Simulink® PLC Coder™

User's Guide

MATLAB&SIMULINK®

MathWorks®

How to Contact MathWorks

	Latest news:	www.mathworks.com
	Sales and services:	www.mathworks.com/sales_and_services
	User community:	www.mathworks.com/matlabcentral
	Technical support:	www.mathworks.com/support/contact_us
	Phone:	508-647-7000

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

Simulink® PLC Coder™ User's Guide

Revision History

March 2010	Online only	New for Version 1.0 (Release 2010a)
September 2010	Online only	Revised for Version 1.1 (Release 2010b)
April 2011	Online only	Revised for Version 1.2 (Release 2011a)
September 2011	Online only	Revised for Version 1.2.1 (Release 2011b)
March 2012	Online only	Revised for Version 1.3 (Release 2012a)
September 2012	Online only	Revised for Version 1.4 (Release 2012b)
March 2013	Online only	Revised for Version 1.5 (Release 2013a)
September 2013	Online only	Revised for Version 1.6 (Release 2013b)
March 2014	Online only	Revised for Version 1.7 (Release 2014a)
October 2014	Online only	Revised for Version 1.8 (Release 2014b)
March 2015	Online only	Revised for Version 1.9 (Release 2015a)
September 2015	Online only	Revised for Version 2.0 (Release 2015b)
March 2016	Online only	Revised for Version 2.1 (Release 2016a)
September 2016	Online only	Revised for Version 2.2 (Release 2016b)
March 2017	Online only	Revised for Version 2.3 (Release 2017a)
September 2017	Online only	Revised for Version 2.4 (Release 2017b)
March 2018	Online only	Revised for Version 2.5 (Release 2018a)
September 2018	Online only	Revised for Version 2.6 (Release 2018b)
March 2019	Online only	Revised for Version 3.0 (Release 2019a)
September 2019	Online only	Revised for Version 3.1 (Release 2019b)
March 2020	Online only	Revised for Version 3.2 (Release 2020a)
September 2020	Online only	Revised for Version 3.3 (Release R2020b)
March 2021	Online only	Revised for Version 3.4 (Release R2021a)
September 2021	Online only	Revised for Version 3.5 (Release R2021b)
March 2022	Online only	Revised for Version 3.6 (Release R2022a)
September 2022	Online only	Revised for Version 3.7 (Release R2022b)

Contents

Getting Started

1

Mapping Simulink Semantics to Structured Text

2

Generating Ladder Diagram

3

Generating Test Bench Code

4

5

6

Working with Tunable Parameters in the Simulink PLC Coder Environment

7

Controlling Generated Code Partitions

8

Integrating Externally Defined Identifiers

9

IDE-Specific Considerations

10

Supported Simulink and Stateflow Blocks

11

Limitations

12

Configuration Parameters for Simulink PLC Coder Models

13

External Mode

14

Ladder Diagram Instructions

15

Ladder Diagram Blocks

16

Fixed Point Code Generation

17

25

PLC Coder Model Advisor

26

Using the PLC Coder Model Advisor

27

Custom Keyword List

28

Plugin Based Targets

29

Variable-Size Code Generation

30

Getting Started

Simulink PLC Coder Product Description

Generate IEC 61131-3 Structured Text and Ladder Diagrams for PLCs and PACs

Simulink PLC Coder generates hardware-independent IEC 61131-3 Structured Text and Ladder Diagrams from Simulink models, Stateflow® charts, and MATLAB® functions. Structured Text is generated in PLCopen XML and other file formats supported by widely used integrated development environments (IDEs) including 3S-Smart Software Solutions CODESYS, Rockwell Automation® Studio 5000, Siemens® TIA Portal, and OMRON® Sysmac® Studio. Ladder diagrams are generated in file formats supported by Rockwell Automation Studio 5000. 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 and Ladder Diagrams using PLC and PAC IDEs and simulation tools. It also provides code generation reports with static code metrics and bidirectional traceability between model and code. Support for industry standards is available through IEC Certification Kit (for IEC 61508 and IEC 61511).

Prepare Model for Structured Text Generation

In this section...
"Tasking Mode" on page 1-3
"Choose a Solver" on page 1-3
"Configure Simulink Models for Structured Text Code Generation" on page 1-3
"Verify System Compatibility for Structured Text Code Generation" on page 1-6

To generate structured text code from Simulink models, Stateflow charts, and MATLAB functions, use the Simulink PLC Coder product. Prepare your model for structured text code generation by performing action such as setting the solver, identifying if your model is a single rate or multirate model, and checking model compatibility for structured text code generation.

Tasking Mode

If your Simulink model contains multirate signals, you must set the tasking mode. If your Simulink model does not contain multirate signals, proceed to solver selection.

Simulink PLC Coder generates code for single-tasking subsystems. For multirate subsystems, you must first explicitly set the tasking mode to single-tasking before selecting a solver. In the model configuration, on the **Solver** pane, clear the check box for **Treat each discrete rate as a separate task**.

Choose a Solver

Choose a solver for your Simulink PLC Coder model.

Model	Solver Setting
Variable-step	Use a continuous solver. Configure a fixed sample time for the subsystem for which you generate code.
Fixed-step	Use a discrete fixed-step solver.

Configure Simulink Models for Structured Text Code Generation

This tutorial uses the example model `plcdemo_simple_subsyystem`.

1 In the Command Window, enter the model name to open the model.

    plcdemo_simple_subsystem
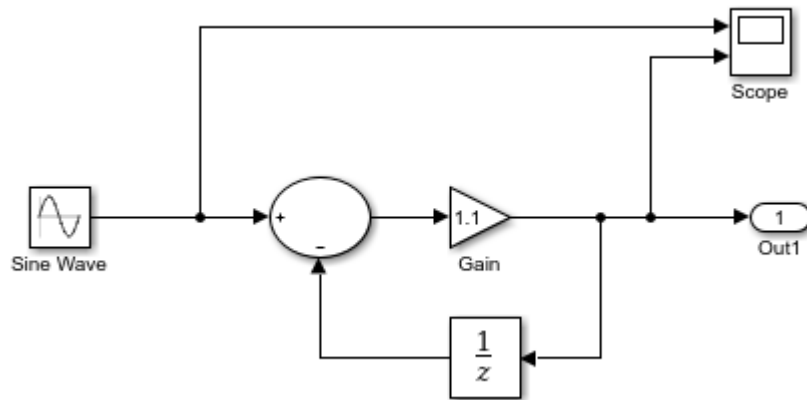
2. Configure the model to use the fixed-step discrete solver. Click the solver link in the lower-right corner of the model window. In the **Solver information** pane, click **View solver settings** to open the **Solver** pane of the model configuration parameters. Under the **Solver** selection, set **Type** to Fixed-step and **Solver** to discrete (no continuous states).

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

3. Save this model as plcdemo_simple_subsystem1.

4. Create a subsystem containing the components for which you want to generate structured text code.

	Cut	Ctrl+X
	Copy	Ctrl+C
	Paste	Ctrl+V
	Comment Through	Ctrl+Shift+Y
	Comment Out	Ctrl+Shift+X
	Delete	Del
	Connect Blocks	
	Create Subsystem from Selection	**Ctrl+G**
	Log Selected Signals	
	Format	▶
	Rotate & Flip	▶
	Arrange	▶
	Requirements Traceability	▶
	C/C++ Code	▶
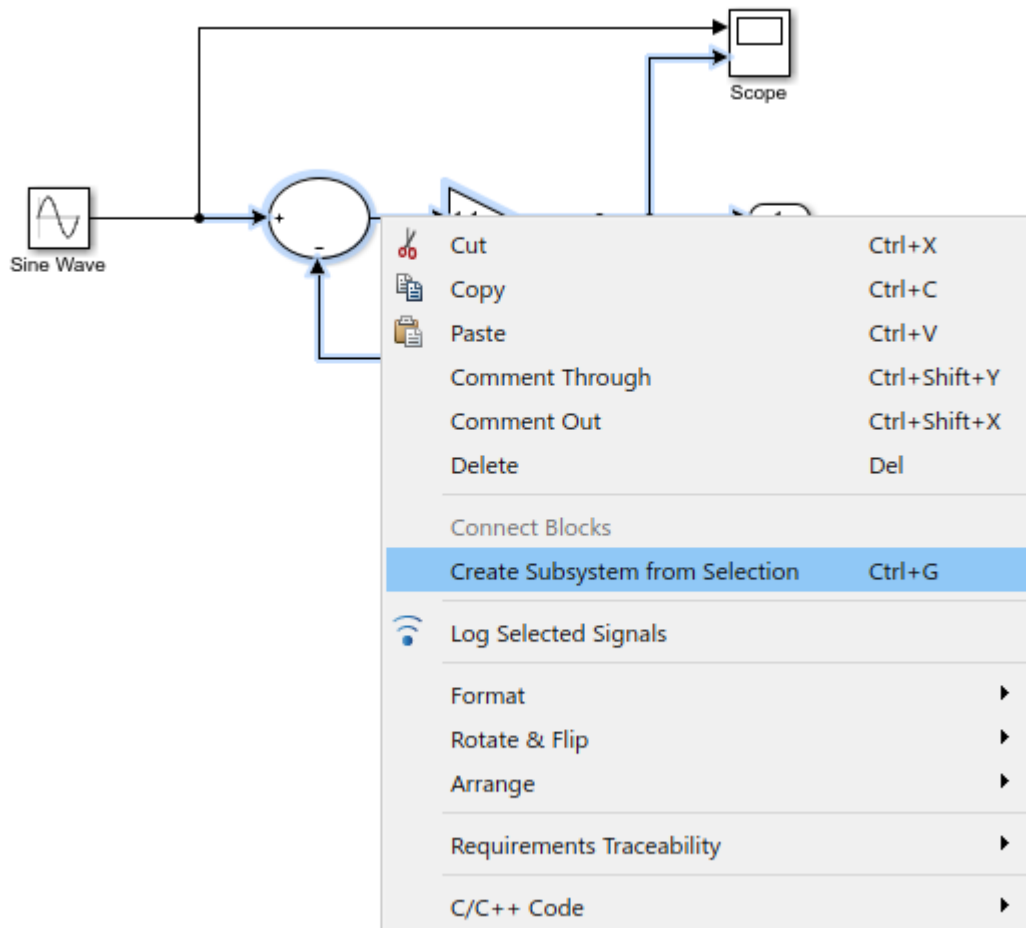
Optionally, rename `In1` and `Out1` to `U` and `Y` respectively, resulting in a subsystem like the following figure:
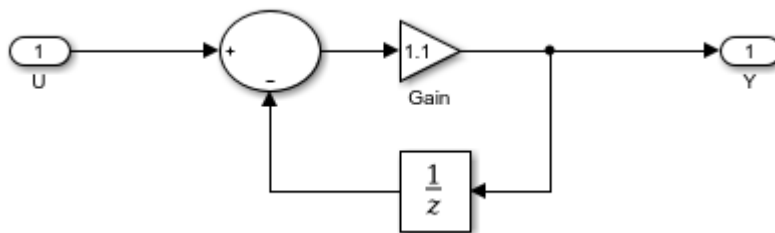
5   Save the model with the new subsystem.

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

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

8   Click **OK**.

9   Simulate, and then save your model.

1-5

You can now:

- Set up your subsystem to generate structured text code. See "Verify System Compatibility for Structured Text Code Generation" on page 1-6.
- Generate structured text code for your IDE. See "Generate and Examine Structured Text Code" on page 1-7.

Verify System Compatibility for Structured Text Code Generation

You must already have a model configured to work with the Simulink PLC Coder software. If not, see "Prepare Model for Structured Text Generation" on page 1-3

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 > Check Subsystem Compatibility**.

The coder verifies whether your model satisfies the Simulink PLC Coder criteria. When the verification is complete, a **View diagnostics** hyperlink appears at the bottom of the model window. Click this hyperlink to open the Diagnostic Viewer window.

If the subsystem is not atomic, right-click the subsystem block and select **PLC Code > Enable "Treat as atomic unit" to generate code**.
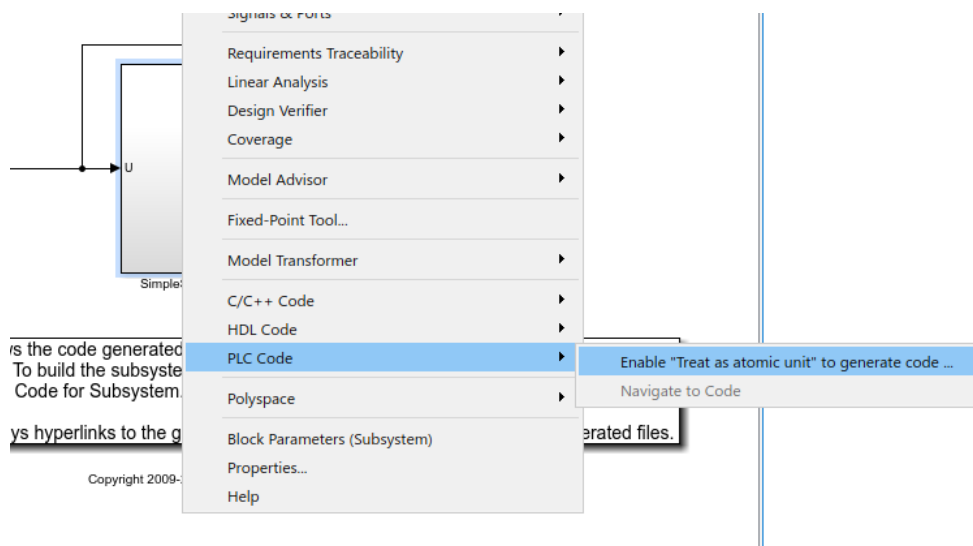
In the block parameter dialog box, select **Treat as atomic unit**.

Generate structured text code for your IDE. See "Generate and Examine Structured Text Code" on page 1-7.

Generate and Examine Structured Text Code

In this section...
"Generate Structured Text from the Model Window" on page 1-7
"Generate Structured Text Through the MATLAB Interface" on page 1-7
"View Generated Code" on page 1-8

To generate structured text code from Simulink models, Stateflow charts, and MATLAB functions, use the Simulink PLC Coder product. Use the generated structured text code in applications such as rapid prototyping, control algorithm validation, and test bench verification.

Generate Structured Text from the Model Window

To generate structured text code from your Simulink model, complete the steps to prepare your model for structured text code generation. For more information, see "Prepare Model for Structured Text Generation" on page 1-3. This tutorial uses the `plcdemo_simple_subsystem`.

1   In the Command Window, enter the model name to open the model.

    `plcdemo_simple_subsystem`

2   Open the **PLC Coder** app. Click the **PLC Code** tab.

3   Click **Settings**.

4   On the **PLC Code Generation** pane, select an option from the **Target IDE** list, for example, `3S CoDeSys 2.3`.

    The default **Target IDE** list displays the full set of supported IDEs. To see a reduced subset of the target IDEs supported by Simulink PLC Coder, clear the **Show full target list** check box. To customize this list, use the `plccoderpref` function.

5   Click **OK**.

6   Click **Generate PLC Code** to:

    - Generate structured text code.
    - Store generated code in *model_name*.exp (for example, `plcdemo_simple_subsystem.exp`)

    When code generation is complete, a **View diagnostics** hyperlink appears at the bottom of the model window. Click this hyperlink to open the Diagnostic Viewer window.

    This window has links that you can click to open the associated files. For more information, see "Files Generated by Simulink PLC Coder" on page 1-11.

Generate Structured Text Through the MATLAB Interface

You can generate structured text code for a subsystem in the Command Window by using the `plcgeneratecode` function. You must have already configured the parameters for the model or, alternatively, you can use the default settings.

For example, to generate code from the `SimpleSubsystem` subsystem in the `plcdemo_simple_subsystem` model:

1.   Open the `plcdemo_simple_subsystem` model:

    `plcdemo_simple_subsystem`

2.   Open the Configuration Parameters dialog box by using the `plcopenconfigset` function:

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

3.   Select a target IDE.

4.   Configure the subsystem by preparing your model for structured text code generation. For more information, see "Prepare Model for Structured Text Generation" on page 1-3.

5.   Generate code for the subsystem:

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

When using `plcgeneratecode` for code generation, all diagnostic messages are printed to the MATLAB Command Window.

View Generated Code

After generating the code, you can view it in the MATLAB editor. For a description of how the generated code for the Simulink components map to structured text components, see "Verify Generated Code by Using Code Tracing" on page 6-2. You can view:

- Matrix data types: Simulink PLC Coder converts matrix data types to single-dimensional vectors (column-major) in the generated structured text.

- Generated code header: If your model has author names, creation dates, and model descriptions, the generated code contains these items in the header comments. The header also lists fundamental sample times for the model and the subsystem block for which you generate code.

- Code comments: You can choose to propagate block descriptions to comments in generated code. See "Propagate Block Descriptions to Code Comments" on page 1-10.

This figure 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 another format.

```
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
```

Once you are satisfied with the generated structured text code, optionally change your workflow to automatically generate and import code to the target IDE. For more information, see "Import Structured Text Code Automatically" on page 1-14.

Propagate Block Descriptions to Code Comments

You can propagate block descriptions from the model to comments in your generated code.

For specific IDEs, you can propagate the block descriptions into specific XML tags in the generated code. The IDEs use the tags to create a readable description of the function blocks in the IDE.

- For Rockwell Automation RSLogix 5000 AOI/routine target IDEs, the coder propagates block descriptions from the model into the L5X `AdditionalHelpText` XML tag. The IDE can then import the tag as part of AOI and routine definition in the generated code.

- For CoDeSys 3.5 IDE, the coder propagates block descriptions from the model into the `documentation` XML tag. When you import the generated code into the CoDeSys 3.5 IDE, the IDE parses the content of this tag and provides readable descriptions of the function blocks in your code.

To propagate block descriptions to comments:

1 Enter a description for the block.

   a Right-click the block for which you want to write a description and select **Properties**.

   b On the **General** tab, enter a block description.

2 Before code generation, specify that block descriptions must propagate to code comments.

   a Right-click the subsystem for which you are generating code and select **PLC Code** > **Options**.

   b Select the option Include block description on page 13-14.

Your block description appears as comments in the generated code.

Files Generated by Simulink PLC Coder

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 "Specify Custom Names for Generated Files" on page 1-13.

Platform	Generated Files
3S-Smart Software Solutions CoDeSys 2.3	*current_folder*\plcsrc\model_name.exp — Structured Text file for importing to the target IDE.
3S-Smart Software Solutions CoDeSys 3.3	*current_folder*\plcsrc\model_name.xml — Structured Text file for importing to the target IDE.
3S-Smart Software Solutions CoDeSys 3.5	*current_folder*\plcsrc\model_name.xml — Structured Text file for importing to the target IDE.
B&R Automation Studio® IDE	The following files in *current_folder*\plcsrc\model_name — Files for importing to the target IDE: • Package.pkg — (If test bench is generated) Top-level package file for function blocks library and test bench main program in XML format. In the main folder (if test bench is generated): • IEC.prg — Test bench main program definition file in XML format. • mainInit.st — Text file. Test bench init program file in Structured Text. • mainCyclic.st — Text file. Test bench cyclic program file in Structured Text. • mainExit.st — Text file. Test bench exit program file in Structured Text. • main.typ — Text file. Main program type definitions file in Structured Text. • main.var — Text file. Main program variable definitions file in Structured Text.
Beckhoff® TwinCAT® 2.11	*current_folder*\plcsrc\model_name.exp — Structured Text file for importing to the target IDE.
Beckhoff TwinCAT 3	*current_folder*\plcsrc\model_name.xml — Structured Text file for importing to the target IDE.
KW-Software MULTIPROG® 5.0	*current_folder*\plcsrc\model_name.xml — Structured Text file, in XML format, for importing to the target IDE.
Phoenix Contact® PC WORX™ 6.0	*current_folder*\plcsrc\model_name.xml — Structured Text file, in XML format, for importing to the target IDE.
Rockwell Automation Studio 5000 IDE: AOI	*current_folder*\plcsrc\model_name.L5X — (If test bench is generated) Structured Text file for importing to the target IDE using Add-On Instruction (AOI) constructs. This file is in XML format and contains the generated Structured Text code for your model.

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

Specify Custom Names for Generated 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. For more information, see "Files Generated by Simulink PLC Coder" on page 1-11.

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 `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 "Prepare Model for Structured Text Generation" on page 1-3, 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`.

Import Structured Text Code Automatically

In this section...
"PLC IDEs for Importing Code Automatically" on page 1-14
"Generate and Automatically Import Structured Text Code" on page 1-14
"Troubleshoot Automatic Import Issues" on page 1-14

PLC IDEs for Importing Code Automatically

You can generate and automatically import structured text code to the 3S-Smart Software Solutions CoDeSys Version 2.3 target PLC IDE.

Generate and Automatically Import Structured Text Code

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

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

Note While the automatic import process is in progress, do not use your mouse or keyboard to avoid disrupting the process. When the process is complete, you can resume normal operations.

You must have already installed your target PLC IDE in a default location. The target IDE must use 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 13-6.

1 If it is not already started, open the Command Window.
2 Open the `plcdemo_simple_subsystem` model.
3 Select the `Subsystem` block. Open the **PLC Coder** app.
4 Open the **PLC Code** tab, click **Settings > Import Code into IDE**.
5 Open the **PLC Code** tab and click **Generate PLC Code**.
6 The software:

    a Generates the code.
    b Starts the target IDE interface.
    c Creates a project.
    d Imports the generated code to the target IDE.

If you want to generate, import, and verify the structured text code, see "Import and Verify Structured Text Code" on page 4-4.

Troubleshoot Automatic Import Issues

Following are 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 3S-Smart Software Solutions CoDeSys Version 2.3 target IDE for automatic code import and verification.

Note Some antivirus softwares falsely identifies the executables that implement the automatic import feature as malware. You can ignore this false identification. For more information, see "Issues with Anti-Virus Software".

Target IDEs Not Supported For Automatic Import

The following target IDEs do not support automatic import. For these target IDEs, the **Import Code into IDE** and **Verify Code in IDE** are disabled.

- 3S-Smart Software Solutions CoDeSys Version 3.3
- 3S-Smart Software Solutions CoDeSys Version 3.5
- B&R Automation Studio IDE
- Beckhoff TwinCAT 2.11, 3
- Generic
- PLCopen
- Rockwell Automation Studio 5000 Logix Designer (both routine and AOI constructs)
- PHOENIX CONTACT (previously KW) Software MULTIPROG 5.0 or 5.50 (English)
- Phoenix Contact PC WORX 6.0 (English)
- Rockwell Automation RSLogix 5000 Series Version 17, 18, 19 (English)

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

Possible Automatic Import Issues

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

- Check that the coder supports the target IDE version and language setting combination.
- Check that you have specified the target IDE path in the subsystem Configuration Parameters dialog box.
- Close 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. Close other unrelated applications on the system and try the automatic import again.

Author, Manage, and Execute Simulation-Based Tests of Generated Code

Author, manage, and execute simulation-based tests of the generated code, by using Simulink Test™ with Simulink PLC Coder.

1. If you do not have the `plcdemo_simple_subsystem` model open, open it.
2. Create a signal build test harness for the subsystem. To create a test harness for a subsystem, select the subsystem, right-click, and from the context menu, select **Test Harness > Create for <subsystem name>**. Set test harness properties through the Create Test Harness dialog box.
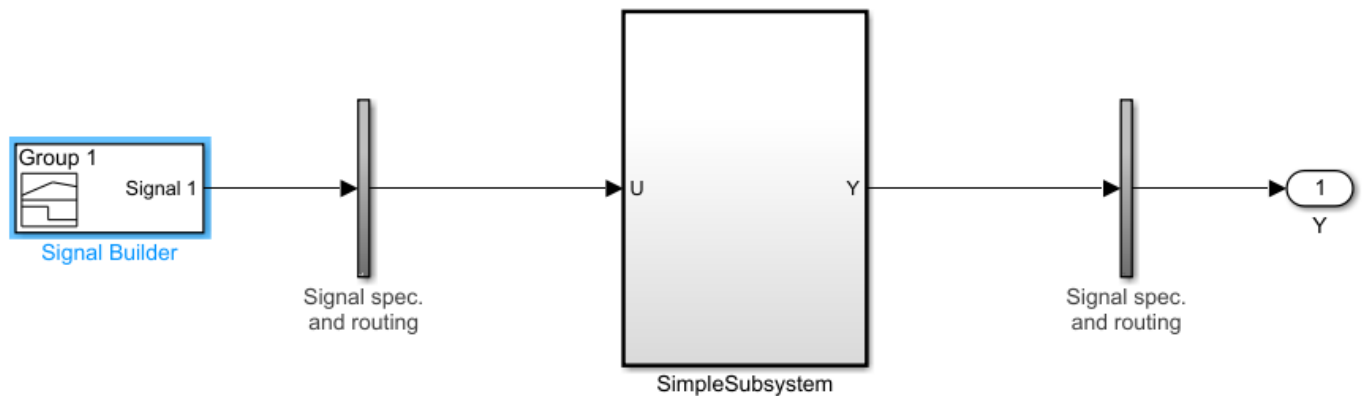
3. Open the **PLC Coder** app. Click the **PLC Code** tab.
4. Click **Settings**.
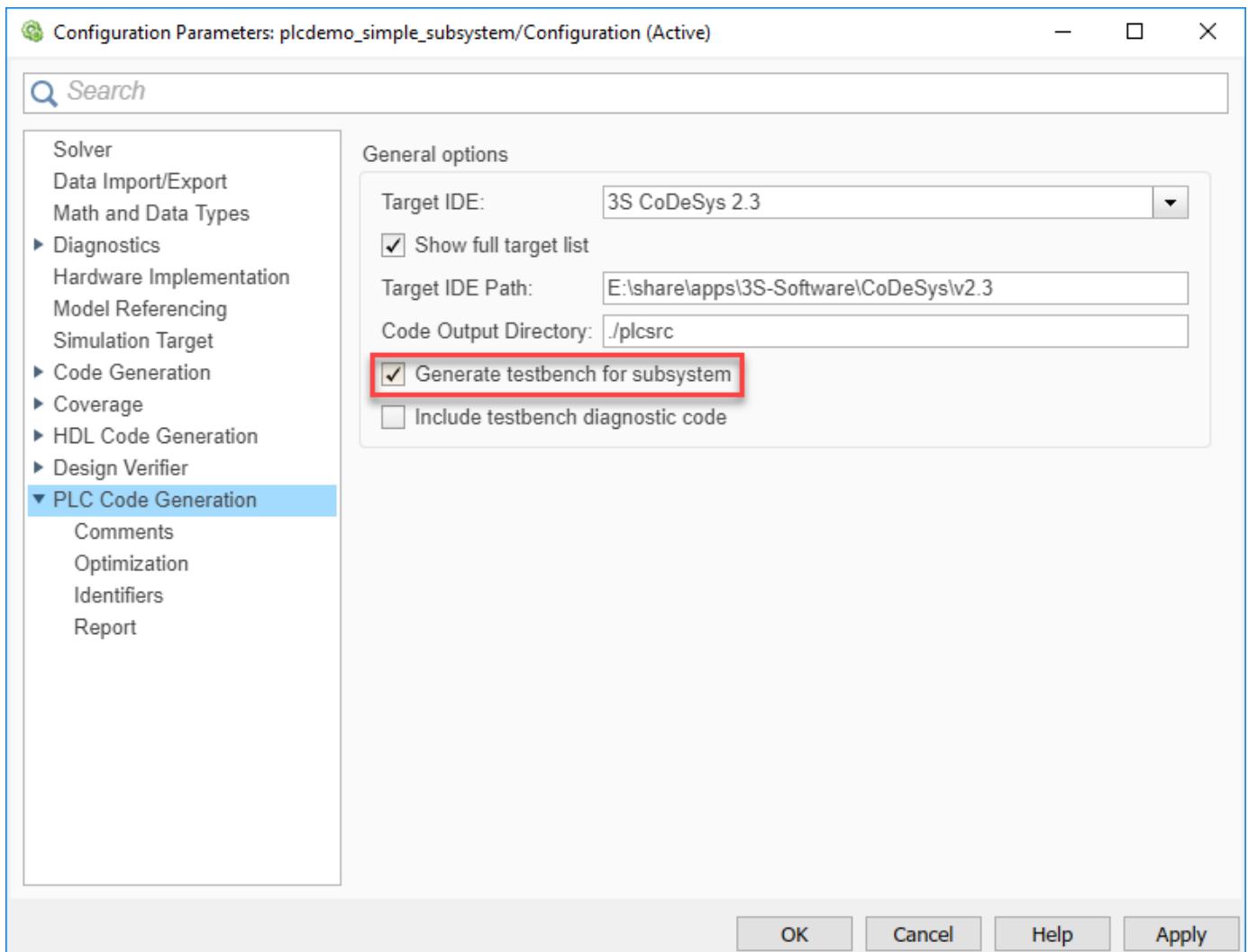
5   In the Configuration Parameters dialog box, on the **PLC Code Generation** pane, select a target and click the **Generate testbench for subsystem** check box.

6   Click **OK**.

7   Select the `Test Harness Window` subsystem, click the **PLC Code** tab and click **Generate PLC Code**. The generated code contains multiple test benches from the signal builder. You can run this code in the PLC emulator to make sure it matches your model simulation.

Limitations

- If you use anything other than a signal builder block in the test harness, you must create a top-level atomic subsystem in the test harness that contains the subsystem under test and the testing blocks (for example, test sequence block) and generate code for this subsystem.

- Simulink PLC Coder does not support the `verify` keyword in the test sequence block

- Simulink PLC Coder supports the `duration` keyword in the test sequence block but it requires that you run the generated code with the same sample rate as in the Simulink model.

Simulation and Code Generation of Motion Instructions

You can use Simulink PLC Coder software for the behavioral simulation and structured text code generation for the Rockwell Automation RSLogix motion control instructions.

Workflow for Using Motion Instructions in Model

This workflow uses the "Simulate and Generate Structured Text Code for Rockwell Automation Motion Instructions" on page 25-46 example in the `plccoderdemos` folder. This example provides a template that you can use with motion instructions. It contains the files listed in this table:

Name	Description
`MotionControllerExample.slx`	Simulink model containing an example Stateflow chart for modeling motion instructions.
`DriveLibrary.slx`	Simulink library with a Stateflow chart that is used for modeling a real world drive (axis) with trajectories, delays, and other parameters.
`MotionTypesForSim.mat`	MAT-file containing the bus data types for the `AXIS_SERVO_DRIVE` and `MOTION_INSTRUCTION`. The `MotioncontrollerExample.slx` model loads the content of the MAT-file into the workspace. If you are creating a model you, must load this MAT-file for simulation and code generation.
`Trajectory.m`	MATLAB class file for implementing trapezoidal velocity profile. This file is used to simulate the behavior of the `Motion Axis Move (MAM)` command.
`MotionApiStubs.slx`	Supporting file for code generation.
`MotionInstructionType.m`	MATLAB enumeration class file that represents the type of motion API calls. For example, `isMAM, isMSF`. This file is used only during simulation.
`plc_keyword_hook.m`	Helper file to avoid name mangling and reserved keyword limitations.
`plcgeneratemotionapicode.p`	Function that transforms the chart in the model to make it suitable for code generation.

Before you start, copy the files in the example to the current working folder.

1. Create a Simulink model with a Stateflow chart.
2. Load the bus data types from the `MotionTypesForSim.mat` file into the workspace by using the `load` function.
3. Create data that represents the drive and motion instructions for the chart. For information on adding data to Stateflow charts, see "Add Stateflow Data" (Stateflow)
4. Copy the drive(axis) model from the `DriveLibrary.slx` file into the Stateflow chart. You must copy the drive model as an atomic subchart.
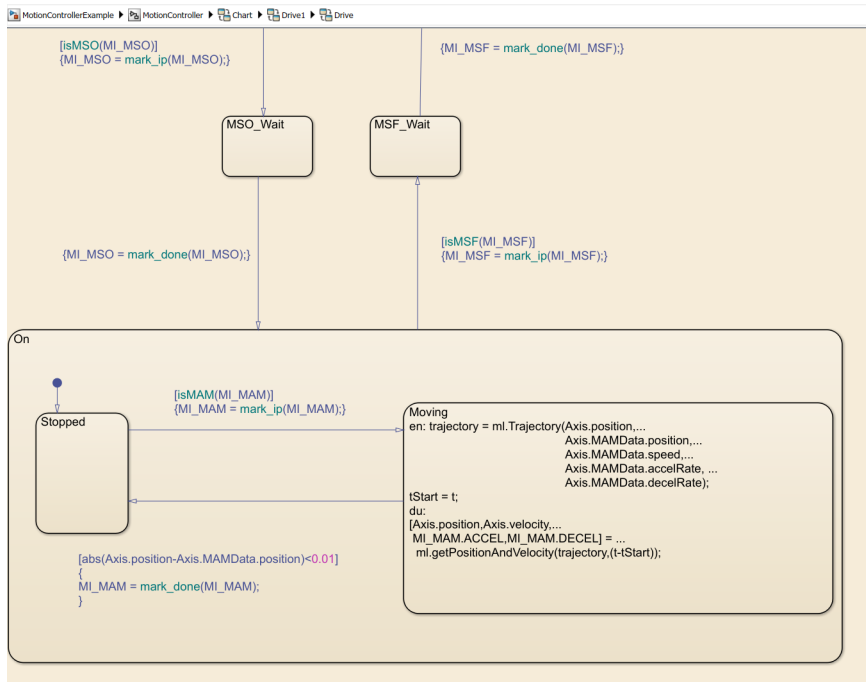
The drive logic Stateflow chart models a real world drive with parameters such as trajectory and delay. Any drive subchart has this data:

5   In the `Subchart Mappings` dialog box ,to map the drive subchart data store memory data with the local data of the appropriate names in the container chart. For more information, see "Map Variables for Atomic Subcharts and Boxes" (Stateflow).

6   Use *graphical functions* to create motion API instructions. For example, for the `Motion Servo On (MSO)` instruction:

The mapping between the inputs to the outputs is through pass by reference.

7 Create the controller logic in another subchart and use the motion instructions created in the previous step in the chart. In the example, `Controller1` has this Stateflow chart.
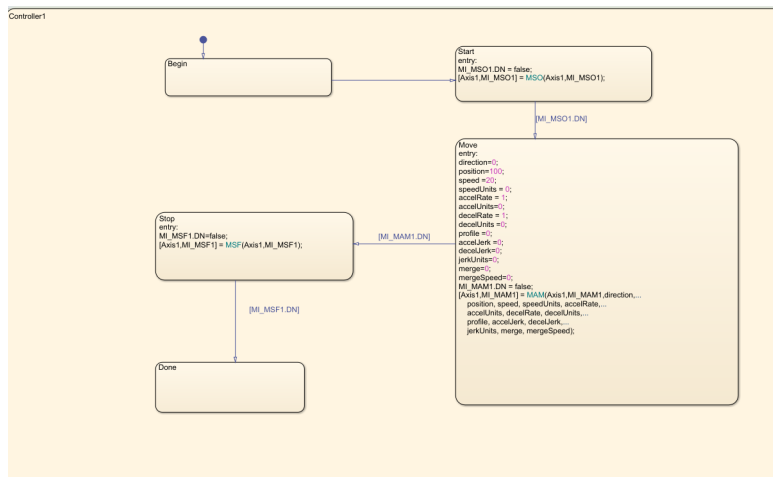
Simulation of Motion API Model

You can run simulation on the model containing the motion instructions and see the state changes in the controller chart and the `Drive` subchart. You can also log the local data of the chart, such as `AXIS` and the `MOTION_INSTRUCTION` variables. For more information, see "Configure States and Data for Logging" (Stateflow).

At the end of simulation, the logged signals are captured in the base workspace as a variable called `logsout`. You can import the logged signals into Simulation Data Inspector. For more information, see "View Data in the Simulation Data Inspector".
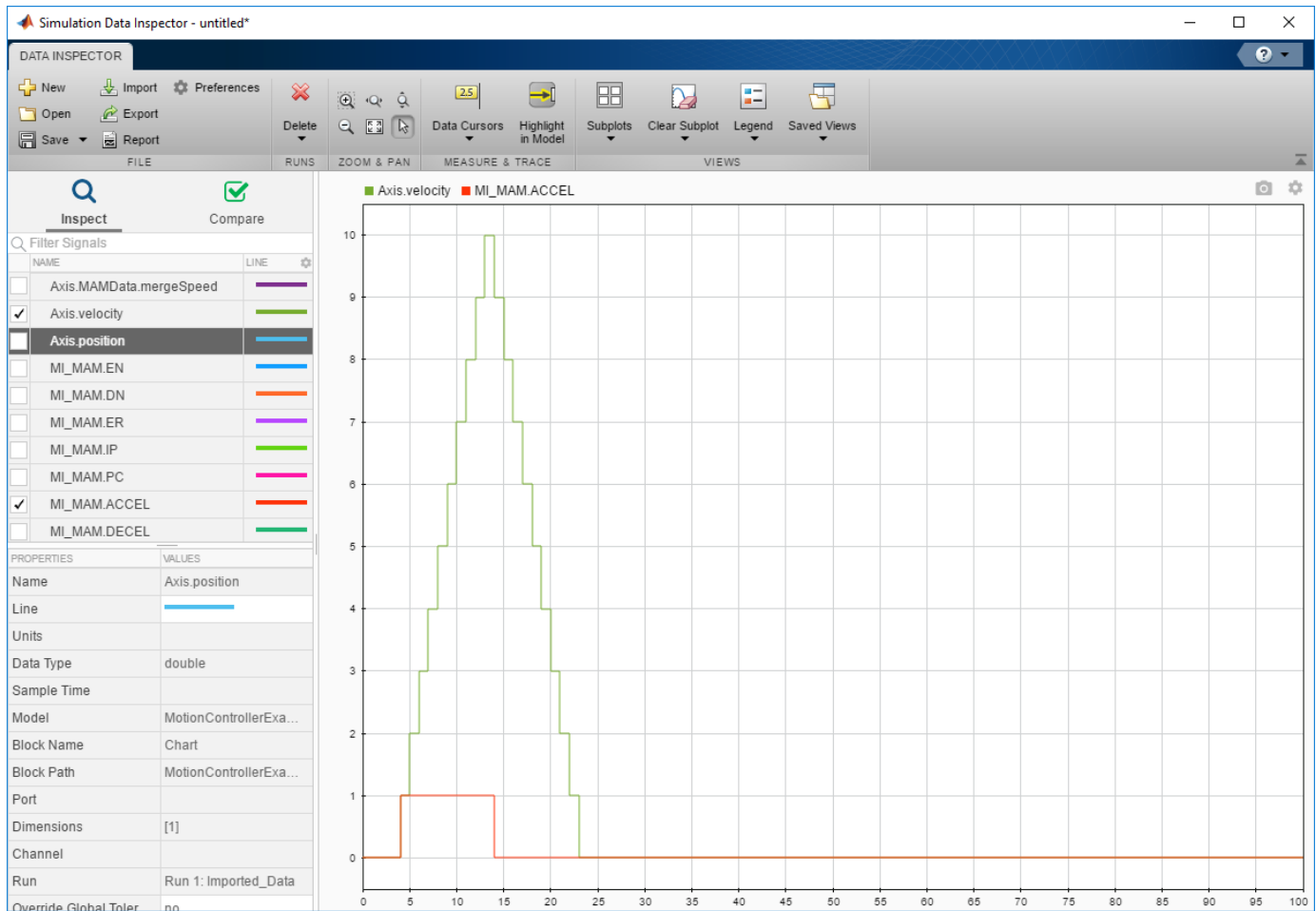
Structured Text Code Generation

To prepare the model for code generation and generate structured text code, use the `plcgeneratemotionapicode` function. The `plcgeneratemotionapicode` takes the full path name of subsystem containing the original chart as an input and creates a model from which you can generate structured text code.

Add Support for Other Motion Instructions

The `plcdemo_motion_api_rockwell` example supports only these motion instructions:

- MAM
- MAS
- MSF
- MSO

To use other Rockwell Automation RSLogix motion instructions in the model (for example, `Motion Axis Jog (MAJ)`,`:`

1.   Because the `MAJ` instruction is similar to MAM instruction, create a bus for `MAJ` with elements similar to that of MAM. See **Type Editor**.

    Name
    - ⟩ ☰ AXIS_SERVO_DRIVE
    - ⌄ ☰ MAMDATA
        - — lockPosition
        - — lockDirection
        - — eventDistance
        - — calculatedDistance
        - — direction
        - — position
        - — speed
        - — speedUnits
        - — accelRate
        - — accelUnits
        - — decelRate
        - — decelUnits
        - — profile
        - — accelJerk
        - — decelJerk
        - — jerkUnits
        - — merge
        - — mergeSpeed
    - ⟩ ☰ MOTION_INSTRUCTION

2.   Update the `MotionTypesForSim.mat` file with the new definitions for `MAJDATA` and `AXIS_SERVO_DRIVE`.

3.   In the Stateflow chart, create a graphical function representing `MAJ` (similar to MAM). Assign the appropriate inputs and outputs.

