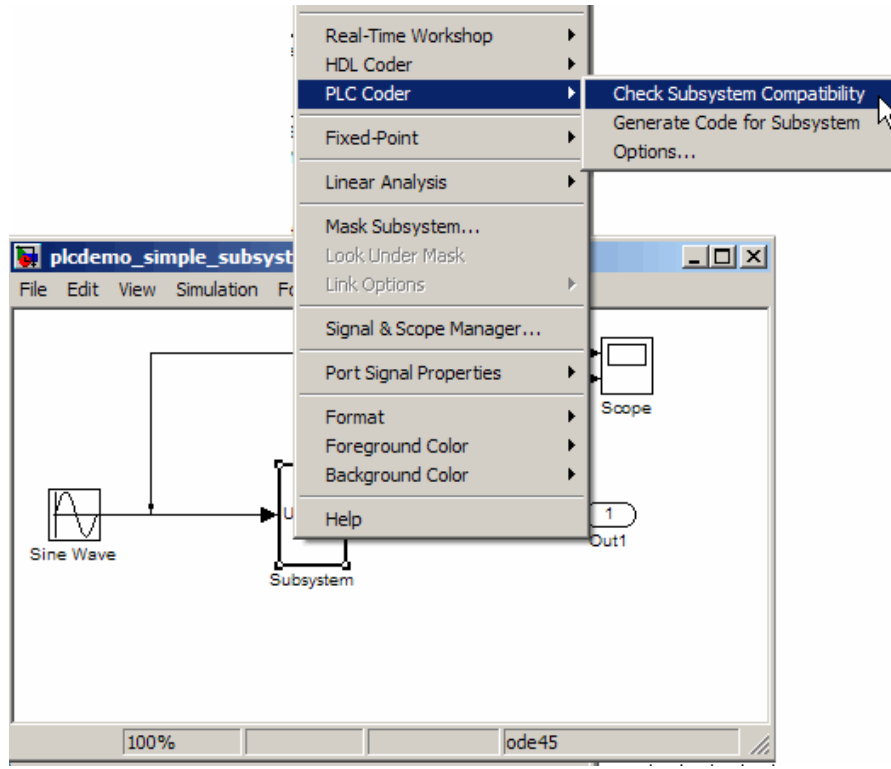
9 Save your model. In later procedures, you can use either this model, or the `plcdemo_simple_subsystem.mdl` model that comes with your software.

You are now ready to generate structured text code for your IDE. See "Generating and Examining Structured Text Code" on page 1-20.

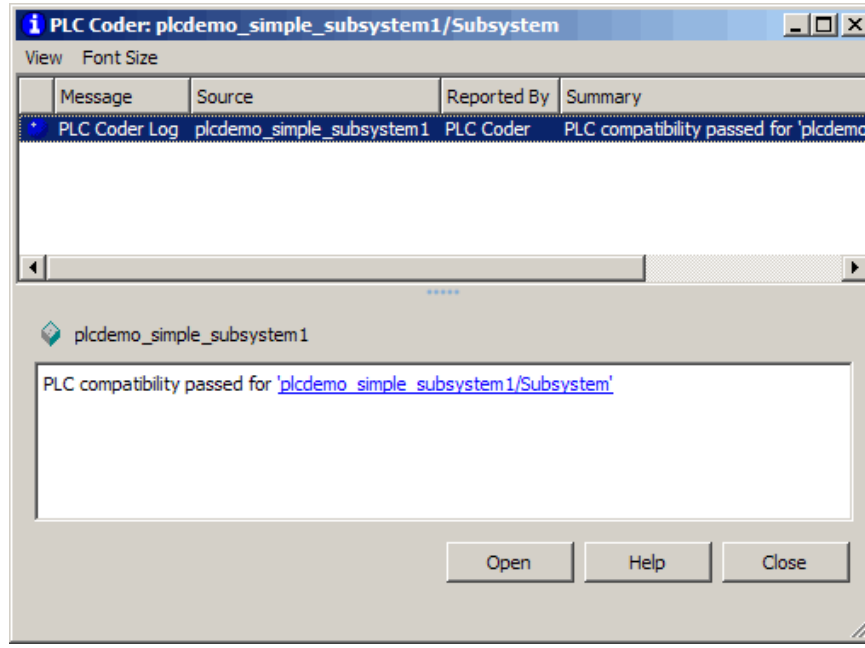
Ensuring System Compatibility for Structured Text Code Generation

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

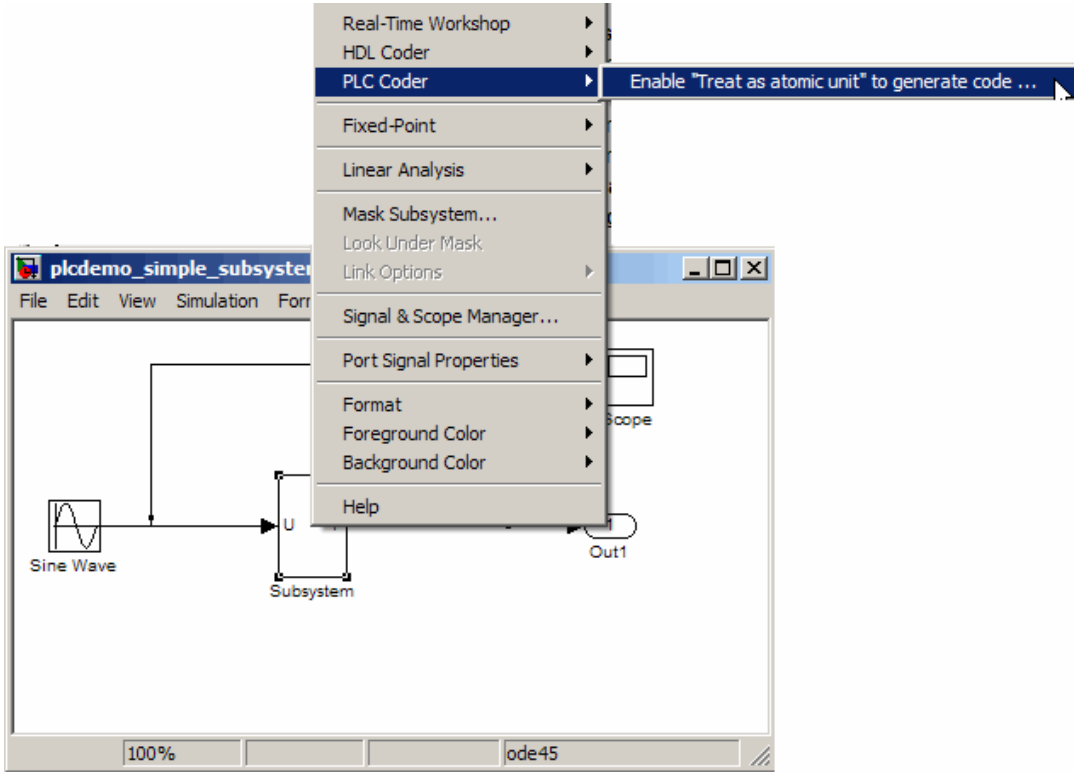
1. In your model, navigate to the subsystem for which you want to generate code.
2. Right-click that Subsystem block and select **PLC Coder > Check Subsystem Compatibility**.