4.   Create a single transition with commands to set the output values.

```
function [AxisTagOut,MITagOut] = MAM(AxisTag,MITag,directionIn, positionIn, speedIn, speedUnitsIn, ...
                accelRateIn, accelUnitsIn, ...
                decelRateIn, decelUnitsIn, ...
                profileIn, accelJerkIn, decelJerkIn, jerkUnitsIn, ...
                mergeIn, mergeSpeedIn,  ...
        lockPositionIn, lockDirectionIn, ...
     eventDistanceIn, calculatedDataIn)

                                    {
                                    AxisTagOut = AxisTag;
                                    MITagOut = MITag;
                                    AxisTagOut.currentInstruction = MotionInstructionType.isMAM;
                                    AxisTagOut.MAMData.direction = directionIn;
                                    AxisTagOut.MAMData.position = positionIn;
                                    AxisTagOut.MAMData.speed = speedIn;
                                    AxisTagOut.MAMData.speedUnits = speedUnitsIn;
                                    AxisTagOut.MAMData.accelRate = accelRateIn;
                                    AxisTagOut.MAMData.accelUnits = accelUnitsIn;
                                    AxisTagOut.MAMData.decelRate = decelRateIn;
                                    AxisTagOut.MAMData.decelUnits = decelUnitsIn;
                                    AxisTagOut.MAMData.profile = profileIn;
                                    AxisTagOut.MAMData.accelJerk = accelJerkIn;
                                    AxisTagOut.MAMData.decelJerk = decelJerkIn;
                                    AxisTagOut.MAMData.jerkUnits = jerkUnitsIn;
                                    AxisTagOut.MAMData.merge = mergeIn;
                                    AxisTagOut.MAMData.mergeSpeed = mergeSpeedIn;
                                    AxisTagOut.MAMData.lockPosition = lockPositionIn;
                                    AxisTagOut.MAMData.lockDirection = lockDirectionIn;
                                    AxisTagOut.MAMData.eventDistance = eventDistanceIn;
                                    AxisTagOut.MAMData.calculatedData = calculatedDataIn;
                                    MITagOut.EN = true;
                                    MITagOut.IP = false;
                                    MITagOut.DN = false;
                                    }
```

5   Remove the transition commands and copy the graphical function to the `MotionApiStubs.slx`.
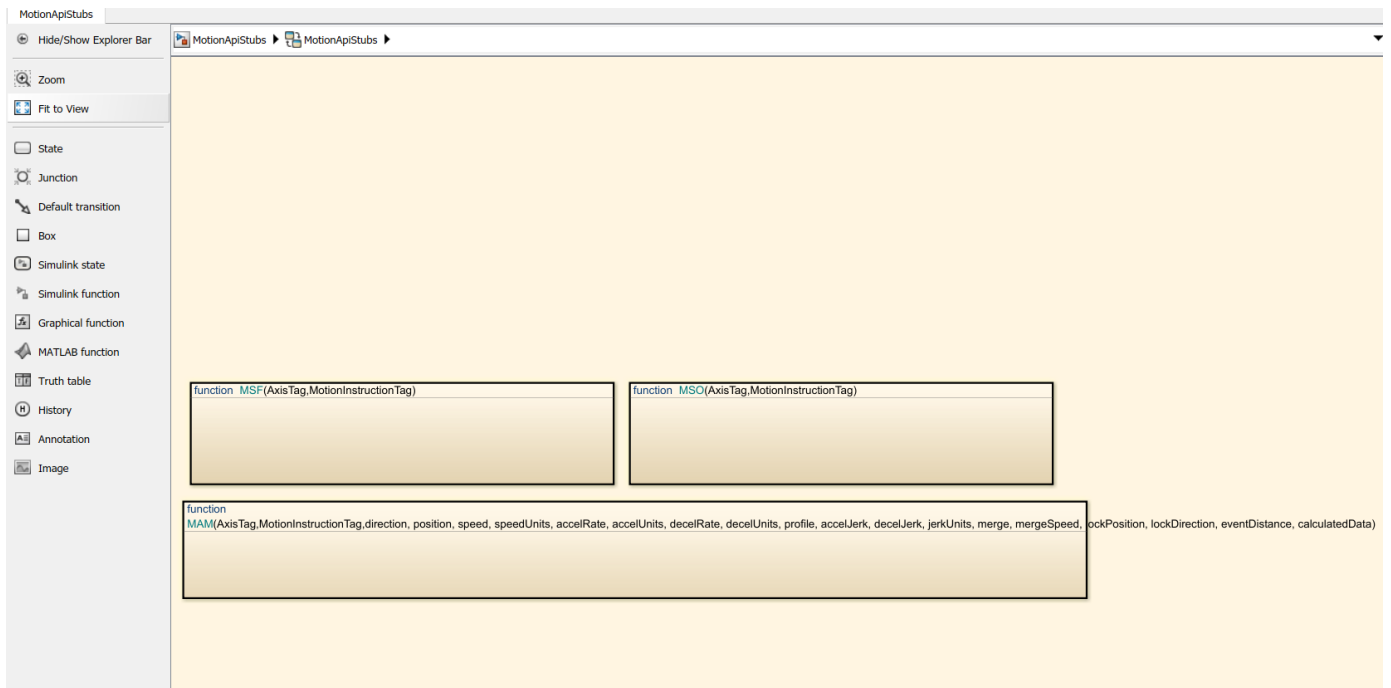
6   Update the `functionName` variable in the `getDriveTemplateNames.m` file to include `MAJ`.

```
Editor - Y:\17\amathewi.ladderFB\plcdemo_motion_api_rockwell\getDriveTemplateNames.m
  getDriveTemplateNames.m    +
11      %          delete them before ST generation. However, these are req
12      %          simulation
13      %
14      %          functionNames : names of the motion api function calls. '
15      %          motion api calls are needed for simulation and also need
16      %          appear in the ST code. However,
17      %              a) these should appear as function calls just as
18      %                 represented in the controller. They should not get
19      %                 b) the definition should not get generated. So we
20      %                 to the plc options symbols that are to not to be g
21      %
22      %          globalDataNames  :  names of the motionAPI global data s
23      %          which behind hte scenes are used to updated the status o
24      %          instruction calls both in the controller and the drive. '
25      %          data store memories in the stateflow chart. The type
26      %          definition for these should not be generated in the ST a
27      %          will be provided by RSLogix. So we add these to the plc
28    - %          symbols that are to not to be generated.
29
30 -        driveNames = {'Drive1', 'Drive2', 'DriveModel'};
31 -        dummyStateNames = {'DummyLogger'};
32 -        functionNames = {'MSF', 'MSO', 'MAM'};
33 -        globalDataNames = {'AXIS_SERVO_DRIVE', 'MOTION_INSTRUCTION', '
34
35 -    end
```

7 Update the `DriveLibrary.slx` file to respond to `MAJ` calls during simulation.

- Create a `isMAJ` graphical function (similar to `isMAM`).
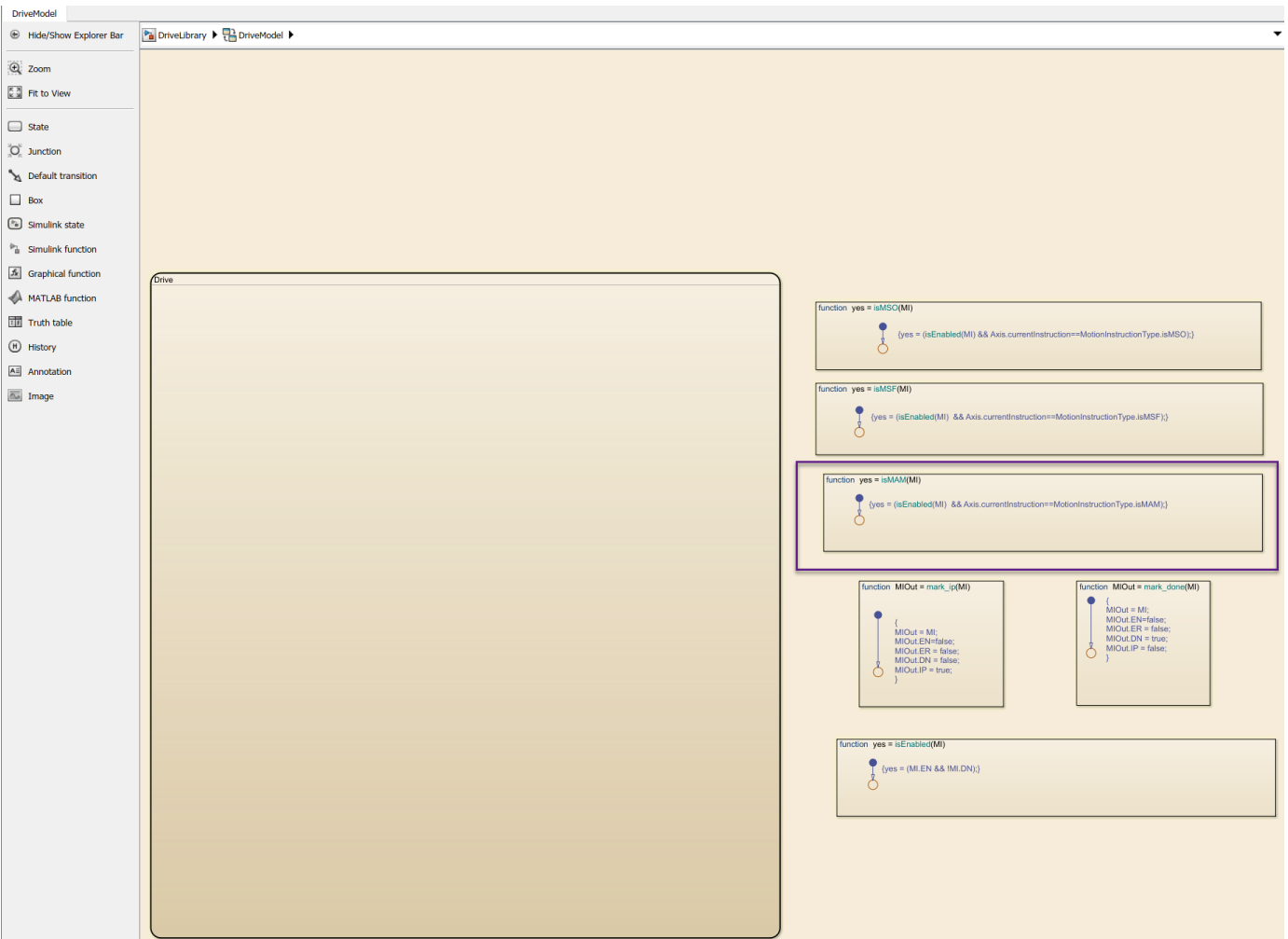
- Update the `Drive` subchart to respond to `MAJ` by implementing required transitions etc (similar to `MAM` as shown).
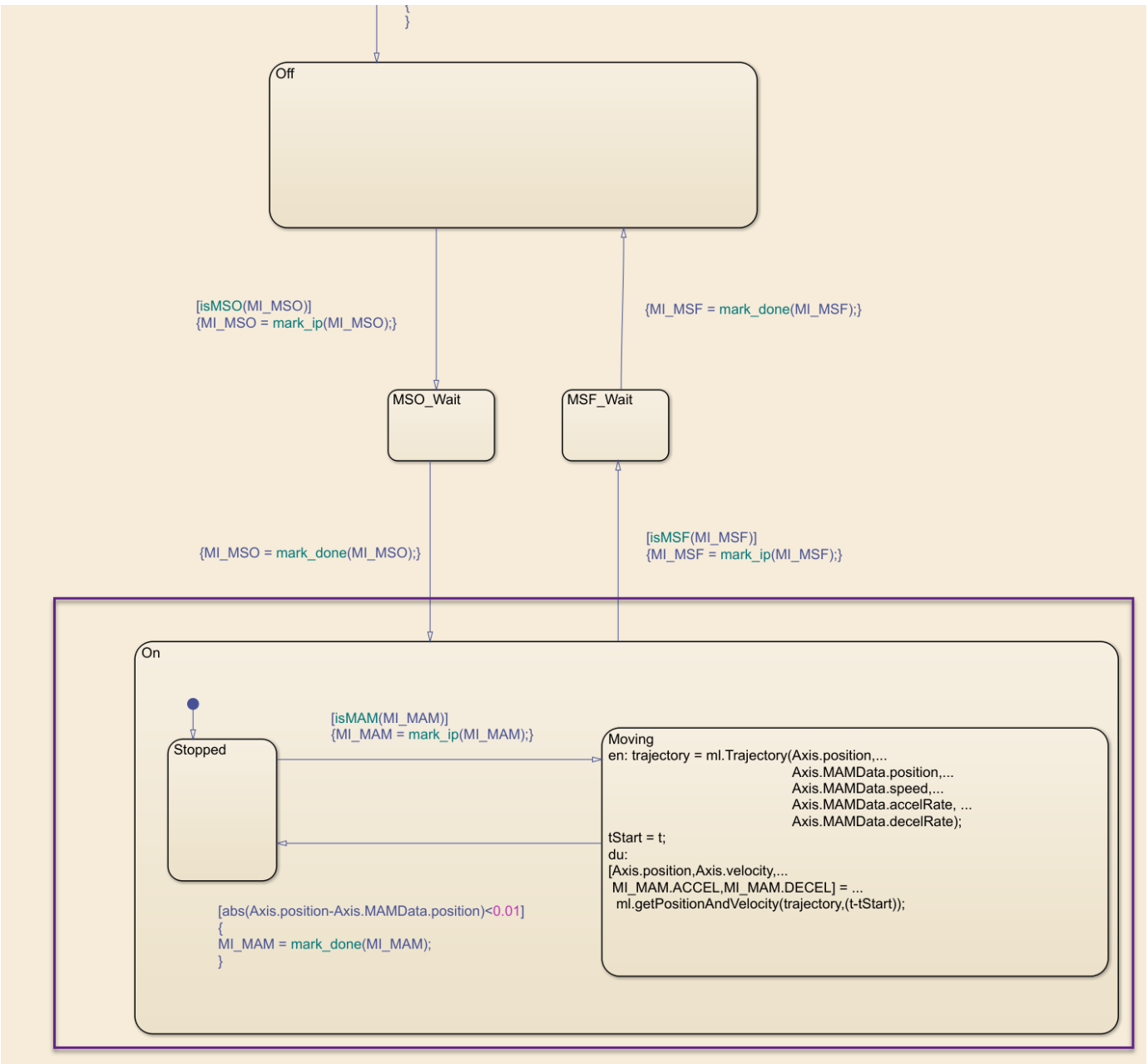
8 Create or update the controller logic as required. Create a new state and add `MAJ` instruction to it (similar to the `MAM`)
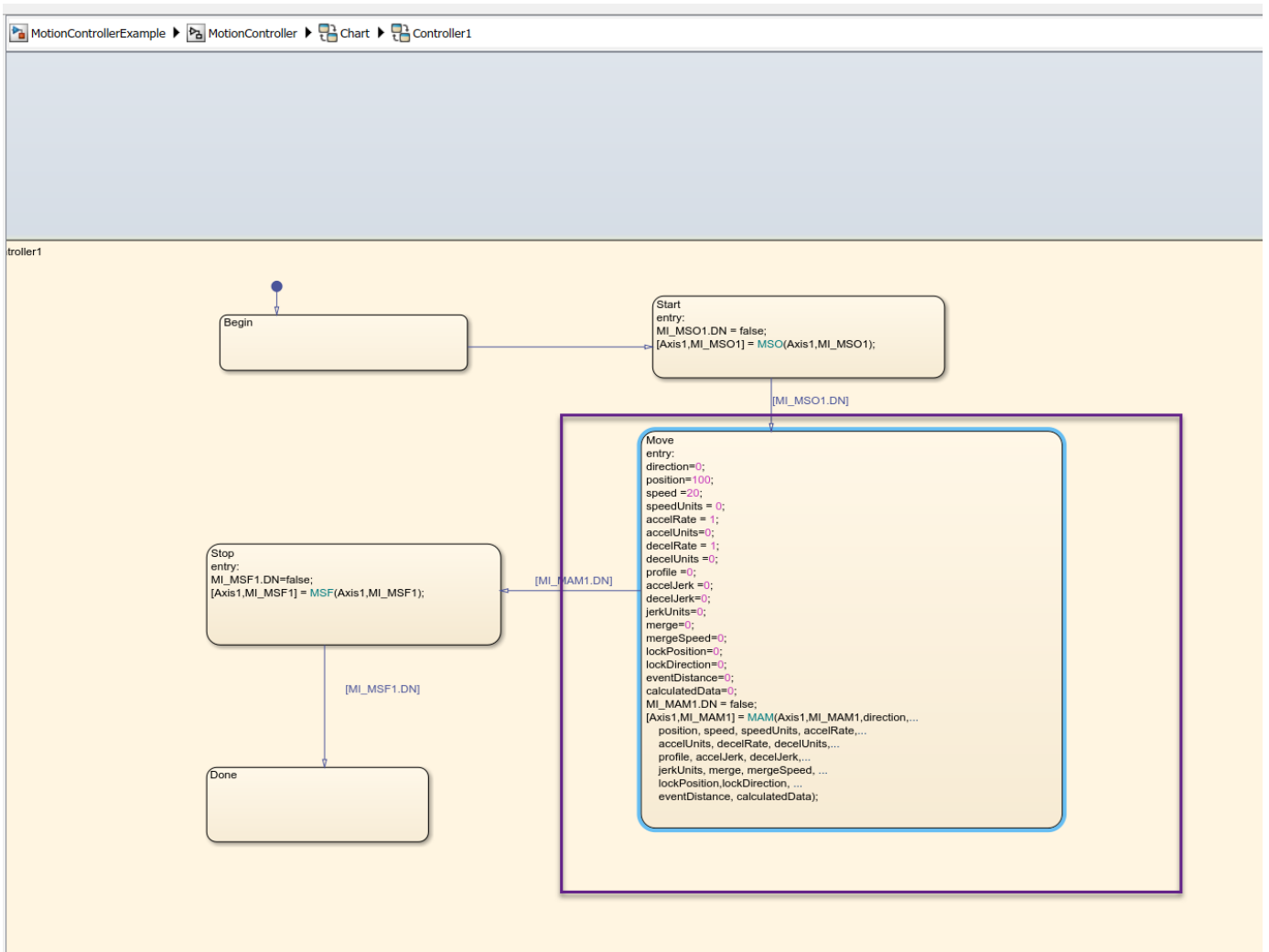
9 Perform simulation and generate code using the steps described earlier.

2

Mapping Simulink Semantics to Structured Text

Generated Code Structure for Simple Simulink Subsystems

This topic assumes that you have generated Structured Text code from a Simulink model. If you have not yet done so, see "Generate Structured Text from the Model Window" on page 1-7.

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

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

    edit plcdemo_simple_subsystem.exp

    A file like the following is displayed.

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

    Note The coder maps alias data types to the base data type in the generated code.
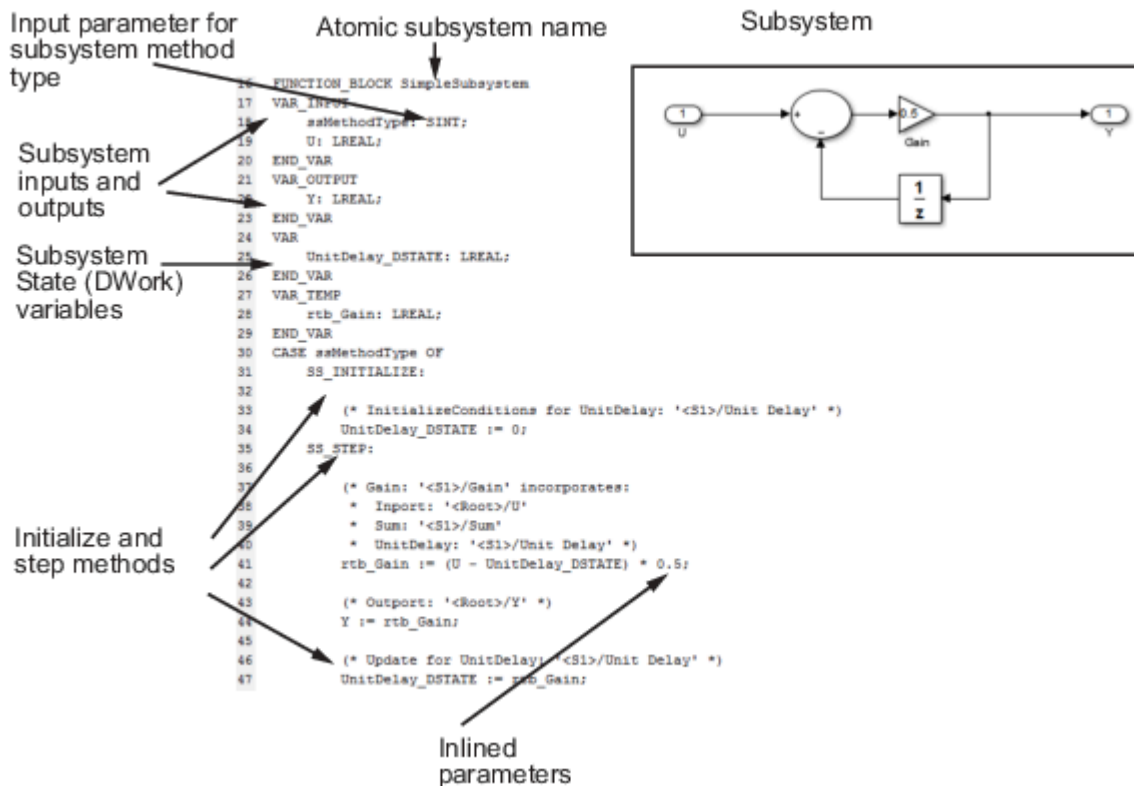
2   Inspect this code as you ordinarily do for PLC code. Check the generated code.

Note The Simulink model for `plcdemo_simple_subsystem` does not contain signal names at the input or output of the `SimpleSubsystem` block. So the generated code has the port names U and Y

as the input and output variable names of the FUNCTION_BLOCK. However, even if your model does contain signal names, coder only uses port names in the generated code.

Generated Code Structure for Reusable Subsystems

This topic assumes that you have generated Structured Text code from a Simulink model. If you have not yet done so, see "Generate Structured Text from the Model Window" on page 1-7.

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 Open the **PLC Coder** app.
3 Click **Generate PLC Code**.

The Simulink PLC Coder software generates Structured Text code and places it in *current_folder*/plcsrc/plcdemo_reusable_subsystem.exp.

4 If you do not have the `plcdemo_reusable_subsystem.exp` file open, open it in the MATLAB editor.

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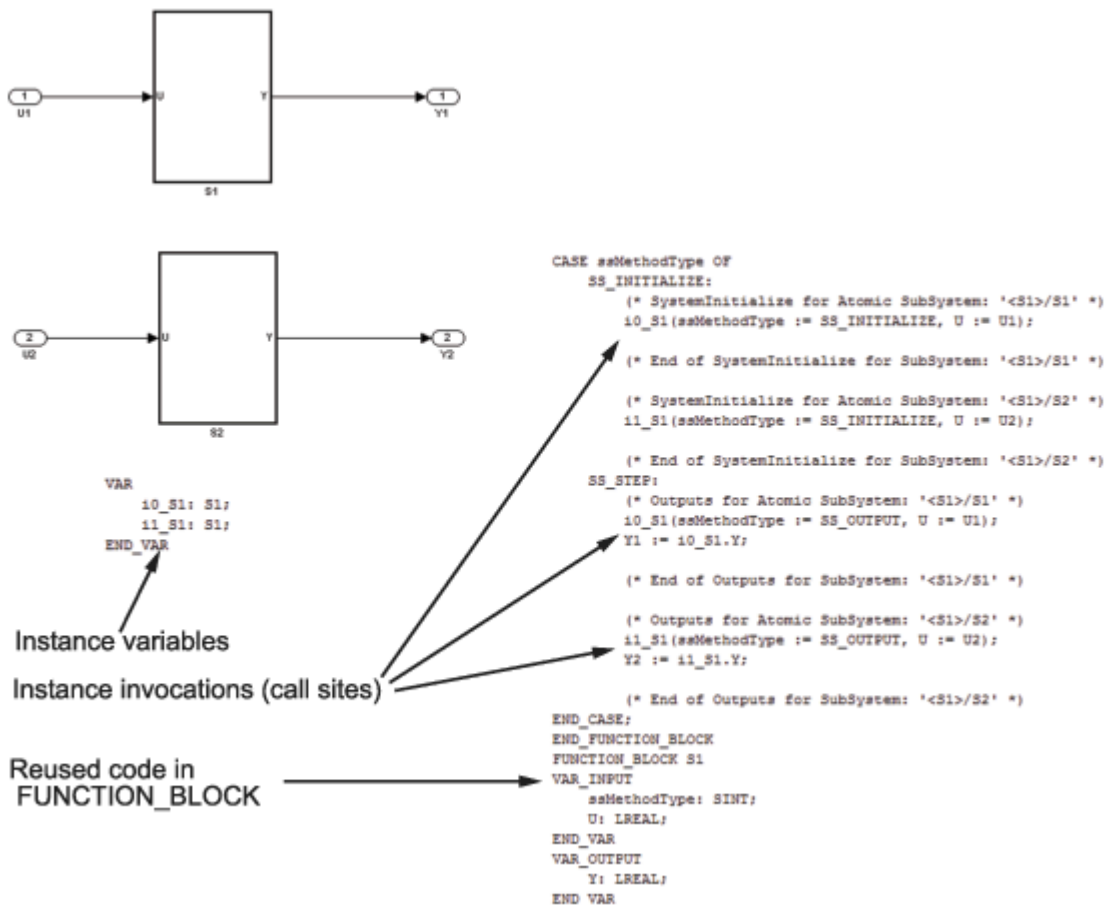
5 Examine the generated Structured Text code. The code defines `FUNCTION_BLOCK S1` once.

Look for two instance variables that correspond to the two instances declared inside the parent `FUNCTION_BLOCK ReusableSubsystem` (`i0_S1: S1` and `i1_S1: S1`). The code invokes these two instances separately by passing in different inputs. The code invokes the outputs per the Simulink execution semantics.

6 For IEC 61131-3 compatible targets, the non-step and the output `ssMethodType` do not use the output variables of the `FUNCTION_BLOCK`. Therefore, the generated Structured Text code for `SS_INITIALIZE` does not contain assignment statements for the outputs `Y1` and `Y2`.

Note This optimization is applicable only to IEC 61131-3 compatible targets.

Generated Code Structure for Triggered Subsystems

This topic assumes that you have generated Structured Text code from a Simulink model. If you have not yet done so, see "Generate Structured Text from the Model Window" on page 1-7.

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 Open the **PLC Coder** app. Click the **PLC Code** tab.

3 Click **Generate PLC Code**.

   The Simulink PLC Coder software generates Structured Text code and places it in *current_folder*`/plcsrc/plcdemo_cruise_control.exp`.

4 If you do not have the `plcdemo_cruise_control.exp` file open, open it in the MATLAB editor.

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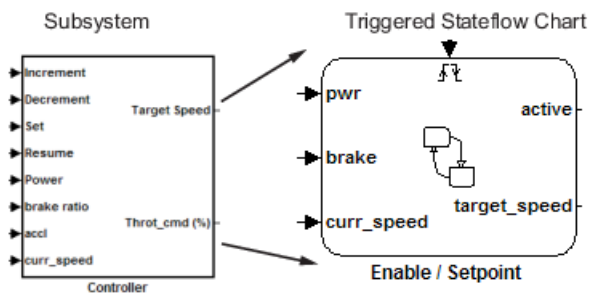
Generated code

```
    EnableSetpoint_Trig_ZCE: ARRAY [0..6] OF USINT := 3,3,3,3,3,3,3;
    i0_ZCFCN_d_ANY: ZCFCN_d_ANY;
END_VAR
...
...
...
    SS_STEP:

        (* DiscretePulseGenerator: '<S1>/Pulse Generator' *)
        IF (clockTickCounter < 1) AND (clockTickCounter >= 0) THEN
            temp1 := 1.0;
        ELSE
            temp1 := 0.0;
        END_IF;
        rtb_PulseGenerator := temp1;
        IF clockTickCounter >= 1 THEN
            clockTickCounter := 0;
        ELSE
            clockTickCounter := clockTickCounter + 1;
        END_IF;
        (* End of DiscretePulseGenerator: '<S1>/Pulse Generator' *)

        (* Chart: '<S1>/Enable // Setpoint ' incorporates:
         *  TriggerPort: '<S2>/ input events ' *)
        tempInputSignal[0] := rtb_PulseGenerator;
        (* Inport: '<Root>/Increment' *)
        tempInputSignal[1] := Increment;
        tempInputSignal[2] := Increment;
        (* Inport: '<Root>/Decrement' *)
        tempInputSignal[3] := Decrement;
        tempInputSignal[4] := Decrement;
        (* Inport: '<Root>/Set' *)
        tempInputSignal[5] := Set;
        (* Inport: '<Root>/Resume' *)
        tempInputSignal[6] := Resume;
        (* Chart: '<S1>/Enable // Setpoint ' incorporates:
         *  TriggerPort: '<S2>/ input events ' *)
        FOR inputEventIndex := 0 TO 6 DO
            i0_ZCFCN_d_ANY(u0 := EnableSetpoint_Trig_ZCE[inputEventIndex],
            callChartStep := i0_ZCFCN_d_ANY.y0;
            tmp := i0_ZCFCN_d_ANY.y1;
            tempOutEvent[inputEventIndex] := callChartStep;
            outState[inputEventIndex] := tmp;
...
...
...
FUNCTION_BLOCK ZCFCN_d_ANY
...
...
...
END_FUNCTION_BLOCK
```

Triggered subsystem semantics

Generated Code Structure for Stateflow Charts

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

Stateflow Chart with Event Based Transitions

Generate code for the Stateflow chart `ControlModule` in the model `plcdemo_stateflow_controller`. Here is the chart:
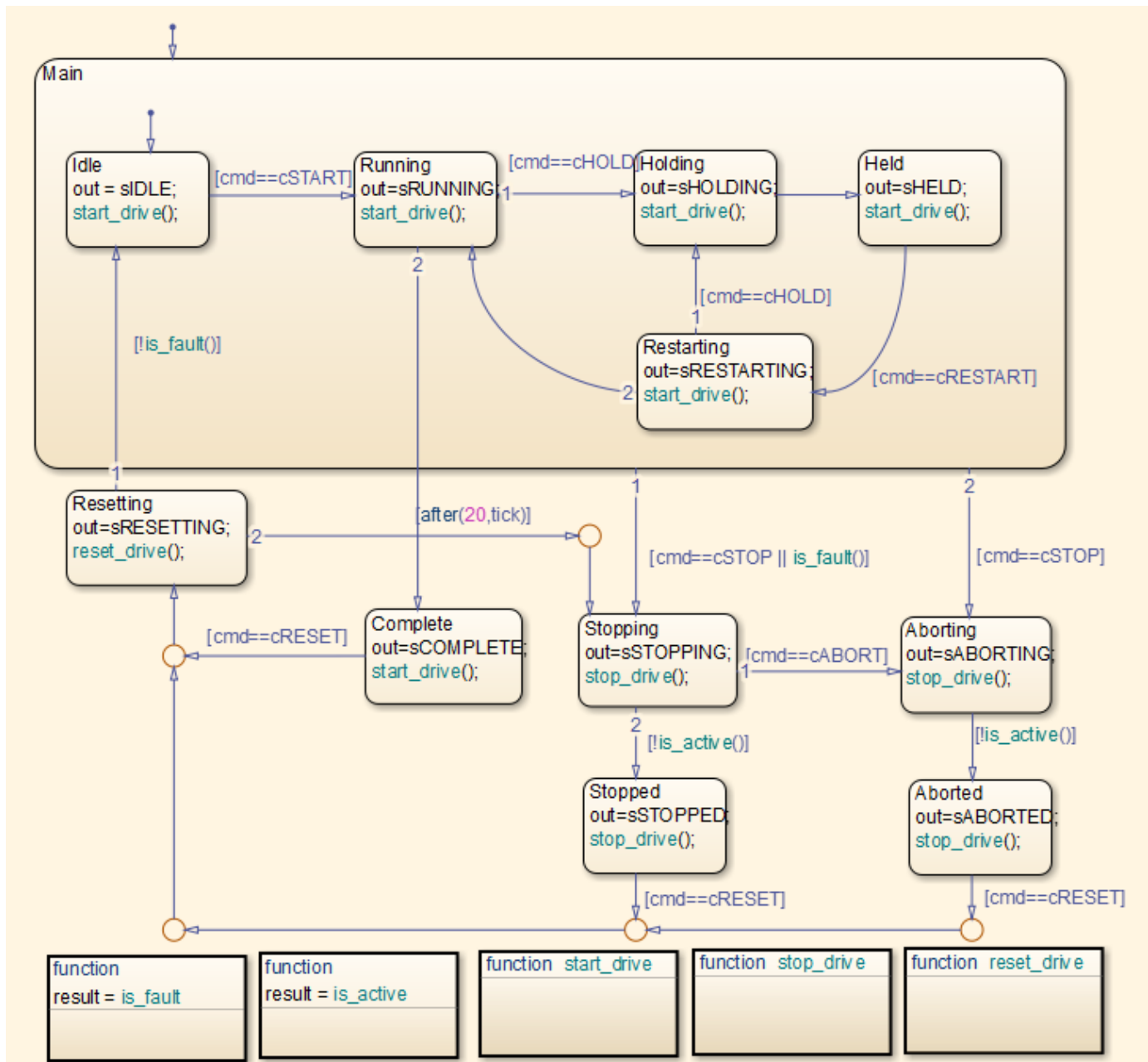
You can map the states and transitions in the chart to the generated code. For instance, the transition from the state `Aborting` to `Aborted` appears in the generated code as:

```
ControlModule_IN_Aborting:
                    rtb_out := sABORTING;
                    (* During 'Aborting': '<S1>:11' *)
                    (* Graphical Function 'is_active': '<S1>:73' *)
                    (* Transition: '<S1>:75' *)
                    IF  NOT drive_state.Active THEN
                        (* Transition: '<S1>:31' *)
                        is_c2_ControlModule := ControlModule_IN_Aborted;
                        (* Entry 'Aborted': '<S1>:12' *)
                        rtb_out := sABORTED;
                        (* Graphical Function 'stop_drive': '<S1>:88' *)
                        (* Transition: '<S1>:90' *)
                        driveOut.Start := FALSE;
                        driveOut.Stop := TRUE;
                        driveOut.Reset := FALSE;
                    END_IF;
```

For more information on the inlining of functions such as `start_drive`, `stop_drive`, and `reset_drive` in the generated code, see "Control Code Partitions for MATLAB Functions in Stateflow Charts" on page 8-8.

Stateflow Chart with Absolute Time Temporal Logic

Generate code for the Stateflow chart `Temporal` in the model `plcdemo_sf_abs_time`. Here is the chart:
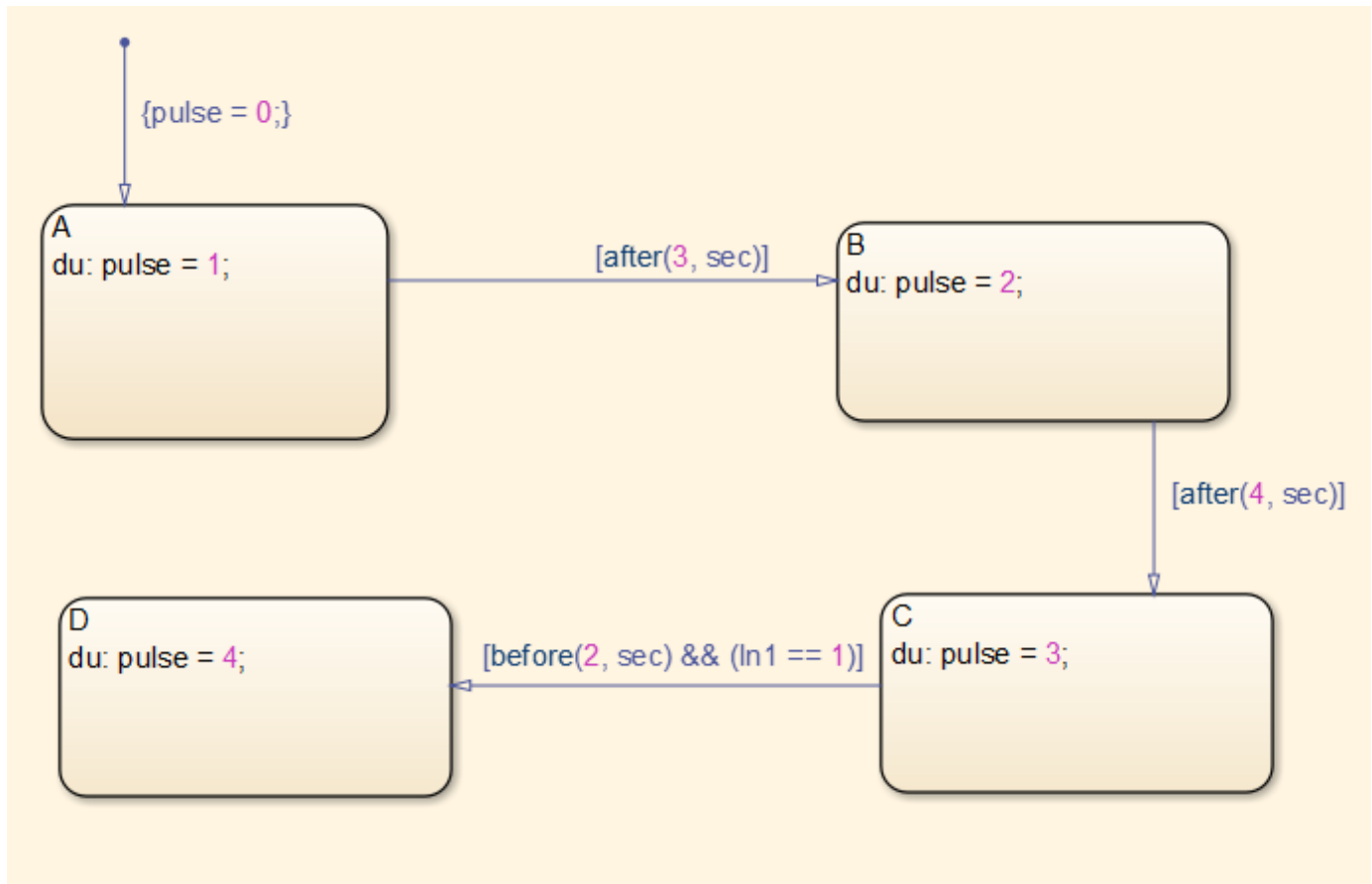
You can map states and transitions in the chart to the generated code. For instance, the transition from state B to C appears as:

```
Temporal_IN_B:
                (* During 'B': '<S1>:2' *)
                temporalCounter_i1(timerAction := 2, maxTime := 4000);
                IF temporalCounter_i1.done THEN
                    (* Transition: '<S1>:8' *)
                    is_c2_Temporal := Temporal_IN_C;
                    temporalCounter_i1(timerAction := 1, maxTime := 0);
                ELSE
                    (* Outport: '<Root>/pulse' *)
                    pulse := 2.0;
                END_IF;
```

The variable `temporalCounter_i1` is an instance of the function block `PLC_CODER_TIMER` defined as:

```
FUNCTION_BLOCK PLC_CODER_TIMER
VAR_INPUT
    timerAction: INT;
    maxTime: DINT;
END_VAR
VAR_OUTPUT
    done: BOOL;
END_VAR
VAR
    plcTimer: TON;
    plcTimerExpired: BOOL;
```

```
END_VAR
CASE timerAction OF
    1:
        (* RESET *)
        plcTimer(IN:=FALSE, PT:=T#0ms);
        plcTimerExpired := FALSE;
        done := FALSE;
    2:
        (* AFTER *)
        IF (NOT(plcTimerExpired)) THEN
            plcTimer(IN:=TRUE, PT:=DINT_TO_TIME(maxTime));
        END_IF;
        plcTimerExpired := plcTimer.Q;
        done := plcTimerExpired;
    3:
        (* BEFORE *)
        IF (NOT(plcTimerExpired)) THEN
            plcTimer(IN:=TRUE, PT:=DINT_TO_TIME(maxTime));
        END_IF;
        plcTimerExpired := plcTimer.Q;
        done := NOT(plcTimerExpired);
END_CASE;
END_FUNCTION_BLOCK
```

Generated Code Structure for MATLAB Function Block

This topic assumes that you have generated Structured Text code from a Simulink model. If you have not yet done so, see "Generate Structured Text from the Model Window" on page 1-7.

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 Open the **PLC Coder** app. Click the **PLC Code** tab.

3 Click **Generate PLC Code**.

The Simulink PLC Coder software generates Structured Text code and places it in *current_folder*/plcsrc/plcdemo_eml_tankcontrol.exp.

4 If you do not have the `plcdemo_eml_tankcontrol.exp` file open, open it in the MATLAB editor.

The following figure illustrates the mapping of the generated code to Structured Text components for a Simulink Subsystem block that contains a MATLAB Function block. The coder tries to perform inline optimization on the generated code for MATLAB local functions. If the coder determines that it is more efficient to leave the local function as is, it places the generated code in a Structured Text construct called `FUNCTION`.

5 Examine the generated Structured Text code.

```
function [InFlow, OutFlow, StirSpeed] = TankControl(Co
%#eml

    % Check the vessel state
    if(Height >= FullHeight)
        % Is it full ?
        vessel = PLCVesselState.FULL;
    elseif(Height <= EmptyHeight)
        % Is it empty ?
        vessel = PLCVesselState.EMPTIED;
    else
        vessel = PLCVesselState.NOT_FULL;
    end
```

MATLAB code

Generated code
for MATLAB
subfunctions

```
FUNCTION_BLOCK TankControl
VAR_INPUT
    Command: PLCCommandState;
    Height: LREAL;
END_VAR
VAR_OUTPUT
    InFlow: LREAL;
    OutFlow: LREAL;
    StirSpeed: LREAL;
END_VAR
VAR
END_VAR
VAR_TEMP
    vessel: PLCVesselState;
    EmptyValve: PLCValveState;
    FillValve: PLCValveState;
END_VAR
(* Check the vessel state *)
IF Height >= 10.0 THEN
    (* Is it full ? *)
    vessel := FULL;
ELSIF Height <= 2.0 THEN
    (* Is it empty ?  *)
    vessel := EMPTIED;
ELSE
    vessel := NOT_FULL;
END_IF;
(* Process the command mode *)
CASE Command OF
    FILL:
        (* Fill Tank *)
        EmptyValve := SHUT;
        IF vessel = FULL THEN
            FillValve := SHUT;
        ELSE
            FillValve := OPEN;
        END_IF;
    HOLD:
        (* Hold Contents *)
        EmptyValve := SHUT;
        FillValve := SHUT;
    EMPTY:
        (* Empty Tank *)
        FillValve := SHUT;
        IF vessel = EMPTIED THEN
            EmptyValve := SHUT;
        ELSE
            EmptyValve := OPEN;
        END_IF;
    ELSE
        EmptyValve := SHUT;
        FillValve := SHUT;
END_CASE;
(* compute inflow and outflow *)
```

Generated Code Structure for Multirate Models

This example assumes that you have generated Structured Text code from a Simulink model. If you have not yet done so, see "Generate Structured Text from the Model Window" on page 1-7.

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_multirate` model. This model has two sample rates.
2   Open the **PLC Coder** app. Click the **PLC Code** tab.
3   Click **Generate PLC Code**.

   The Simulink PLC Coder software generates Structured Text code and places it in *current_folder*/plcsrc/plcdemo_multirate.exp.

4   If you do not have the `plcdemo_multirate.exp` file open, open it in the MATLAB editor and examine the Structured Text code.