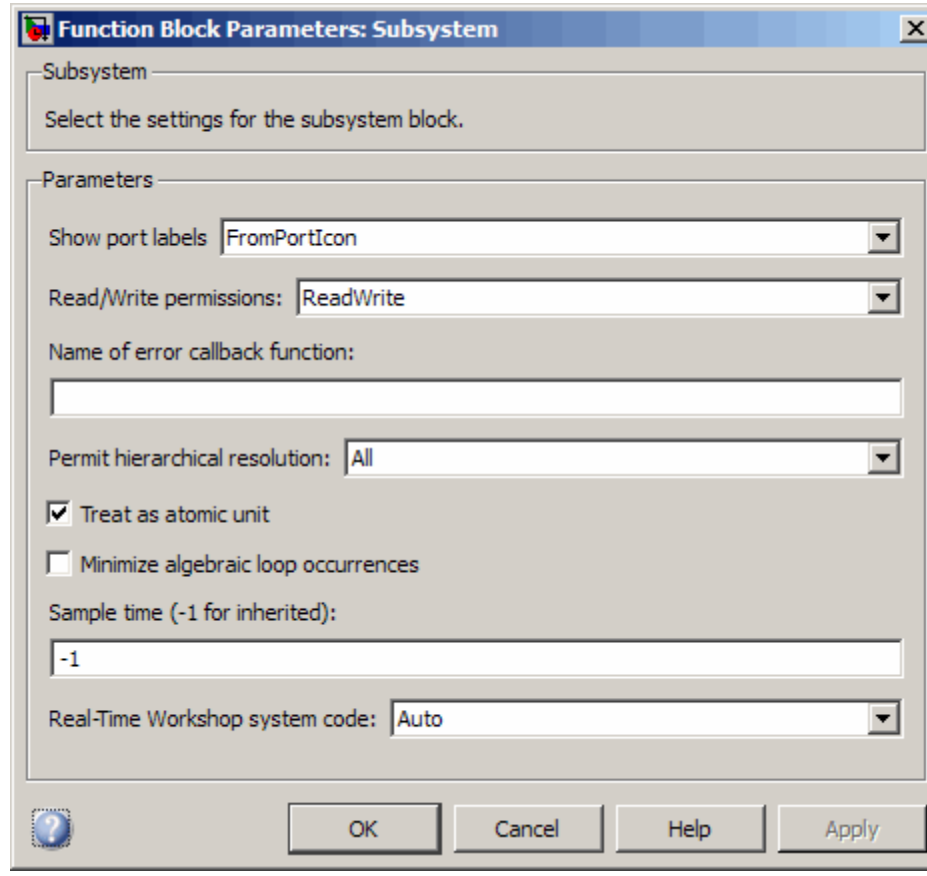
The coder verifies that your model satisfies the Simulink PLC Coder criteria and displays an information window when done.



If the subsystem is not atomic, right-clicking the Subsystem block and selecting **PLC Coder** 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 1-20.

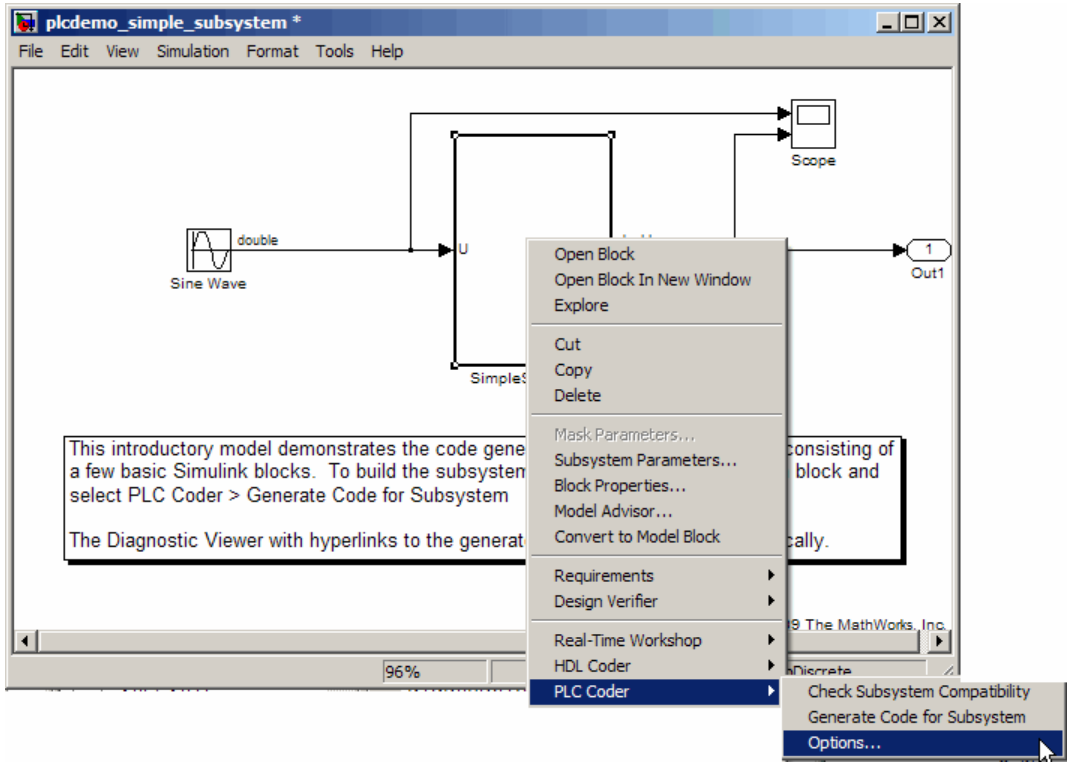
Generating and Examining Structured Text Code

In this section...

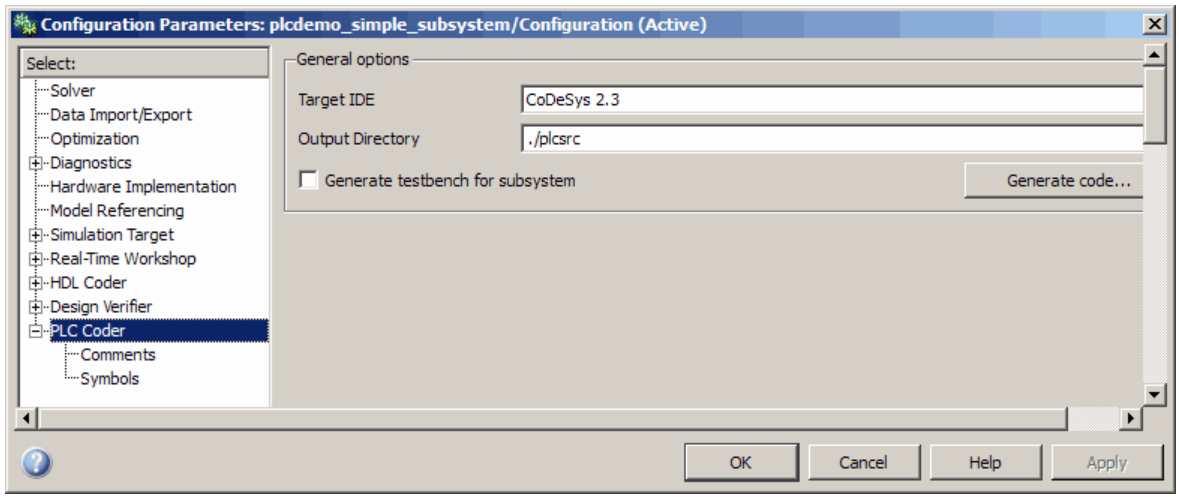
Generating Structured Text Code from the Model Window

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 1-11.

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



The Configuration Parameters dialog box is displayed.



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

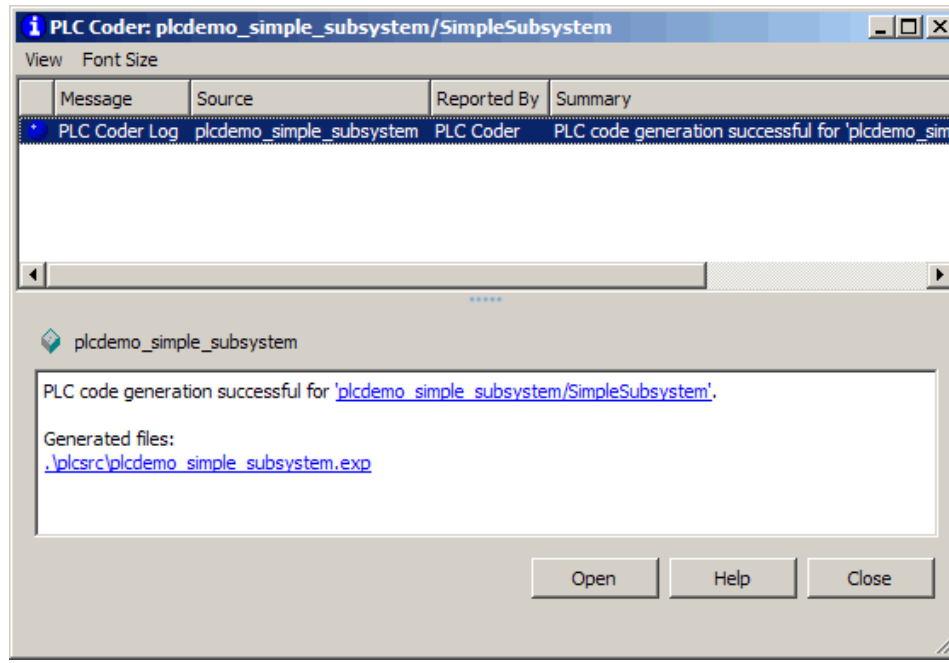
4 Click **Apply**.

5 Click the **Generate code** button.

This button:

- Generates structured text code (same as the **PLC Coder > 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 depending on the target IDE platform.

Platform	Generated Files
CoDeSys IDE	*current_folder*\plcsrc\model_name.exp — Structured text file appropriate for downloading to the target IDE.
Rockwell Automation RSLogix 5000 IDE	*current_folder*\plcsrc\plcdemo_simple_subsystem.L5X (*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.

Platform	Generated Files
B&R Automation Studio IDE	Five or six files in folder *current_folder*\plcsrc*model_name**subsystem* — Files appropriate for downloading to the target IDE. Note that the *subsystem* folder name might be truncated. • `IEC.lby` Main project 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.
PLCOpen XML	*current_folder*\plcsrc\plcdemo_simple_subsystem.xml (*model_name*`.xml`) — Structured text file formatted using the PLCOpen XML standard.
Beckhoff TwinCAT 2.11	*current_folder*\plcsrc\plcdemo_simple_subsystem.exp (*model_name*`.exp`). — Structured text file appropriate for downloading to the target IDE.
Generic	*current_folder*\plcsrc\plcdemo_simple_subsystem.st (*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.

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 format and looks different.

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

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

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

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

END_CASE;
END_FUNCTION_BLOCK
```

After generating structured text code, examine it 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.

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 1-20.

To generate the code for the subsystem, type:

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

2

Mapping Simulink Semantics to Structured Text

When you examine generated code, you evaluate how well the Simulink PLC Coder software has generated code from your model. The following topics describe how the coder maps Simulink subsystem semantics to function block semantics in structured text. As examples, the topics describe the mapping in the context of the different subsystem types that Simulink supports. The examples assume that you have already generated code (see "Generating Structured Text Code from the Model Window" on page 1-20). 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