   The generated code contains a global time step counter variable:

```
VAR_GLOBAL
    plc_ts_counter1: DINT;
END_VAR
```

In this example, there are two rates, and the fast rate is twice as fast as the slow rate, so the time step counter counts to 1, then resets:

```
IF plc_ts_counter1 >= 1 THEN
     plc_ts_counter1 := 0;
ELSE
     plc_ts_counter1 := plc_ts_counter1 + 1;
END_IF;
```

The generated code for blocks running at slower rates executes conditionally based on the corresponding time step counter values. In this example, the generated code for `Gain1`, `Unit Delay1`, and `Sum1` executes every other time step, when `plc_ts_counter1 = 0`, because those blocks run at the slow rate. The generated code for `Gain`, `Unit Delay`, `Sum`, and `Sum2` executes every time step because those blocks run at the fast rate.

```
SS_STEP:

        (* Gain: '<S1>/Gain' incorporates:
         *  Inport: '<Root>/U1'
         *  Sum: '<S1>/Sum'
         *  UnitDelay: '<S1>/Unit Delay' *)
        rtb_Gain := (U1 - UnitDelay_DSTATE) * 0.5;

        (* Outport: '<Root>/Y1' *)
        Y1 := rtb_Gain;
        IF plc_ts_counter1 = 0 THEN

            (* UnitDelay: '<S1>/Unit Delay1' *)
            UnitDelay1 := UnitDelay1_DSTATE;

            (* Gain: '<S1>/Gain1' incorporates:
             *  Inport: '<Root>/U2'
```

```
     *  Sum: '<S1>/Sum1' *)
    rtb_Gain1 := (U2 - UnitDelay1) * 0.5;

    (* Outport: '<Root>/Y2' *)
    Y2 := rtb_Gain1;
END_IF;

(* Outport: '<Root>/Y3' incorporates:
 *  Sum: '<S1>/Sum2'
 *  UnitDelay: '<S1>/Unit Delay' *)
Y3 := UnitDelay_DSTATE - UnitDelay1;

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

IF plc_ts_counter1 = 0 THEN

    (* Update for UnitDelay: '<S1>/Unit Delay1' *)
    UnitDelay1_DSTATE := rtb_Gain1;

END_IF;
```

In general, for a subsystem with n different sample times, the generated code has `n-1` time step counter variables, corresponding to the `n-1` slower rates. Code generated from parts of the model running at the slower rates executes conditionally, based on the corresponding time step counter values.

Generated Code Structure for Subsystem Mask Parameters

In the generated code for masked subsystems, the mask parameters map to function block inputs. The values you specify in the subsystem mask are assigned to these function block inputs in the generated code.

For example, the following subsystem, `Subsystem`, contains two instances, `Filt1` and `Filt2`, of the same masked subsystem.
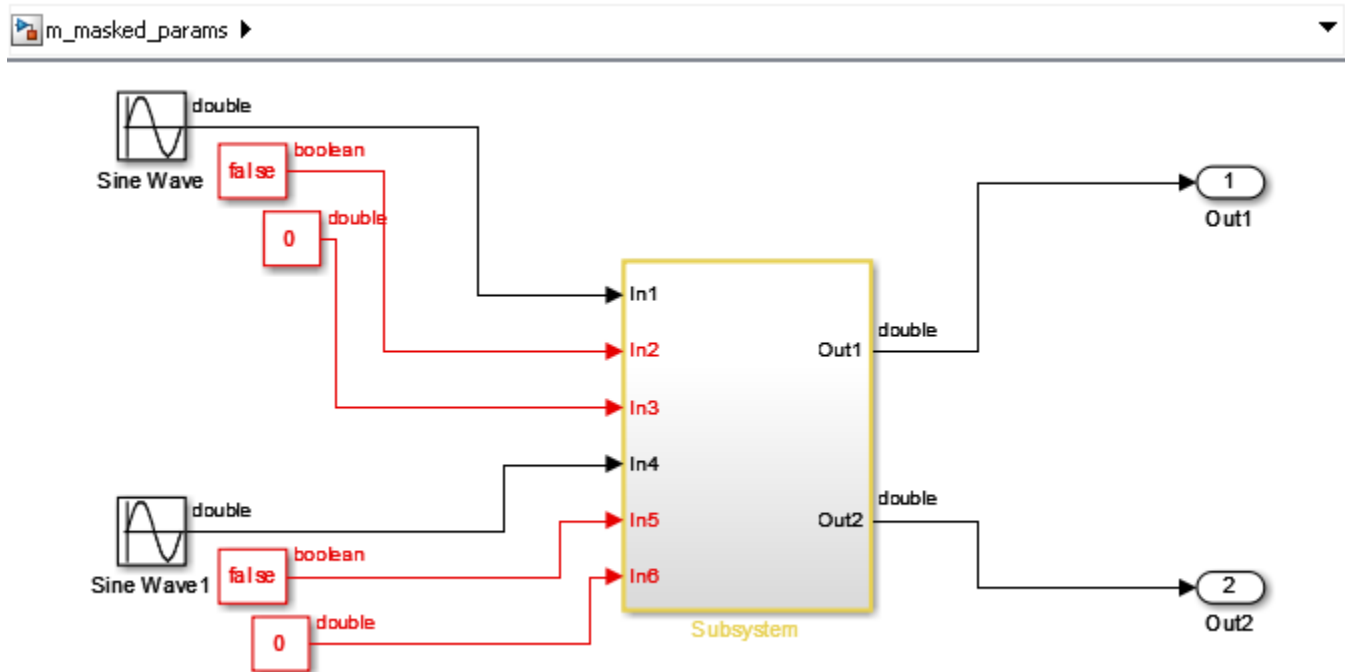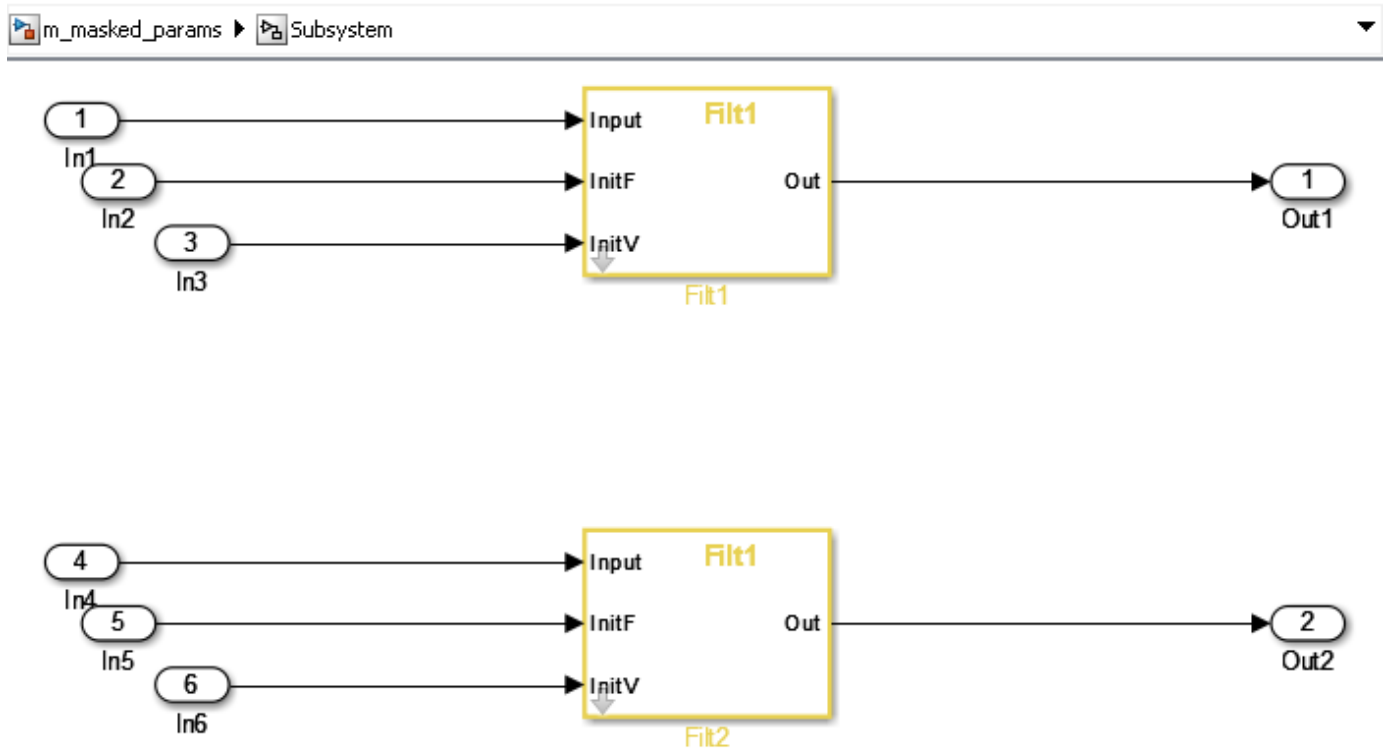
The two subsystems, `Filt1`, and `Filt2`, have different values assigned to their mask parameters. In this example, `Filt1_Order_Thau` is a constant with a value of 5.
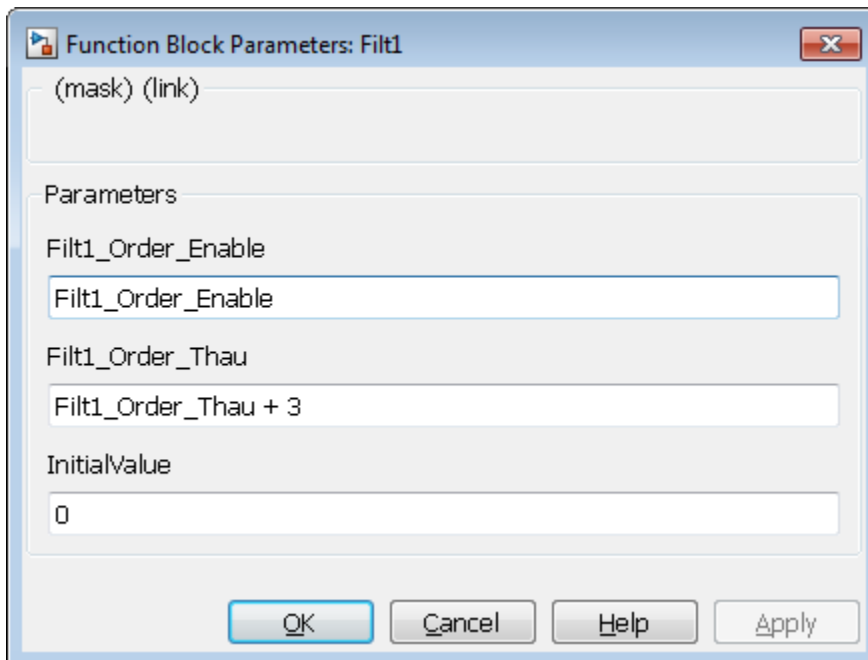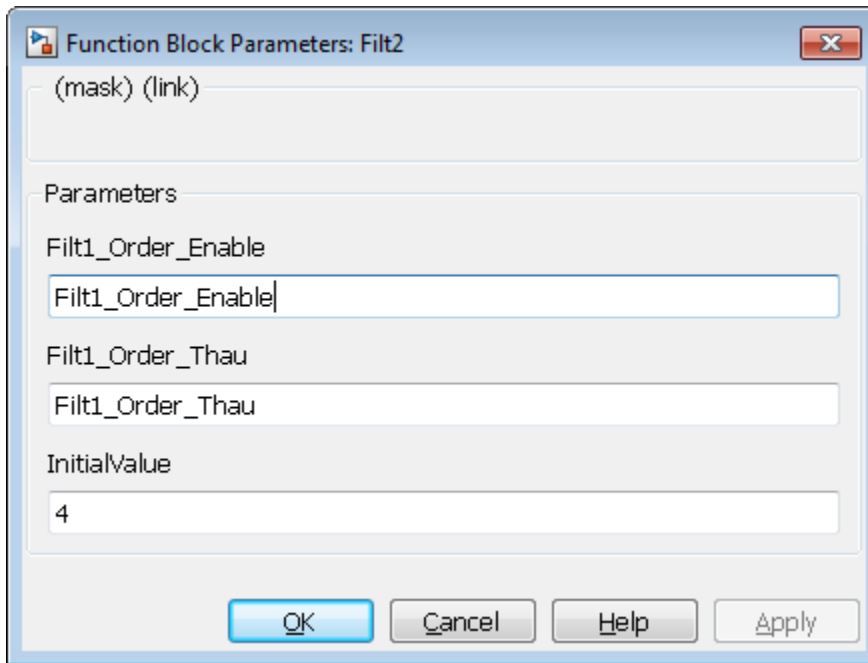
Therefore, for the `Filt1` subsystem, the `Filt1_Order_Thau` parameter has a value of 8, and for the `Filt2` subsystem, the `Filt1_Order_Thau` parameter has a value of 5.

The following generated code shows the `Filt1` function block inputs. The `rtp_Filt1_Order_Thau` input was generated for the `Filt1_Order_Thau` mask parameter.

```
FUNCTION_BLOCK Filt1
VAR_INPUT
    ssMethodType: SINT;
    InitV: LREAL;
    InitF: BOOL;
    Input: LREAL;
    rtp_Filt1_Order_Thau: LREAL;
    rtp_InitialValue: LREAL;
    rtp_Filt1_Order_Enable: BOOL;
END_VAR
```

The following generated code is from the `FUNCTION_BLOCK Subsystem`. The function block assigns a value of 8 to the `rtp_Filt1_Order_Thau` input for the `i0_Filt1` instance, and assigns a value of 5 to the `rtp_Filt1_Order_Thau` input for the `i1_Filt1` instance.

```
SS_INITIALIZE:
        (* InitializeConditions for Atomic SubSystem: '<S1>/Filt1' *)

        i0_Filt1(ssMethodType := SS_INITIALIZE, InitV := In3,
                InitF := In2, Input := In1,
                rtp_Filt1_Order_Thau := 8.0,
                rtp_InitialValue := 0.0,
                rtp_Filt1_Order_Enable := TRUE);
        Out1 := i0_Filt1.Out;

        (* End of InitializeConditions for SubSystem: '<S1>/Filt1' *)
```

```
        (* InitializeConditions for Atomic SubSystem: '<S1>/Filt2' *)
        i1_Filt1(ssMethodType := SS_INITIALIZE, InitV := In6,
                InitF := In5, Input := In4,
                rtp_Filt1_Order_Thau := 5.0,
                rtp_InitialValue := 4.0,
                rtp_Filt1_Order_Enable := TRUE);
        Out2 := i1_Filt1.Out;

        (* End of InitializeConditions for SubSystem: '<S1>/Filt2' *)
SS_STEP:
        (* Outputs for Atomic SubSystem: '<S1>/Filt1' *)

        i0_Filt1(ssMethodType := SS_OUTPUT, InitV := In3, InitF := In2,
                Input := In1, rtp_Filt1_Order_Thau := 8.0,
                rtp_InitialValue := 0.0,
                rtp_Filt1_Order_Enable := TRUE);
        Out1 := i0_Filt1.Out;

        (* End of Outputs for SubSystem: '<S1>/Filt1' *)

        (* Outputs for Atomic SubSystem: '<S1>/Filt2' *)
        i1_Filt1(ssMethodType := SS_OUTPUT, InitV := In6, InitF := In5,
                Input := In4, rtp_Filt1_Order_Thau := 5.0,
                rtp_InitialValue := 4.0,
                rtp_Filt1_Order_Enable := TRUE);
        Out2 := i1_Filt1.Out;

        (* End of Outputs for SubSystem: '<S1>/Filt2' *)
```

Global Tunable Parameter Initialization for PC WORX

For PC WORX, the coder generates an initialization function, PLC_INIT_PARAMETERS, to initialize global tunable parameters that are arrays and structures. This initialization function is called in the top-level initialization method.

For example, suppose that your model has a global array variable, `ParArrayXLUT`:

```
ParArrayXLUT=[0,2,6,10];
```

In the generated code, the PLC_INIT_PARAMETERS function contains the following code to initialize `ParArrayXLUT`:

```
(* parameter initialization function starts *)<br/>
ParArrayXLUT[0] := LREAL#0.0;<br/>
ParArrayXLUT[1] := LREAL#2.0;<br/>
ParArrayXLUT[2] := LREAL#6.0;<br/>
ParArrayXLUT[3] := LREAL#10.0;<br/>
(* parameter initialization function ends *)<br/></div></html>
```

The PLC_INIT_PARAMETERS function is renamed `i0_PLC_INIT_PARAMETERS`, and called in the top-level initialization method:

```
CASE SINT_TO_INT(ssMethodType) OF<br/>
    0: <br/>
        i0_PLC_INIT_PARAMETERS();<br/>
```

Considerations for Nonintrinsic Math Functions

When Simulink PLC Coder encounters a math function that is not intrinsic, it generates structured text by replacing the nonintrinsic function with an equivalent IEC-61131 compatible intrinsic function. For such cases, an input value that is larger than the allowed input range causes overflow and generates a NaN value.

For example, hyperbolic *tan* is not an intrinsic function. Simulink PLC Coder uses *exp* in the generated code to represent *tanh*. More specifically, it uses *(exp(2*x)-1)/(exp(2*x)+1)*. For large values of *x*, this function overflows. You can address this issue by adding validation code or, using blocks before calling the *tanh* function to check that the range of the input is within acceptable values. In MATLAB, *tanh(x)* for *x>19* is 1.0000. If *x>19*, return a value of 1.0000.

See Also

Generating Ladder Diagram

Supported Elements in Ladder Diagram

The ladder diagram import capability of Simulink PLC Coder enables you to import ladder diagrams created by using Rockwell Automation IDEs, such as RSLogix 5000 and Studio 5000, into the Simulink environment as a model.

Supported Ladder Elements

Simulink PLC Coder supports these ladder elements:

- Boolean variables
- Data access to array elements, bus elements, bit, and constant variables.
- Multiple rungs
- Simple `Jump`, `Temporary End`, and other supported execution control elements.
- Ladder diagram blocks. See `plcladderlib`.
- Ladder Diagram Instructions. See "Instructions Supported in Ladder Diagram" on page 15-2
- The L5X data types listed in this table:

L5X Data Types	Simulink Types
BOOL	Boolean datatype
SINT	Int8 datatype
INT	Int16 datatype
DINT	Int32 datatype
REAL	Single datatype
TIMER	Timer bustype
COUNTER	Counter bustype
CONTROL	Control bustype
UDT	UDT bustype
AOI	AOI bustype

- Supported ladder diagram tags:
  - Controller
  - Program
  - AOI Tags, such as Input, Output, and InOut

See Also
`plcimportladder` | `plcgeneraterunnertb` | `plcgeneratecode` | `plcladderlib` | `plcladderoption` | `plcloadtypes` | `plccleartypes`

More About
- "Import L5X Ladder Diagram Files into Simulink" on page 3-4
- "Model and Simulate Ladder Diagrams in Simulink" on page 3-8

- "Generating Ladder Diagram Code from Simulink" on page 3-13
- "Generating C Code from Simulink Ladder" on page 3-15
- "Verify Generated Ladder Diagram Code" on page 3-17
- "Simulink PLC Coder Workflow vs. Rockwell Automation RSLogix IDE Workflow" on page 3-21

Import L5X Ladder Diagram Files into Simulink

Simulate, test, and validate your .L5X ladder diagram files by importing your ladder diagram files into Simulink®. Use the `plcimportladder` function to import your ladder diagram files into Simulink. Simulink PLC Coder™ supports only import of ladder diagram files created by using Rockwell Automation® RSLogix 5000® and Studio 5000 ® integrated development environments (IDEs).

Ladder Diagram Description

The ladder diagram in the `simple_timer.L5X` file controls a motor by using an input switch (`Switch_A`) and a timer (`Motor_timer`). This ladder diagram was created using the Studio 5000 IDE.
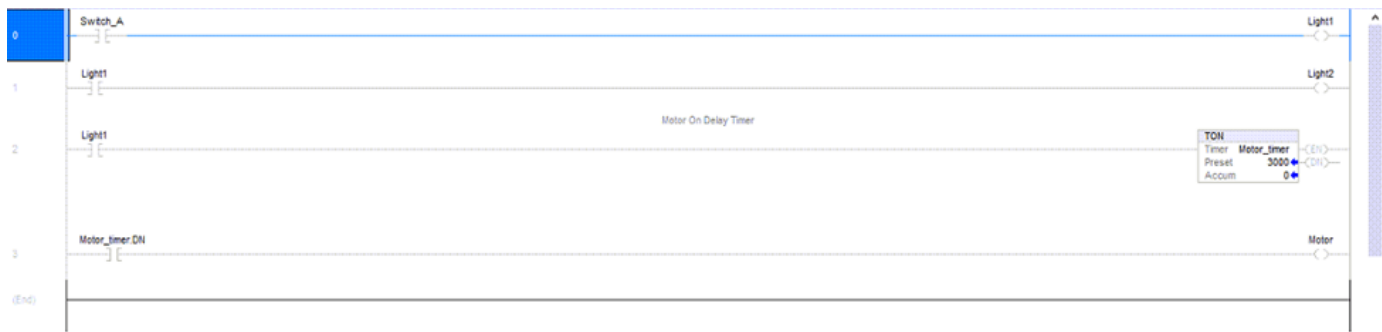
`Light1, Light2, and Motor` are the outputs of this ladder diagram.

Import Ladder Diagram

Before using the `plcimportladder` function to import your ladder diagram files into Simulink:

- Verify that your .L5X ladder diagram file has no errors by compiling the file in the Rockwell Automation IDE.
- Verify that the .L5X ladder diagram file uses blocks that are supported by Simulink PLC Coder. For a list of supported blocks, see Simulink PLC Coder Ladder Diagram Blocks. If your ladder diagram contains custom instructions that are not supported use the `Custom Instruction` block to create your instructions in Simulink. For more information, see Custom Instruction. To create a custom instruction, see "Create Custom Instruction in PLC Ladder Diagram Models" on page 3-23.

To import the `simple_timer.L5X` ladder diagram file into Simulink, use the `plcimportladder` function.

```
plcimportladder('simple_timer.L5X');
```

The ladder diagram is imported into Simulink and a `simple_timer.slx` file is created. The current folder also contains a `simple_timer_value.mat` file that loads the initial values for `Motor_timer` into the model data store memory. The data store memory also contains state information of elements of the ladder diagram. This state information is updated by the model during simulation.

During the ladder diagram import, Simulink PLC Coder:

- Imports rung comments. For example, rung two of `simple_timer.L5X` has the comment `Motor On Delay Timer.` This comment also appears in the Simulink model as well.
- Imports Add On Instruction (AOI) with mixed-order arguments, while preserving the order of the arguments. This order argument is preserved during ladder diagram code generation as well.

Imported Ladder Diagram Structure

The `simple_timer.L5X ladder diagram file is located in` Controller simple_timer > MainTask > MainProgram > MainRoutine.
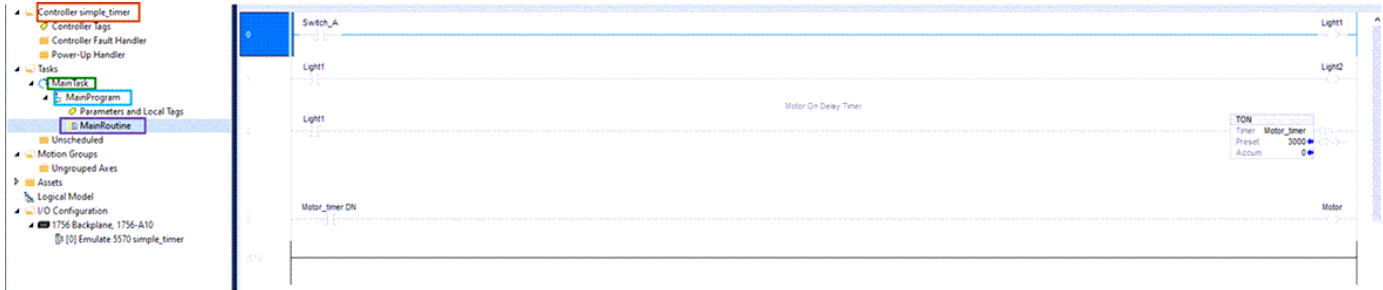
The `simple_timer.slx` ladder diagram is located in `simple_timer > MainTask > MainProgram > _Logic.` This structure is similar to the structure in the Rockwell Automation IDE.
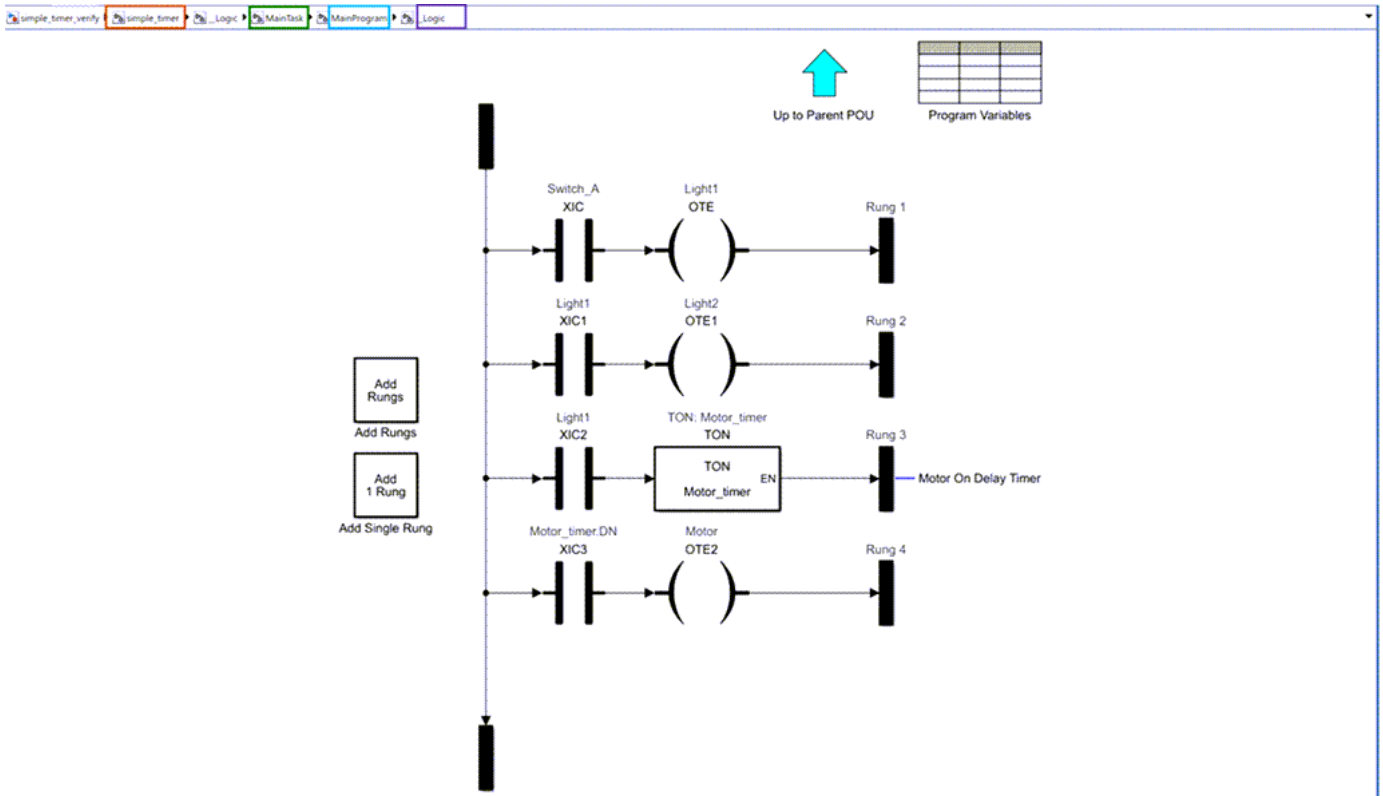
Verify Imported Ladder Diagram

To verify the imported ladder diagram:

- Connect a `Signal Builder` block to `Switch_A`.
- Connect a `Scope` block to the `Motor` and `Switch_A` signals.
- Open the `simple_timer_verify.slx` model.
- Open the `Scope` block and click the **Run** button.

```
% open_system("simple_timer_verify.slx")
```

This image shows the `Scope` block output for the model simulation. The `Motor (simple_timer)` output turns on three seconds after `Switch_A is turned on` and turns off as soon as `Switch_A` is turned off. This behavior is the expected behavior of the ladder diagram.
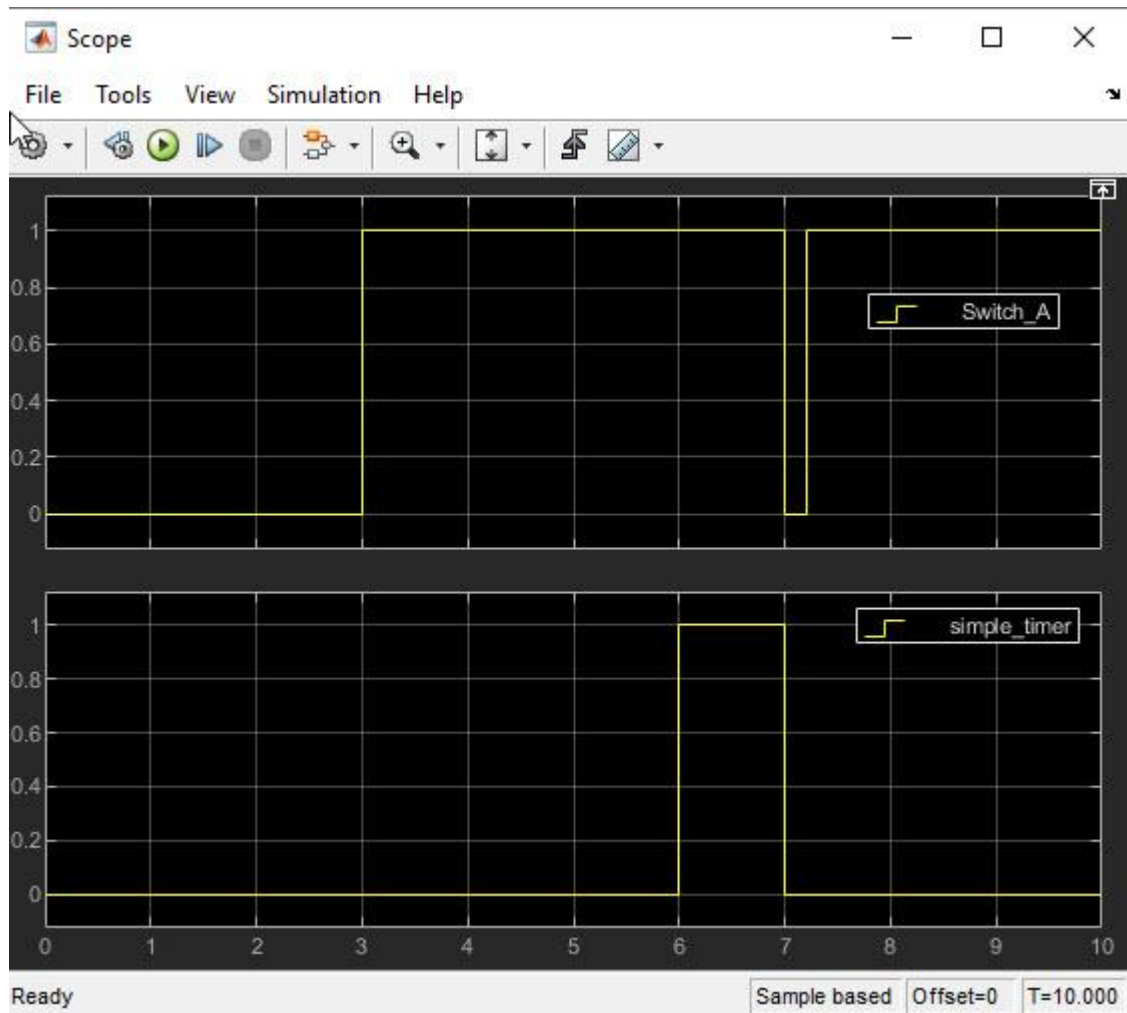
See Also
plcimportladder | plcgeneraterunnertb | plcgeneratecode | plcladderlib | plcladderoption | plcloadtypes | plccleartypes

More About
- "Supported Elements in Ladder Diagram" on page 3-2

- "Model and Simulate Ladder Diagrams in Simulink" on page 3-8
- "Generating Ladder Diagram Code from Simulink" on page 3-13
- "Generating C Code from Simulink Ladder" on page 3-15
- "Verify Generated Ladder Diagram Code" on page 3-17
- "Simulink PLC Coder Workflow vs. Rockwell Automation RSLogix IDE Workflow" on page 3-21

Model and Simulate Ladder Diagrams in Simulink

Simulate and validate your ladder diagrams by modeling them in Simulink PLC Coder.

You can then simulate and generate code for the ladder diagram models from within the Simulink environment.

1. To create a ladder diagram, open the Simulink PLC Coder ladder diagram library. At the MATLAB command line, enter:

    ```
    plcladderlib
    ```

    The ladder library opens containing all the blocks required for building the ladder diagram in Simulink.
2. Create a blank Simulink model. You can drag appropriate blocks from the library to build your ladder logic model in Simulink. For each block, double-click the block to see the block parameters. Use the `help` menu to view the block parameter description. For more information on the ladder diagram instructions and these blocks, refer to the LOGIX 5000 Controllers General Instructions Reference Manual. Go to https://literature.rockwellautomation.com/idc/groups/literature/documents/rm/1756-rm003_-en-p.pdf.
3. The Simulink PLC Coder ladder diagram library contains top-level ladder logic blocks such as PLCControllerSuite, PLC Controller, Task, Ladder Diagram Program, Ladder Diagram Subroutine, Ladder Diagram add-on instruction block (AOI), and AOI Runner. All these blocks are organization blocks (ladder diagram containers) that you cannot put on ladder diagram rungs. Other blocks from the library cannot be top-level ladder logic blocks for simulation.

    *   The PLCControllerSuite block can hold controller tags that are visible for all ladder logic blocks in this controller and contain the Task block.

    *   The PLCController block enables you to build ladder logic directly. All the tags in the controller level ladder diagram are controller tags (global variables or I/O symbols)

    *   The Task block contains ladder diagram programs that are using the same sample time and priority.

        Code generation for empty Task blocks is not supported. If a Task block is empty, the software does not issue warnings or errors during code generation, but the generated code produces errors in Rockwell Automation IDEs.

    *   The Ladder Diagram Program block enables you to build ladder logic directly. Program-level ladder diagrams can have program scope variables and can access controller tags if defined.

    *   The Ladder Diagram Subroutine block enables you to create and define a named ladder routine. You can edit the logic implemented by the subroutine by clicking the `Routine Logic` button under the block parameters menu of this block.

    *   The Ladder Diagram Function Block (AOI) block enables you to create the ladder diagram function block. You can edit the parameters and specifications of this block by using the various options available in the block parameters menu of this block.

    *   The AOI Runner block is a program block that can contain only one Ladder Diagram Function Block designed for add-on instruction (AOI) testing (test bench generation and verification).
4. Drag a PLCControllerSuite block into the blank model that you created. You can double-click each organizational unit to traverse to the lower-level ladder logic semantics and build your ladder diagram. This image shows the empty ladder logic diagram.
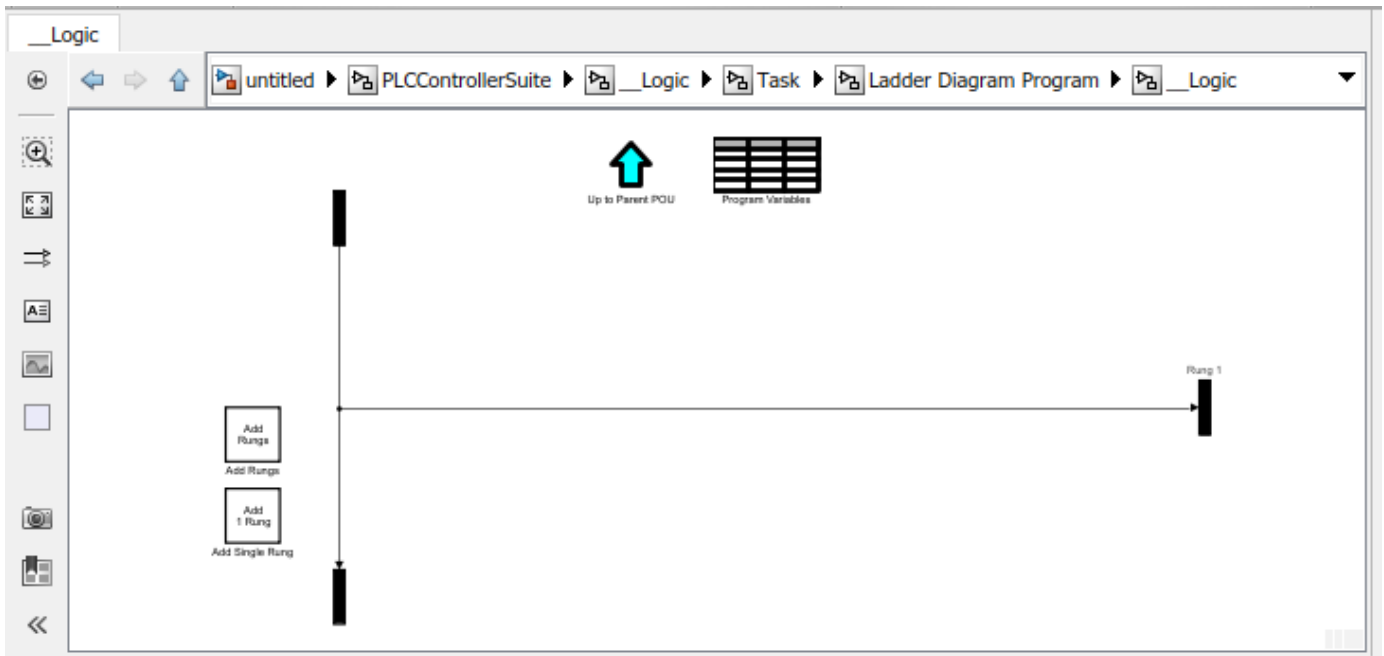
5 Use the XIC and OTE blocks from the library to construct a simple ladder diagram. Use the **Add Rungs** or **Add 1 Rung** buttons in the ladder logic semantic to add a new rung. All added blocks must be on the rung. Use the Junction block to merge rung branches.
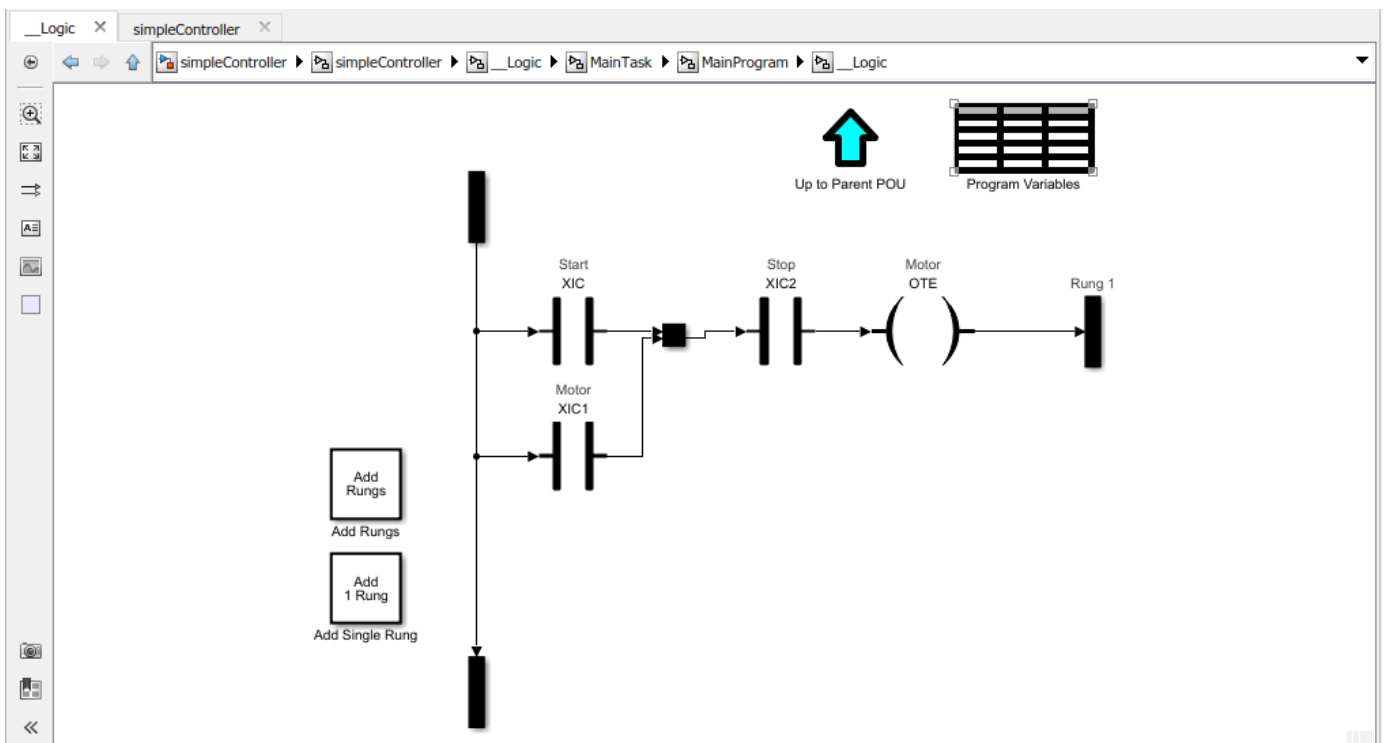
6 Double-click each new block added to the rung and specify the tags. In ladder diagrams, tags (variables) represent all inputs, outputs, and internal memory. The tag can be a variable name or an expression, such as:

- Variable Name: Start, Stop, Switch
- Bit Access: MyInt.0, MyInt.31
- Array Element: A[1], B[2,3], C[idx], D[i, j]. Use of braces for indexing is not allowed in a tag expression. For example, A(2) is illegal.
- Structure: A.B, C.D, E.F.G
- Mixture: A[1].B[i,j].C[3].D
- Expressions: A[3].B > C.D; A[3]+B[4].C

7   To change the attributes of the tag, open the **Program Variables** table within the Ladder Diagram Program block. The tags can have attributes such as Data Type, Initial Value, and Size. You can delete the unused variables in the variable table by selecting the **Delete** option. Select **Apply** for the changes to take effect. Go to the Controller Level block and double-click the **Controller Tags** table to specify the global variable and I/O symbol attributes.