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 1-20.

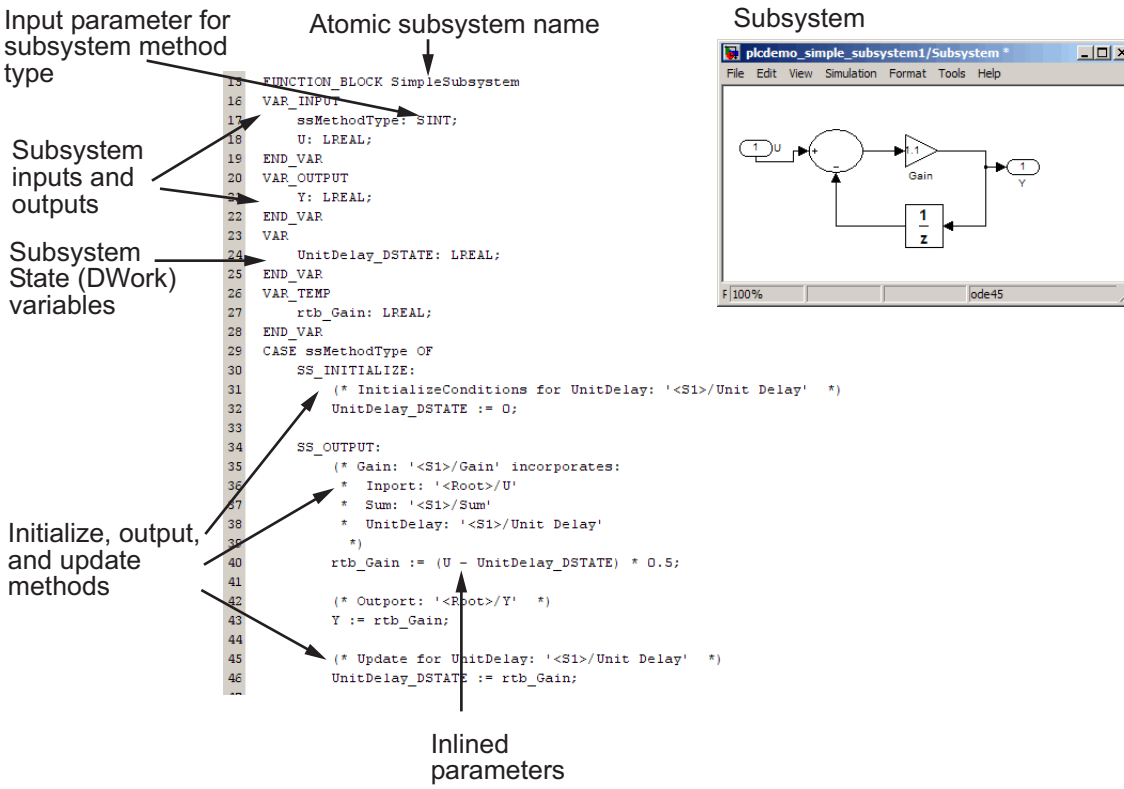
The example in this topic shows generated code for the CoDeSys Version 2.3 PLC IDE. Generated code for other IDE platforms looks different.

1. 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`.



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

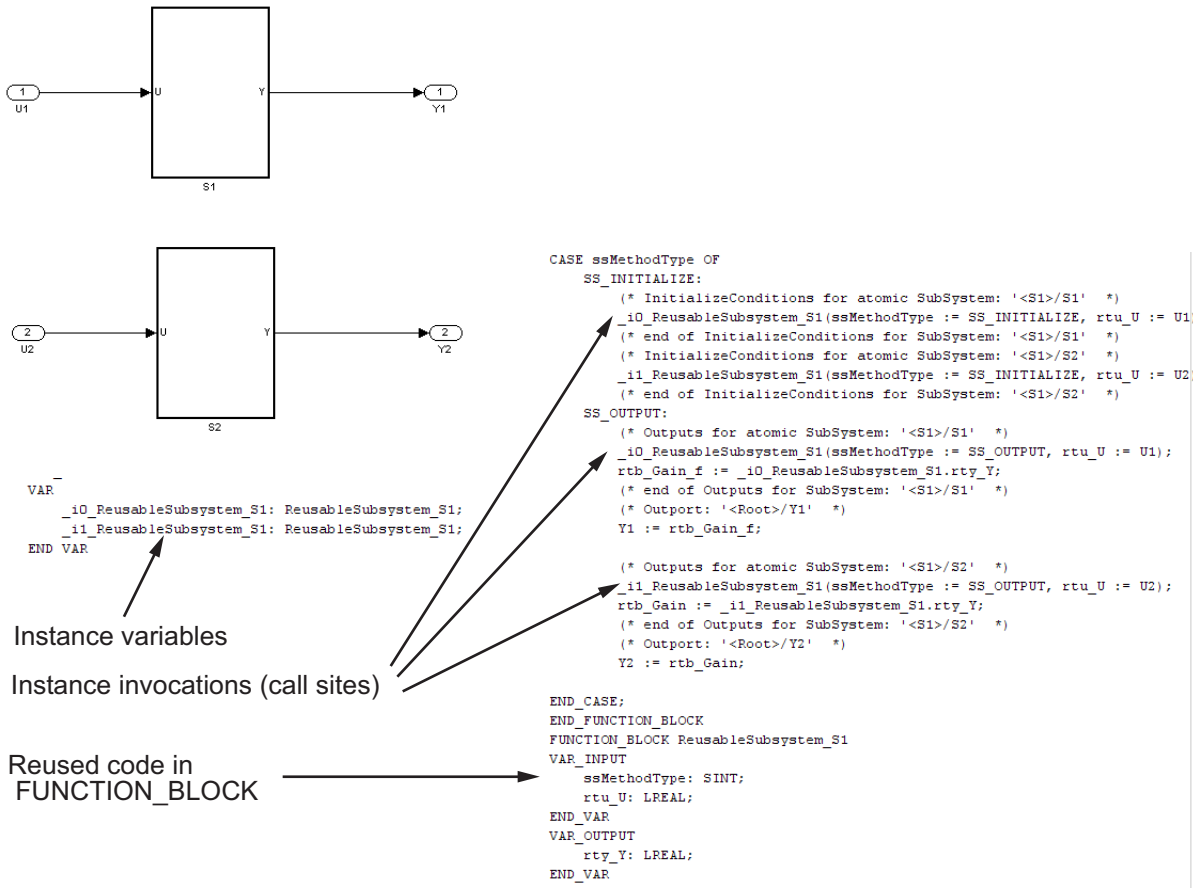
This topic assumes that you have generated structured text code from a Simulink model. If you have not yet done so, see “Generating Structured Text Code from the Model Window” on page 1-20.

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_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 `ReusableSubysstem` string in the code).



4 Examine the generated structured text code. The code defines `FUNCTION_BLOCK S0_S1` once.

5 Look for two instance variables that correspond to the two instances declared inside the parent `FUNCTION_BLOCK S0` (`_instance_S0_S1_1: S0_S1` and `_instance_S0_S1_0: S0_S1`). The code invokes these two instances separately by passing in different inputs. The code invokes the outputs per the Simulink execution semantics.

How Stateflow Subsystem Code Maps to Function Blocks

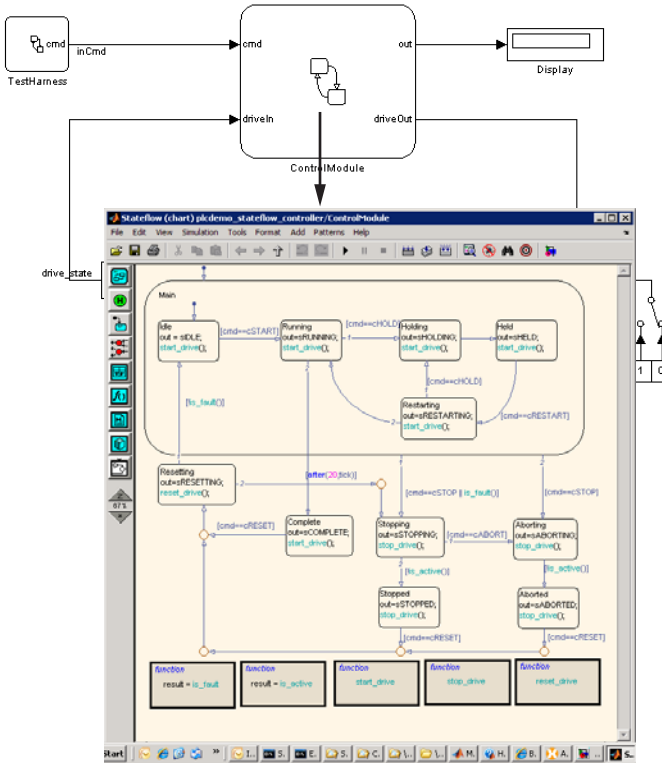
This topic assumes that you have generated structured text code from a Simulink model. If you have not yet done so, see “Generating Structured Text Code from the Model Window” on page 1-20.

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 Embedded MATLAB Subsystem Code Maps to Function Blocks

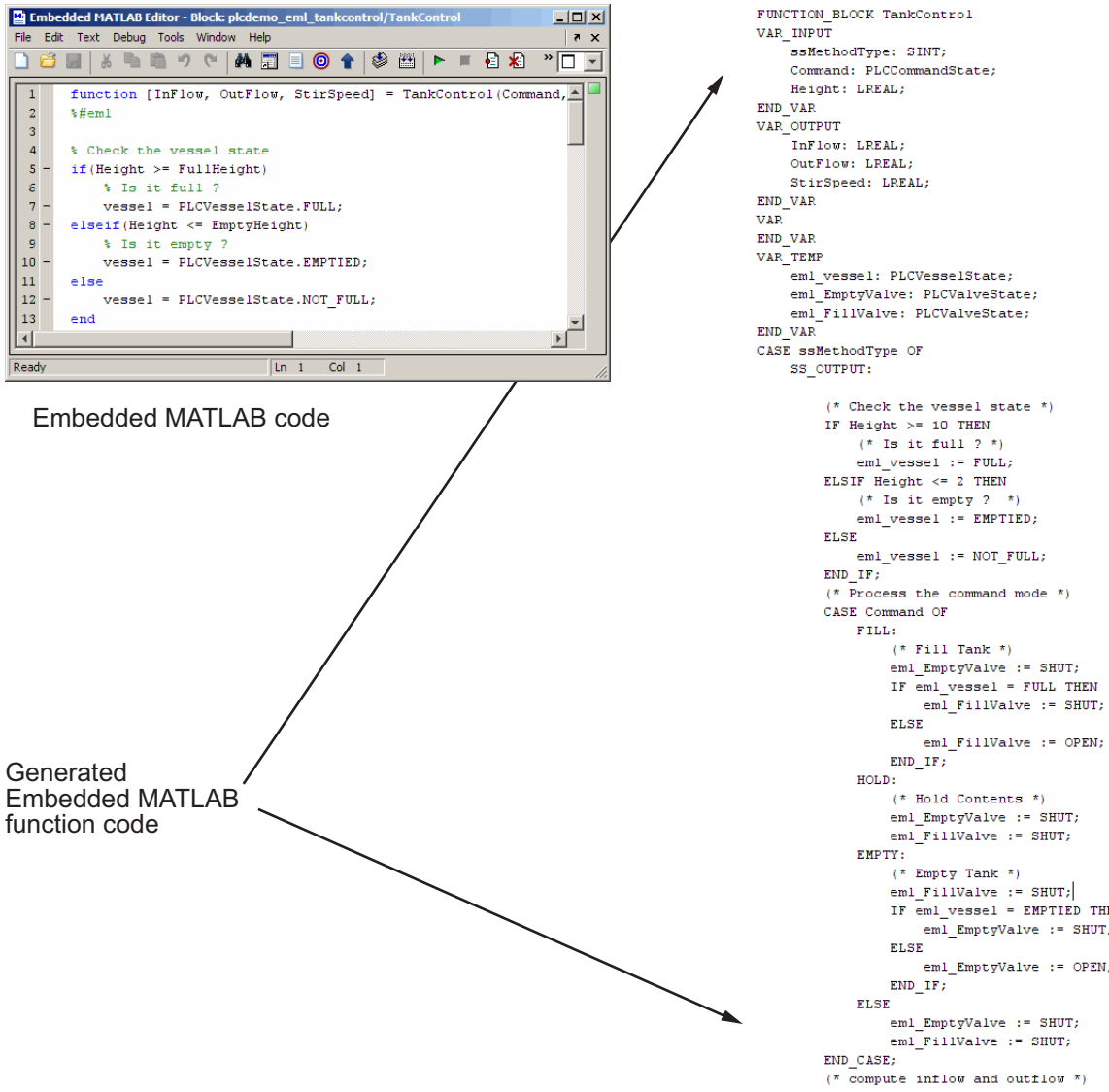
This topic assumes that you have generated structured text code from a Simulink model. If you have not yet done so, see "Generating Structured Text Code from the Model Window" on page 1-20.

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_eml_tankcontrol` model.
2. Right-click the TankControl 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_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 an Embedded MATLAB Function block to structured text components. The coder tries to perform inline optimization on the generated code for Embedded 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`. The main difference between a `FUNCTION_BLOCK` and `FUNCTION` is that a `FUNCTION` cannot have states and cannot access global variables.



4 Examine the generated structured text code.

How Alias Data Types Map in Generated Code

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

3

Generating Test Bench Code

Working with Generated Structured Text

In this section...

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 verifies that the generated code is functionally and numerically equivalent to the Simulink model. To perform this verification, import the generated structured text and the test bench data into your target IDE.

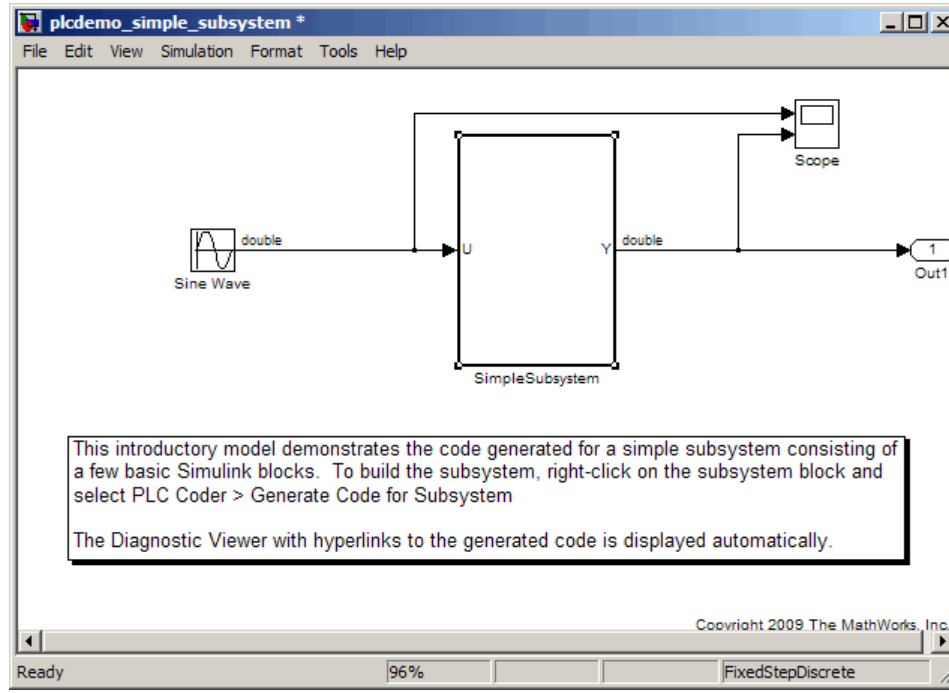
Generating and Importing Structured Text

This topic describes the basic procedure for how to generate test bench code. It uses the CoDeSys environment 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 1-11. 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 simulation results with Simulink output. For example:



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

The Configuration Parameters dialog box is displayed.

4 In **PLC Coder > 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**.

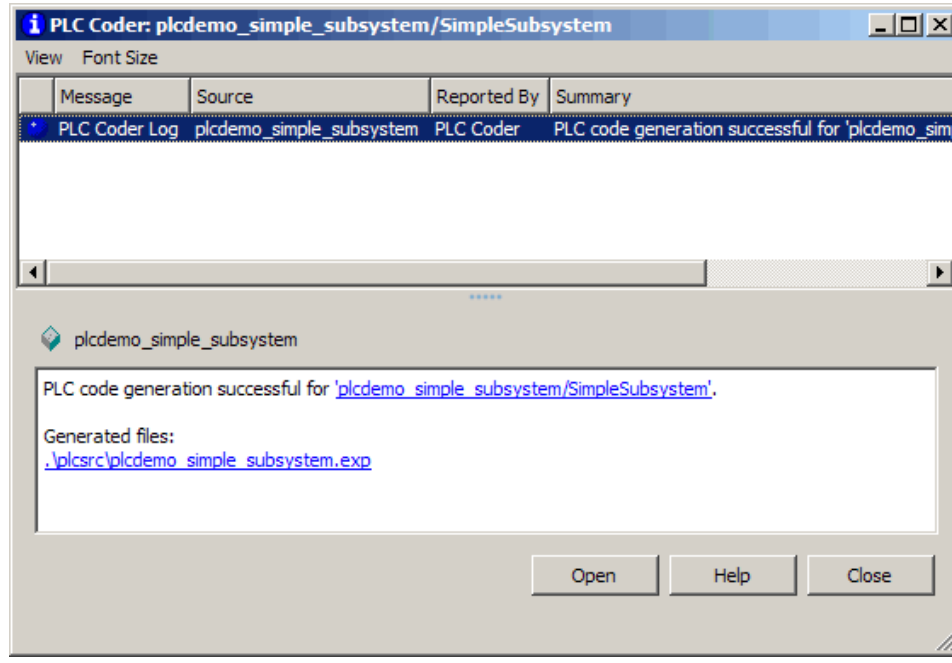
7 Click the **Generate code** button.

This button:

- Generates structured text code (same as the **PLC Coder > Generate Code for Subsystem** option)
- Generates test bench for code through Simulink simulation

- Combines 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. 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 1-20 for list of the target IDE platforms and the possible generated files.

4

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

Dialog 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.

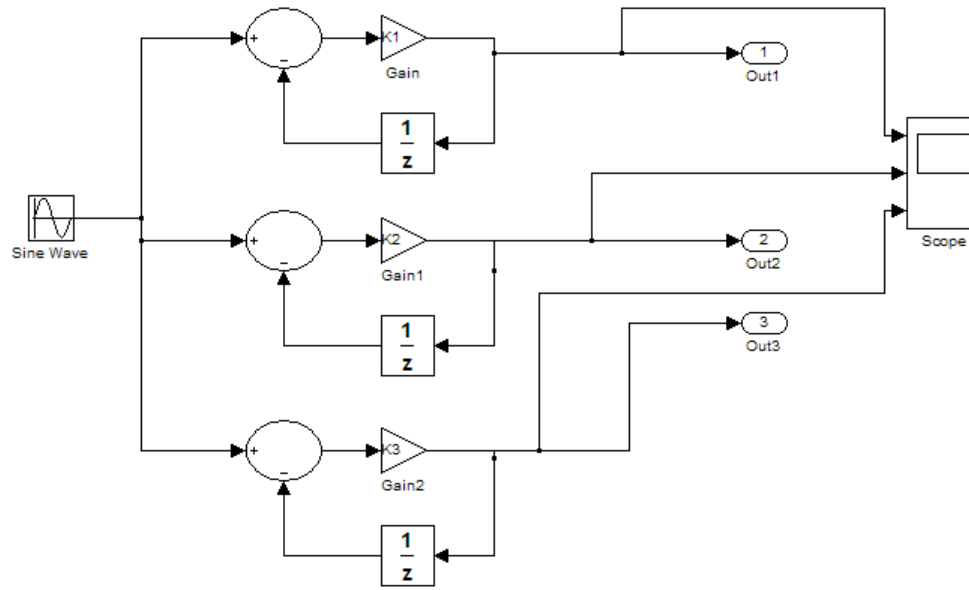
The general workflow for configuring a model to enable tunable parameters is:

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 > Model Parameter Configuration** dialog box.

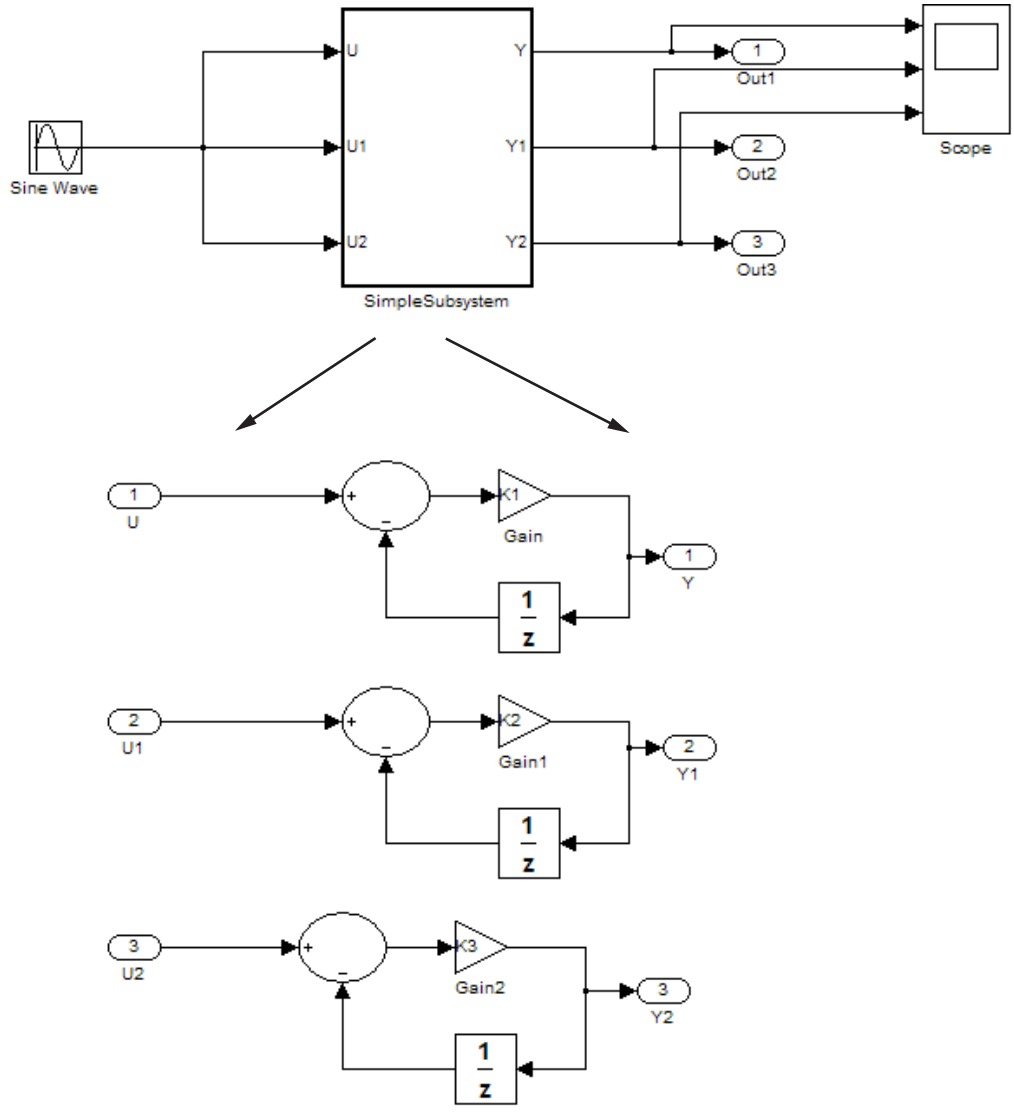
Identifying Tunable Parameters

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:

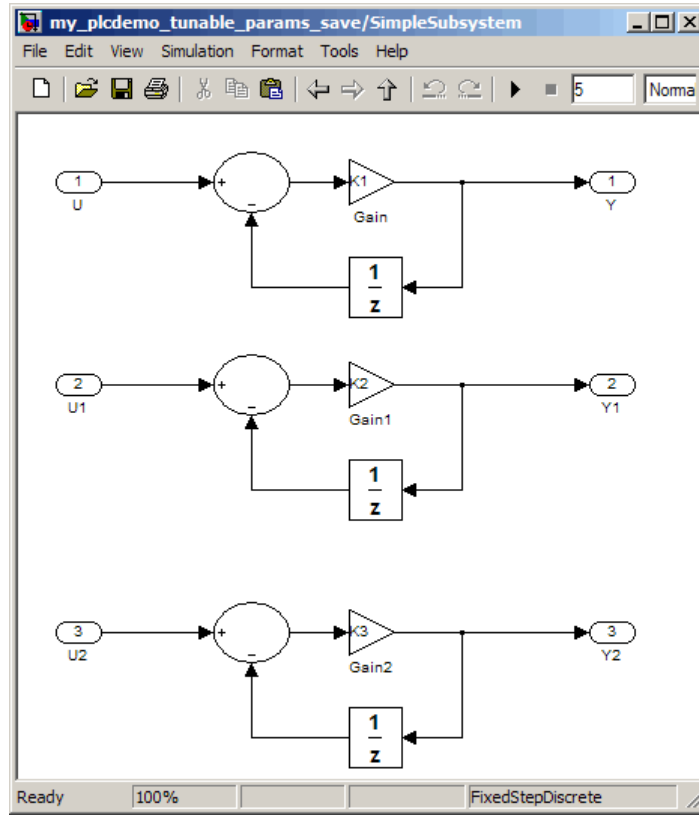


2 Select the Sum, Gain, and Unit Delay blocks and create an atomic subsystem with inputs U, U1, and U2 and outputs Y, Y1, and Y2. Rename the Subsystem block as SimpleSubystem. When you are finished, the top model and atomic subsystem model should look like the following:



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 4-5.

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.

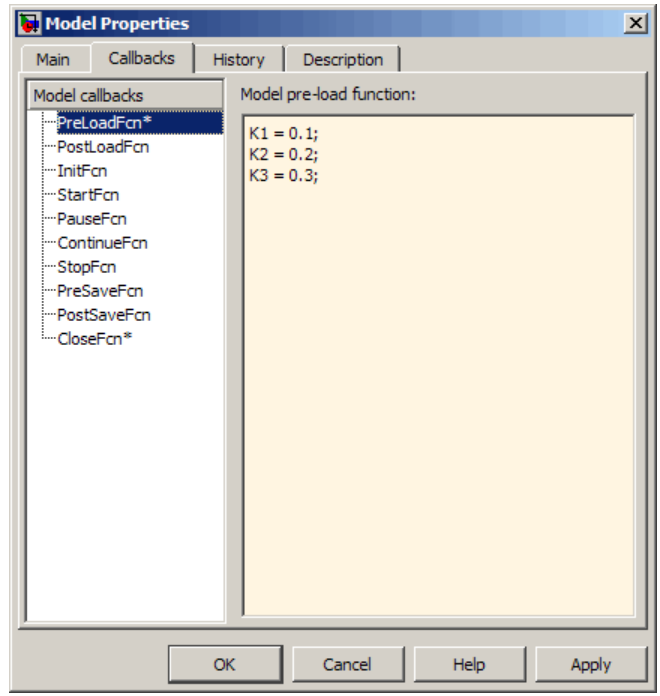
1. In the `my_plcdemo_tunable_params` model, select **File > Model Properties**.

   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;
   ```