8   To add rung comments to your model in Simulink, create a connected annotation (see `Motor Control Logic` in image) to the rung terminal block. See "Associate Annotations with Blocks and Areas".
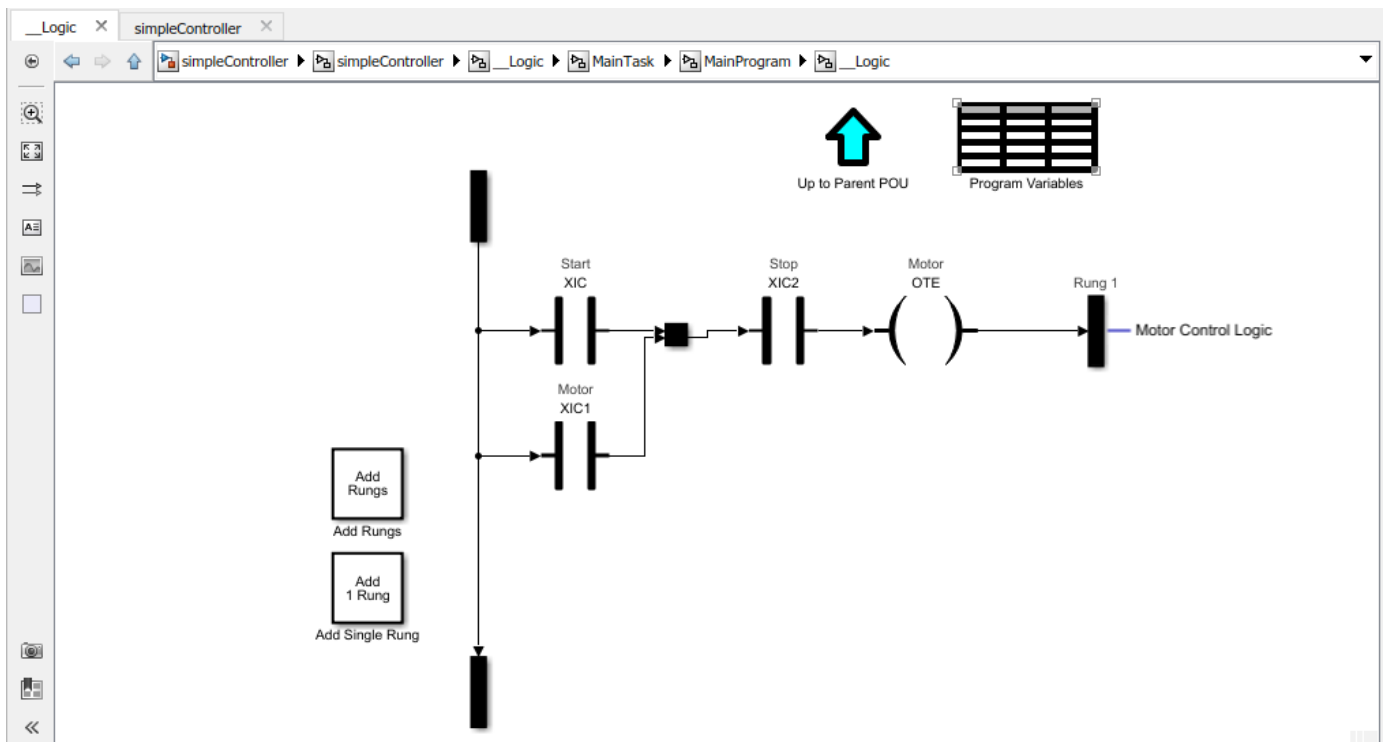
9   Use `Ctrl+D` to update the ladder logic model to reflect changes. You have now created a simple ladder model in Simulink.

Ladder Model Simulation

To perform ladder diagram simulation in Simulink, you must connect input and output blocks to the ladder model that match the actions that the ladder diagram inputs and outputs perform.

1   For simulation, to enable animation, use the `plcladderoption` function. At the MATLAB command line, enter:

```
plcladderoption('simpleController','Animation','on')
```

2   Connect input and output ports to the PLCControllerSuite block to provide inputs for simulation and to read the outputs. You must modify the attributes of the switch and motor tags. To change the attributes of the tag, open the **Program Variables** table within the Ladder Diagram Program block and set them to the values shown.
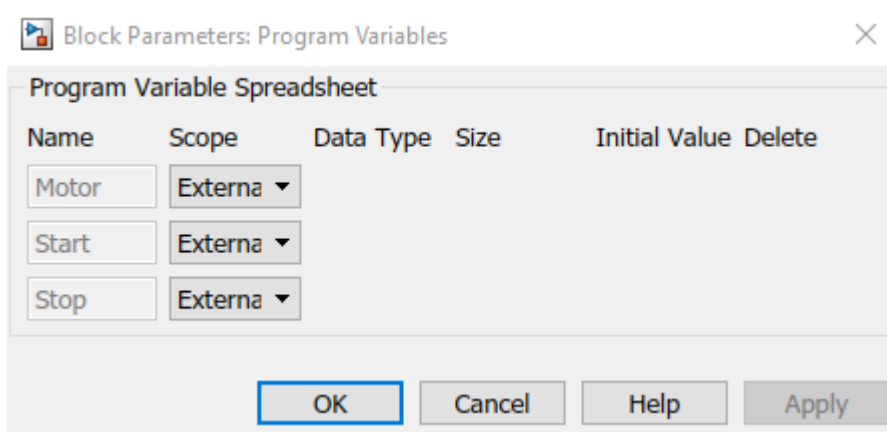
3   Go to the Controller Level block and double-click the **Controller Tags** table to specify the global variable and I/O symbol attributes.
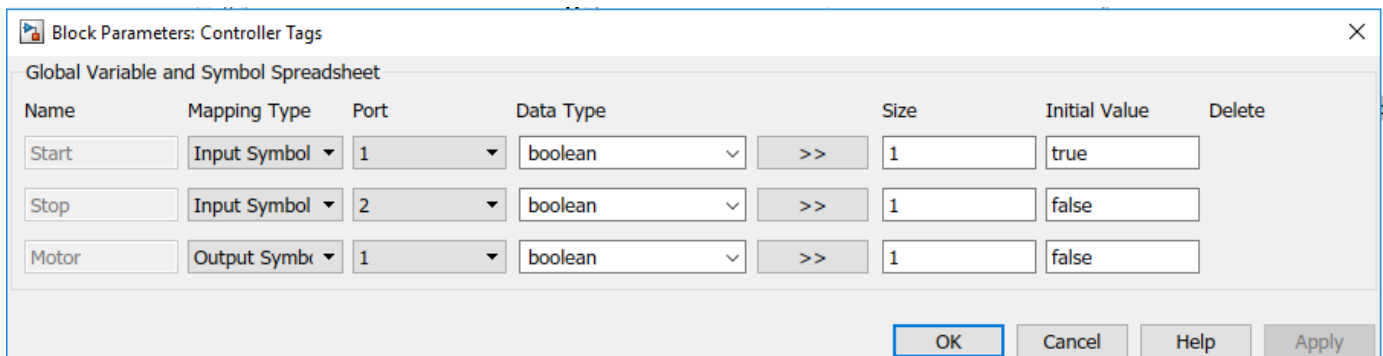
4   The software adds input and output ports to the PLCControllerSuite block. You can use Simulink blocks to add inputs to the ladder model. For example, you can use the Constant block to add Boolean inputs to mimic switch behavior.

5   Navigate to the Ladder Diagram Program block of the ladder model and click **Step Forward** to step through the simulation. The software uses the inputs provided, runs a behavioral simulation, and animates the ladder rungs and blocks based on the execution state.
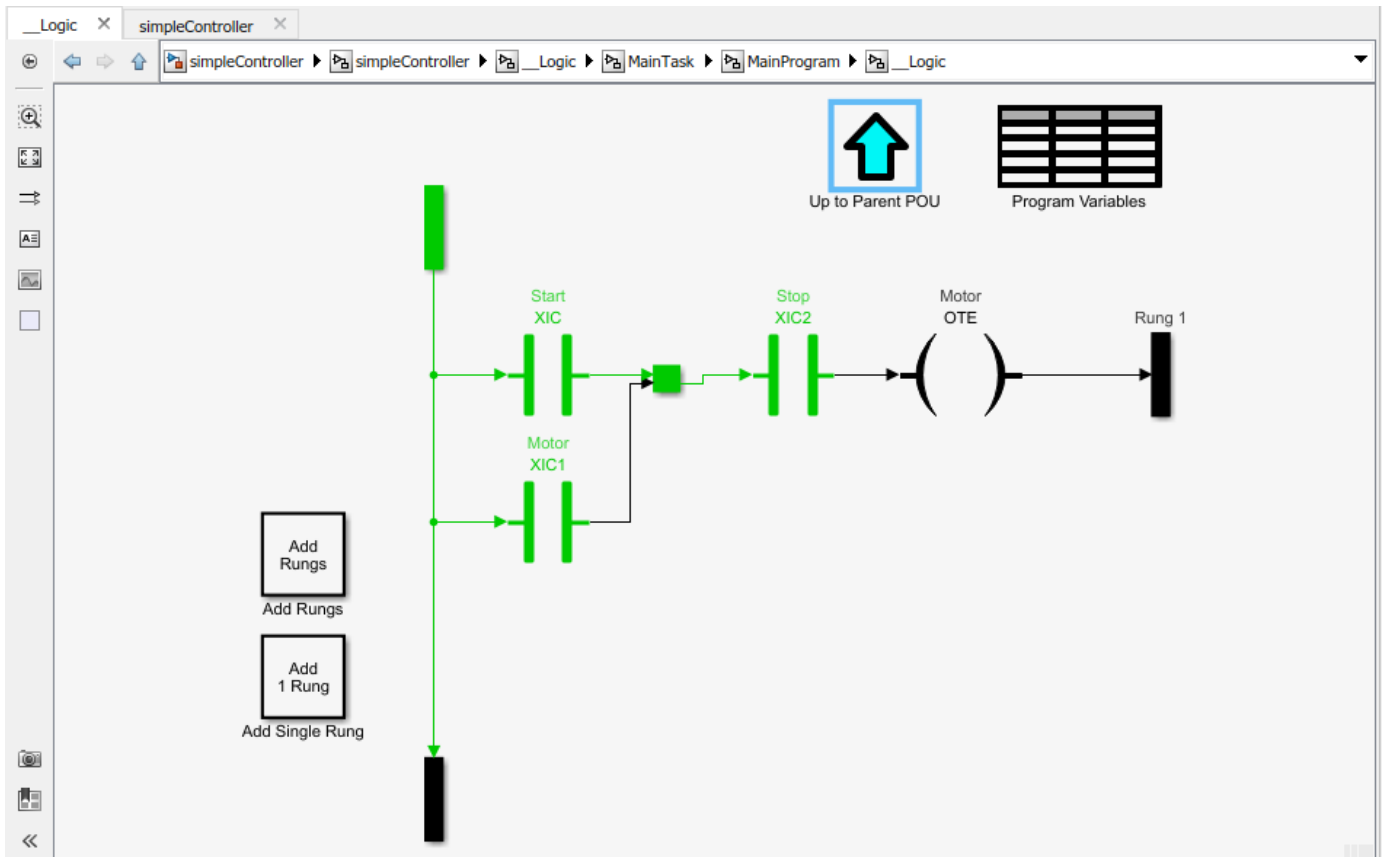
6 You can continue stepping forward or run a continuous simulation until the simulation stop time.

See Also
plcimportladder | plcgeneraterunnertb | plcgeneratecode | plcladderlib |
plcladderoption | plcloadtypes | plccleartypes

More About
- "Supported Elements in Ladder Diagram" on page 3-2
- "Import L5X Ladder Diagram Files into Simulink" on page 3-4
- "Generating Ladder Diagram Code from Simulink" on page 3-13
- "Generating C Code from Simulink Ladder" on page 3-15
- "Verify Generated Ladder Diagram Code" on page 3-17
- "Simulink PLC Coder Workflow vs. Rockwell Automation RSLogix IDE Workflow" on page 3-21

Generating Ladder Diagram Code from Simulink

This example shows how to import a simple Ladder Diagram from an `.L5X` file (`simpleController.L5X`) into the Simulink environment, and then generate Ladder Diagram (L5X) from the imported model. The Ladder Diagram `.L5X` file was created using RSLogix 5000 IDE and contains contacts and coils representing switches and motor. This figure shows the ladder structure.
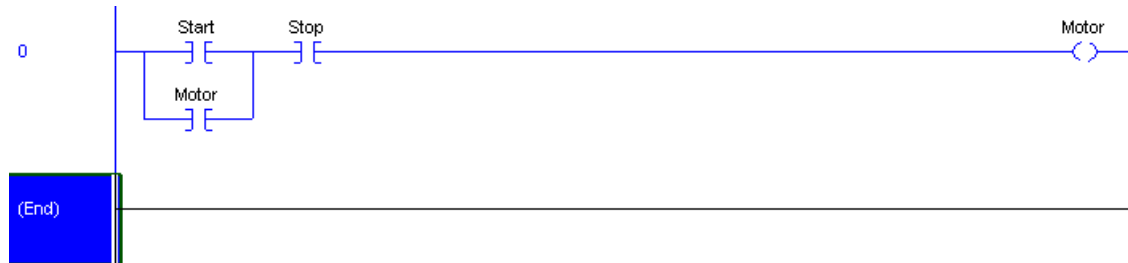
1 Use the `plcladderimport` function to import the ladder into Simulink.

```
[mdlName,mdlLib,busScript] = plcimportladder('simpleController.L5X','OpenModel','On')
```

2 The imported model contains a PLC Controller block named `simpleController`, followed by a Task block named `MainTask`, and finally a Ladder Diagram Program block named `MainProgram`. The model imported into Simulink has blocks that implement the functionality of the contacts and coils.
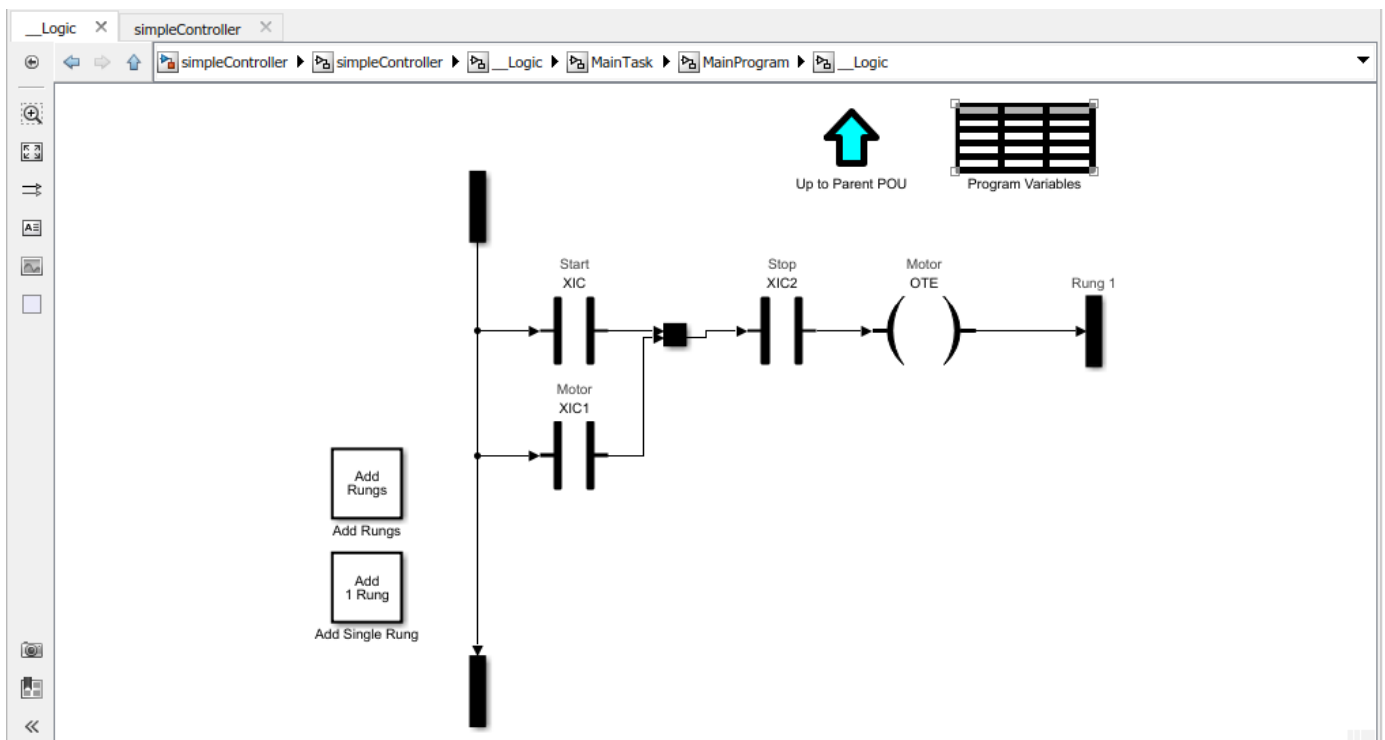
3 Generate code for the subsystem `simpleController/simpleController`.

```
generatedFiles = plcgeneratecode('simpleController/simpleController');
```

```
PLC code generation successful for 'simpleController/simpleController'.
```

3-13

```
Generated ladder files:
plcsrc\simpleController_gen.L5X
```

> **Note** You cannot generate structured text code from the Ladder Diagram blocks. The Ladder feature supports only ladder code generation.

See Also
plcimportladder | plcgeneraterunnertb | plcgeneratecode | plcladderlib | plcladderoption | plcloadtypes | plccleartypes

More About

- "Supported Elements in Ladder Diagram" on page 3-2
- "Import L5X Ladder Diagram Files into Simulink" on page 3-4
- "Model and Simulate Ladder Diagrams in Simulink" on page 3-8
- "Generating C Code from Simulink Ladder" on page 3-15
- "Verify Generated Ladder Diagram Code" on page 3-17
- "Simulink PLC Coder Workflow vs. Rockwell Automation RSLogix IDE Workflow" on page 3-21

Generating C Code from Simulink Ladder

This example shows how to import a simple ladder diagram from an `.L5X` file (`simpleController.L5X`) into the Simulink environment and then generate C code from the imported model. You must have Simulink Coder and necessary compilers to generate C code from the model. For more information, see "Get Started with Simulink Coder" (Simulink Coder).

The `.L5X` file was created using RSLogix 5000 IDE and contains contacts and coils representing switches and motor. This figure shows the ladder structure.
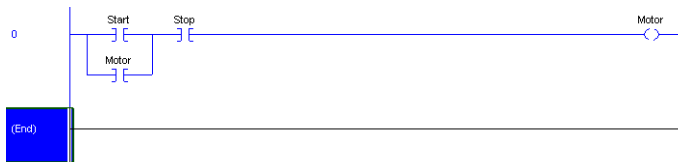
Use the `plcladderimport` function to import the ladder into Simulink.

```
[mdlName,mdlLib,busScript] = plcimportladder('simpleController.L5X','OpenModel','On')
```

The imported model contains a PLC Controller block named `simpleController`, followed by a Task block named `MainTask`, and finally a Ladder Diagram Program block named `MainProgram`. The model imported into Simulink has blocks that implement the functionality of the contacts and coils.
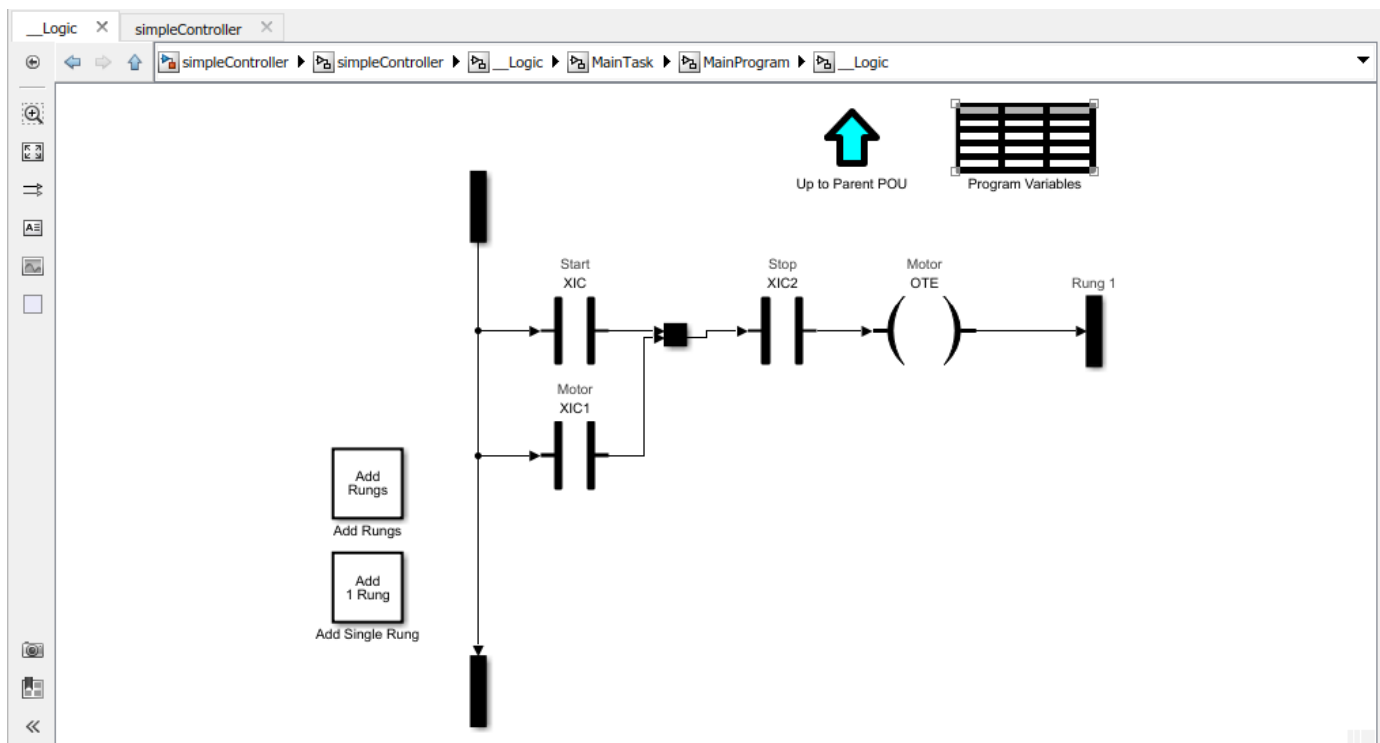
To generate C code for the subsystem `simpleController/simpleController`, you must first enable the `'FastSim'` option for the Simulink Ladder Diagram model.

```
currentState = plcladderoption('simpleController/simpleController','FastSim','on');
```

Open the Configuration Parameters dialog box from the model editor by clicking **Modeling > Model Settings**.

Alternatively, type these commands at the MATLAB command prompt:

```
cs = getActiveConfigSet(model);
openDialog(cs);
```

Ensure that a valid **Toolchain** is selected.

In the model window, initiate code generation and the build process for the model by using any of the following options:

- Click the **Build Model** button.
- Press **Ctrl+B**.
- In the **Apps** gallery, under **Code Generation**, click **Embedded Coder**. On the **C Code** tab, select **Build> Build**.
- Invoke the `slbuild` command at the MATLAB command line.

See Also
plcimportladder | plcgeneraterunnertb | plcgeneratecode | plcladderlib | plcladderoption | plcloadtypes | plccleartypes

More About
- "Supported Elements in Ladder Diagram" on page 3-2
- "Import L5X Ladder Diagram Files into Simulink" on page 3-4
- "Model and Simulate Ladder Diagrams in Simulink" on page 3-8
- "Generating Ladder Diagram Code from Simulink" on page 3-13
- "Verify Generated Ladder Diagram Code" on page 3-17

Verify Generated Ladder Diagram Code

This example shows how to import a simple Ladder Diagram from an `.L5X` file (`simpleXIC.L5X`) into the Simulink environment and generate test bench code for it. The Ladder Diagram `.L5X` file was created using RSLogix 5000 IDE and contains an AOI named `simpleXIC` with contact and coil representing a switch and a light. This figure shows the ladder structure.
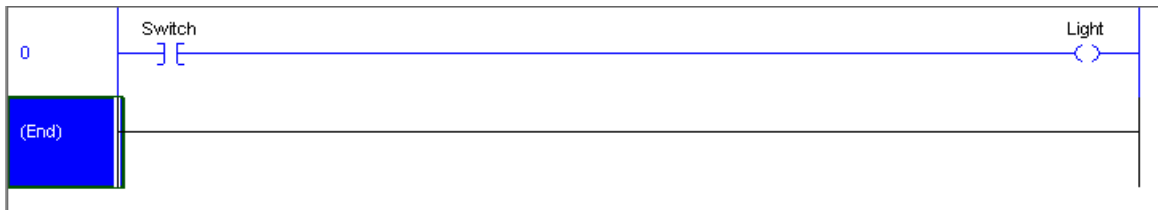
1   Use the `plcladderimport` function to import the ladder into Simulink.

```
[mdlName,mdlLib,busScript] = plcimportladder('simpleXIC.L5X',...
'OpenModel','On','TopAOI','simpleXIC')
```

2   The imported model contains an AOI Runner block named `simpleXIC_runner`, followed by a Ladder Diagram Function (AOI) block named `simpleXIC`.
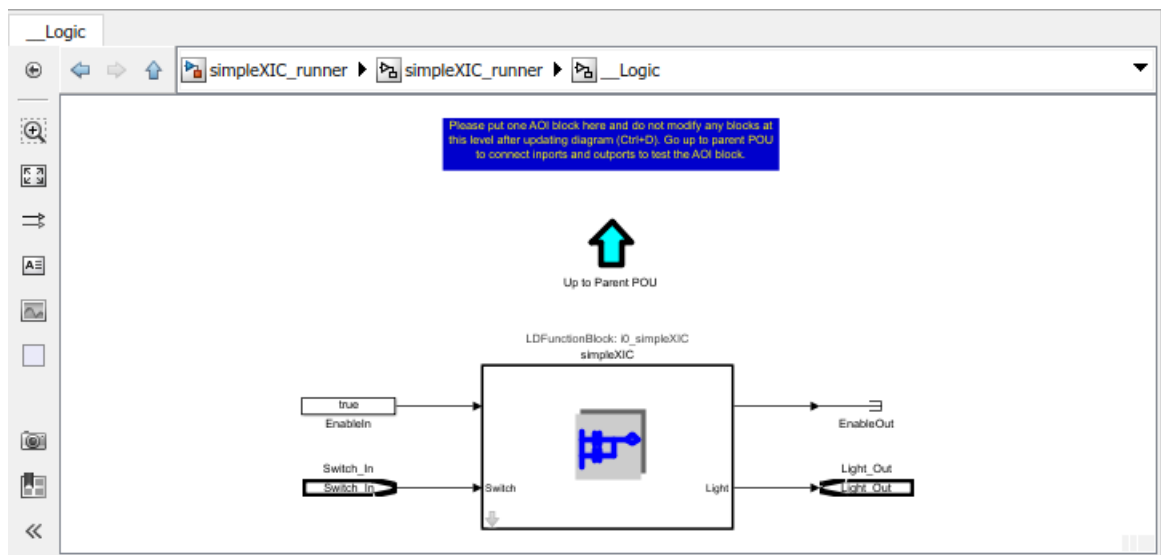
3   Add the Signal Builder input block, Scope, and output ports as shown.
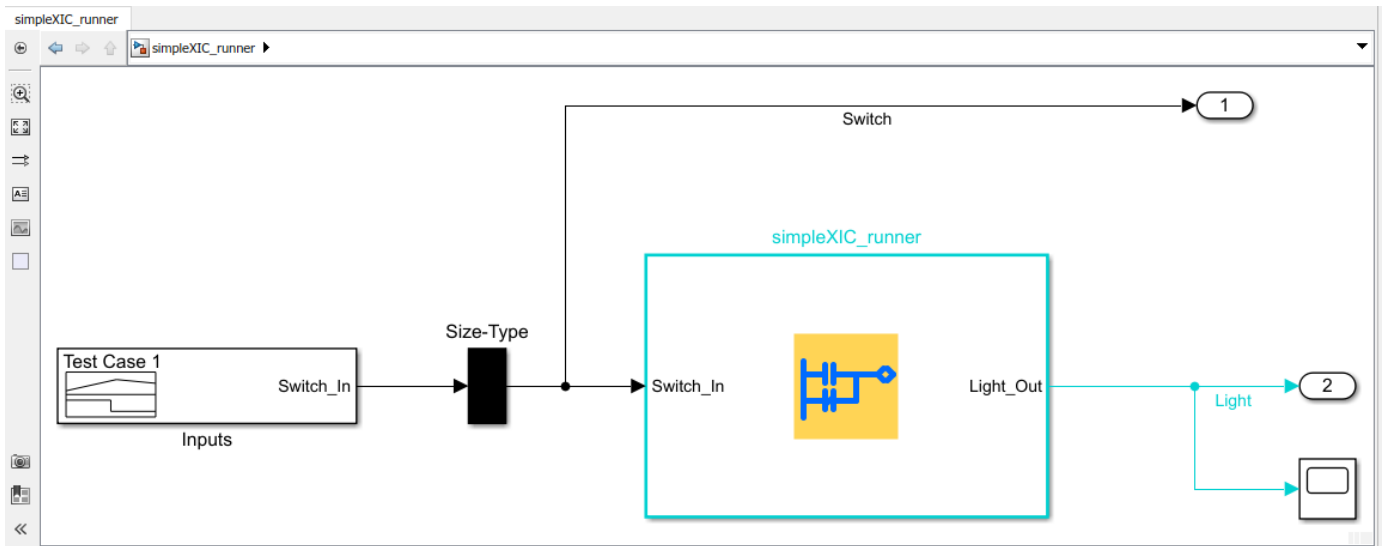
4 Modify the Signal Builder input to mimic a switch operation.
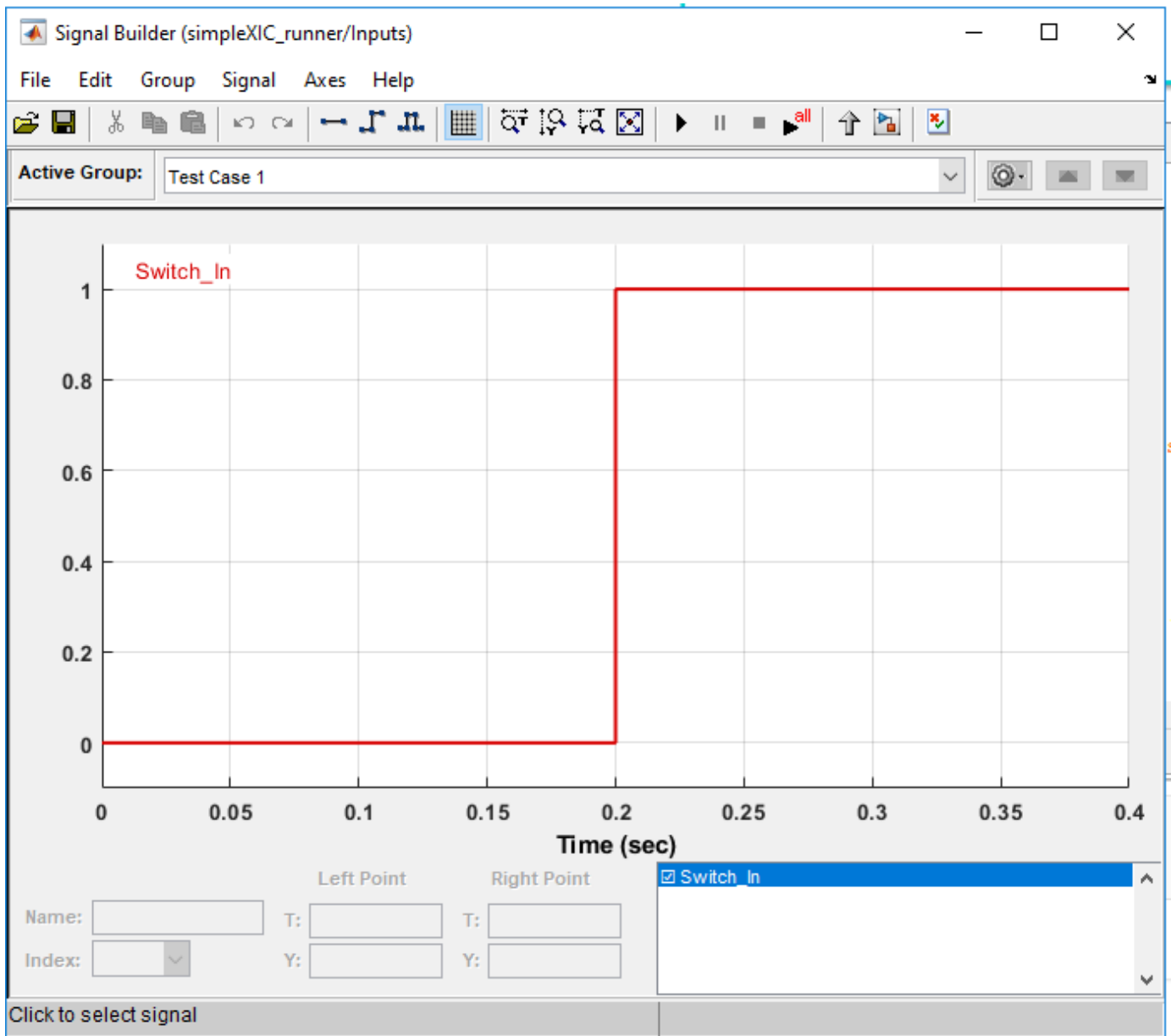
5 Generate a test bench for the Ladder Diagram model.

```
Tbcode = plcgeneraterunnertb('simpleXIC_runner/simpleXIC_runner')

Tbcode =

  1×1 cell array

    {'C:\runnerTB\simpleXIC_runner.L5X'}
```

If the test bench code generation is successful, a test bench file `simpleXIC_runner.L5X` is created. You can verify the generated AOI test bench file on the Rockwell Automation IDE.

If you have created the Ladder Diagram model in Simulink and are generating Ladder Diagram (L5X) code, you can also use the **Generate testbench for subsystem** option on the **PLC Code**

Generation pane in the Configuration Parameters dialog box to generate test bench code along with ladder code. When the selected subsystem is a the ladder AOI Runner block and the test bench option is on, the generated code includes test bench, selected AOI, and dependent AOI and user define tag (UDT) types.
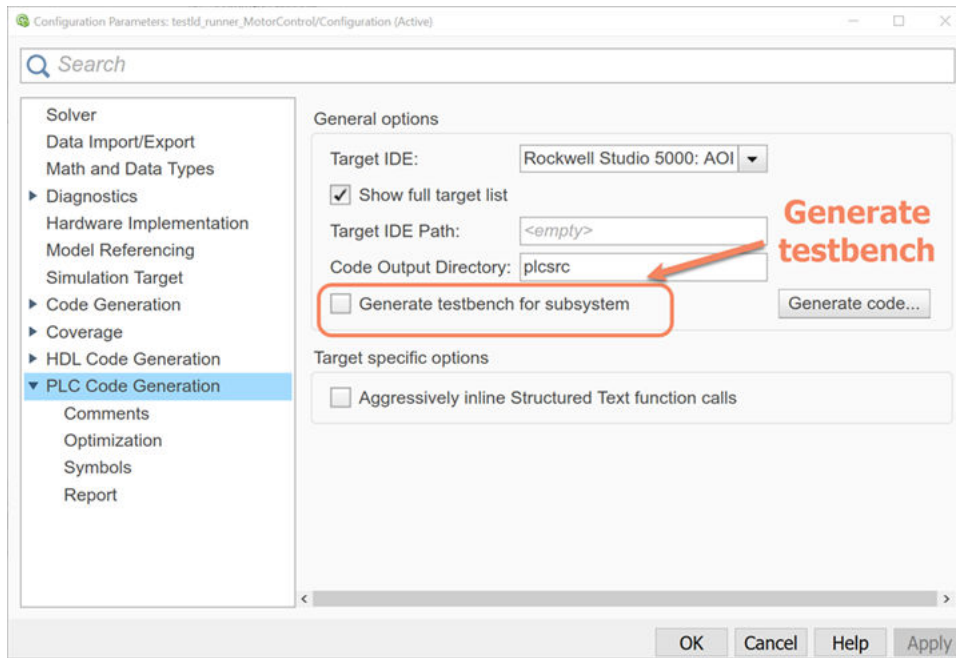
See Also

plcimportladder | plcgeneraterunnertb | plcgeneratecode | plcladderlib | plcladderoption | plcloadtypes | plccleartypes

More About

- "Supported Elements in Ladder Diagram" on page 3-2
- "Import L5X Ladder Diagram Files into Simulink" on page 3-4
- "Model and Simulate Ladder Diagrams in Simulink" on page 3-8
- "Generating Ladder Diagram Code from Simulink" on page 3-13
- "Generating C Code from Simulink Ladder" on page 3-15
- "Simulink PLC Coder Workflow vs. Rockwell Automation RSLogix IDE Workflow" on page 3-21

Simulink PLC Coder Workflow vs. Rockwell Automation RSLogix IDE Workflow

These flowcharts show the workflow comparison in a ladder diagram created by Simulink PLC Coder versus a ladder diagram created in the Rockwell Automation RSLogix IDE.

- You first place either the PLC Controller or PLC Controller Suite block onto the blank Simulink model page. This block contains all the tasks, programs, program tags, controller tags, routines, AOI blocks and so on. For more information, see PLC Controller.

- You place the Task block inside the PLC Controller or PLC Controller suite block. The Task blocks house the programs, program tags, routines, AOI blocks, and so on. For more information see, Task.

- You place the Ladder Diagram Program block or blocks inside the Task block. The Ladder Diagram Program block contains program tags, routines , AOI blocks, and so on. For more information see, Program

- You next place JSR (Jump To Subroutine) blocks within the Ladder Diagram Program block. The JSR blocks contain the ladder rungs, ladder logic and AOI blocks within them. For more information see ,Subroutine.

- You can place the AOI block either inside the JSR block or inside the Ladder Diagram Program block. For more information see, Function Block (AOI).
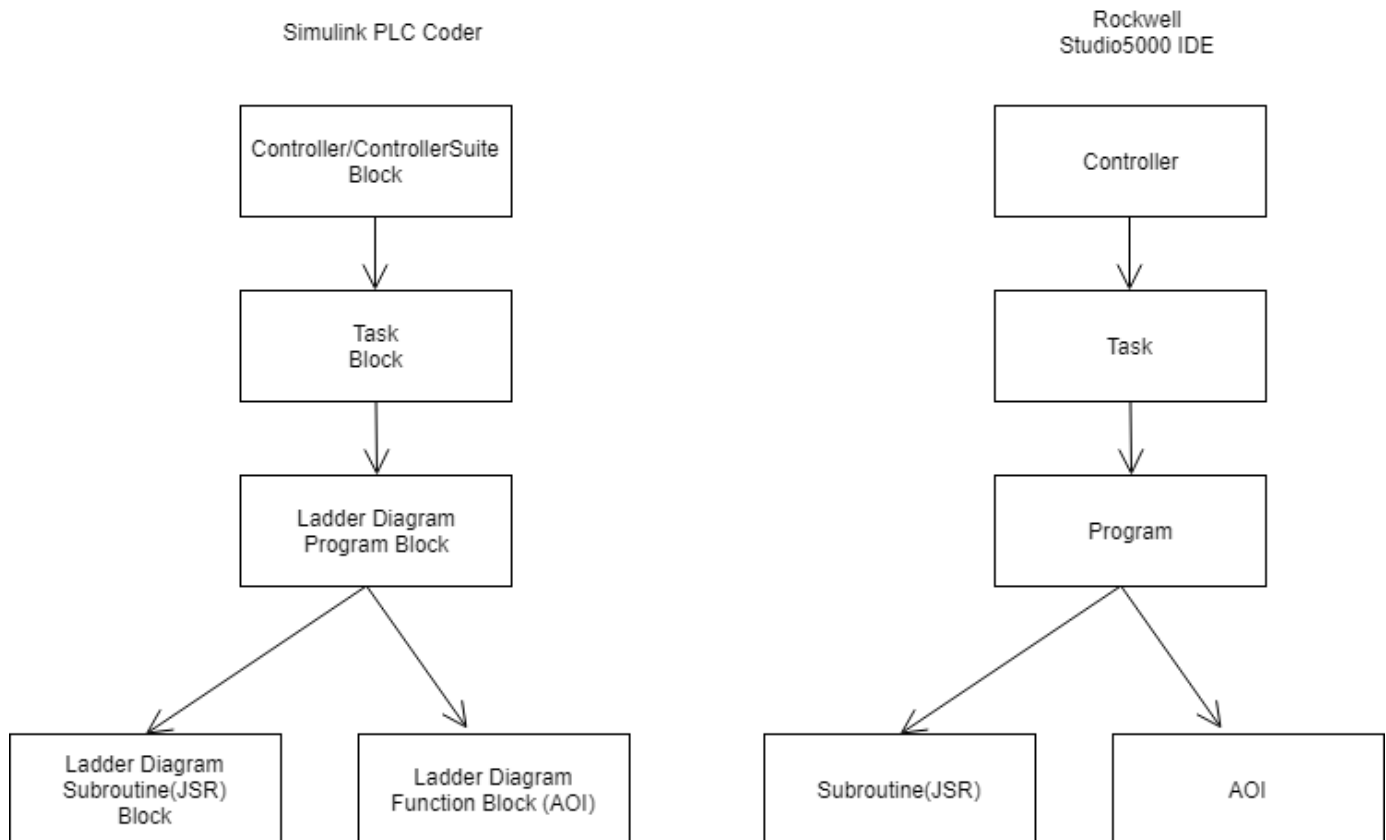
See Also

PLC Controller | PLC Controller Suite | Task | Program | Subroutine | Function Block
(AOI)

More About

- "Supported Elements in Ladder Diagram" on page 3-2
- "Import L5X Ladder Diagram Files into Simulink" on page 3-4
- "Model and Simulate Ladder Diagrams in Simulink" on page 3-8
- "Generating Ladder Diagram Code from Simulink" on page 3-13
- "Verify Generated Ladder Diagram Code" on page 3-17

Create Custom Instruction in PLC Ladder Diagram Models

You can create user-defined instructions for your ladder models by using the Custom Instruction block. You can store these blocks containing custom instructions in a user-defined library named `plcuserlib.slx`. You can also import, simulate, and export your ladder instructions by using your custom blocks.

Create User-Defined Custom Instruction

To create a user-defined instruction, use a Custom Instruction block that you add to the Simulink PLC Coder Ladder Library.

1   To open the Ladder Library, at the MATLAB command line, enter:

```
plcladderlib
```

The Ladder Library opens all the blocks required for building the Ladder Diagram in Simulink.

2   To create a Simulink library, in the **Library** tab, click **New > Library**. From the Simulink start page, select **Blank Library** and click **Create Library**.

3   Drag a Custom Instruction block from the Ladder Library to the new library that you created. See Custom Instruction.

4   To build your own ladder logic model, double-click your Custom Instruction block to see the block parameters. Use the **Help** menu to view their descriptions.
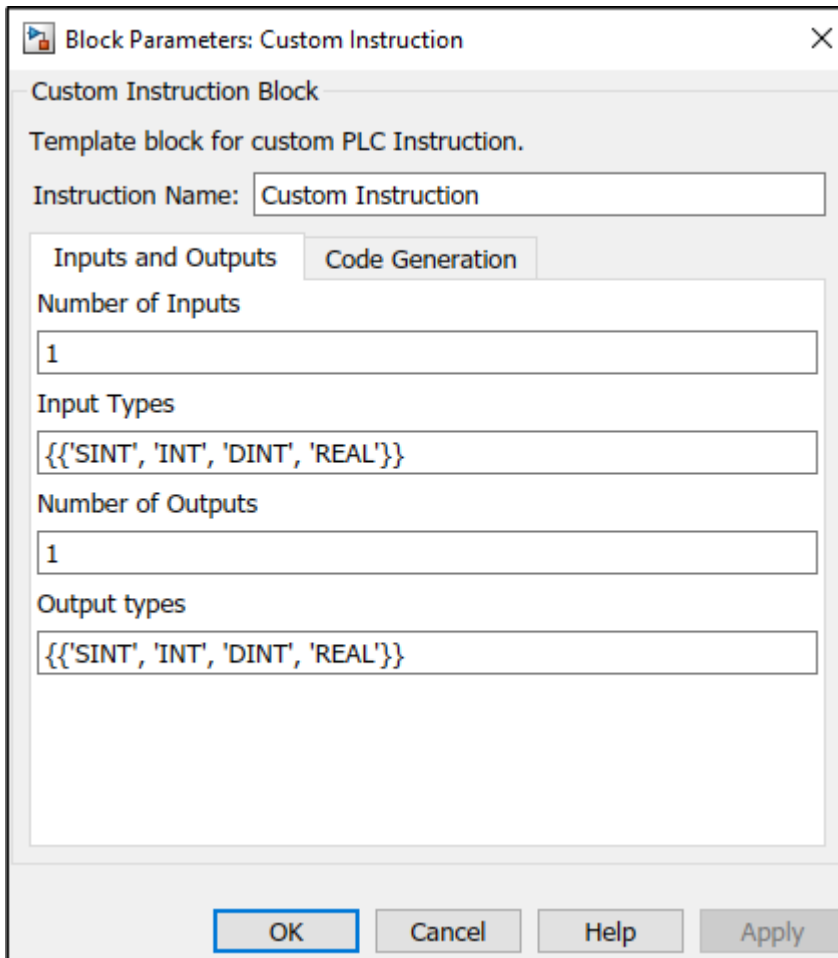
5 In **Instruction Name** text field, give a name to your instruction. Specify the inputs and outputs required for your instruction block. Click **Apply**, and then click **OK**.