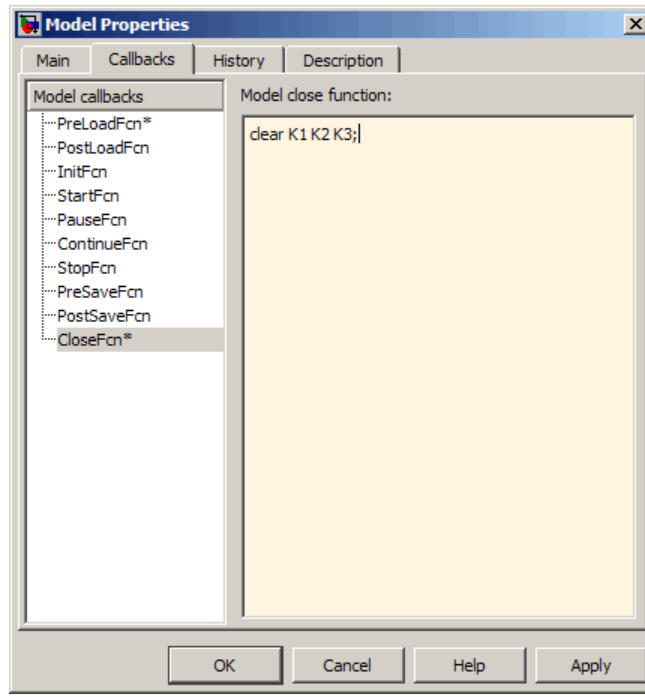


4 Click **Apply**.

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.

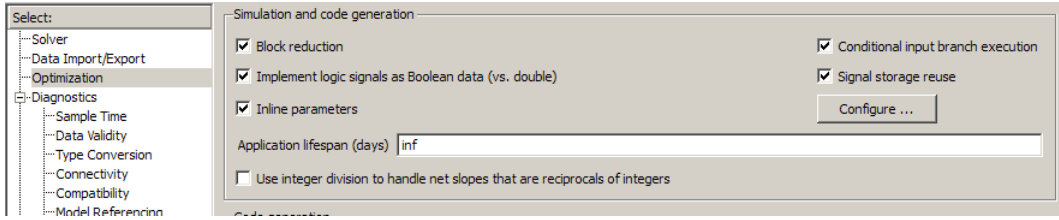
Configuring Parameters to Be Tunable

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 4-5.