6 To look inside the mask, click ⬇ in the Custom Instruction block. The blocks inside the mask enable the instruction to simulate with other PLC Ladder instructions. The user-defined logic is included in the Instruction_Enable block.
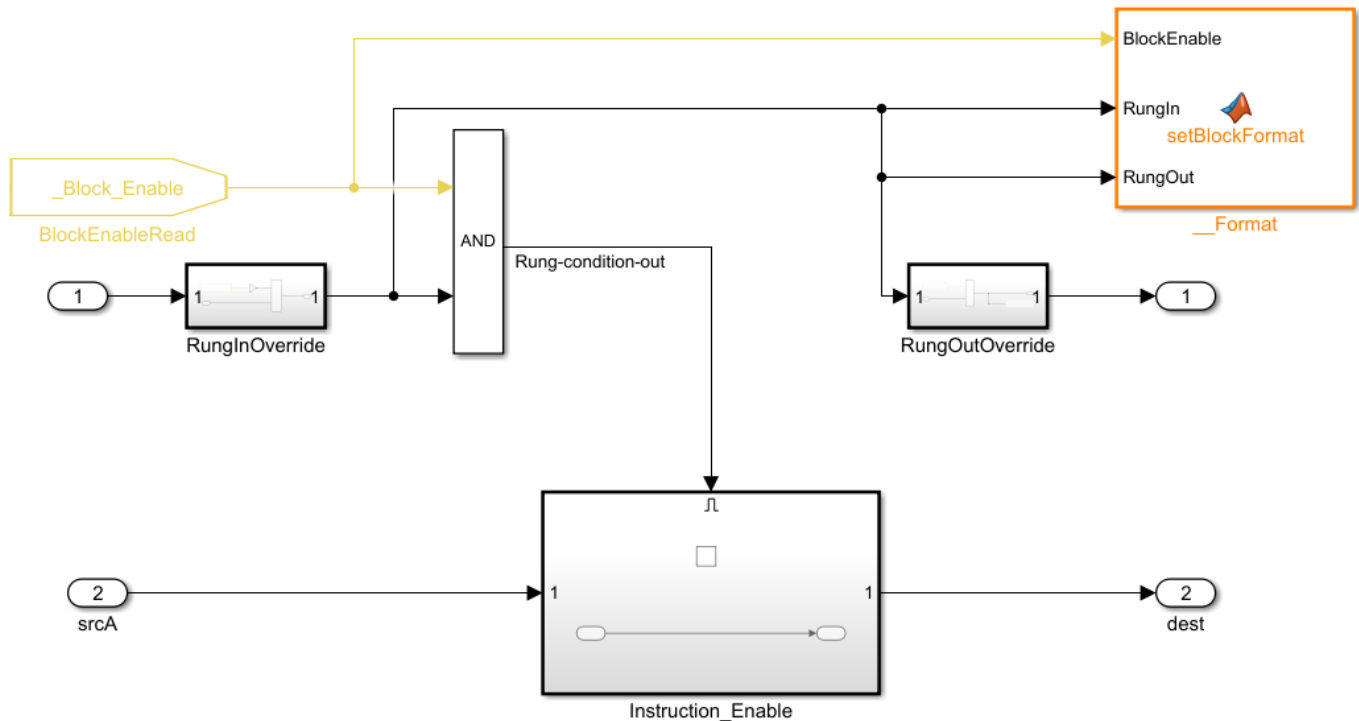
7   Save the library as `plcuserlib.slx`. You can add multiple instruction blocks to this library.

Calculate Square Root by Using Custom Instruction Block

This example shows how to calculate square root of an input signal by using a Custom Instruction block.

1   To open the Simulink Start Page, on the MATLAB **Home** tab, click **Simulink**.

2   Select **Blank Library** and click **Create Library**.

3   Save the library as `plcuserlib.slx` to a folder on the MATLAB path.

4   To open the PLC Ladder Library, at the MATLAB command line, enter:

`plcladderlib`

5   Drag the Custom Instruction block from `plcladderlib` to your user-defined library `plcuserlib.slx`.

6   Double-click the Custom Instruction block to open the Block Parameters.

7   Specify the **Instruction Name** as `SQR`. Make sure that the **Number of Inputs** is 1 and **Input Types** is specified as a cell array of allowed data types. Make sure that the **Number of Outputs** is 1 and **Output Types** is specified as a cell array of allowed data types. Click **OK**.

8   Click ⬇ in the SQR block and double-click the Instruction_Enable subsystem.

9   Inside the Instruction_Enable subsystem, add a Sqrt block from the Simulink / Math Operations Library. Double-click this block and select `signedSqrt` from **Main>Function**, and then click **OK**.

10   Connect the input and output ports to the input and output ports of Sqrt block by using Data Type Conversion blocks.
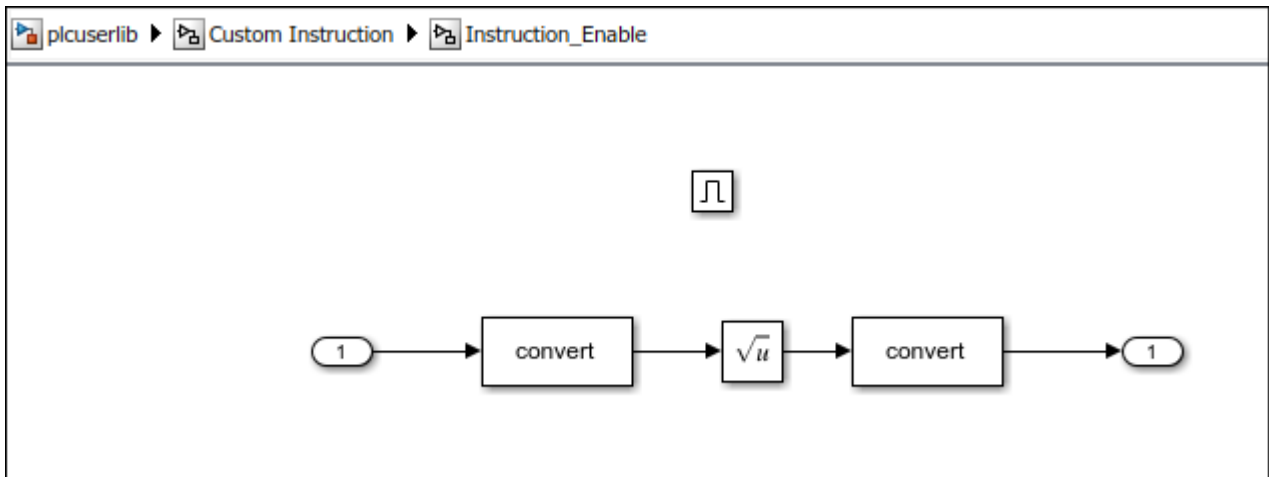
11 Navigate to the top level of the library. Click **Lock Links** and **Unlock Library** in the **Library** tab, and then save the library. Simulink PLC Coder can now use the SQR instruction when `plcuserlib.slx` is on the MATLAB path. You can drag this instruction to your models from the library that you have created and saved.

12 To verify if Simulink PLC Coder has identified the newly created instruction, at the MATLAB command line, enter:

```
plcladderinstructions
```

This command lists the instructions that Simulink PLC Coder can use. The supported instructions displayed in the output includes the SQR instruction.

The example in the image shows the use of the SQR instruction inside an Add-On Instruction block.
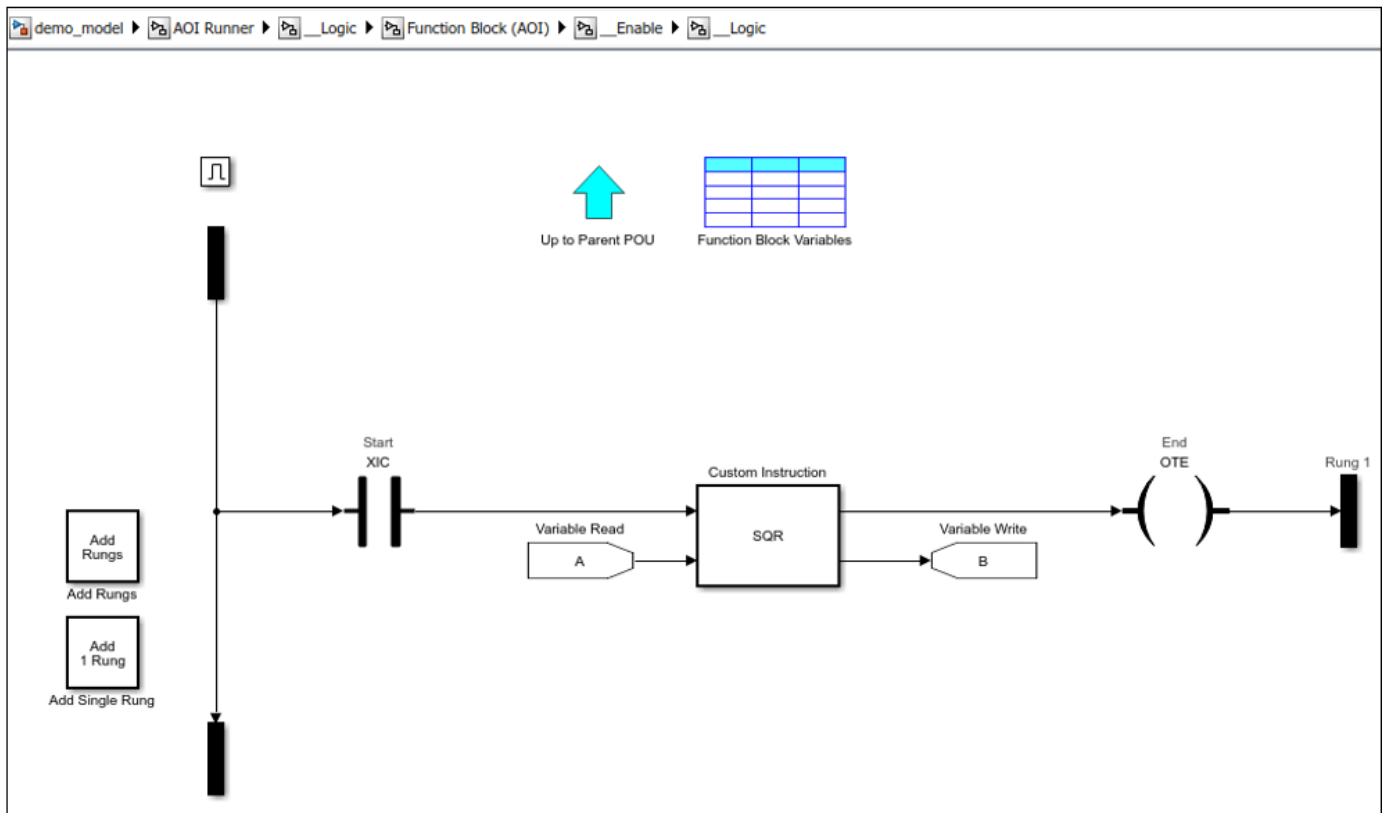
Limitations

The Custom Instruction block does not support instructions:

• With data type `array` and `struct` (composite) as arguments.
• That require internal data storage (states).

See Also
`plcimportladder` | `plcladderinstructions` | Custom Instruction | `plcladderlib`

More About
• "Supported Elements in Ladder Diagram" on page 3-2
• "Import L5X Ladder Diagram Files into Simulink" on page 3-4
• "Model and Simulate Ladder Diagrams in Simulink" on page 3-8
• "Generating Ladder Diagram Code from Simulink" on page 3-13

Generating Test Bench Code

- "Test Bench Verification" on page 4-2
- "Integrate Generated Code with Custom Code" on page 4-3
- "Import and Verify Structured Text Code" on page 4-4
- "Generate Code That Has Multiple Test Benches" on page 4-6

Test Bench Verification

The Simulink PLC Coder software simulates your model and captures the input and output signals for a subsystem. The captured input and output signal data is the test bench data. You can generate a test bench or test harness by using the test bench data. See "Generate Testbench for Subsystem" on page 13-7.

You can verify that the output of the generated code is numerically and functionally equivalent to the output of the Simulink model by using the generated test bench. This table shows the error tolerance for the different data types. The comparison is between the outputs of the generated code (expected values) and outputs of the model (actual values).

Data Type	Comparison	Error Tolerance
`integer`	absolute	0
`boolean`	absolute	0
`single`	relative	0.0001
`double`	relative	0.00001

The relative tolerance comparison for single or double data types uses this logic:

```
IF ABS(actual_value - expected_value) > (ERROR_TOLERANCE * expected_value) THEN
        testVerify := FALSE;
END_IF;
```

To verify the generated code by using the test bench, import the generated structured text and the test bench data into your target IDE. You can import test bench code either manually or automatically. See "Import and Verify Structured Text Code" on page 4-4.

See Also

See Also

Related Examples

- "Generate Code That Has Multiple Test Benches" on page 4-6

Integrate Generated Code with Custom Code

For the top-level subsystem that has internal state, the generated FUNCTION_BLOCK code has ssMethodType. ssMethodType is a special input argument that the coder adds to the input variables section of the FUNCTION_BLOCK section during code generation. ssMethodType enables you to execute code for Simulink Subsystem block methods such as initialization and computation steps. The generated code executes the associated CASE statement based on the value passed in for this argument.

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

Has Internal State	ssMethodType Contains...
Yes	The generated function block for the block has an extra first parameter ssMethodType of integer type. This extra parameter is in addition to the function block I/O parameters mapped from Simulink block I/O ports. To use the function block, first initialize the block by calling the function block with ssMethodType set to integer constant SS_INITIALIZE. If the IDE does not support symbolic constants, set ssMethodType to integer value 0. For each follow-up invocation, call the function block with ssMethodType set to constant SS_STEP. If the IDE does not support symbolic constants, set ssMethodType to integer value 1. These settings cause the function block to initialize or compute and return output for each time step. If you select Keep top level ssMethod name same as non-top level, the ssMethodType SS_STEP will be generated as SS_OUTPUT with integer value 3.
No	The function block interface only has parameters mapped from Simulink block I/O ports. There is no ssMethodType parameter. To use the function block in this case, call the function block with I/O arguments.

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

Has Internal State	ssMethodType Contains...
Yes	The function block interface has the ssMethodType parameter. The generated code might have SS_INITIALIZE, SS_OUTPUT, or other ssMethodType constants to implement Simulink semantics. If non top-level subsystems have blocks with constant sample time the generated code could have SS_CONST_CODE constants to implement Simulink semantics.
No	The function block interface only has parameters mapped from Simulink block I/O ports. There is no ssMethodType parameter.

Import and Verify Structured Text Code

Generate structured text code and test bench from your model. Verify the generated code by importing the generated code and test bench into your target IDE. You can verify that the output of the generated code matches the output of the model simulation by using the test bench data.

Generate, Import, and Verify Structured Text

This example shows how to import and verify your generated code by using the generated test bench:

1 Open the `plcdemo_simple_subsystem` example.
2 Open the **PLC Coder** app. Click the **PLC Code** tab > **Settings** > **PLC Code Generation**.
3 Select the **Generate testbench for subsystem** check box. Click **OK**.
4 Click the **PLC Code** tab. Click **Settings** > **Verify Code in IDE**.
5 In the **PLC Code** tab, click **Generate PLC Code**.

When you select **Verify Code in IDE**, the software:

1 Generates the code and test bench.
2 Starts the target IDE.
3 Creates a project.
4 Imports the generated code and test bench to the new project in the target IDE.
5 Runs the generated code on the target IDE to verify it.

If you do not specify that the test bench code must be generated, when you verify the generated code, you see the error `Testbench not selected`.

For information on:

- IDEs not supported for automatic import and verification, see "Troubleshoot Automatic Import Issues" on page 1-14.
- Possible reasons for long code generation time, see "Troubleshooting: Long Test Bench Code Generation Time" on page 4-4.

Troubleshooting: Long Test Bench Code Generation Time

When generating code that has a testbench and the test bench data size exceeds the limit that Simulink PLC Coder can handle, it might result in long code generation times. The test bench data size depends on the number of times the input signal is sampled during simulation. When the simulation time is long or the simulation signals are sampled at a high frequency, the test bench data can be large.

To reduce test bench data size and code generation time, you can:

- Reduce the duration of the simulation.
- Increase the simulation step size.
- If you want to retain the simulation duration and the step size, divide the simulation into multiple parts. For a simulation input signal that have the duration [0, t], divide the input into multiple parts with durations [0, t_1], [t_1, t_2], [t_2, t_3], and so on, where $t_1 < t_2 < t_3 < .. < t$.

Generate test bench code for each part separately and manually import them together to your IDE.

See Also

Related Examples

* "Generate Code That Has Multiple Test Benches" on page 4-6

Generate Code That Has Multiple Test Benches

You can generate code that has multiple test benches from your subsystem. For the generated code to have multiple test benches, the input to your subsystem must consist of multiple signal groups.

To generate multiple test benches for your subsystem:

1 Provide multiple signal groups as inputs by using a Signal Builder block that has multiple signal groups.

Use Simulink Design Verifier™ to create a test harness model from the subsystem. In the test harness model, a Signal Builder block that has one or more signal groups provides input to the subsystem. Use this Signal Builder block to provide inputs to your subsystem. If your model is complex, Simulink Design Verifier can create a large number of signal groups. See "Troubleshooting: Test Data Exceeds Target Data Size" on page 4-7.

To create your Signal Builder block:

a Right-click the subsystem and select **Design Verifier** > **Generate Tests for Subsystem**.

b In the Simulink Design Verifier Results Summary window, select **Create harness model**.
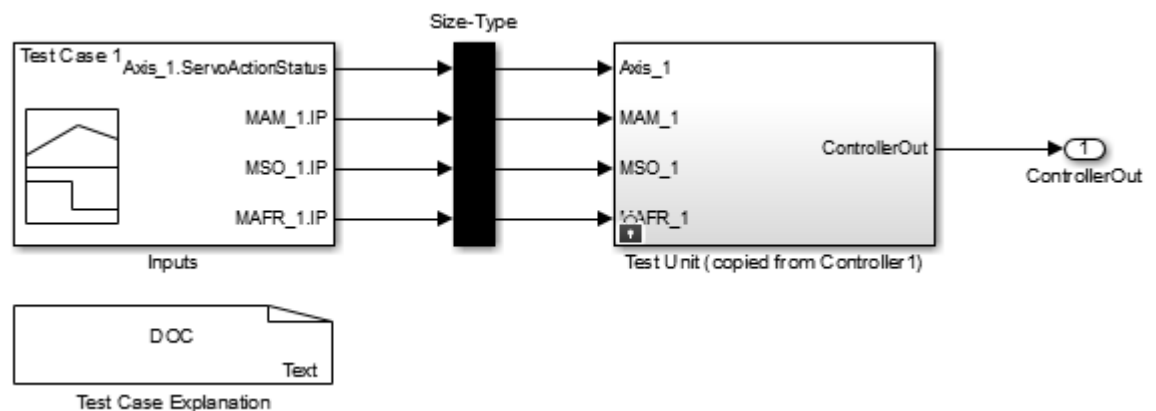
c Open the Inputs block in the test harness model. The Inputs block is a Signal Builder block that can have one or more signal groups.

In the Signal Builder window, make sure that more than one signal group is available in the **Active Group** drop-down list.
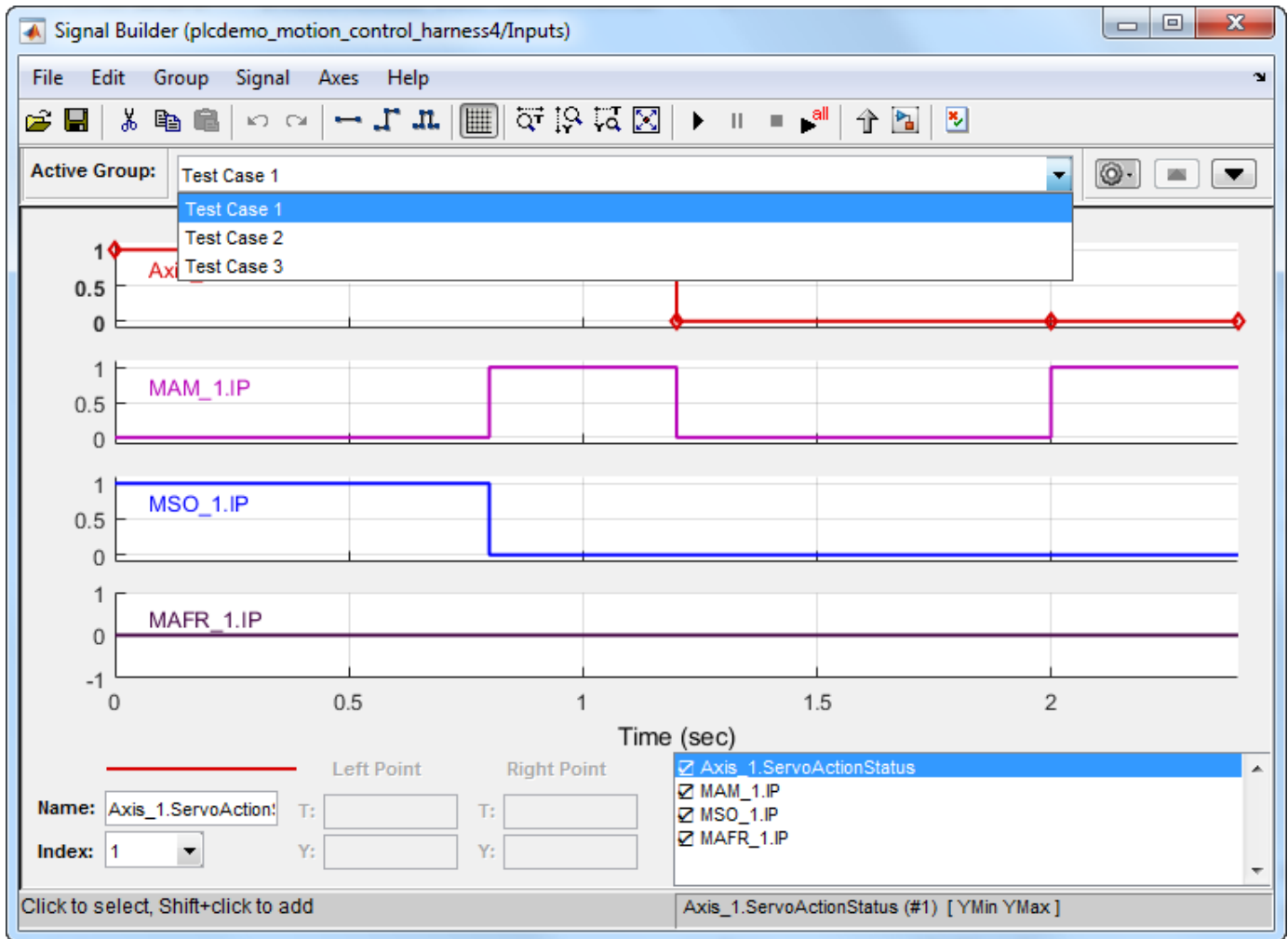
d Copy the Signal Builder block from the test harness mode. Use this block to provide inputs to your original subsystem.

2 Generate test benches for the subsystem:

a Open the **PLC Coder** app. Click **PLC Code** tab > **Settings** > **PLC Code Generation**.

b Select the **Generate testbench for subsystem** option.

3 Open your model and open the **PLC Coder App**. Click **Settings**, and then select **Verify Code in IDE**.

In your target IDE, you can see multiple test benches. Each test bench corresponds to a signal group.

Troubleshooting: Test Data Exceeds Target Data Size

If the test data from the multiple signal groups exceeds the maximum data size on your target, you can encounter compilation errors. If you encounter compilation errors when generating multiple test benches, try one of the following:

• Reduce the number of signal groups in the Signal Builder block and regenerate the test benches.

- Increase the simulation step size for the subsystem.

Limitations

When you are switching between signal groups, the model simulation time must remain the same for the entire simulation. Do not change the model simulation time.

See Also

Related Examples

- "Import and Verify Structured Text Code" on page 4-4

5

Code Generation Reports

Information in Code Generation Reports

The coder creates and displays a Traceability Report file when you select one or more of these options:

GUI Option	Command-Line Property	Description
Generate traceability report	PLC_GenerateReport	Specify whether to create code generation report.
Generate model web view	PLC_GenerateWebview	Include the model web view in the code generation report to navigate between the code and model within the same window. You can share your model and generated code outside of the MATLAB environment.

In the Configuration Parameters dialog box, in the **Report** panel, you see these options.
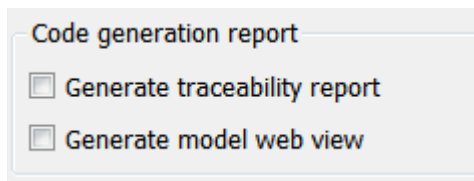
Note You must have a Simulink Report Generator™ license to generate traceability reports.

The coder provides the traceability report to help you navigate more easily between the generated code and your source model. When you enable code generation report, the coder creates and displays an HTML code generation report. You can generate reports from the Configuration Parameters dialog box or the command line. Traceability report generation is disabled when generating Ladder Diagrams from Stateflow chart. See "Traceability Report Limitations" on page 13-33 . A typical traceability report looks something like this figure:

Code Generation Report — □ ×

← → ↻ Find: [   ] ↑ ↓ Match Case

Traceability Report
Code Metrics Report

Generated Files

SimpleSubsystem.exp

Traceability Report for plcdemo_simple_subsystem

Table of Contents

1. Eliminated / Virtual Blocks
2. Traceable Simulink Blocks / Stateflow Objects / MATLAB Functions
   - plcdemo_simple_subsystem/SimpleSubsystem

Eliminated / Virtual Blocks

Block Name	Comment
<S1>/U	Inport
<S1>/Y	Outport

Traceable Simulink Blocks / Stateflow Objects / MATLAB Functions

Subsystem: plcdemo_simple_subsystem/SimpleSubsystem

Object Name	Code Location
<S1>/Gain	SimpleSubsystem.exp:43
<S1>/Sum	SimpleSubsystem.exp:45
<S1>/Unit Delay	SimpleSubsystem.exp:40, 46, 50

OK   Help

Create Code Generation Report

In this section...
"Generate a Traceability Report" on page 5-4
"Limitation" on page 5-6

Generate a Traceability Report

Generate a Simulink PLC Coder code generation report by using the Configuration Parameters dialog box:

1. Verify that the model is open.
2. Open the **PLC Coder** app. Click the **PLC Code** tab.
3. Click **Settings** and navigate to the **PLC Code Generation** pane.
4. To enable report generation, select the **Report > Generate traceability report** check box.
5. To open the code generation report upon completion of code generation, select the **Report> Open report automatically** check box. Click **OK**.
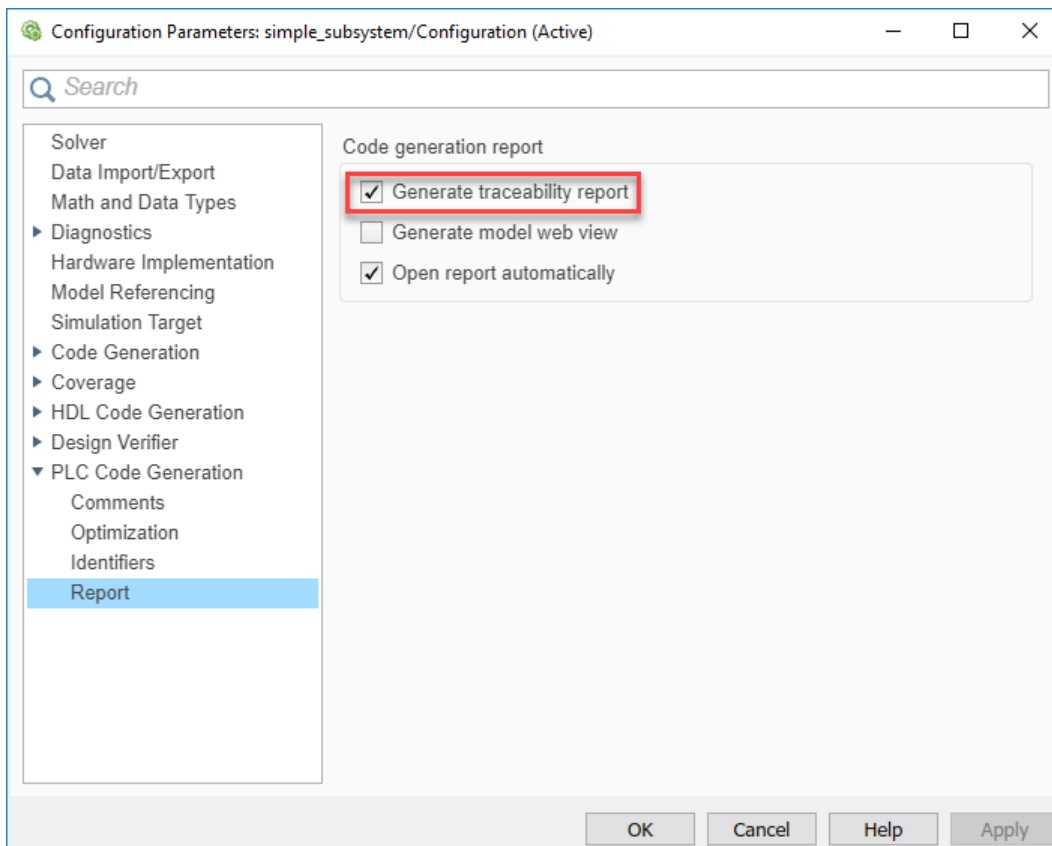
6. Click **Generate PLC Code** to initiate code and report generation. The coder generates HTML report files as part of the code generation process. THE HTML report opens.
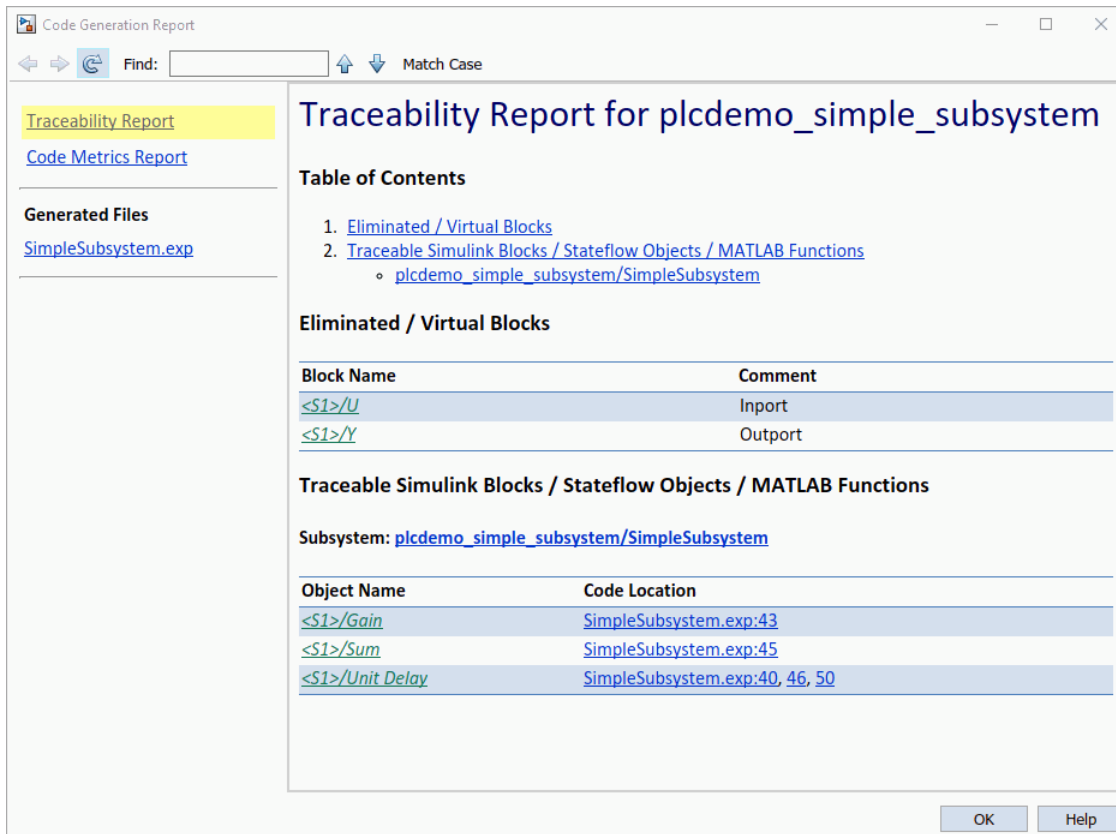
- The **Traceability Report** section enables you to account for **Eliminated / Virtual Blocks** that are untraceable, versus the listed **Traceable Simulink Blocks / Stateflow Objects / MATLAB Scripts**, providing a complete mapping between model elements and code.

- The **Static Code Metrics Report** section provides generated code statistics. Metrics are estimated from static analysis of the generated code.

In the **Generated Files** section, you can click the names of source code files generated from your model to view their contents in a MATLAB web browser window. In the displayed source code:

- Global variable instances are hyperlinked to their definitions.

- You can use the hyperlinks within the displayed source code to view the blocks or subsystems from which the code was generated. Click the hyperlinks to view the relevant blocks or subsystems in a Simulink model window.

- You can view the generated code for a block in the model. To highlight a block's generated code in the HTML report, click the block and in the **PLC Coder** app, **REVIEW RESULTS** pane, click **Navigate to Code**.

For more information, see:

- "Trace Simulink Model Elements in Generated Code" on page 6-8
- "Trace Stateflow Elements in Generated Code" on page 6-11

Limitation

If you generate a code generation report for a model, and then change the model, the report becomes invalid. To keep your code generation report current, after modifying the source model, regenerate code and the report. If you close and then reopen a model, regenerate the report.

Model Web View in Code Generation Report

Model Web Views

To review and analyze the generated code, it is helpful to navigate between the code and model. You can include a web view of the model within the HTML code generation report. You can then share your model and generated code outside of the MATLAB environment. You must have a Simulink Report Generator license to include a Web view (Simulink Report Generator) of the model in the code generation report.

Browser Requirements for Web Views

Web views require a web browser that supports Scalable Vector Graphics (SVG). Web views use SVG to render and navigate models.

You can use these web browsers:

- Mozilla® Firefox® Version 1.5 or later, which has native support for SVG. To download the Firefox browser, go to `https://www.mozilla.org/en-US/firefox/`.
- Apple Safari
- Microsoft® Internet Explorer® that has Adobe® SVG Viewer plugin.

Note Web views do not support Microsoft Internet Explorer 9.

Generate HTML Code Generation Report with Model Web View

This example shows how to create an HTML code generation report that includes a web view of the model diagram.

1 Open the `plcdemo_simple_subsystem` model.
2 Open the **PLC Coder** app. Click the **PLC Code** tab.
3 Click **Settings** and navigate to the Code Generation pane.
4 To enable report generation, select the **Report>Generate traceability report** check box.
5 To enable model web view, select the **Report > Generate model web view** check box.
6 Click **OK**.
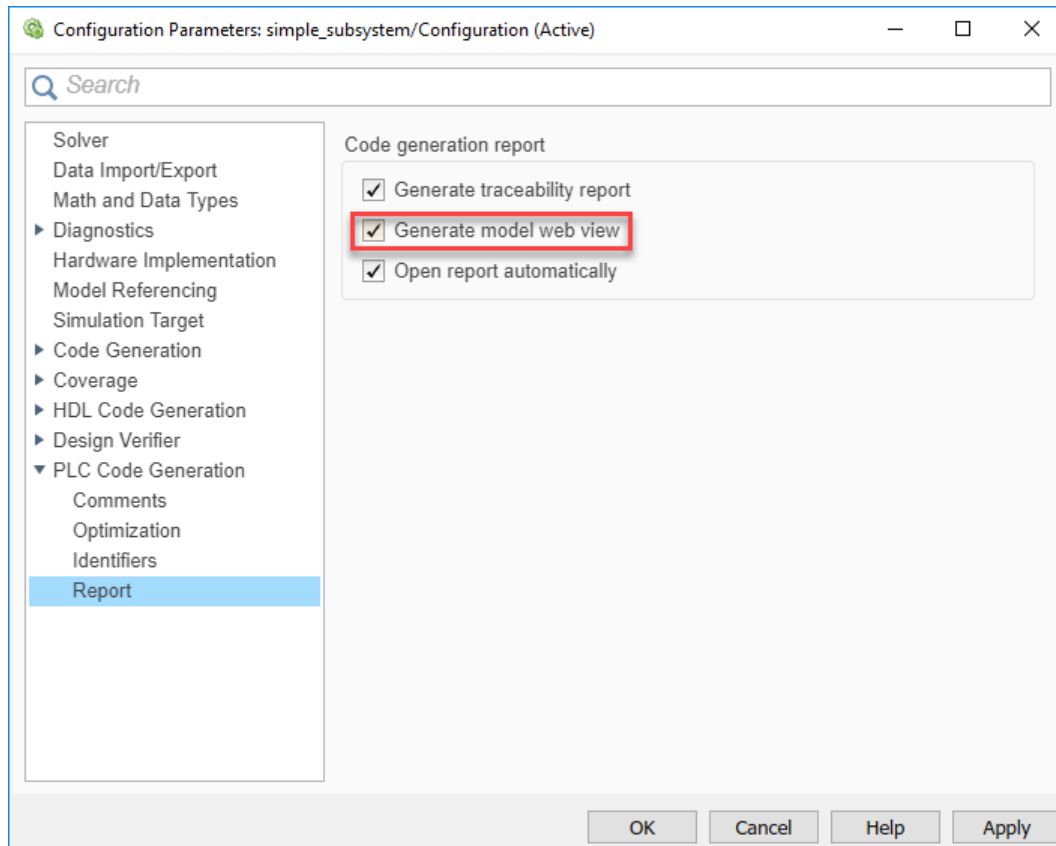
7 On the **PLC Code** tab, click **Generate PLC Code** to initiate code and report generation. The code generation report for the top model opens in a MATLAB web browser.

8 In the left navigation pane, select a source code file. The corresponding traceable source code is displayed in the right pane and includes hyperlinks.

10   Click a hyperlink in the code. The model web view displays and highlights the corresponding block in the model.

11   To go back to the code generation report for the top model, at the top of the left navigation pane, click the **Back** button until the report for the top model is displayed.

For more information about navigating between the generated code and the model diagram, see "Trace Simulink Model Elements in Generated Code" on page 6-8.

Model Web View Limitations

When you are using the model web view, the HTML code generation report has these limitations:

- Code is not generated for virtual blocks. In the model web view, if you click a virtual block, the code generation report clears highlighting in the source code files.
- Stateflow truth tables, events, and links to library charts are not supported.
- Searching in the code generation report does not find or highlight text.
- In a subsystem build, the traceability hyperlinks of the root-level inports and outports blocks are disabled.

- If you navigate from the actual model diagram (not the model web view in the report), to the source code in the HTML code generation report, the model web view is disabled and not visible. To enable the model web view, open the report again.

Generate Static Code Metrics Report

The Static Code Metrics report is a section included in the HTML code generation report. The report provides generated code statistics. The report is generated when you select **Generate Traceability Report** in the Configuration Parameters dialog box. You can use the Static Code Metrics Report to evaluate the generated PLC code before implementation in your IDE. You can use information in the report to:

- Find the number of files and lines of code in each file.
- Estimate the number of lines of code and stack usage per function.

For more information, see "Working with the Static Code Metrics Report" on page 5-14.

This example shows how to generate a code generation report that contains the Static Code Metrics section.

1   Open the `AirportConveyorBeltControlSystemExample`.

    `openExample('plccoder/AirportConveyorBeltControlSystemExample')`

    Select the `Controller` subsystem.

2   Open the **PLC Coder** app. Click the **PLC Code** tab.

3   Click **Settings** and navigate to the Code Generation pane.

4   To enable report generation, select **Report > Generate traceability report**.

5   Click **OK**.
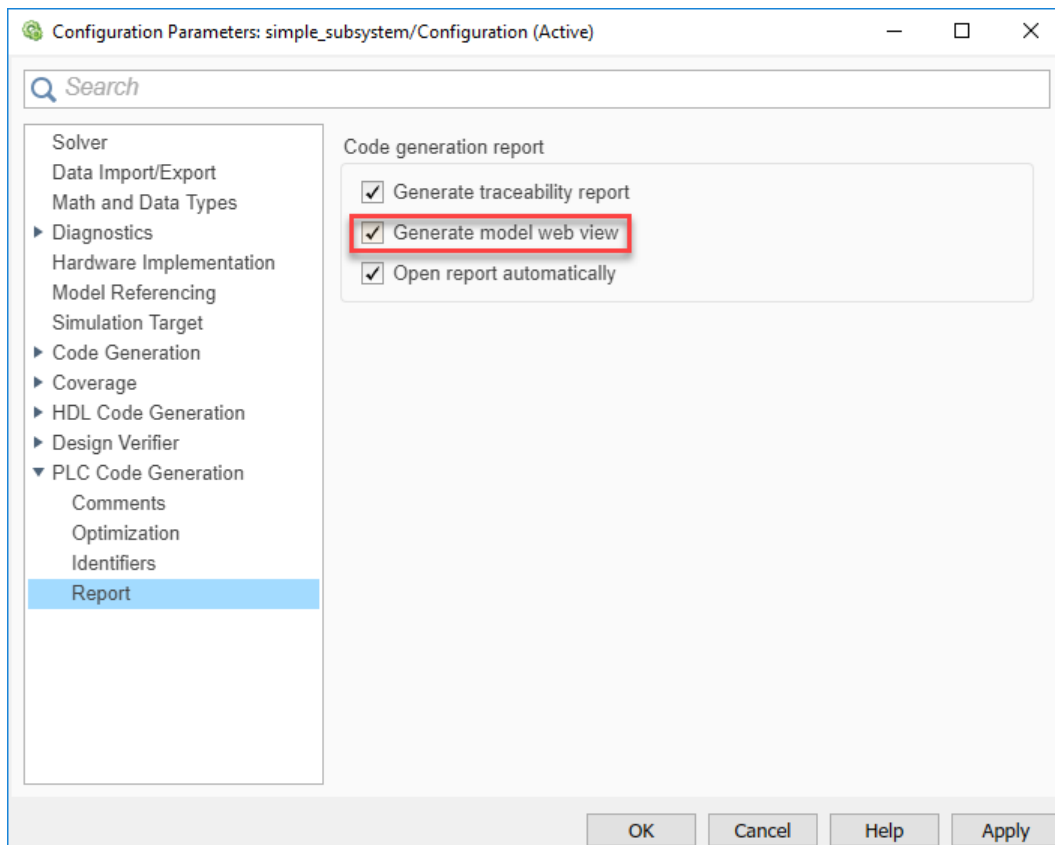
6 Click **Generate PLC Code** to initiate code and report generation. The code generation report for the top model opens in a MATLAB web browser.

7 On the left navigation pane, in the **Contents** section, select **Code Metrics report**.
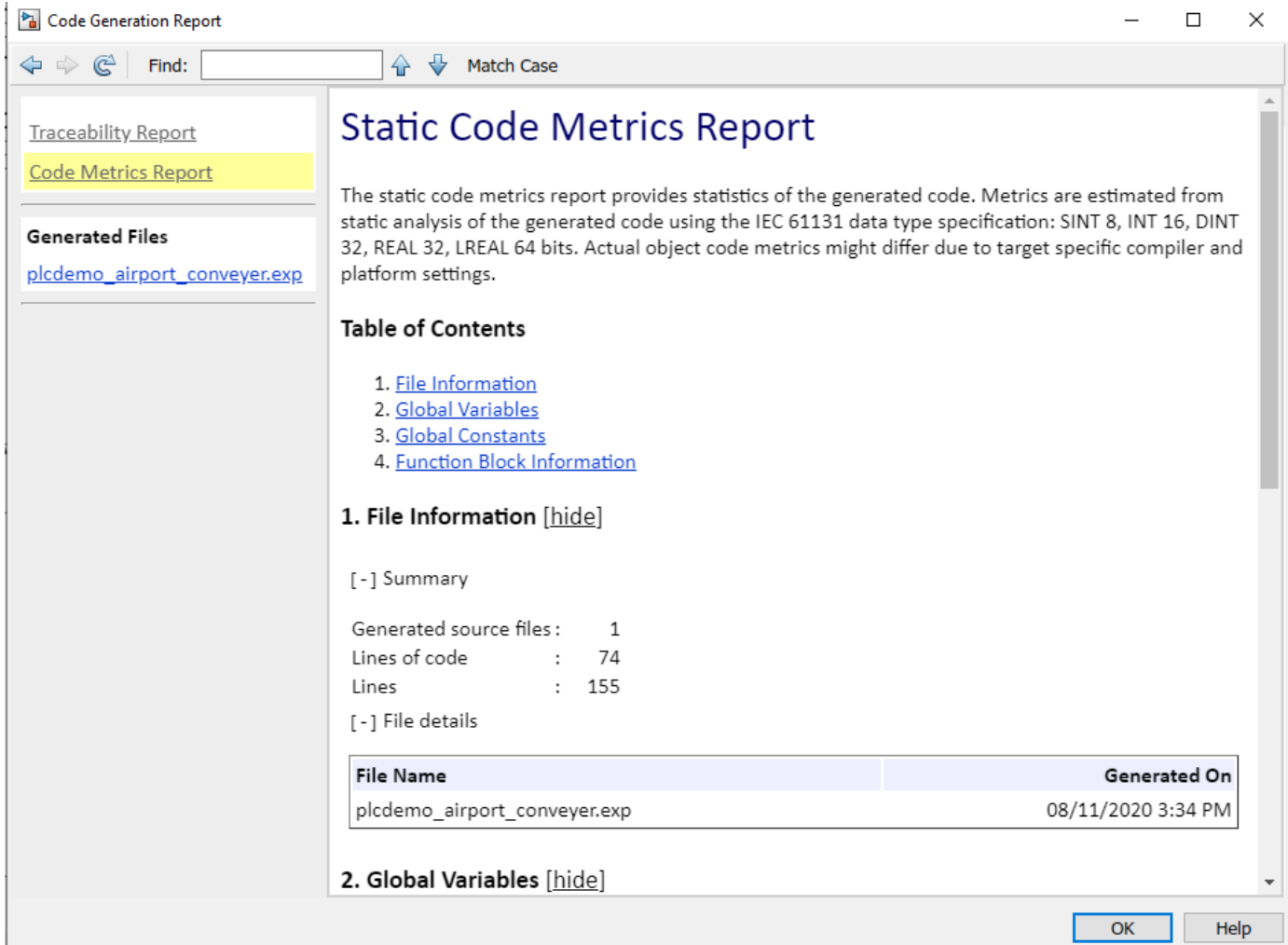
8 To see the generated files and how many lines of code are generated for each file, look at the **File Information** section.

9 To view the global constants in their generated code and their size, see the **Global Constants** section.

3. Global Constants [hide]

Global constants defined in the generated code.

Global Constant	Size (bytes)
c_Controller_IN_NO_ACTIVE_C	1
Controller_IN_IntervalStop	1
Controller_IN_Run_F1	1
Controller_IN_Stop_F1	1
Controller_IN_Run_MC	1
Controller_IN_Stop_MC	1
SS_INITIALIZE	1
SS_STEP	1
Total	**8**

10 To view the function metrics such as stack size, number of inputs, and number of outputs, see the **Function Block Information** section.

4. Function Block Information [hide]

Function block metrics in table format. "Number of Locals" includes state and temporary variables (does not include other function block instance variables).

Name	Self Stack Size (bytes)	Lines of Code	Lines	Number of Inputs	Number of Outputs	Number of Locals
Controller	15	74	155	6	2	5

Working with the Static Code Metrics Report

In this section...
"Workflow for Static Code Metrics Report" on page 5-14
"Report Contents" on page 5-14
"Function Block Information" on page 5-15

You can use the information in the Static Code Metrics Report to assess the generated code and make model changes before code implementation in your target IDE.

Before starting, familiarize yourself with potential code limitations of your IDE. For example, some IDEs have limits on the number of variables or lines of code in a function block.

For detailed instructions on generating the report, see "Generate Static Code Metrics Report" on page 5-11.

Workflow for Static Code Metrics Report

This workflow is the basic workflow for using the Static Code Metrics Report with your model.
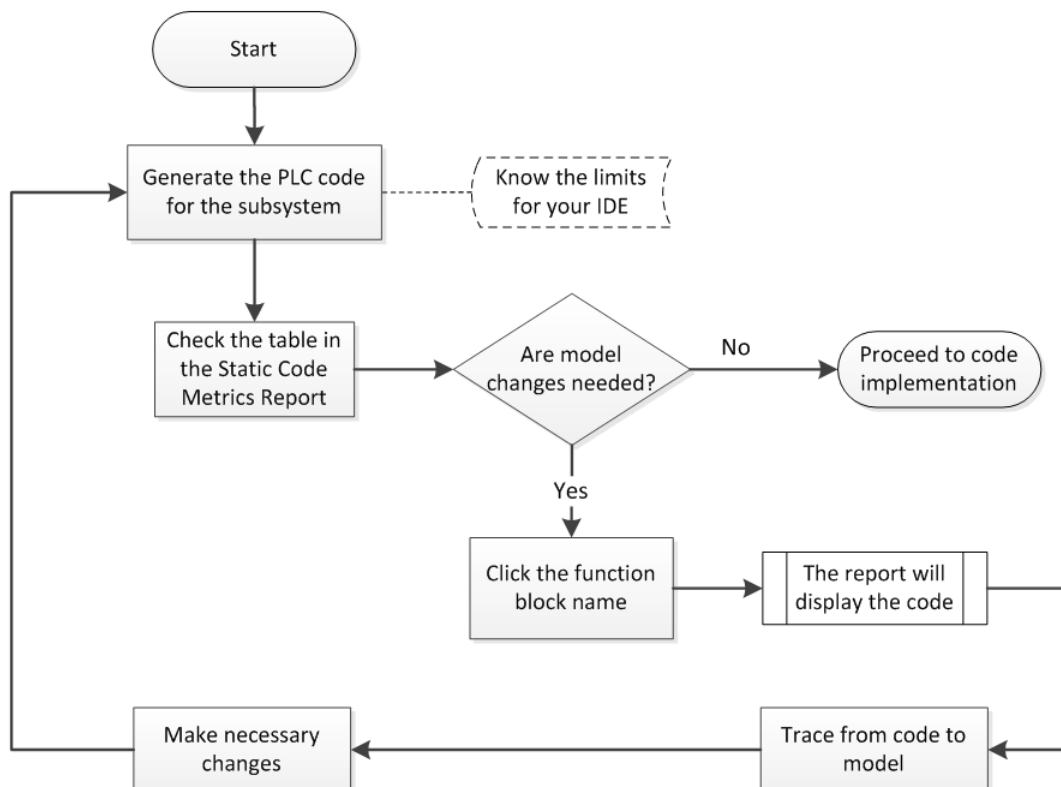
Report Contents

The Static Code Metrics Report is divided into these sections:

- **File Information**: Reports high-level information about generated files, such as lines and lines of code.
- **Global Variables**: Reports information about global variables defined in the generated code.
- **Global Constants**: Reports information about global constants defined in the generated code.
- **Function Block Information**: Reports a table of metrics for each function block generated from your model.

Function Block Information

You can use the information in the Function Block Information table to assess the generated code before implementation in your IDE. The leftmost column of the table lists function blocks with hyperlinks. Clicking a function block name leads you to the function block location in the generated code. You can then trace from the code to the model. For more information, see "Verify Generated Code by Using Code Tracing" on page 6-2.

View Requirements Links from Generated Code

For requirements reviews, design reviews, traceability analysis, or project documentation, you can create links to requirements documents from your model with the Requirements Toolbox™ software. If your model has links to requirements documents, you can also view the links from the generated code.

Note The requirement links must be associated with a model object. If requirements links are associated with the code in a MATLAB Function block, they do not appear in generated code comments.

To view requirements from generated code:

1 From your model, create links to requirements documents.

See, "Requirements Management Interface" (Requirements Toolbox).

2 For the subsystem for which you want to generate code, specify the following configuration parameters.

Option	Purpose
Include comments on page 13-13	Model information must appear in code comments.
Generate traceability report on page 13-33	After code is generated, a Code Generation Report must be produced.

3 Generate code.

The code generation report opens. The links to requirements documents appear in generated code comments. When you view the code in the code generation report, you can open the links from the comments.

6

Code Traceability

- "Verify Generated Code by Using Code Tracing" on page 6-2
- "Trace Simulink Model Elements in Generated Code" on page 6-8
- "Trace Stateflow Elements in Generated Code" on page 6-11

Verify Generated Code by Using Code Tracing

Code tracing (traceability) uses hyperlinks to navigate between a line of generated code and its corresponding elements in a model. To find the lines of code and their corresponding elements, you can use the PLC Coder App **Navigate To Code** on an element in the model. This two-way navigation is bidirectional traceability.

Using code tracing, you can:

- Verify that the generated code is as you expect. You can identify which model elements correspond to a line of code. You can track code from different elements that you have or have not reviewed.
- Verify that generated code meets design requirements. You can link requirements to model elements and use code tracing to verify that the generated code for a model element meets the design requirements.

When you generate code from a Simulink model, traceability information is embedded in the generated code, unless explicitly unspecified. The traceability information includes links for tracing between the generated source code and the model. You can view the generated code by using the code generation report.

The generated code includes these resources that support code tracing:

- Code element hyperlinks (indicated by underlining when you place your cursor over the code) to trace variables or types in the generated code to their declarations or definitions.
- Tags in code comments that identify elements in a model from which lines of code are generated.
- Line number hyperlinks that link to the model component from which the line of code was generated.

Traceable Elements

Bidirectional traceability is supported for Simulink blocks and these Stateflow elements:

- States
- Transitions
- State transition tables
- MATLAB functions. Traceability is not supported for external code that you call from a MATLAB function.
- Truth table blocks
- Graphical functions

- Simulink functions

Traceability in one direction is supported for these Stateflow elements:

- Events (code-to-model)

  Code-to-model traceability works for explicit events, but not implicit events. Clicking a hyperlink for an explicit event in the generated code highlights that item in the **Contents** pane of the Model Explorer.
- Junctions (model-to-code)

  Model-to-code traceability works for junctions with at least one outgoing transition. Right-clicking such a junction in the Stateflow Editor highlights the line of code that corresponds to the first outgoing transition for that junction.

For more information, see

Traceability in Generated Code

This example shows how to verify generated code by using the code generation report.

1   Open the example `GeneratingStructuredTextForAFeedForwardPIDControllerExample`.

   `openExample('plccoder/GeneratingStructuredTextForAFeedforwardPIDControllerExample')`
2   Open the **PLC Coder** app. Click the **PLC Code** tab.
3   Click **Settings** and navigate to the Code Generation pane.
4   To enable report generation, select **Report > Generate traceability report**.
5   Click **OK**.
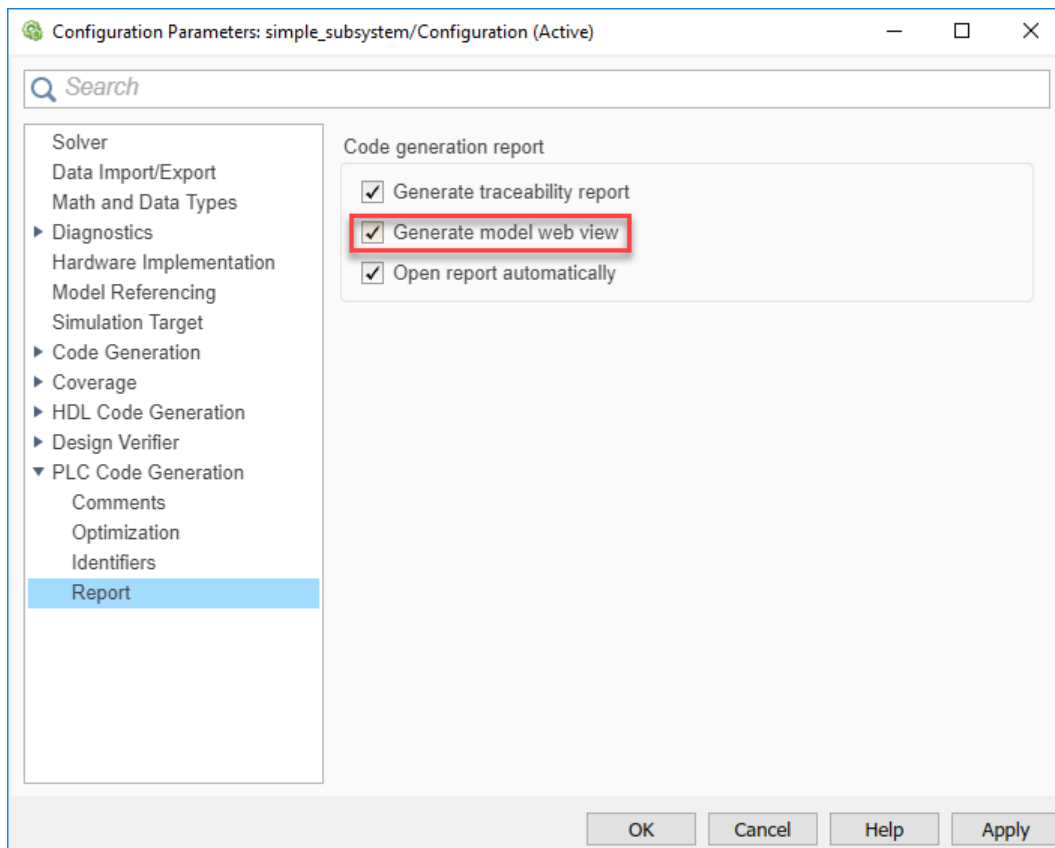
6. Click **Generate PLC Code** to initiate code and report generation. The code generation report for the top model opens in a MATLAB web browser.

7. In the left navigation pane, select the `plcdemo_pid_feedforward.exp` file.

8. Click a comment or line number hyperlink. The Simulink Editor displays and highlights the corresponding block in the model.
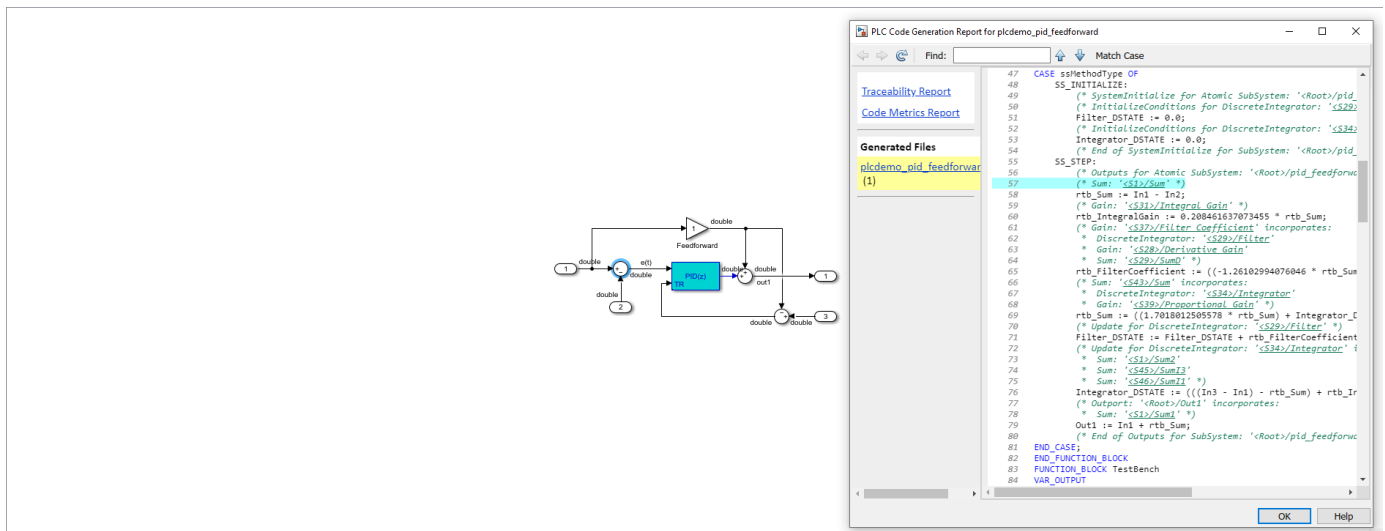
9 To highlight the generated code for a block in the model, select the block and in the **PLC Coder** tab, click **Navigate to Code**. The generated code for the block is highlighted in the HTML code generation report.

10 In the left navigation pane, you can click the **Back** button to go back to the previous code generation report.

Traceability Tags

A traceability tag appears in a comment above the corresponding line of generated code. The format of the tag is *<system>/block_name* .

- *system* is a unique number assigned by the Simulink engine.
- *block_name* is the name of the source block.

This code shows a tag comment above the generated line of code. A Sum block within a subsystem one level below the root level of the source model generates this code:

```
(* Sum: '<S1>/Sum' *)
rtb_Sum := In1 - In2;
```

Operator Traceability

The generated code provides traceability between operators in the generated code and Simulink blocks, Stateflow elements, or MATLAB Function blocks.

To verify the generated code by using operator traceability, in the generated code, click an operator hyperlink to highlight the source block in the model.

These operators are supported.

Operator Type	Operators
Arithmetic	+, -, *, /, %
	+=, -=, *=, /=, %=
	++, -- (prefix and postfix)
Logical	!, &&, ||
Relational	==, !=, <, >, <=, >=
Bit	~, |, ^, &, >>, <<
	&=, ^=, |=, <<=, >>=
Conditional	?:

These operators are not supported.

Operator Type	Operator Examples
Assignment operator	=
Member of and pointer operators	Array subscript: a[b]
	Address of and pointer dereference: &a, *a
	Member of: a.b, a->b

Operator Type	Operator Examples
Other operators	Parenthesis in function call: `foo(a,b)`
	Comma: `a, b`
	Scope resolution: `a::b`
	Cast: `type(a)`
	`new`, `new[]`
	`delete`, `delete[]`

Generate a Traceability Report from the Command Line

To generate a Simulink PLC Coder code generation report from the command-line code for the subsystem `plcdemo_simple_subsystem/SimpleSubsystem`:

1 Open a Simulink PLC Coder model, for example:

```
open_system('plcdemo_simple_subsystem');
```

2 Enable the code generation parameter `PLC_GenerateReport`. To view the output in the model web view, also enable `PLC_GenerateWebview`:

```
set_param('plcdemo_simple_subsystem', 'PLC_GenerateReport', 'on');
set_param('plcdemo_simple_subsystem', 'PLC_GenerateWebView', 'on');
```

3 Generate the code.

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

A traceability report is displayed. In your model, a **View diagnostics** hyperlink appears at the bottom of the model window. Click this hyperlink to open the Diagnostic Viewer window.

If the model web view is also enabled, that view is displayed.

Traceability Limitations

These limitations apply to reports generated by the Simulink PLC Coder software:

- Under the following conditions, model-to-code traceability is disabled for a block if the block name contains:

  - A single quote (').
  - An asterisk (*) that causes a name-mangling ambiguity relative to other names in the model. This name-mangling ambiguity occurs if in a block name or at the end of a block name, an asterisk precedes or follows a slash (/).
  - The character ÿ (`char(255)`).

- If a block name contains a newline character (\n), the generated code comment for the block path hyperlink replaces the newline character with a space for readability.
- You cannot trace blocks representing these types of subsystems to generated code:

  - Virtual subsystems
  - Masked subsystems
  - Nonvirtual subsystems for which code has been removed due to optimization

If you cannot trace a subsystem at subsystem level, you can trace individual blocks within the subsystem.

- If you open a model on a platform that is different from the platform used to generate code, you cannot use model-to-code and code-to-model traceability.

- Inline traceability is not available for files that are generated in `shared_utils` folder.

See Also

More About

- "Trace Simulink Model Elements in Generated Code" on page 6-8
- "Trace Stateflow Elements in Generated Code" on page 6-11

Trace Simulink Model Elements in Generated Code

To verify the generated code, Simulink PLC Coder provides bidirectional traceability between the Simulink model and the generated code. For traceability, you can use either method:

- Code-to-model: In comment lines in the generated code, the generated code displays these hyperlinks:

  - Block/subsystems names
  - Line numbers
  - Operators

  To highlight the corresponding block or subsystem in the Simulink Editor, click the hyperlinks.

- Model-to-code: You can select a single block of a model in the Simulink Editor and navigate to the corresponding generated code.

Code-To-Model Traceability

This example shows how to use hyperlinks for tracing code-to-model elements:

1 Open the example `GeneratingStructuredTextForAFeedForwardPIDControllerExample`.

```
openExample('plccoder/GeneratingStructuredTextForAFeedforwardPIDControllerExample')
```

2 Open the **PLC Coder** app. Click the **PLC Code** tab.

3 Click **Settings** and navigate to the Code Generation pane.

4 To enable report generation, select **Report** > **Generate traceability report**.

5 Click **OK**.

6 Click **Generate PLC Code** to initiate code and report generation. The code generation report for the top model opens in a MATLAB web browser.

7 In the left navigation pane, select the `plcdemo_pid_feedforward.exp` file.

8 Click the hyperlink on line 73. In the model window, the corresponding `Sum` block is highlighted.
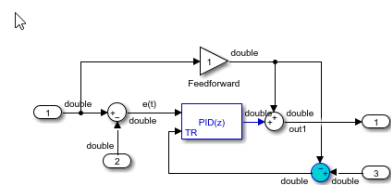
Model-to-Code Traceability

This example shows how to trace model elements to their corresponding generated code:

1 Open the example `GeneratingStructuredTextForAFeedForwardPIDControllerExample`.

```
openExample('plccoder/GeneratingStructuredTextForAFeedforwardPIDControllerExample')
```

2 Open the **PLC Coder** app. Click the **PLC Code** tab.

3 Click **Settings** and navigate to the Code Generation pane.

4　To enable report generation, select **Report > Generate traceability report**.

5　Click **OK**.

6　Click **Generate PLC Code** to initiate code and report generation. The code generation report for the top model opens in a MATLAB web browser.

7　Select the highlighted Sum block in the image,and in the **PLC Coder** tab, click **Navigate to Code**. The generate code for the block is highlighted in the HTML code generation report.

See Also

More About

- "Verify Generated Code by Using Code Tracing" on page 6-2
- "Trace Stateflow Elements in Generated Code" on page 6-11

Trace Stateflow Elements in Generated Code

To verify the generated code for your Stateflow elements, you can trace Stateflow elements in your model to the generated code by using these types of navigation:

- Code-to-model: Trace generated code back to the model by clicking hyperlinks in the comments or the hyperlinked line numbers, which highlights the corresponding model element in the Simulink Editor.
- Model-to-code: Trace the model elements in the Simulink Editor to corresponding lines in generated code by right-clicking the model element and navigating to the generated code. This traceability is not supported for some Stateflow elements in Code view.

Inline Traceability for Stateflow Elements

Inline traceability refers to the line-level traceability available in the generated code. You can click the hyperlinked line numbers to trace the corresponding Stateflow elements.

This example shows how to use hyperlinks for tracing code-to-Stateflow elements:

1 Open the example `AirportConveyorBeltControlSystemExample`.

```
openExample('plccoder/AirportConveyorBeltControlSystemExample')
```

2 Open the **PLC Coder** app. Click the **PLC Code** tab.

3 Click **Settings** and navigate to the Code Generation pane.

4 To enable report generation, select **Report > Generate traceability report**.

5 Click **OK**.
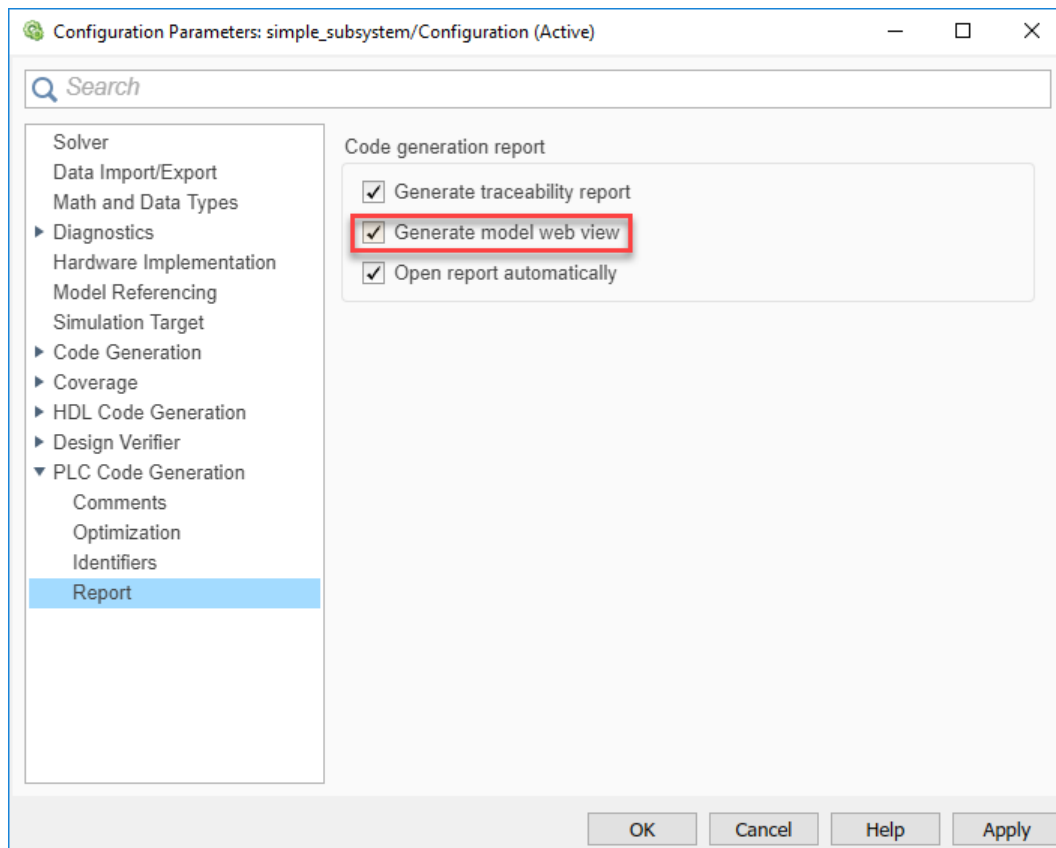
6. Click **Generate PLC Code** to initiate code and report generation. The code generation report for the top model opens in a MATLAB web browser.

7. In the left navigation pane, select the `plcdemo_airport_conveyor.exp` file.

8. Click the hyperlink on line 65. In the Stateflow chart, the corresponding `Stop_F1` state is highlighted.

Trace States and Transitions to Code

This example shows how to trace Stateflow states and transitions to the generated code:

1. Open the example `AirportConveyorBeltControlSystemExample`.

   ```
   openExample('plccoder/AirportConveyorBeltControlSystemExample')
   ```

2 Open the **PLC Coder** app. Click the **PLC Code** tab.

3 Click **Settings** and navigate to the Code Generation pane.

4 To enable report generation, select **Report > Generate traceability report**.

5 Click **OK**.

6 Click **Generate PLC Code** to initiate code and report generation. The code generation report for the top model opens in a MATLAB web browser.

7 In the **Controller** subsystem, **Control** block, select the Run_F1 state. In the **PLC Coder** tab, click **Navigate to Code**. The generated code for the state is highlighted in the HTML code generation report.

See Also

More About

- "Trace Simulink Model Elements in Generated Code" on page 6-8
- "Verify Generated Code by Using Code Tracing" on page 6-2

Working with Tunable Parameters in the Simulink PLC Coder Environment

Block Parameters in Generated Code

To control how the block parameters appear in the generated code, you can either define the parameters as `Simulink.Parameter` objects in the MATLAB workspace or use the Model Parameter Configuration dialog box. For more information, see "Control Appearance of Block Parameters in Generated Code" on page 7-4.

Simulink PLC Coder exports tunable parameters as exported symbols and preserves the names of these parameters in the generated code. It does not mangle these names. As a result, if you use a reserved IDE keyword as a tunable parameter name, the code generation can cause compilation errors in the IDE. As a best practice, do not use IDE keywords as tunable parameter names.

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

Target IDE	Parameter Storage Class			
	`Model default`	`ExportedGlobal`	`ImportedExtern`	`Imported-ExternPointer`
CoDeSys 2.3	Local function block variables	Global variable	Variable is not defined in generated code and expected to be defined externally.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
CoDeSys 3.3	Local function block variables	Global variable	Variable is not defined in generated code and expected to be defined externally.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
CoDeSys 3.5	Local function block variables	Global variable	Variable is not defined in generated code and expected to be defined externally.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
B&R Automation Studio 3.0	Local function block variable	Local function block variable	Local function block variable.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
B&R Automation Studio 4.0	Local function block variable	Local function block variable	Local function block variable.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`
Beckhoff TwinCAT 2.11	Local function block variable	Global variable	Variable is not defined in generated code and expected to be defined externally.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.

Target IDE	Parameter Storage Class			
	Model default	**ExportedGlobal**	**ImportedExtern**	**Imported-ExternPointer**
KW-Software MULTIPROG 5.0	Local function block variable	Local function block variable	Local function block variable.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
Phoenix Contact PC WORX 6.0	Local function block variable	Global variable	Variable is not defined in generated code and expected to be defined externally.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
RSLogix 5000 17, 18: AOI	AOI local tags	AOI input tags	AOI input tags.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
RSLogix 5000 17, 18: Routine	Instance fields of program UDT tags	Program tags	Variable is not defined in generated code and expected to be defined externally.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
Siemens SIMATIC STEP 7	Local function block variable	Global variable	Variable is not defined in generated code and expected to be defined externally.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
Siemens TIA Portal	Local function block variable	Global variable	Variable is not defined in generated code and expected to be defined externally.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`
Generic	Local function block variable	Global variable	Variable is not defined in generated code and expected to be defined externally.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
PLCopen	Local function block variable	Global variable	Variable is not defined in generated code and expected to be defined externally.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.

Control Appearance of Block Parameters in Generated Code

Unless you use constants for block parameters in your model, they appear in the generated code as variables. You can choose how these variables appear in the generated code. For more information, see "Block Parameters in Generated Code" on page 7-2.

To control how the block parameters appear in the generated code:

1   Use variables instead of constants for block parameters.
2   Define these parameters in the MATLAB workspace in one of the following ways:

   • Use a MATLAB script to create a `Simulink.Parameter` object. Run the script every time that the model loads.

      Simulink stores `Simulink.Parameter` objects outside the model. You can then share `Simulink.Parameter` objects between multiple models.

   • Use the Model Configuration Parameters dialog box to make the parameters tunable.

      Simulink stores global tunable parameters specified using the Configuration Parameters dialog box with the model. You cannot share these parameters between multiple models.

Note The MATLAB workspace parameter value must be of the same data type as used in the model. Otherwise, the value of the variable in the generated code is set to zero. See "Workspace Parameter Data Type Limitations" on page 19-4.

Configure Tunable Parameters with Simulink.Parameter Objects

This example shows how to create and modify a `Simulink.Parameter` object.

The model `plcdemo_tunable_params_slparamobj` illustrates these steps. The model contains a Subsystem block `SimpleSubsystem` that has three Gain blocks with tunable parameters, K1, K2, and K3.

1   Write a MATLAB script that defines the tunable parameters.

   The following script `setup_tunable_params.m` creates the constants K1, K2, and K3 as `Simulink.Parameter` objects, assigns values, and sets the storage classes for these constants. For more information on the storage classes, see "Block Parameters in Generated Code" on page 7-2.

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

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

% tunable parameter mapped to global const
K3 = Simulink.Parameter;
K3.Value = 0.3;
```

```
K3.CoderInfo.StorageClass = 'Custom';
K3.CoderInfo.CustomStorageClass = 'Const';
```

2 Specify that the script `setup_tunable_params.m` must execute before the model loads and that the MATLAB workspace must be cleared before the model closes.

   a In the model window, go to the **Modeling** tab and select **Model Properties** from the **Model Settings** drop-down.

   b In the Model Properties dialog box, on the **Callbacks** tab, select `PreLoadFcn`. Enter `setup_tunable_params` for **Model pre-load function**.
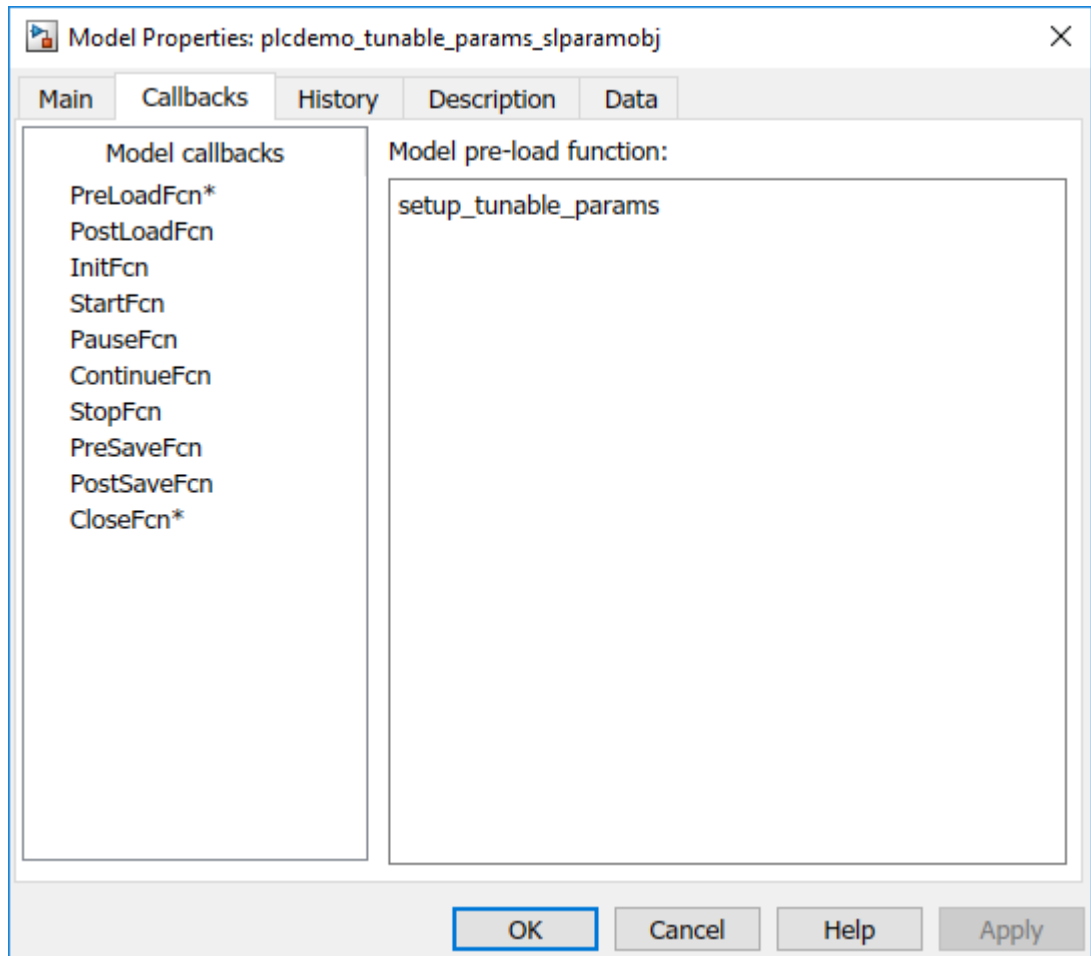
   c On the **Callbacks** tab, select `CloseFcn`. Enter `clear K1 K2 K3;` for **Model close function**.

Every time that you open the model, the variables K1, K2, and K3 are loaded into the base workspace. You can view the variables and their storage classes in the Model Explorer.

3 Generate code and inspect it.

Variable	Storage Class	Generated Code (3S CoDeSys 2.3)

K1	`Model default`	K1 is a local function block variable.
		
		`FUNCTION_BLOCK SimpleSubsystem`
		`.`
		`.`
		`VAR`
		`    K1: LREAL := 0.1;`
		`    .`
		`    .`
		`END_VAR`
		`.`
		`.`
		`END_FUNCTION_BLOCK`

K2	`ExportedGlobal`	K2 is a global variable.
		
		`VAR_GLOBAL`
		`    K2: LREAL := 0.2;`
		`END_VAR`

K3	`CoderInfo.CustomStorageClass` set to `Const`.	K3 is a global constant.
		
		`VAR_GLOBAL CONSTANT`
		`    SS_INITIALIZE: SINT := 0;`
		`    K3: LREAL := 0.3;`
		`    SS_STEP: SINT := 1;`
		`END_VAR`

Make Parameters Tunable Using Configuration Parameters Dialog Box

This example shows how to make parameters tunable using the Model Configuration Parameters dialog box.

The model `plcdemo_tunable_params` illustrates these steps. The model contains a Subsystem block `SimpleSubsystem` that has three Gain blocks with tunable parameters, K1, K2, and K3.

1. Specify that the variables K1, K2, and K3 must be initialized before the model loads and that the MATLAB workspace must be cleared before the model closes.

   a. In the **Modeling** tab and select **Model Properties** from the **Model Settings** drop-down.

   b. In the Model Properties dialog box, on the **Callbacks** tab, select `PreLoadFcn`. Enter `K1=0.1; K2=0.2; K3=0.3;` for **Model pre-load function**.

   c. On the **Callbacks** tab, select `CloseFcn`. Enter `clear K1 K2 K3;` for **Model close function**.

2. On the **Modeling** tab and select **Model Settings** to open the Configuration Parameters dialog box.

3. Navigate to **Optimization** pane. Specify that all parameters must be inlined in the generated code. Select `Inlined` for **Default Parameter Behavior**.

4. To override the inlining and make individual parameters tunable, click **Configure**. In the Model Parameter Configuration dialog box, from the **Source list**, select `Referenced workspace variables`.

5. **Ctrl**+select the parameters and click **Add to table >>**.

By default, this dialog box sets all parameters to the `SimulinkGlobal` storage class. Set the **Storage class** and **Storage type qualifier** as shown in this figure. For more information on the storage classes, see "Block Parameters in Generated Code" on page 7-2.
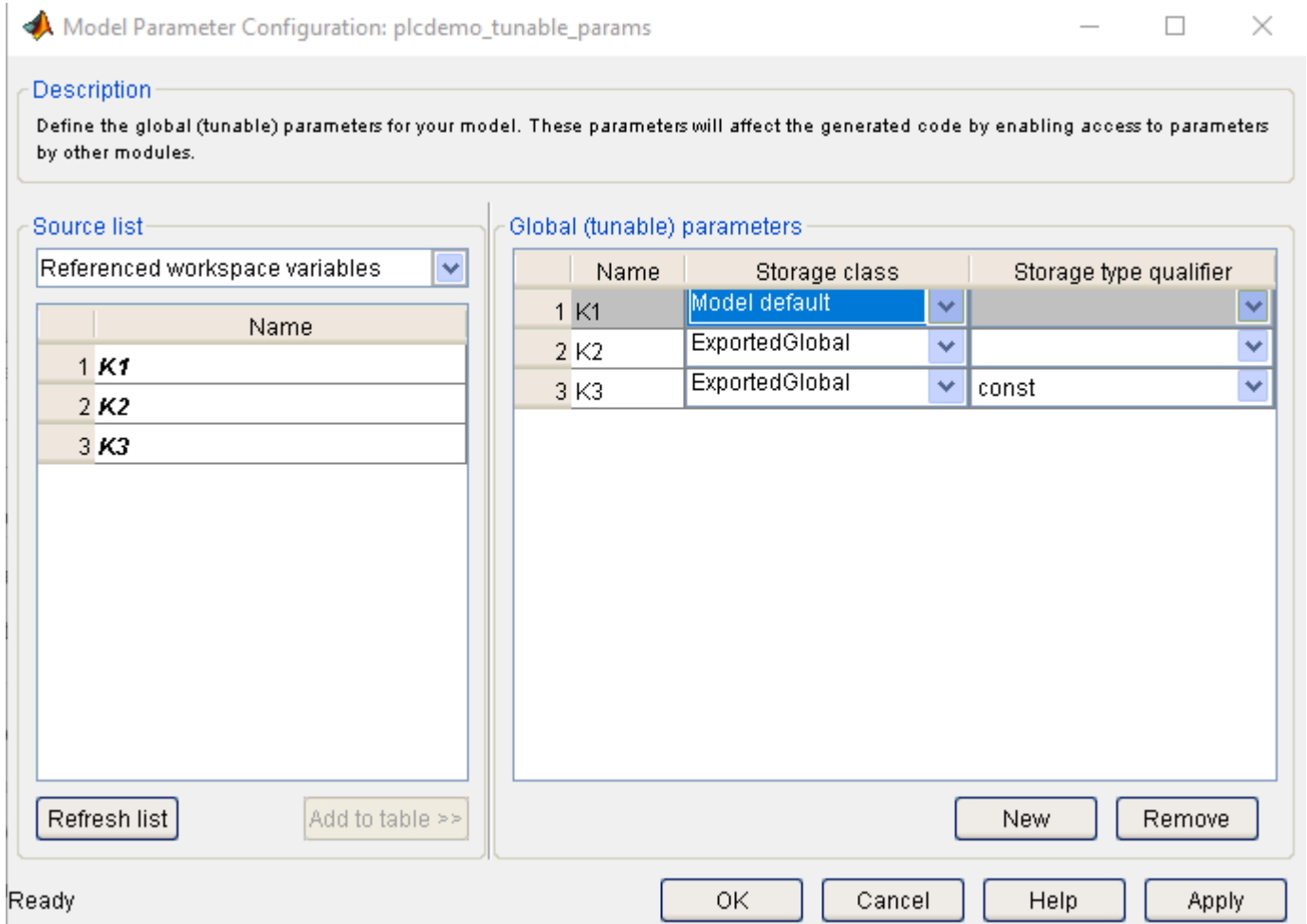
6 Generate code and inspect it.

Variable	Storage Class	Generated Code (3S CoDeSys 2.3)

K1	Model default	K1 is a local function block variable.
		
		FUNCTION_BLOCK SimpleSubsystem
		.
		.
		VAR
		    K1: LREAL := 0.1;
		    .
		    .
		END_VAR
		.
		.
		END_FUNCTION_BLOCK

Variable	Storage Class	Generated Code (3S CoDeSys 2.3)
K2	ExportedGlobal	K2 is a global variable. VAR_GLOBAL K2: LREAL := 0.2; END_VAR

K3	CoderInfo.CustomStorageClass and **Storage type qualifier** set to Const.	K3 is a global constant.
		
		VAR_GLOBAL CONSTANT
		   SS_INITIALIZE: SINT := 0;
		   K3: LREAL := 0.3;
		   SS_STEP: SINT := 1;
		END_VAR

Controlling Generated Code Partitions

Generate Global Variables from Signals in Model

If you want to generate a global variable in your code, use a global Data Store Memory block based on a `Simulink.Signal` object in your model.

1 Set up a data store in your model by using a Data Store Memory block.

2 Associate a `Simulink.Signal` object with the data store.

  a In the base workspace, define a `Simulink.Signal` object with the same name as the data store. Set the storage class of the object to `ExportedGlobal` or `ImportedExtern`.

  b Use the Model Data Editor to enable the **Data store name must resolve to Simulink signal object** parameter of the Data Store Memory block. To use the Model Data Editor in a model, on the **Modeling** tab, select **Model Data Editor** under the **Design** category. On the **Data Stores** tab, set the **Change View** drop-down to `Design`. Enable **Resolve** for the Data Store Memory block. For more information, see **Model Data Editor**.

3 In your model, attach the signals that you want to Data Store Read blocks that read from the data store and Data Store Write blocks that write to the data store.

The `Simulink.Signal` object that is associated with the global Data Store Memory block appears as a global variable in generated code.

Note If you follow this workflow for Rockwell Automation RSLogix 5000 AOIs, the generated code uses `INOUT` variables for the global data.

Control Code Partitions for Subsystem Block

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

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

For code generation from a subsystem with no inputs or outputs, you must set the **Function packaging** parameter of the block to `Reusable function`.

These topics use code generated with CoDeSys Version 2.3.

Control Code Partitions 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`	Generates a function with arguments that allows reuse of subsystem code when a model includes multiple instances of the subsystem.
`Nonreusable function`	The Simulink PLC Coder does not support `Nonreusable function` packaging. See, "Restrictions" on page 12-3.
`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` or `Auto`. This setting creates two function blocks, one for the parent, one for S2.
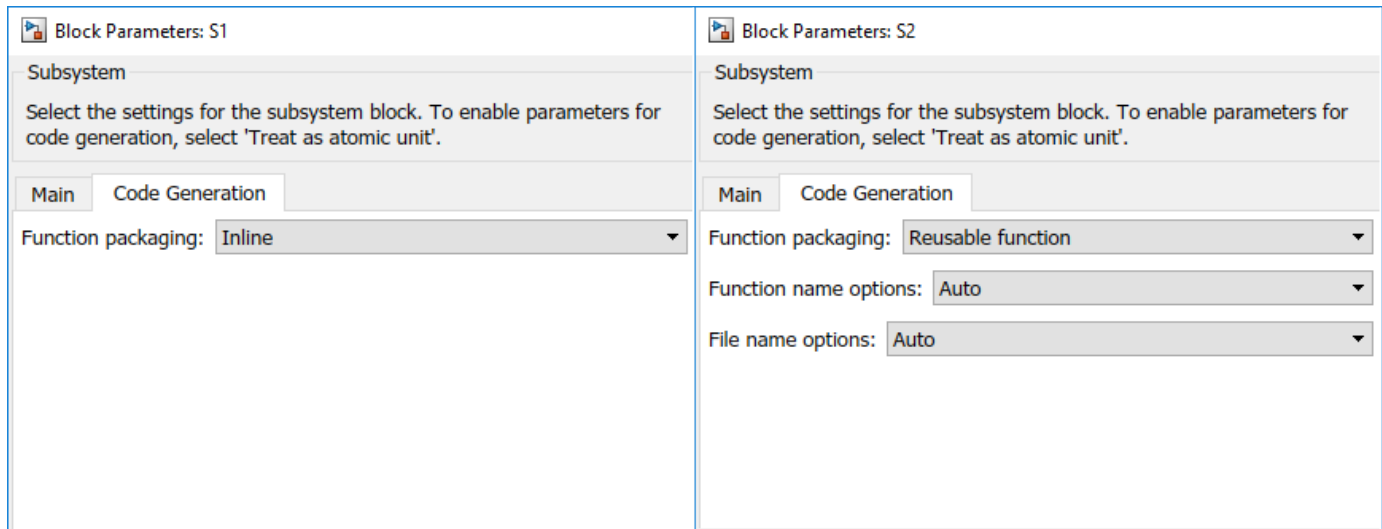
Changing the Name of a Subsystem

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, see `plcdemo_hierarchical_virtual_subsystem`:

1. Open the S1 subsystem block parameter dialog box.
2. If the **Treat as atomic unit** check box is not yet selected, select it.
3. Click the **Code Generation** tab.
4. Set **Function packaging** to `Reusable function`.
5. Set **Function name options** to `User specified`.
6. In the **Function name** field, specify a custom name. For example, type `my_own_subsystem`.
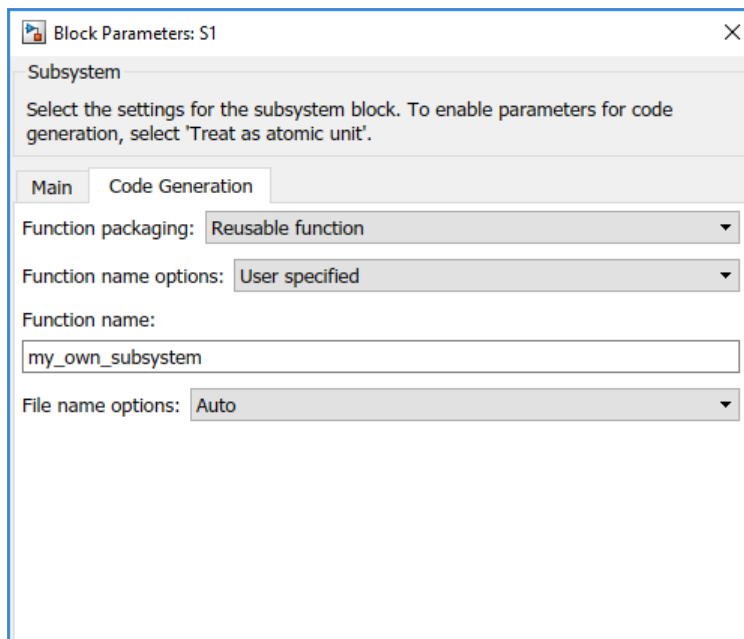
7 Save the new settings.

8 Generate code for the parent subsystem.

9 Observe the renamed function block.