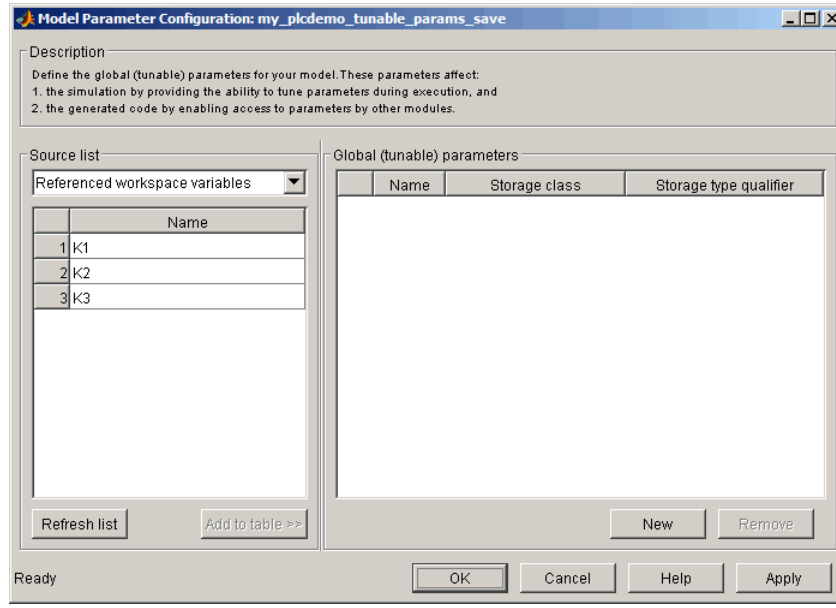
This topic uses code generated with CoDeSys Version 2.3.

1. In the model, right-click SimpleSubsystem and select **PLC Coder > Options**.
2. Navigate to the **Optimization** node.
3. In the **Simulation and code generation** section of the **Optimization** pane, ensure that the **Inline parameters** check box is selected. (This check box is selected by default.)



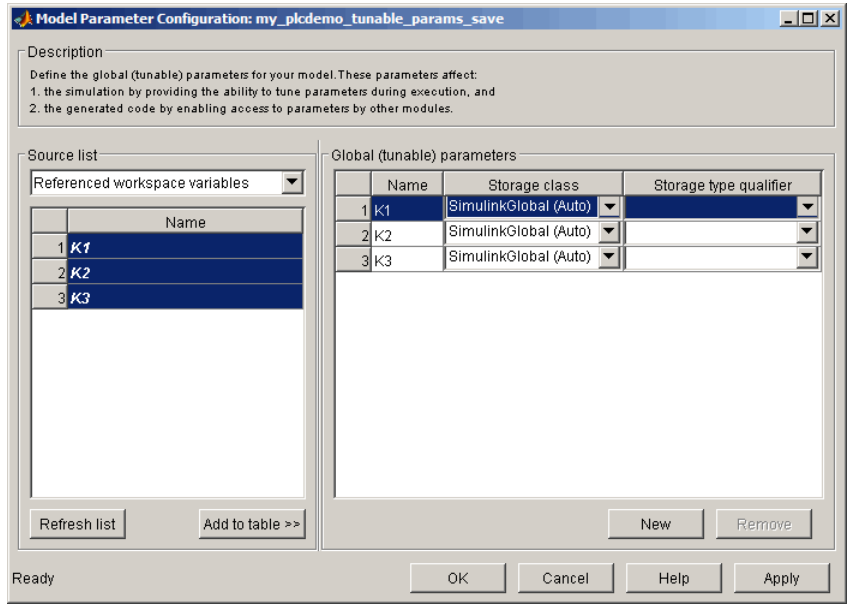
4. In this section, click **Configure**.

   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 Coder > 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 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.

```
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.

5

Controlling Generated Code Partitions

Function Block Partitions

In this section...

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 **Real-Time Workshop system code** parameter of the Subsystem block.

These topic use code generated with CoDeSys Version 2.3.

Example: One Function Block for Atomic Subsystems

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

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

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

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

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

END_CASE;
END_FUNCTION_BLOCK
```

Example: One Function Block for Virtual Subsystems

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

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

Example: Multiple Function Blocks for Nonvirtual Subsystems

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

Function Block for Hierarchical Subsystem

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

Function Block for Hierarchical S1

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

Function Block for Hierarchical S2

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

Controlling Generated Code Using Subsystem Block Parameters

You can partition generated code using the following Subsystem block parameters:

- **Real-Time Workshop system code**
- **Real-Time Workshop function name options**

Leave the **Real-Time Workshop file name options** set to the default, `Auto`.

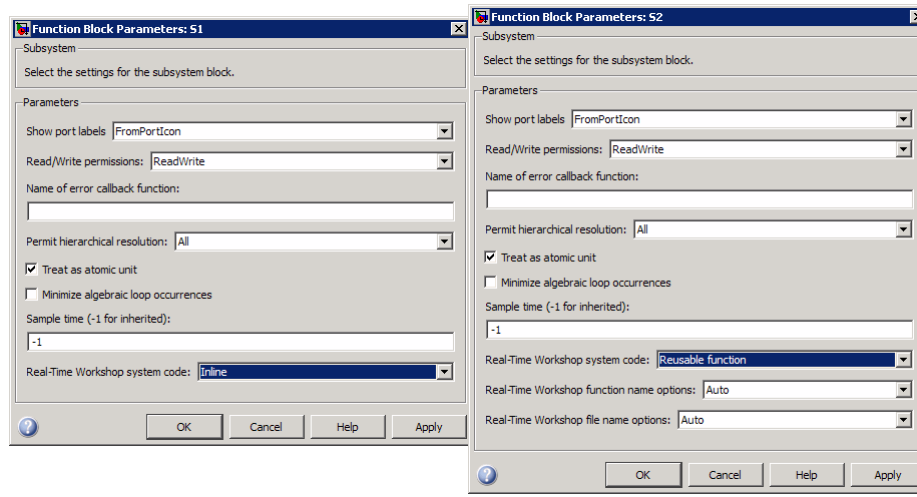
Generating Separate Partitions and Inlining Subsystem Code

Use the **Real-Time Workshop system code** parameter to specify the code format to generate for an atomic (nonvirtual) subsystem. The Simulink PLC Coder software interprets this parameter:

If Real-Time Workshop system code is	Coder Interpretation
`Auto`,	Chooses the optimal format based on the type and number of instances of the subsystem that exist 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:

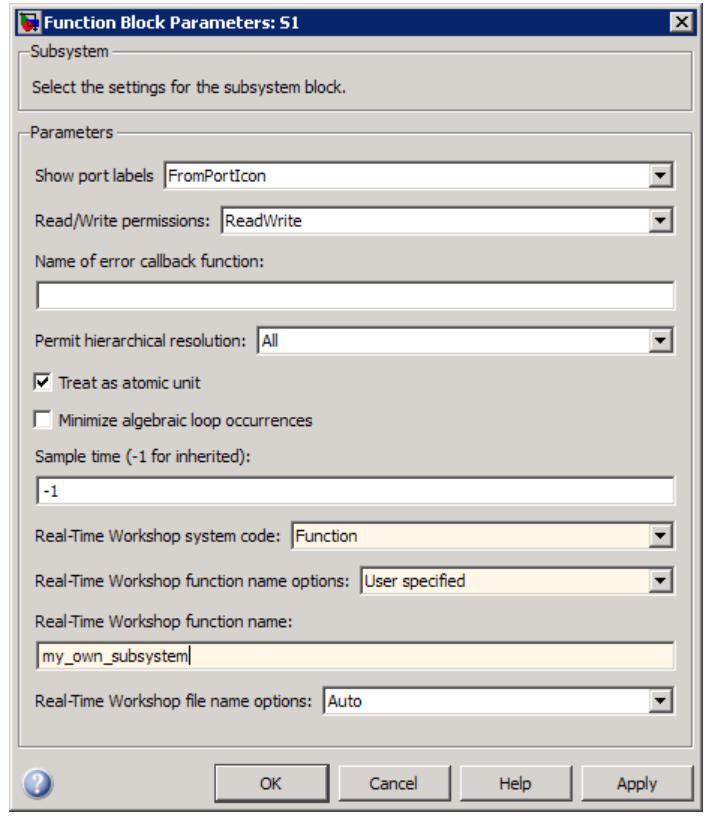
- Inline the S1 subsystem code by setting **Real-Time Workshop system code** to `Inline`. This setting creates one function block for the parent with the S1 subsystem inlined.
- Create a function block for the S2 subsystem by setting **Real-Time Workshop system code** to `Reusable function`, `Auto`, 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 **Real-Time Workshop function name options** parameter to change the name of a subsystem from the one on the block label. When the Simulink PLC Coder software 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. Set **Real-Time Workshop system code** to `Function`.
3. Set **Real-Time Workshop function name options** to `User specified`.
4. In the **Real-Time Workshop function name** field, specify a custom name. For example, type `my_own_subsystem`.



5 Save the new settings.

6 Generate code for the parent subsystem.

7 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.

In this section...

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 during code generation.

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 at code generation.

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 types in generated code.

7

Limitations

Coder Limitations

In this section...

Current Limitations

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

- Fixed-point data types
- 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`.
- Triggered subsystems
- Merge block
- Multi-rate models
- Signal and state storage classes

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.

8

Functions — Alphabetical List

plccoderdemos

Purpose

Product demos

Syntax

```
plccoderdemos
```

Description

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

Example

Display demos in the MATLAB Help browser.

```
plccoderdemos
```

See Also

`plcopenconfigset`

Purpose

Set target IDE

Syntax

```
plccoderpref
plccoderpref('plctargetide')
plccoderpref('plctargetide', IDEcodename)
plccoderpref('plctargetide', '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 previoiusly, or the factory default. The factory default is `'codesys23'`.

`plccoderpref('plctargetide', IDEcodename)` sets the default target IDE to the one that you specify in *`IDEcodename`*. This command ensures that *IDEcodename* 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'`).

Input Arguments

`'plctargetide'`

String directive that specifies the default target IDE.

`'IDEcodename'`

String that specifies the default target IDE. String must be one of:

Name	Description
`codesys23`	3S-Smart Software Solutions CoDeSys Version 2.3 (default) target IDE
`codesys33`	3S-Smart Software Solutions CoDeSys Version 3.3 target IDE
`rslogix5000`	Rockwell Automation RSLogix 5000 Series target IDE

Name	Description
`brautomation30`	B&R Automation Studio 3.0 target IDE
`plcopen`	PLCOpen XML target IDE
`twincat211`	Beckhoff TwinCAT 2.11 target IDE
`generic`	Generic target IDE

`'default'`

String directive that sets `'plctargetide'` to factory default (`'codesys23'`).

Example

Return the current default target IDE.

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

Set `rslogix5000` as the new default target IDE.

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

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.

Example

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.

Example

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

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

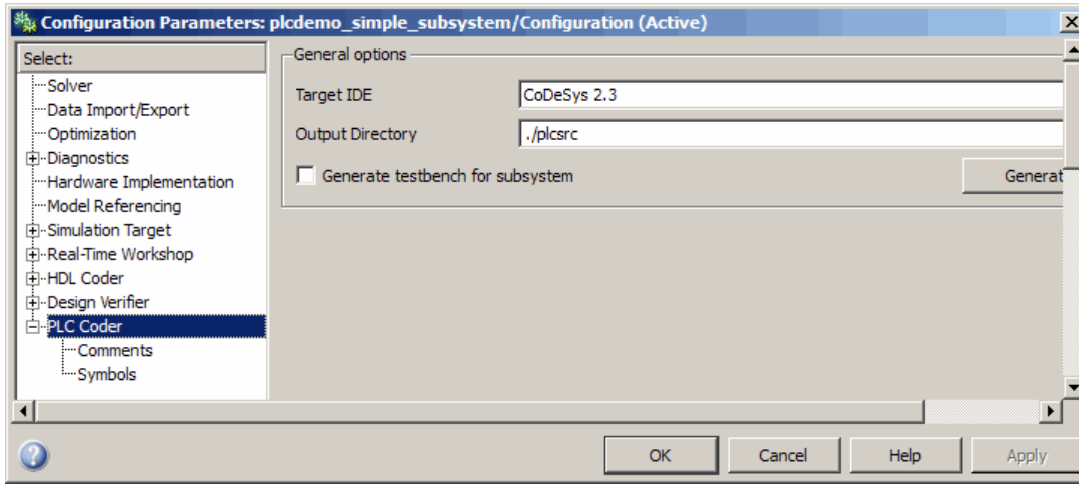
See Also

`plcgeneratecode`

9

Configuration Parameters for Simulink PLC Coder Models

PLC Coder: General



In this section...

PLC Overview

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

Configuration

This pane appears only if your model contains an atomic Subsystem block.

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 Coder > 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

“Generating Structured Text Code from the Model Window” on page 1-20

Target IDE

Select the target language in which to generate code.

Settings

Default: `Structured Text (CoDeSys)`

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

`CoDeSys 3.3`
: Generates structured text (IEC 61131) code for 3S-Smart Software Solutions CoDeSys software Version 3.3.

`RSLogix 5000`
: Generates structured text code for Rockwell Automation RSLogix 5000 software.

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

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

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

`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.

Tip

- 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_TargetIDE`
Type: string
Value: `'codesys23' | 'codesys33' | 'rslogix5000' | 'brautomation30' | 'plcopen' | 'twincat211' | 'generic'`
Default: `'codesys23'`

See Also

"Generating Structured Text Code from the Model Window" on page 1-20

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 1-20

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.

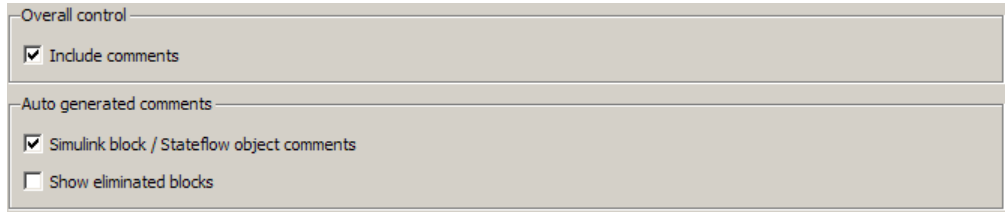
Command-Line Information

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

See Also

“Generating Structured Text Code from the Model Window” on page 1-20

PLC Coder: Comments



In this section...

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 1-20

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 1-20

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 1-20

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.

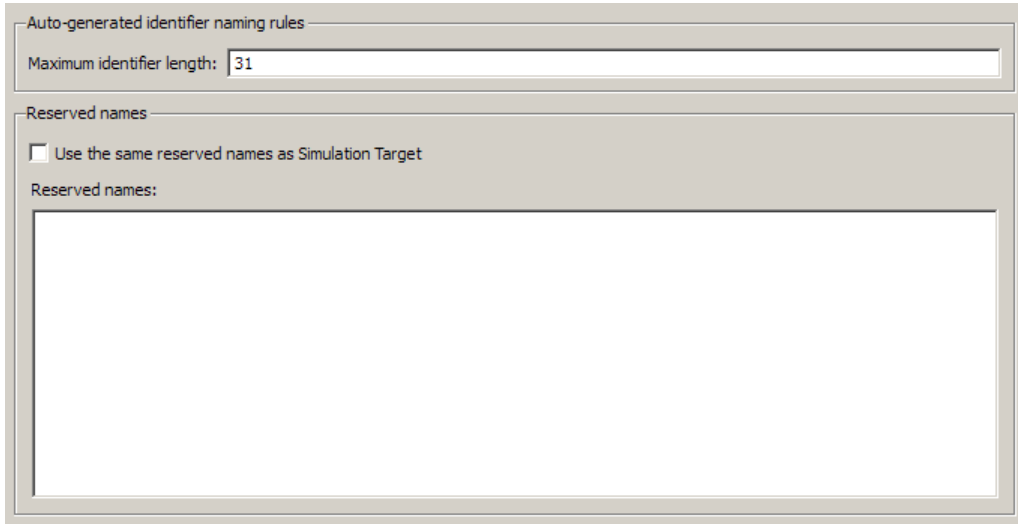
Command-Line Information

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

See Also

“Generating Structured Text Code from the Model Window” on page 1-20

PLC Coder: Symbols



In this section...

Symbols Overview

Select the automatically generated identifier naming rules.

See Also

"Generating Structured Text Code from the Model Window" on page 1-20

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.

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 1-20

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 1-20

Reserved names

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

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 1-20

Index

T

U

V

W