```
FUNCTION_BLOCK my_own_subsystem
VAR_INPUT
    ssMethodType: SINT;
    U: LREAL;
END_VAR
```

One Function Block for Atomic Subsystems

The code for `plcdemo_simple_subsystem` is an example of generating code with one function block. The atomic subsystem for which you generate code does not contain 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
CASE ssMethodType OF
    SS_INITIALIZE:
        (* InitializeConditions for UnitDelay: '<S1>/Unit Delay' *)
        UnitDelay_DSTATE := 0.0;
    SS_STEP:
        (* Gain: '<S1>/Gain' incorporates:
         *  Sum: '<S1>/Sum'
         *  UnitDelay: '<S1>/Unit Delay' *)
        Y := (U - UnitDelay_DSTATE) * 0.5;
        (* Update for UnitDelay: '<S1>/Unit Delay' *)
        UnitDelay_DSTATE := Y;
END_CASE;
END_FUNCTION_BLOCK
VAR_GLOBAL CONSTANT
    SS_INITIALIZE: SINT := 0;
    SS_STEP: SINT := 1;
END_VAR
```

One Function Block for Virtual Subsystems

The `plcdemo_hierarchical_virtual_subsystem` example 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
    UnitDelay1_DSTATE: LREAL;
    UnitDelay_DSTATE: LREAL;
    UnitDelay_DSTATE_i: LREAL;
    UnitDelay_DSTATE_a: LREAL;
END_VAR
VAR_TEMP
    rtb_Gain_n: LREAL;
END_VAR
CASE ssMethodType OF
    SS_INITIALIZE:
        (* InitializeConditions for UnitDelay: '<S1>/Unit Delay1' *)
        UnitDelay1_DSTATE := 0.0;
```

Multiple Function Blocks for Nonvirtual Subsystems

The `plcdemo_hierarchical_subsystem` example 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 S1`, and `FUNCTION_BLOCK S2` components.

Function block for Hierarchical Subsystem

```
FUNCTION_BLOCK HierarchicalSubsystem
VAR_INPUT
    ssMethodType: SINT;
    In1: LREAL;
    In2: LREAL;
    In3: UINT;
    In4: LREAL;
END_VAR
```

Function block for S1

```
FUNCTION_BLOCK S1
VAR_INPUT
    ssMethodType: SINT;
    U: LREAL;
END_VAR
```

Function block for S2

```
FUNCTION_BLOCK S2
VAR_INPUT
    ssMethodType: SINT;
    U: LREAL;
END_VAR
```

Control Code Partitions for MATLAB Functions in Stateflow Charts

Simulink PLC Coder inlines MATLAB functions in generated code based on your inlining specifications. To specify whether to inline a function:

1 Right-click the MATLAB function and select **Properties**.
2 For **Function Inline Option**, select Inline if you want the function to be inlined. Select Function if you do not want the function to be inlined. For more information, see "Specify Properties of MATLAB Functions" (Stateflow).

However, Simulink PLC Coder does not follow your inlining specifications exactly in the following cases:

• If a MATLAB function accesses data that is local to the chart, it is inlined in generated code even if you specify that the function must not be inlined.

Explanation: The chart is converted to a function block in generated code. If the MATLAB function in the chart is converted to a Structured Text function, it cannot access the data of an instance of the function block. Therefore, the MATLAB function cannot be converted to a Structured Text function in generated code and is inlined.

• If a MATLAB function has multiple outputs and you specify that the function must not be inlined, it is converted to a function block in generated code.

Explanation: A Structured Text function cannot have multiple outputs, therefore the MATLAB function cannot be converted to a Structured Text function.

The following simple example illustrates the different cases. The model used here has a Stateflow chart that contains four MATLAB functions fcn1 to fcn4.

Here is the model.
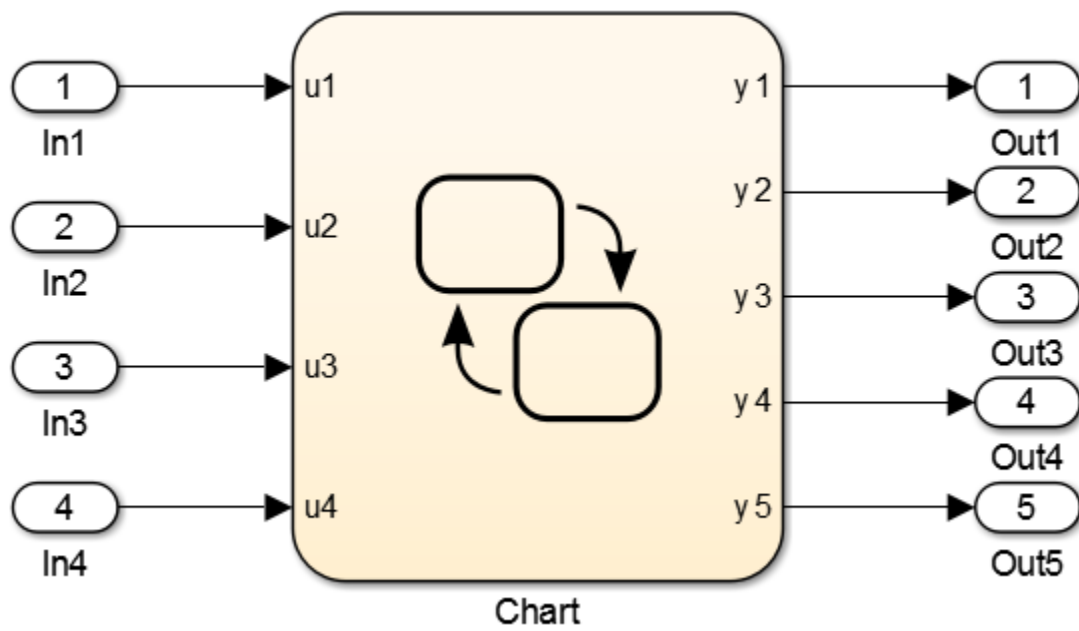
Here is the Stateflow chart.
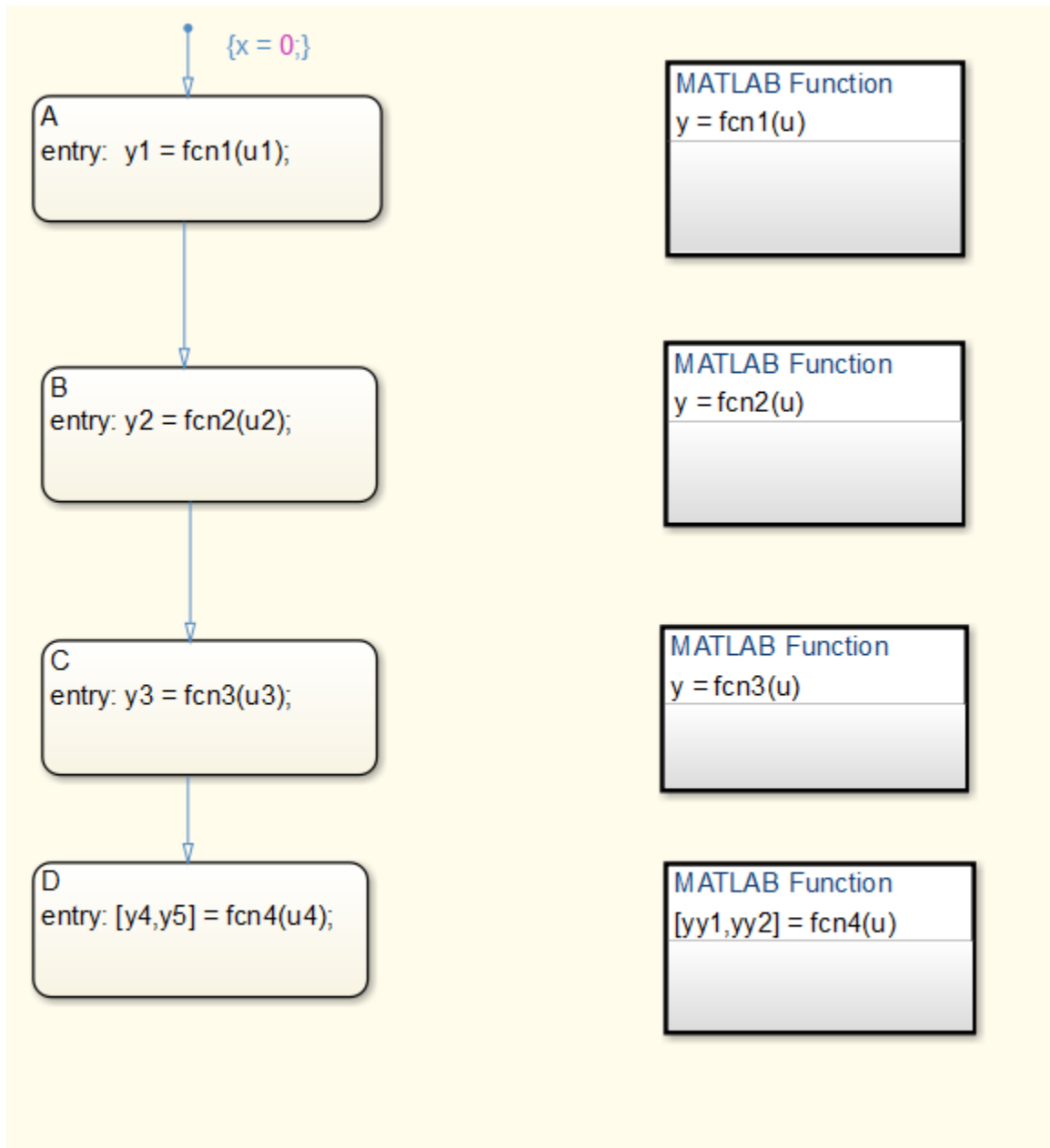
The functions fcn1 to fcn4 are defined as follows.

Function	Inlining Specification	Generated Code

fcn1:
 	
`function y = fcn1(u)`
`y = u+1;`
	Specify that the function must be inlined.	`fcn1` is inlined in the generated code.
		
		`is_c3_Chart := Chart_IN_A;`
		`(* Outport: '<Root>/y1'`
		`        incorporates:`
		` *  Inport: '<Root>/u1' *)`
		`(* Entry 'A': '<S1>:10' *)`
		`(* MATLAB Function 'fcn1':`
		`        '<S1>:1' *)`
		`(* '<S1>:1:3' *)`
		`y1 := u1 + 1.0;`

fcn2:
 	
`function y = fcn2(u)`
`y = u+2;`
	Specify that the function must not be inlined.	`fcn2` is not inlined in the generated code.
		
		`is_c3_Chart := Chart_IN_B;`
		`(* Outport: '<Root>/y2'`
		`        incorporates:`
		` *  Inport: '<Root>/u2' *)`
		`(* Entry 'B': '<S1>:11' *)`
		` y2 := fcn2(u := u2);`
		`.`
		`.`
		`.`
		`FUNCTION fcn2: LREAL`
		`VAR_INPUT`
		`    u: LREAL;`
		`END_VAR`
		`VAR_TEMP`
		`END_VAR`
		`(* MATLAB Function 'fcn2':`
		`        '<S1>:4' *)`
		`(* '<S1>:4:3' *)`
		`fcn2 := u + 2.0;`
		`END_FUNCTION`

fcn3:
 	
`function y = fcn3(u)`
`% The function accesses`
`% local data x of`
`% parent chart`
`y = u+3+x;`
	Specify that the function must not be inlined.	`fcn3` is inlined in the generated code because it accesses local data from the Stateflow chart.
		
		`is_c3_Chart := Chart_IN_C;`
		`(* Outport: '<Root>/y3'`
		`        incorporates:`
		` *  Inport: '<Root>/u3' *)`
		`(* Entry 'C': '<S1>:15' *)`
		`(* MATLAB Function 'fcn3':`
		`            '<S1>:9' *)`
		`(* The function accesses`
		`   local data x of parent`
		`    chart *)`
		`(* '<S1>:9:4' *)`
		`y3 := (u3 + 3.0) + x;`

Function	Inlining Specification	Generated Code

fcn4:
 	
`function [yy1,yy2] =`
`        fcn4(u)`
`yy1 = u+4;`
`yy2 = u+5;`
	Specify that the function must not be inlined.	fcn4 is converted to a function block in the generated code because it has multiple outputs.
		
		`is_c3_Chart := Chart_IN_D;`
		`(* Entry 'D': '<S1>:28' *)`
		`i0_fcn4(u := u4);`
		`b_y4 := i0_fcn4.yy1;`
		`b_y5 := i0_fcn4.yy2;`
		`(* Outport: '<Root>/y4'`
		`        incorporates:`
		` *  Inport: '<Root>/u4' *)`
		`y4 := b_y4;`
		`(* Outport: '<Root>/y5' *)`
		`y5 := b_y5;`
		`.`
		`.`
		`.`
		`FUNCTION_BLOCK fcn4`
		`VAR_INPUT`
		`    u: LREAL;`
		`END_VAR`
		`VAR_OUTPUT`
		`    yy1: LREAL;`
		`    yy2: LREAL;`
		`END_VAR`
		`VAR`
		`END_VAR`
		`VAR_TEMP`
		`END_VAR`
		`(* MATLAB Function 'fcn4':`
		`        '<S1>:26' *)`
		`(* '<S1>:26:3' *)`
		`yy1 := u + 4.0;`
		`(* '<S1>:26:4' *)`
		`yy2 := u + 5.0;`
		`END_FUNCTION_BLOCK`

Integrating Externally Defined Identifiers

- "Integrate Externally Defined Identifiers" on page 9-2
- "Integrate Custom Function Block in Generated Code" on page 9-3

Integrate Externally Defined Identifiers

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

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

For a description of how to integrate a custom function block, see "Integrate Custom Function Block in Generated Code" on page 9-3. For a description of the **Externally Defined Identifiers** parameter, see "Externally Defined Identifiers" on page 13-29.

Integrate Custom Function Block in Generated Code

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

1   In a Simulink model, add a MATLAB Function block.

2   Double-click the MATLAB Function block.

3   In the MATLAB editor, minimally define inputs, outputs, and stubs. For example:

Functions that have only one output, no states, and do not access global variables are generated as FUNCTION in the generated structured text code.

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

4   Change the MATLAB Function block name to ExternallyDefinedBlock.

5   Create a subsystem from this MATLAB Function block.

6   Complete the model to look like plcdemo_external_symbols.
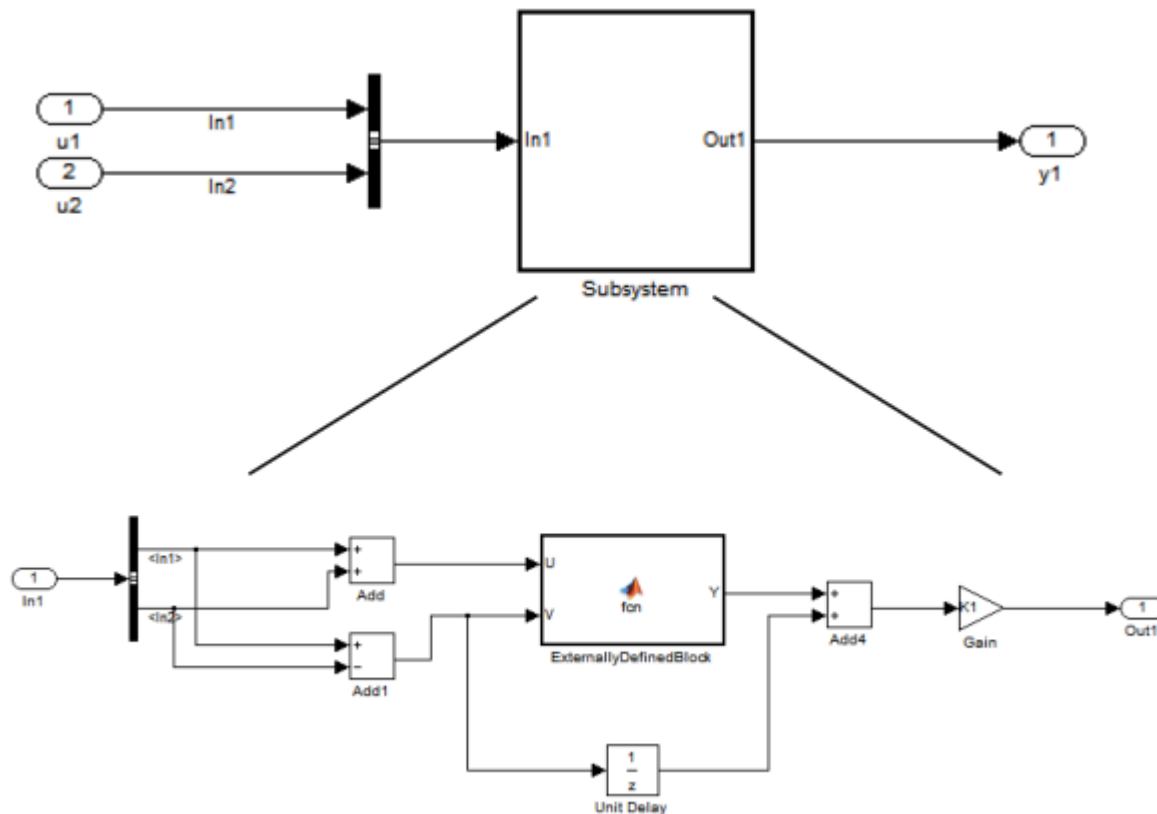
7   Open the Configuration Parameters dialog box for the model.

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

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

Externally Defined Symbols

ExternallyDefinedBlock InBus K1
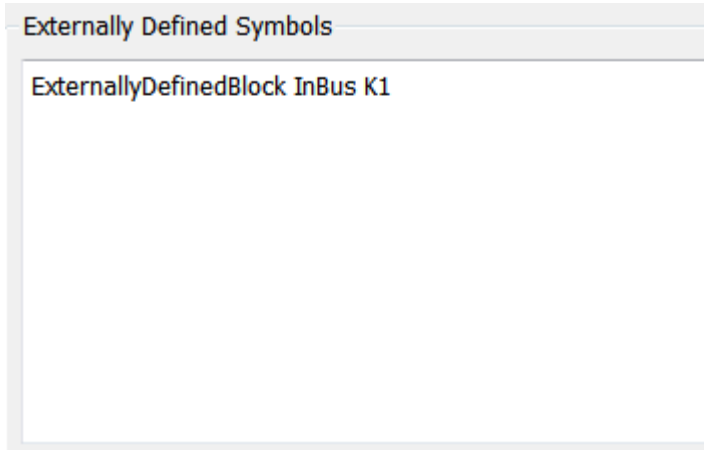
10 Save and close your new model. For example, save it as `plcdemo_external_symbols_mine`.

11 Generate code for the model.

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

The reference of `ExternallyDefinedBlock` is:

```
VAR
    UnitDelay_DSTATE: LREAL;
    i0_ExternallyDefinedBlock: ExternallyDefinedBlock;
END_VAR
```

The omission of `ExternallyDefinedBlock` is:

```
(* MATLAB Function: '<S1>/ExternallyDefinedBlock' *)
i0_ExternallyDefinedBlock(U := rtb_Add, V := rtb_Add1);
rtb_Y := i0_ExternallyDefinedBlock.Y;
```

IDE-Specific Considerations

Integrate Generated Code with Siemens IDE Project

You can integrate generated code with an existing Siemens SIMATIC STEP 7 or Siemens TIA Portal project. For more information on:

- How to generate code, see "Generate and Examine Structured Text Code" on page 1-7.
- The location of generated code, see "Files Generated by Simulink PLC Coder" on page 1-11.

Integrate Generated Code with Siemens SIMATIC STEP 7 Projects

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

2 Navigate to the folder containing the generated code and open the file.

The custom file name unless assigned differently, is the *model_name*.scl. After you open the file, a new entry called *model_name*.scl appears under the **Sources** node.

3 Double-click the new entry. The generated code is listed in the SCL editor window.

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

5 In the customize window, select **Create block numbers automatically**, and click **OK**.

Symbol addresses are automatically generated for Subsystem blocks.

6 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.

Integrate Generated Code with Siemens TIA Portal Projects

1 In the **Project tree** pane, on the **Devices** tab, under the **External source files** node in your project, select **Add new external file**.

2 Navigate to the folder containing the generated code and open the file.

The custom file name unless assigned differently, is the *model_name*.scl. After you open the file, a new entry called *model_name*.scl appears under the **External source files** node.

3 Right-click the new entry and select **Generate blocks from source**.

The Siemens TIA Portal IDE compiles the new file and generates TIA Portal program blocks from the code. The program blocks appear under the **Program blocks** node. They are available for use with the existing Siemens TIA Portal project.

Use Internal Signals for Debugging in RSLogix 5000 IDE

For debugging, you can generate code for test point outputs from the top-level subsystem of your model. The coder generates code that maps the test pointed output to optional AOI output parameters for RSLogix 5000 IDEs. In the generated code, the variable tags that correspond to the test points have the property `Required=false`. This example assumes that you have a model appropriately configured for the coder, such as `plcdemo_simple_subsystem`.

1   Open the `plcdemo_simple_subsystem` model.

   `plcdemo_simple_subsystem`

2   In the Configuration Parameters dialog box, set **Target IDE** to `Rockwell RSLogix 5000: AOI`.

3   In the top-level subsystem of the model, right-click the output signal of `SimpleSubsystem` and select **Properties**.

   The Signal Properties dialog box is displayed.

4   On the **Logging and accessibility** tab, click the **Test point** check box.
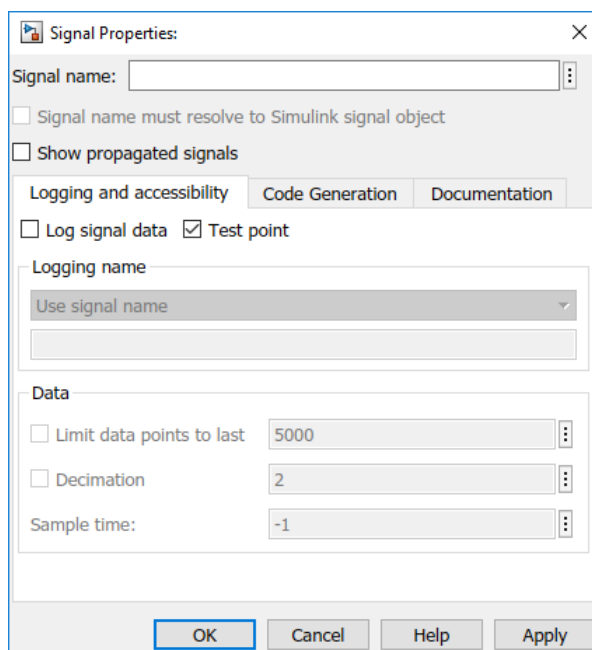
5   Click **OK**.

6   Generate code for the top-level subsystem.

7   Inspect the generated code for the string `Required=false`.

For more information on signals with test points, see "What Is a Test Point?".

Rockwell Automation RSLogix Requirements

Following are considerations 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 the AOIs for Add-On instruction format, but not FUNCTION_BLOCK.

Double-Precision Data Types

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

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, the numeric results produced by the generated Structured Text can 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, Simulink PLC Coder converts this data type to signed integer data types in generated code.

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. For more information about coder limitations for fixed-point data type support, see "Fixed Point Simulink PLC Coder Structured Text Code Generation" on page 20-2.

Enumerated Data Types

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

Reserved Keywords

The Rockwell Automation RSLogix target IDE has reserved keywords. Do not use them as tag names in subsystems from which code will be for be generated for Rockwell Automation RSLogix IDE.

ABS	ACS	AND	ASN	ATN	COS	DEG	FRD	LN	LOG	MOD
NOT	OR	RAD	SIN	SQR	TAN	TOD	TRN	XOR	acos	asin
atan	by	case	do	else	elsif	end_case	end_for	end_if	end_repeat	end_while
exit	for	if	of	repeat	return	then	to	trunc	until	while

These keywords are case insensitive. If your code generation target IDE is the Rockwell Automation RSLogix 5000 or Studio 5000 IDE do not use these keywords as variable names.

Rockwell Automation IDE selection

Based on the L5X import file target IDE version you will choose the PLC target IDE to be RSLogix 5000 or Studio 5000. If importing into v24 or later choose Studio 5000 else for versions prior to v24 choose RSLogix 5000.

Siemens IDE Requirements

Target PLCs and Supported Data Types

To choose your target PLC based on supported data types, see the options in this table.

Data Type	S7-300/400	S7-1200	S7-1500
BOOL	Yes	Yes	Yes
BYTE	Yes	Yes	Yes
WORD	Yes	Yes	Yes
DWORD	Yes	Yes	Yes
LWORD	No	No	Yes
SINT	No	Yes	Yes
INT	Yes	Yes	Yes
DINT	Yes	Yes	Yes
USINT	No	Yes	Yes
UINT	No	Yes	Yes
UDINT	No	Yes	Yes
LINT	No	No	Yes
ULINT	No	No	Yes
REAL	Yes	Yes	Yes
LREAL	No	Yes	Yes

To generate code for your S7-300/400 series PLCs use the SIMATIC STEP 7 or TIA Portal as the target IDE.. To generate code for your S7-1200 or S7-1500 series PLCs, use the TIA Portal: Double Precision as the target IDE .

Double-Precision Floating-Point Data Types

To generate code for your Siemens targets that do not support double-precision, floating-point data types, use the SIMATIC STEP 7 or TIA Portal as the target IDE. At code generation, Simulink PLC Coder converts this data type to single-precision real data types in the generated code. Design your model so that the possible precision loss of generated code numeric results does not change the expected semantics of the model.

To generate code for your Siemens targets that support double-precision, floating-point types, use Siemens TIA Portal: Double Precision as the target IDE. The generated code uses the LREAL type for double-precision, floating-point types in the model. For more information, see "Target IDE" on page 13-3.

int8 Data Type and Unsigned Integer Types

To generate code for your Siemens targets that do not support the int8 data type and unsigned integer data types, use Siemens SIMATIC Step 7 or Siemens TIA Portal as the target IDE. At code

generation, Simulink PLC Coder converts the int8 data type and unsigned integer data types to int16 or int32 in the generated code.

Design your model to use int16 and int32 data types as much as possible instead of int8 or unsigned integer data types. The Simulink numeric results by using the int8 data type or unsigned integer data types can differ from the numeric results produced by the generated structured text.

Design your model so that the effects of integer data type conversion of the generated code do not change the expected semantics of the model.

To generate code for your Siemens targets that support the int8 data type and unsigned integer data types, use Siemens TIA Portal: Double Precision as the target IDE. The generated code preserves the int8 data type and unsigned integer data types. For more information, see "Target IDE" on page 13-3.

Unsigned Fixed-Point Data Types

Do not use unsigned, fixed-point data types in your model to generate code for your Siemens targets that do not support unsigned integer data types. For more information about coder limitations for fixed-point data type support, see "Fixed Point Simulink PLC Coder Structured Text Code Generation" on page 20-2.

Enumerated Data Types

The Siemens SIMATIC STEP 7 and TIA Portal target IDEs do not support enumerated data types. Simulink PLC Coder converts this data type to 16-bit signed integer data type in the generated code for Siemens targets.

Naming Constraints

The Siemens TIA Portal target IDEs earlier than v17 throw an error when variable names or function block names in the generated code contain numbers. For example, if your generated code is:

```
FUNCTION_BLOCK xg1479016
VAR_INPUT
    In1: LREAL;
END_VAR
VAR_OUTPUT
    Out1: LREAL;
END_VAR
Out1 := In1*2;
END_FUNCTION_BLOCK
```

. When you import this generated code the TIA Portal IDE , this error message or warning is generated 'HEX value or unsigned number exceeds the BYTE-WORD-DWORD limit'. This warning is thrown as the TIA Portal importer considers xg1479016 as a number and the value exceeds 65535.

Selectron CAP1131 IDE Requirements

For the Selectron CAP1131 target IDE platform, consider these limitations:

Double-Precision Floating-Point Data Types

The Selectron CAP1131 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 the generated code. Design your model so that the possible precision loss of numerical results of the generated code numeric results does not change the model semantics that you expect.

Enumerated Data Types

The Selectron CAP1131 target IDE does not support enumerated data types. The Selectron CAP1131 IDE converts this data type to 32-bit signed integer data type in the generated code.

See Also

Supported Simulink and Stateflow Blocks

Supported Blocks

For Simulink semantics not supported by Simulink PLC Coder, see "Structured Text Code Generation Limitations" on page 12-2.

View Supported Blocks Library

To view a Simulink library of blocks that the Simulink PLC Coder software supports, type `plclib` in the Command Window. The coder can generate Structured Text code for subsystems that contain these blocks. The library window is displayed.
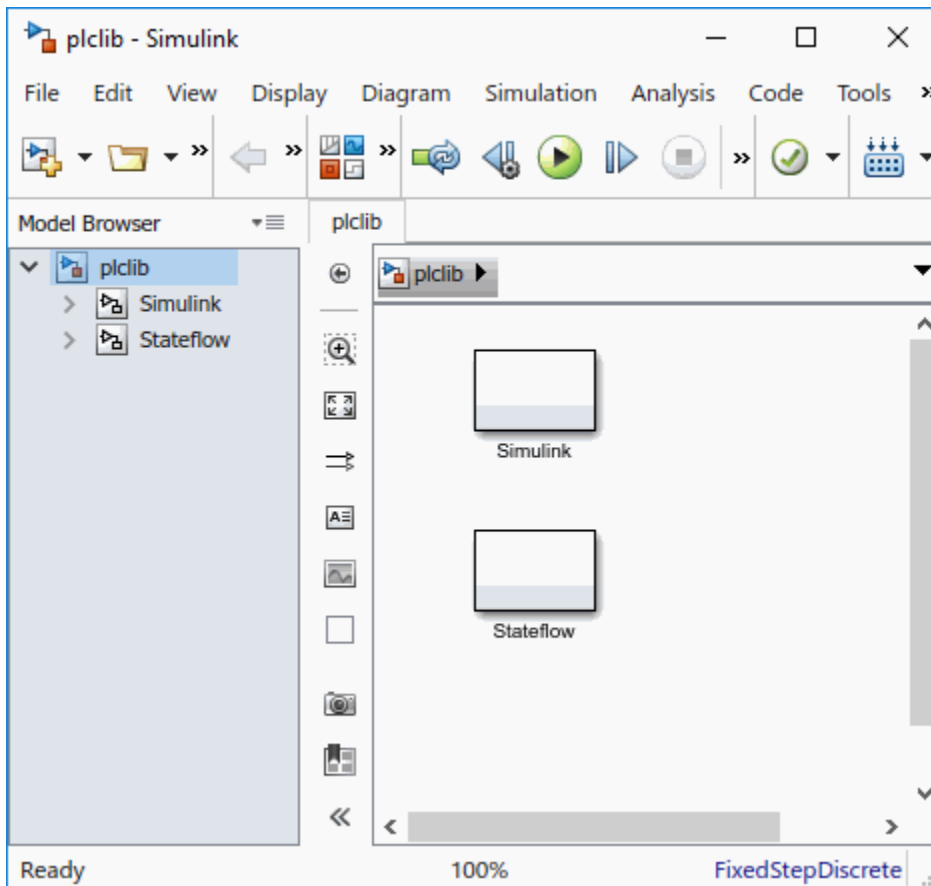
This library contains two sublibraries, Simulink and Stateflow. Each sublibrary contains the blocks that you can include in a Simulink PLC Coder model.

Supported Simulink Blocks

The coder supports the following Simulink blocks.

Additional Math & Discrete/Additional Discrete

Transfer Fcn Direct Form II

Transfer Fcn Direct Form II Time Varying

Unit Delay Enabled (Obsolete)

Unit Delay Enabled External IC (Obsolete)

Unit Delay Enabled Resettable (Obsolete)

Unit Delay Enabled Resettable External IC (Obsolete)

Unit Delay External IC (Obsolete)

Unit Delay Resettable (Obsolete)

Unit Delay Resettable External IC (Obsolete)

Unit Delay With Preview Enabled (Obsolete)

Unit Delay With Preview Enabled Resettable (Obsolete)

Unit Delay With Preview Enabled Resettable External RV (Obsolete)

Unit Delay With Preview Resettable (Obsolete)

Unit Delay With Preview Resettable External RV (Obsolete)

Commonly Used Blocks

Inport

Bus Creator

Bus Selector

Constant

Data Type Conversion

Demux

Discrete-Time Integrator

Gain

Ground

Logical Operator

Mux

Product

Relational Operator

Saturation

Scope

Subsystem

Inport

Outport

Sum

Switch

Terminator

Unit Delay

Discontinuities

Coulomb and Viscous Friction

Dead Zone Dynamic

Rate Limiter

Rate Limiter Dynamic

Relay

Saturation

Saturation Dynamic

Wrap To Zero

Discrete

Difference

Discrete Transfer Fcn

Discrete Derivative

Discrete FIR Filter

Discrete Filter

Discrete PID Controller

Discrete PID Controller (2 DOF)

Discrete State-Space

Discrete-Time Integrator

FIR Interpolation

Integer Delay

Memory

Tapped Delay

Transfer Fcn First Order

Transfer Fcn Lead or Lag

Transfer Fcn Real Zero

Unit Delay

Zero-Order Hold

Logic and Bit Operations

Bit Clear

Bit Set

Bitwise Operator

Compare To Constant

Compare To Zero

Detect Change

Detect Decrease

Detect Increase

Detect Fall Negative

Detect Fall Nonpositive

Detect Rise Nonnegative

Detect Rise Positive

Extract Bits

Interval Test

Interval Test Dynamic

Logical Operator

Shift Arithmetic

Lookup Tables

Dynamic-Lookup

Interpolation Using Prelookup

PreLookup

n-D Lookup Table

Math Operations

Abs

Add

Assignment

Bias

Divide

Dot Product

Gain

Math Function

Matrix Concatenate

MinMax

MinMax Running Resettable

Permute Dimensions

Polynomial

Product

Product of Elements

Reciprocal Sqrt

Reshape

Rounding Function

Sign

Slider Gain

Sqrt

Squeeze

Subtract

Sum

Sum of Elements

Trigonometric Function

Unary Minus

Vector Concatenate

Model Verification

Assertion

Check Discrete Gradient

Check Dynamic Gap

Check Dynamic Range

Check Static Gap

Check Static Range

Check Dynamic Lower Bound

Check Dynamic Upper Bound

Check Input Resolution

Check Static Lower Bound

Check Static Upper Bound

Model-Wide Utilities

DocBlock

Model Info

Ports & Subsystems

Atomic Subsystem

CodeReuse Subsystem

Enabled Subsystem

Enable

Function-Call Subsystem

Subsystem

Inport

Outport

Signal Attributes

Data Type Conversion

Data Type Duplicate

Signal Conversion

Signal Routing

Bus Assignment

Bus Creator

Bus Selector

Data Store Memory

Demux

From

Goto

Goto Tag Visibility

Index Vector

Multiport Switch

Mux

Selector

Sinks

Display

Floating Scope

Scope

Stop Simulation

Terminator

To File

To Workspace

Sources

Constant

Counter Free-Running

Counter Limited

Enumerated Constant

Ground

Pulse Generator

Repeating Sequence Interpolated

Repeating Sequence Stair

User-Defined Functions

MATLAB Function (MATLAB Function Block)

Supported Stateflow Blocks

The coder supports the following Stateflow blocks.

Stateflow

Chart

State Transition Table

Truth Table

Blocks with Restricted Support

Simulink Block Support Exceptions

The Simulink PLC Coder software supports the `plclib` blocks with the following exceptions. Also, see "Structured Text Code Generation Limitations" on page 12-2 for a list of limitations of the software.

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

Stateflow Chart Exceptions

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

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

    The chart Stateflow Editor dialog box is displayed.
2   Right-click in this dialog box.
3   In the context-sensitive menu, select **Properties**.

    The Chart dialog box is displayed.
4   In the Chart dialog box, navigate to the **States When Enabling** parameter and select `Held`.
5   Click **Apply** and **OK** and save the model.

Data Store Memory Block

To generate PLC code for a model that uses a Data Store Memory block, first define a `Simulink.Signal` object in the base workspace. Then, in the **Signal Attributes** tab of the block parameters, set the data store name to resolve to that `Simulink.Signal` object.

For more information, see "Data Stores with Data Store Memory Blocks".

Reciprocal Sqrt Block

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

Lookup Table Blocks

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

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

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

Limitations

Structured Text Code Generation Limitations

General Limitations

The Simulink PLC Coder software does not support:

- Complex data types
- String data types
- Model reference blocks
- Stateflow machine-parented data and events
- Stateflow messages
- Stateflow empty chart output functions.
- Limited support for math functions
- Merge block
- Step block
- Clock block
- Signal and state storage classes
- Shared state variables between subsystems
- For Each Subsystem block
- Variable-size signals and parameters
- MATLAB System block or system objects
- MATLAB classes.
- The `Simulink.CoderInfo Identifier` name property with `Simulink.Parameter` and `Simulink.Signal` objects.
- The `Simulink.LookupTable`, `Simulink.Breakpoint`, and `Simulink.DualScaledParameter` objects.
- Code generation for Simulink signals that do not resolve to a `Simulink.Signal` data store memory object.
- Code generation when `UseRowMajorAlgorithm='on'`.
- The use of `enum` datatype numeric values for comparison inside model subsystem blocks. Use a data type conversion block to perform an enum to integer conversion, to perform the numeric comparison.
- The use of special characters in comments. This could lead to errors when importing the generated code.
- Signal lines named using `Simulink.Signal` mappings.
- Half precision fixed-point data types.
- Testbench generation for models using software-in-the-loop (SIL) simulation mode.
- Testbench generation for models using processor-in-the-loop (PIL) simulation mode.
- Non top-level Stateflow function call output events that call Simulink subsystems.
- Half-precision data types.
- Code generation inside subsystem reference blocks.

- Code generation for nested subsystem reference blocks.
- Code generation in folders that have the `Folder is ready for archiving` option enabled on the Windows® system prior to R2022a.

Restrictions

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

- Supports code generation only for atomic subsystems.
- Supports automatic, inline, or reusable function packaging for code generation. Nonreusable function packaging is not supported.
- Does not support blocks that require continuous time semantics. This restriction includes integrator blocks, zero-crossing detection blocks, physical blocks, such as Simscape™ library blocks and so on.
- Does not support pointer data types.
- Does not support recursion (including recursive events).
- Does not support nonfinite data, for example `NaN` or `Inf`.
- Does not support MATLAB 64-bit integer data types.

Negative Zero

In a floating-point data type, the value `0` has either a positive sign or a negative sign. Arithmetically, `0` is equal to `-0`, but some operations are sensitive to the sign of a `0` input. Examples include `rdivide`, `atan2`, `atan2d`, and `angle`. Division by `0` produces `Inf`, but division by `-0` produces `-Inf`. Similarly, `atan2d(0,-1)` produces `180`, but `atan2d (-0,-1)` produces `-180`.

Simulink PLC Coder stores `-0` as `0` because there is no representation of `-0` in IEC61131.This leads to division by `-0` producing `-Inf` in Simulink, but `Inf` in PLC IDE. Similarly, `atan2d(-0,-1)` produces `-180` in Simulink, but `180` in PLC IDE as the `-0` is converted to `0`.

Divide by Zero

In Simulink, division by zero produces either `Inf` or the largest number for the data type. In the Codesys target IDE, division by zero results in a `-1`. Code generation by using a testbench might result in testbench verification failures due to a difference in results from divide by zero operations.

Fixed-Point Data Type Multiword Operations

Simulink PLC Coder does not support code generation for block parameter settings that require fixed-point data type multiword operations. For example, the square root block that has int32 integer data type as input and output data type setting of `Inherit via internal rule` is not supported for code generation.

Inplace Variables Code Generation

Inplace argument semantics could be broken if the datatypes between inputs and outputs differ in the number of dimensions. To fix the problem, set the input variable size to `-1`. For more information, see "Declare Variable-Size MATLAB Function Block Variables".

Simulink Data Dictionary

Simulink PLC Coder does not support:

- The mixed use of the base workspace and Simulink Data Dictionary (SLDD) files. Use the Simulink migration utility to migrate your entire base workspace to SLDD files.
- Model workspace parameters and signals for code generation.
- MATLAB variables in SLDD files for code generation. To generate code convert these variables to `Simulink.Parameter` objects.

`Simulink.parameter` types that have `StorageClass` options other than `ExportedGlobal` and `ImportedExtern` are auto converted to `ExportedGlobal StorageClass` during code generation.

Ladder Logic Code Generation Limitations

plcladderlib Limitations

Simulink PLC Coder `plcladderlib` has these limitations:

- Only Rockwell Automation RSLogix 5000 and Studio 5000 IDEs can import ladder logic generated using the `plcladderlib` library.

Ladder Diagram Import Limitations

- When importing an `.L5X` file that contains a continuous task, the imported Simulink model has a sample time of `-1`. For periodic tasks, the sample time is the value specified in the `.L5X` file. Event tasks are not supported.
- Simulink PLC Coder may not follow the same initialization order specified in the `Prescan` mode. Do not read variables that are read by the `Prescan` mode because this leads to different behavior in simulation of the model when compared to execution in the IDE. The affected Simulink PLC Coder `plcladderlib` blocks are: OTE, ONS, OSF, OSR, CTD, CTU, TON, TOF, RTO, JSR, AOI, and FBC
- If you Ladder Diagram implementation has multiple AOI or subroutine instances with the same name, the software does not check if these instances refer to the same implementation. It is recommended to use different names if these structures contain different functionality.

Ladder Diagram Modeling and Simulation Limitations

- Ladder models do not support unsigned integer types. Use signed integer instead.
- Ladder models do not support double type. Instead, use single type.
- The Rockwell Automation IDEs have limitations on the character length used for names. The length should not be more than 40 characters. For supported name lengths consult the Rockwell documentation.
- Label the **Port** numbers in the **Controller Tags** uniquely and sequentially, when modeling Ladder Diagrams in Simulink.

Ladder Diagram Code Generation Limitations

- Code generation requires a controller, task, program model, AOI runner, or AOI model hierarchy
- AOI input argument should be either non-array or 1-D array type. Test bench generation does not support 2-D or 3-D array types. This limitation includes nested 2-D, 3-D array types in structure fields.
- The Rockwell Automation IDEs have limitations on the character length used for names. The length should not be more than 40 characters. For supported name lengths consult the Rockwell documentation.

Ladder Diagram Verification Limitations

- Ladder test bench generation is supported for only AOI Runner block.

- AOI input argument should be either non-array or 1-D array type. Test bench generation does not support 2-D or 3-D array types. This limitation includes nested 2-D, 3-D array types in structure fields.

- AOI input argument in the L5X file should not be single-element array type for runner test bench generation.

- Test bench generation for Ladder Diagram models containing timer blocks such as `TON`, `TOF` and `RTO` fails. To generate test-bench code for these models, modify the Ladder Diagram structure while maintaining the logic.

- If the Simulink model is set as read-only, the model can become corrupted during the test bench generation process. When the code generation process completes, it reverts all code generation changes performed on the model. You can ignore or close the model during this process.

"Generating Ladder Diagram Code from Simulink" on page 3-13 | "Import L5X Ladder Diagram Files into Simulink" on page 3-4 | "Model and Simulate Ladder Diagrams in Simulink" on page 3-8 | "Verify Generated Ladder Diagram Code" on page 3-17

Configuration Parameters for Simulink PLC Coder Models

PLC Coder: General

In this section...

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 > 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.
- Certain options are target-specific and are displayed based on the selection for **Target IDE**.

See Also

"Prepare Model for Structured Text Generation" on page 1-3

"Generate Structured Text from the Model Window" on page 1-7

Target IDE

Select the target IDE for which you want to generate code. This option is available in the Configuration Parameters dialog box, **PLC Code Generation** pane.

The default **Target IDE** list shows the full set of supported targets. See "Show Full Target List" on page 13-5.

To see a reduced subset of targets, clear the option **Show full target list**. To customize this list and specify IDEs that you use more frequently, use the `plccoderpref` function.

For version numbers of supported IDEs, see "Supported IDE Platforms".

Settings

Default: `3S CoDeSys 2.3`

`3S CoDeSys 2.3`

Generates Structured Text (IEC 61131-3) 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.

`3S CoDeSys 3.5`

Generates Structured Text code in PLCopen XML for 3S-Smart Software Solutions CoDeSys Version 3.5.

B&R Automation Studio 3.0

Generates Structured Text code for B&R Automation Studio 3.0.

B&R Automation Studio 4.0

Generates Structured Text code for B&R Automation Studio 4.0.

Beckhoff TwinCAT 2.11

Generates Structured Text code for Beckhoff TwinCAT 2.11 software.

Beckhoff TwinCAT 3

Generates Structured Text code for Beckhoff TwinCAT 3 software.

KW-Software MULTIPROG 5.0

Generates Structured Text code in PLCopen XML for PHOENIX CONTACT (previously KW) Software MULTIPROG 5.0 or 5.50.

Phoenix Contact PC WORX 6.0

Generates Structured Text code in PLCopen XML for Phoenix Contact PC WORX 6.0.

Rockwell RSLogix 5000: AOI

Generates Structured Text code for Rockwell Automation RSLogix 5000 using Add-On Instruction (AOI) constructs.

Rockwell RSLogix 5000: Routine

Generates Structured Text code for Rockwell Automation RSLogix 5000 routine constructs.

Rockwell Studio 5000: AOI

Generates Structured Text code for Rockwell Automation Studio 5000 Logix Designer using Add-On Instruction (AOI) constructs.

Rockwell Studio 5000: Routine

Generates Structured Text code for Rockwell Automation Studio 5000 Logix Designer routine constructs.

Siemens SIMATIC Step 7

Generates Structured Text code for Siemens SIMATIC STEP 7.

Siemens TIA Portal

Generates Structured Text code for Siemens TIA Portal S7-300/400 CPUs.

Siemens TIA Portal: Double Precision

Generates Structured Text code for Siemens TIA Portal S7-1200 and S7-1500 CPUs. THE IDE also supports the int8 data type, unsigned integer data types, and double-precision, floating-point data types. The code uses LREAL type for double data type in the model and can be used on Siemens PLC devices that support the LREAL type.

Generic

Generates a pure Structured Text file. If the target IDE that you want is not available for the Simulink PLC Coder product, consider generating and downloading a generic Structured Text file.

PLCopen XML

Generates Structured Text code formatted using PLCopen XML standard.

Rexroth Indraworks

Generates Structured Text code for Rexroth IndraWorks version 13V12 IDE.

OMRON Sysmac Studio

Generates Structured Text code for OMRON® Sysmac® Studio Version 1.04, 1.05, or 1.09.

Selectron CAP1131

Generates Structured Text code for Selectron CAP1131 v 11 IDE.

Tips

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

Command-Line Information
Parameter: PLC_TargetIDE
Type: string
Value: 'codesys23' | 'codesys33' | 'codesys35' | 'rslogix5000' | 'rslogix5000_routine' | 'studio5000' | 'studio5000_routine' | 'brautomation30' | 'brautomation40' | 'multiprog50' | 'pcworx60' | 'step7' | 'plcopen' | 'twincat211' | 'twincat3' | 'generic' | 'indraworks' | 'omron' | 'tiaportal' | 'tiaportal_double'
Default: 'codesys23'

See Also

"Generate Structured Text from the Model Window" on page 1-7

Show Full Target List

View the full list of supported target IDEs in the **Target IDE** drop-down list. For more information, see "Target IDE" on page 13-3. This option is available on the **PLC Code Generation** pane in the Configuration Parameters dialog box.

Settings

Default: On

☑ On

The **Target IDE** list displays the full set of supported IDEs. For more information, see "Supported IDE Platforms".

☐ Off

The **Target IDE** list displays only the more commonly used IDEs. The default subset contains the following IDEs:

- codesys23 — 3S-Smart Software Solutions CoDeSys Version 2.3 (default) target IDE
- studio5000 — Rockwell Automation Studio 5000 Logix Designer target IDE for AOI format
- step7 — Siemens SIMATIC STEP 7 target IDE
- omron — OMRON Sysmac Studio
- plcopen — PLCopen XML target IDE

You can customize the entries in the reduced **Target IDE** list by using the plccoderpref function.

Command-Line Information
Parameter: PLC_ShowFullTargetList
Type: string
Value: 'on' | 'off'
Default: 'on'

You can change the contents of the reduced **Target IDE** list using the plccoderpref function. See plccoderpref.

Target IDE Path

Specify the target IDE installation path. The path already specified is the default installation path for the target IDE. Change this path if your IDE is installed in a different location. This option is available on the **PLC Code Generation** pane in the Configuration Parameters dialog box.

Settings

Default: C:\Program Files\3S Software

C:\Program Files\3S Software

    Default installation path for 3S-Smart Software Solutions CoDeSys software Version 2.3.

C:\Program Files\3S CoDeSys

    Default installation path for 3S-Smart Software Solutions CoDeSys software Version 3.3 and 3.5.

C:\Program Files\BrAutomation

    Default installation path for B&R Automation Studio 3.0 and 4.0 software.

C:\TwinCAT

    Default installation path for Beckhoff TwinCAT 2.11 and 3 software.

C:\Program Files\KW-Software\MULTIPROG 5.0

    Default installation path for PHOENIX CONTACT (previously KW) Software MULTIPROG 5.0 software. For MULTIPROG 5.50, the installation path may be different, change accordingly.

C:\Program Files\Phoenix Contact\Software Suite 150

    Default installation path for Phoenix Contact PC WORX 6.0 software.

C:\Program Files\Rockwell Software

    Default installation path for Rockwell Automation RSLogix 5000 software.

C:\Program Files\Siemens

    Default installation path for Siemens SIMATIC STEP 7 5.4 software.

C:\Program Files\Siemens\Automation

    Default installation path for Siemens TIA Portal software.

Tips

- When you change the **Target IDE** value, the value of this parameter changes.
- If you right-click the Subsystem block, the **PLC Code > 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 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

"Import Structured Text Code Automatically" on page 1-14

Code Output Directory

Enter a path to the target folder into which code is generated. This option is available on the **PLC Code Generation** pane in the Configuration Parameters dialog box.

Settings

Default: `plcsrc` subfolder in your working folder

Command-Line Information
Parameter: `PLC_OutputDir`
Type: string
Value: `string`
Default: `'plcsrc'`

Tips

- If the target folder path is empty, a default value of `./plcsrc` is used as the **Code Output Directory**.
- If, you want to generate code in the current folder use `.` as the output directory.
- The **Code Output Directory** can have the same name as your current working folder.

See Also

"Generate Structured Text from the Model Window" on page 1-7

Generate Testbench for Subsystem

Specify the generation of test bench code for the subsystem. This option is available on the **PLC Code Generation** pane in the Configuration Parameters dialog box.

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

"Generate Structured Text from the Model Window" on page 1-7

Include Testbench Diagnostic Code

Specify the generation of test bench code with additional diagnostic information that will help you identify output variables causing test bench failures. This option is available on the **PLC Code Generation** pane in the Configuration Parameters dialog box. To enable this parameter, you must select the **Generate testbench for subsystem** option

Settings

Default: off

☑ On

Enables generation of test bench code with additional diagnostic information.

☐

Disables generation of test bench code with additional diagnostic information.

Command-Line Information
Parameter: `PLC_GenerateTestbenchDiagCode`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

See Also

"Generate Structured Text from the Model Window" on page 1-7

Generate Functions Instead of Function Block

Use this option to control whether the generated Structured Text code contains `Function` instead of `Function Block`. This option is available for only the Phoenix Contact PC WORX or the PHOENIX CONTACT (previously KW) Software MULTIPROG target. There are certain cases where you may not be able to generate code with `Function` instead of `Function Block`. For example, if your Simulink subsystem or MATLAB Function block has internal state or persistent variables. In such cases, the software issues a diagnostic warning.

This option is available on the **PLC Code Generation** pane in the Configuration Parameters dialog box, when the **Target IDE** is set to `Phoenix Contact PC WORX 6.0` or `KW-Software MULTIPROG 5.0`.

Settings

Default: off

☑ On

> The generated Structured Text code contains `Function` instead of `Function Block` where possible.

☐ Off

> Switch to the default behavior of the software.

Command-Line Information
Parameter: `PLC_EmitAsPureFunctions`
Type: string
Value: `'on'` | `'off'`
Default: `'off'`

See Also

"Generate Structured Text from the Model Window" on page 1-7

Allow Functions with Zero Inputs

Emit a function with no inputs as a function instead of a function block. This option is available for only the Phoenix Contact PC WORX or the PHOENIX CONTACT (previously KW) Software MULTIPROG target.

When the **Target IDE** is set to `Phoenix Contact PC WORX 6.0` or `KW-Software MULTIPROG 5.0`, in the Configuration parameters dialog box, **PLC Code Generation** pane, this option is available.

Settings

Default: off

☑ On

> The generated Structured Text code contains `Function` instead of `Function Blocks` when there is a function with no inputs.

☐ Off

> The generated Structured Text code contains function blocks and no functions.

Command-Line Information
Parameter: `PLC_PureFunctionNoInputs`
Type: string
Value: `'on'` | `'off'`

Default: `'off'`

See Also

"Generate Structured Text from the Model Window" on page 1-7

Suppress Auto-Generated Data Types

Use this option to control whether the generated Structured Text code contains auto-generated data types for array types. This option is available for only the Phoenix Contact PC WORX or the PHOENIX CONTACT (previously KW) Software MULTIPROG target.

This option is available on the **PLC Code Generation** pane in the Configuration Parameters dialog box, when the **Target IDE** is set to `Phoenix Contact PC WORX 6.0` or `KW-Software MULTIPROG 5.0`.

Settings

Default: off

☑ On

    The software automatically generates named types for array types in your Simulink model.

☐ Off

    Switch to the default behavior of the software.

Command-Line Information
Parameter: PLC_SuppressAutoGenType
Type: string
Value: `'on'` | `'off'`
Default: `'off'`

See Also

"Generate Structured Text from the Model Window" on page 1-7

Emit Data type Worksheet Tags for PCWorx

Use this option to control whether `datatypeWorksheet` tags are represented in code generated for Phoenix Contact PC WORX target. This option allows you to have finer control and generate multiple `datatypeWorksheet` definitions.

This option is available on the **PLC Code Generation** pane in the Configuration Parameters dialog box, when the **Target IDE** is set to `Phoenix Contact PC WORX 6.0`.

Settings

Default: off

☑ On

The datatypeWorksheet tags are marked as separate tags in the generated code.

☐ Off

No separate datatypeWorksheet tags are in the generated code.

Command-Line Information
Parameter: PLC_EmitDatatypeWorkSheet
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

"Generate Structured Text from the Model Window" on page 1-7

Aggressively Inline Structured Text Function Calls

Using this option, you can control inlining of Structured Text function calls for Rockwell Automation targets. By default, the software attempts to inline only math functions where possible. With this option, the software aggressively inlines all function calls so that the generated code has less number of Function blocks.

This option is available on the **PLC Code Generation** pane in the Configuration Parameters dialog box, when the **Target IDE** is set to Rockwell Automation targets such as Rockwell Studio 5000: AOI, Rockwell Studio 5000: Routine, Rockwell RSLogix 5000: AOI, or Rockwell RSLogix 5000: Routine.

Settings

Default: off

☑ On

Aggressively inlines Structured Text function calls for RSLogix IDE.

☐ Off

Reverts to its default behavior and inlines only math function calls in the generated code.

Command-Line Information

Parameter: PLC_EnableAggressiveInlining
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

- "Generate Structured Text from the Model Window" on page 1-7
- "Generated Code Structure for Simple Simulink Subsystems" on page 2-2

Signal Builder Block Time Range to Generate Multi Testbench

Use this option to generate multiple testbenches of varying sizes. The generated testbench size depends on the time duration of the respective signal group in the Signal Builder block. This option is available on the **PLC Code Generation** pane in the Configuration Parameters dialog box. Select the **Generate testbench for subsystem** option.

Settings

Default: Off

☑ On

Generate multiple testbenches with testbench size dependent on time duration of respective signal group in the Signal Builder block..

☐ Off

Generate multiple testbenches with testbench size dependent on model simulation time.

Command-Line Information
Parameter: PLC_MultiTBSigbuilderTimeRange
Type: string
Value: 'on' | 'off'
Default: 'off'

PLC Coder: Comments
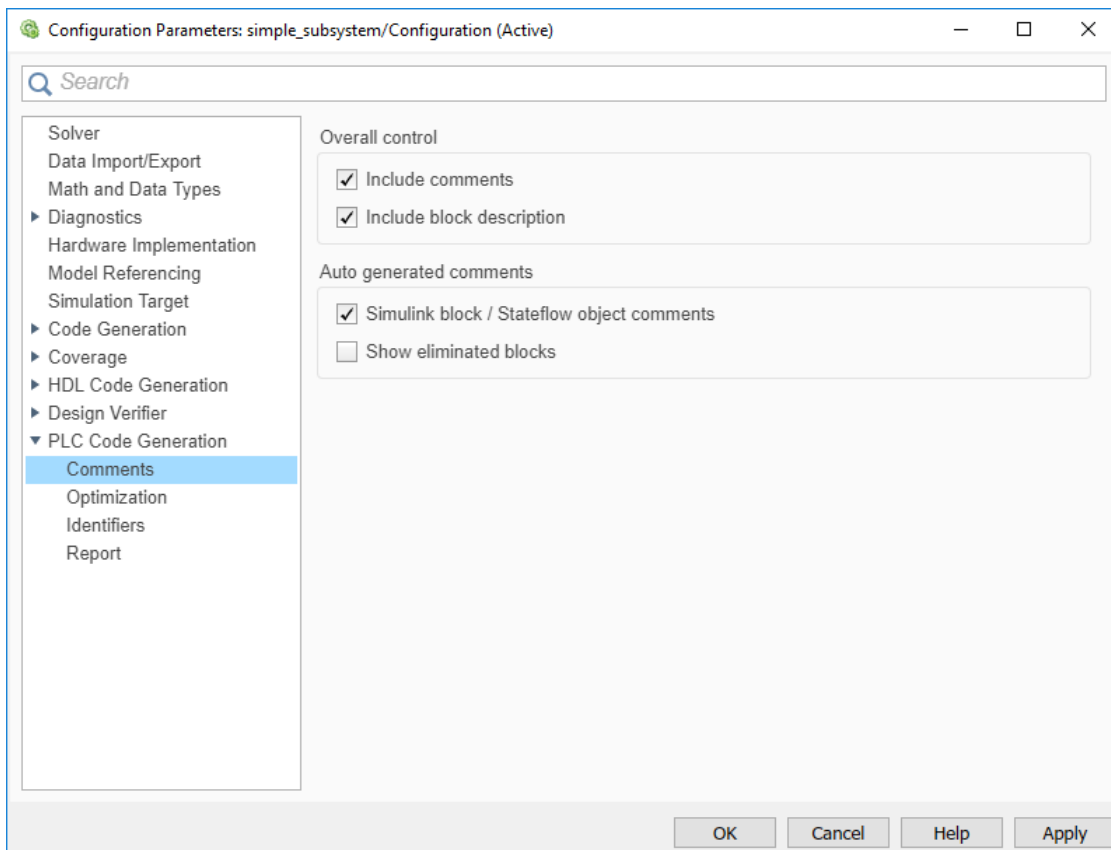
In this section...
"Comments Overview" on page 13-13
"Include Comments" on page 13-13
"Include Block Description" on page 13-14
"Simulink Block / Stateflow Object Comments" on page 13-15
"Show Eliminated Blocks" on page 13-15

Comments Overview

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

See Also

"Generate Structured Text from the Model Window" on page 1-7

Include Comments

Specify which comments are in generated files. This option is available on the **PLC Code Generation > Comments** pane in the Configuration Parameters dialog box.

Settings

Default: on

☑ On

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

> If you create links to requirements documents from your model using the Requirements Toolbox software, the links also appear in generated code comments.

☐ Off

> Omits comments from the generated files.

Command-Line Information
Parameter: PLC_RTWGenerateComments
Type: string
Value: 'on' | 'off'
Default: 'on'

See Also

"Generate Structured Text from the Model Window" on page 1-7

Include Block Description

Specify which block description comments are in generated files. This option is available on the **PLC Code Generation > Comments** pane in the Configuration Parameters dialog box.

Settings

Default: on

☑ On

> Places comments in the generated files based on the contents of the block properties **General** tab.

☐ Off

> Omits block descriptions from the generated files.

Command-Line Information
Parameter: PLC_PLCEnableBlockDescription
Type: string
Value: 'on' | 'off'
Default: 'on'

Simulink Block / Stateflow Object Comments

Specify whether to insert Simulink block and Stateflow object comments. This option is available on the **PLC Code Generation > Comments** pane in the Configuration Parameters dialog box.

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'

Show Eliminated Blocks

Specify whether to insert eliminated block comments. This option is available on the **PLC Code Generation > Comments** pane in the Configuration Parameters dialog box.

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

"Generate Structured Text from the Model Window" on page 1-7

PLC Coder: Optimization
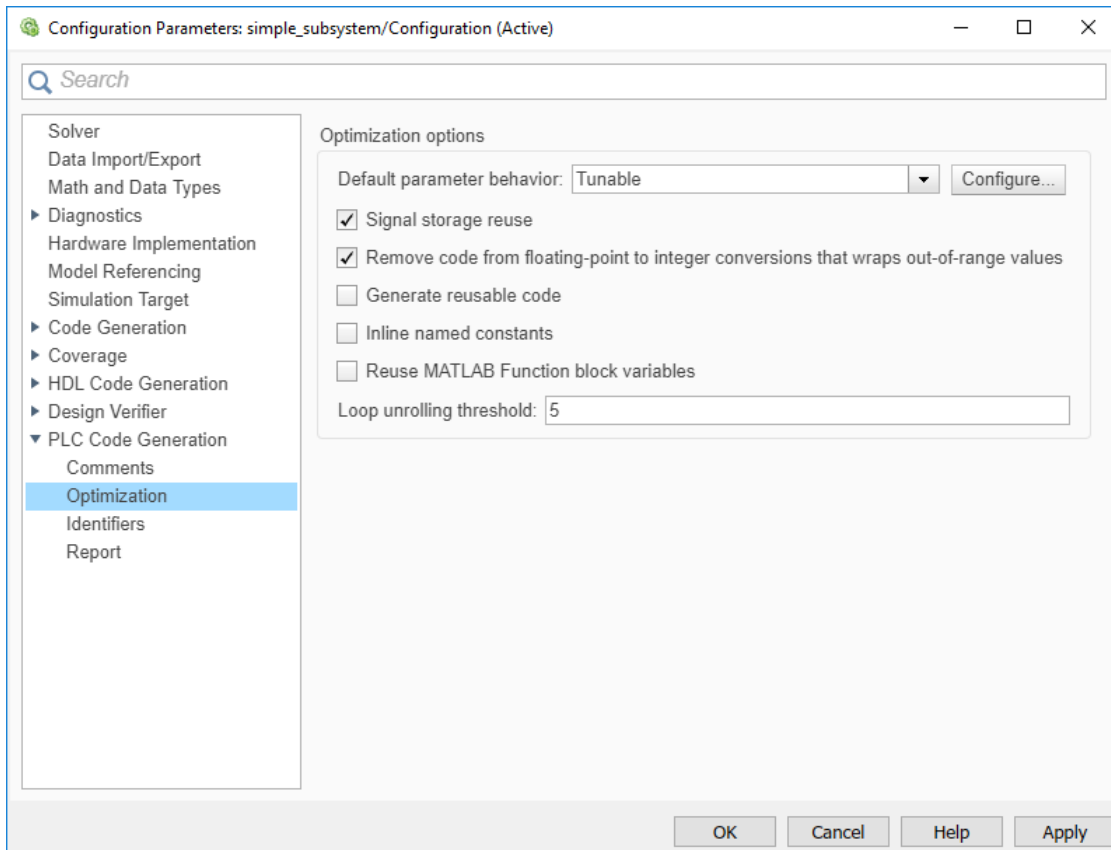
In this section...

Optimization Overview

Select the code generation optimization settings.

Default Parameter Behavior

Transform numeric block parameters into constant inlined values in the generated code. This option is available on the **PLC Code Generation > Optimization** pane in the Configuration Parameters dialog box.

Description

Transform numeric block parameters into constant inlined values in the generated code.

Category: Optimization

Settings

Default: `Tunable` for GRT targets | `Inlined` for ERT targets

`Inlined`

Set **Default parameter behavior** to `Inlined` to reduce global RAM usage and increase efficiency of the generated code. The code does not allocate memory to represent numeric block parameters such as the **Gain** parameter of a Gain block. Instead, the code inlines the literal numeric values of these block parameters.

`Tunable`

Set **Default parameter behavior** to `Tunable` to enable tunability of numeric block parameters in the generated code. The code represents numeric block parameters and variables that use the storage class `Auto`, including numeric MATLAB variables, as tunable fields of a global parameters structure.

Tips

- Whether you set **Default parameter behavior** to `Inlined` or to `Tunable`, create parameter data objects to preserve tunability for block parameters. For more information, see "Create Tunable Calibration Parameter in the Generated Code" (Simulink Coder).

- When you switch from a system target file that is not ERT-based to one that is ERT-based, **Default parameter behavior** sets to `Inlined` by default. However, you can change the setting of **Default parameter behavior** later.

- When a top model uses referenced models, or if a model is referenced by another model:

  - All referenced models must set **Default parameter behavior** to `Inlined` if the top model has **Default parameter behavior** set to `Inlined`.

  - The top model can specify **Default parameter behavior** as `Tunable` or `Inlined`.

- If your model contains an Environment Controller block, you can suppress code generation for the branch connected to the Sim port if you set **Default parameter behavior** to `Inlined` and the branch does not contain external signals.

Command-Line Information

Parameter:PLC_PLCEnableVarReuse
Type: string
Value: 'on' | 'off'
Default: 'on'

See Also

"Generate Structured Text from the Model Window" on page 1-7

Signal Storage Reuse

Reuse signal memory. This option is available on the **PLC Code Generation** > **Optimization** pane in the Configuration Parameters dialog box.

Settings

Default: on

☑ On

> Reuses memory buffers allocated to store block input and output signals, reducing the memory requirement of your real-time program.

☐ Off

> Allocates a separate memory buffer for each block's outputs. This allocation makes block outputs global and unique, which in many cases significantly increases RAM and ROM usage.

Tips

- This option applies only to signals with storage class Auto.

- Signal storage reuse can occur among only signals that have the same data type.

- Clearing this option can substantially increase the amount of memory required to simulate large models.

- Clear this option if you want to:

  - Debug a C-MEX S-function.

  - Use a Floating Scope or a Display block with the **Floating display** option selected to inspect signals in a model that you are debugging.

- If you select **Signal storage reuse** and attempt to use a Floating Scope or floating Display block to display a signal whose buffer has been reused, an error dialog box opens.

Command-Line Information

Parameter:PLC_PLCEnableVarReuse
Type: string
Value: 'on' | 'off'
Default: 'on'

See Also

"Generate Structured Text from the Model Window" on page 1-7

Remove Code from Floating-Point to Integer Conversions That Wraps Out-Of-Range Values

Enable code removal for efficient casts. This option is available on the **PLC Code Generation** > **Optimization** pane in the Configuration Parameters dialog box.

Settings

Default: on

☑ On

    Removes code from floating-point to integer conversions.

☐ Off

    Does not remove code from floating-point to integer conversions.

Tips

Use this parameter to optimize code generation.

Command-Line Information

Parameter: PLC_PLCEnableEfficientCast
Type: string
Value: 'on' | 'off'
Default: 'on'

See Also

"Generate Structured Text from the Model Window" on page 1-7

Generate Reusable Code

Using this option, you can generate better reusable code for reusable subsystems. For instance, if your model contains multiple instances of the same subsystem and some instances have constant inputs, by default, the generated code contains separate function blocks for each instance. If you select this option, the software does not consider whether the inputs to the subsystem are constant and generates one function block for the multiple instances.

This option is available on the **PLC Code Generation** > **Optimization** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

   Generates better reusable code for reusable subsystems.

☐ Off

   Reverts to its default behavior. Instead of a single reusable function block, the software generates separate function blocks for individual instances of a reusable subsystem because of certain differences in their inputs.

Tips

*   If you find multiple function blocks in your generated code for multiple instances of the same subsystem, select this option. The software performs better identification of whether two instances of a subsystem are actually the same and whether it can combine the multiple blocks into one reusable function block.

*   If different instances of a subsystem have different values of a block parameter, you cannot generate reusable code. Clear this option or use the same block parameter for all instances.

*   Despite selecting this option, if you do not see reusable code for different instances of a subsystem, you can determine the reason. To determine if two reusable subsystems are identical, the code generator internally uses a checksum value. You can compare the checksum values for two instances of a subsystem and investigate why they are not identical.

    To get the checksum values for the two instances that you expect to be identical, use the function `Simulink.SubSystem.getChecksum`. If the checksum values are different, investigate the checksum details to see why the values are not identical.

Command-Line Information

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

See Also

*   "Generate Structured Text from the Model Window" on page 1-7
*   "Generated Code Structure for Reusable Subsystems" on page 2-4

Inline Named Constants

Using this option, you can control inlining of global named constants. By default, the generated code contains named `ssMethodType` constants for internal states or other Simulink semantics. If you select this option, the software replaces the named constants with its integer value.

This option is available on the **PLC Code Generation > Optimization** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

Inlines named constants.

☐ Off

Reverts to its default behavior and uses named constants in the generated code.

Command-Line Information

Parameter:PLC_InlineNamedConstant
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

- "Generate Structured Text from the Model Window" on page 1-7
- "Generated Code Structure for Simple Simulink Subsystems" on page 2-2

Reuse MATLAB Function Block Variables

You can use this option to enable reuse of MATLAB function block variables in the generated code.

This option is available on the **PLC Code Generation** > **Optimization** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

Generates code that reuses MATLAB Function block variables where appropriate.

☐ Off

Reverts to its default behavior and does not reuse variables in the generated code.

Command-Line Information

Parameter:PLC_ReuseMLFcnVariable
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

- "Generate Structured Text from the Model Window" on page 1-7
- "Generated Code Structure for MATLAB Function Block" on page 2-12

Fold temporary variable using function block instance output variable

Use this to optimize the code for the output variables of function blocks generated from reusable Simulink subsystems, Stateflow charts, or MATLAB Function blocks by removing temporary variable assignments that are not required. This option is available on the **PLC Code Generation> Optimization** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

   Generates optimized code for output variables of function blocks by folding temporary variables into a single expression.

☐ Off

   The generated code for function block instance output variables uses temporary variable assignments.

Command-Line Information

Parameter:PLC_FoldFBCallOutputVar
Type: string
Value: 'on' | 'off'
Default: 'off'

Loop Unrolling Threshold

Specify the minimum signal or parameter width for which a for loop is generated. This option is available on the **PLC Code Generation > Optimization** pane in the Configuration Parameters dialog box.

Settings

Default: 5

Specify the array size at which the code generator begins to use a `for` loop instead of separate assignment statements to assign values to the elements of a signal or parameter array.

When the loops are perfectly nested loops, the code generator uses a `for` loop if the product of the loop counts for all loops in the perfect loop nest is greater than or equal to this threshold.

Command-Line Information

Parameter: PLC_RollThreshold
Type: string
Value: any valid value
Default: '5'

See Also

"Generate Structured Text from the Model Window" on page 1-7

PLC Coder: Identifiers

In this section...

In this section...
"Inline Enum Cast Function" on page 13-30

Identifiers Overview

Select the automatically generated identifier naming rules.

See Also

"Generate Structured Text from the Model Window" on page 1-7
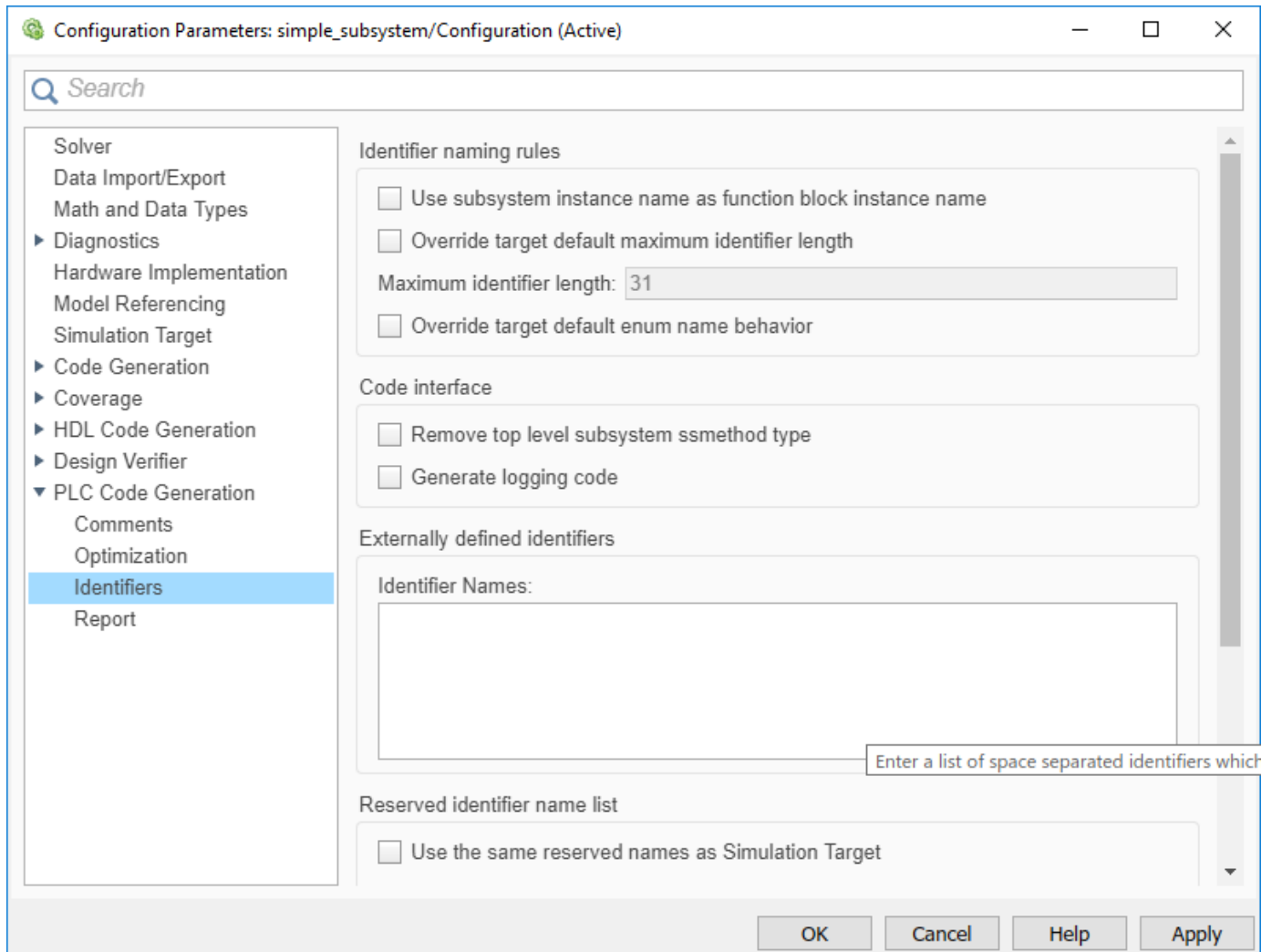
Use Subsystem Instance Name as Function Block Instance Name

Specify how you want the software to name the Function block instances it generates for the subsystem. When you select this option, the software uses the subsystem instance name as the name of the Function blocks in the generated code. By default, the software generates index-based instance names.

This option is available on the **PLC Code Generation** > **Identifiers** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

> Uses the subsystem instance name as the name of the Function block instances in the generated code.

☐ Off

> Uses auto-generated index-based instance names for the Function blocks in the generated code.

Command-Line Information
Parameter: PLC_FBUseSubsystemInstanceName
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

"Generate Structured Text from the Model Window" on page 1-7

Override Target Default Maximum Identifier Length

If your custom target IDE version supports long name identifiers, you can use this option along with the **Maximum identifier length** to specify the maximum number of characters in the generated function, type definition, and variable names. By default, the software complies with the maximum

identifier length of standard versions of the target IDE and ignores unsupported values specified in the **Maximum identifier length**.

This option is available on the **PLC Code Generation** > **Identifiers** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

   Override target default maximum identifier length in the generated code.

☐ Off

   The generated code uses the default identifier length of the target IDE.

Command-Line Information
Parameter: `PLC_OverrideDefaultNameLength`
Type: string
Value: `'on'` | `'off'`
Default: `'off'`

See Also

"Generate Structured Text from the Model Window" on page 1-7

Maximum Identifier Length

Specify the maximum number of characters in generated function, type definition, and variable names. This option is available on the **PLC Code Generation** > **Identifiers** pane in the Configuration Parameters dialog box.

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 for these names. Simulink PLC Coder complies with target IDE limitations.

Command-Line Information
Parameter: `PLC_RTWMaxIdLength`
Type: int
Value: 31 to 256
Default: 31

See Also

"Generate Structured Text from the Model Window" on page 1-7

Override Target Default enum Name Behavior

Use this option to enable enum names to be used as the identifier names instead of enum values. The PLC target IDE must support enum type.

This option is available on the **PLC Code Generation** > **Identifiers** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

> Override target default enum behavior and always have enum names instead of enum values.

☐ Off

> The generated code uses the enum behavior of the target IDE.

Command-Line Information
Parameter: PLC_GenerateEnumSymbolicName
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

"Generate Structured Text from the Model Window" on page 1-7

Generate enum Cast Function

Autogenerate the enum type conversion code. The target PLC IDE must support enum type.

This option is available in the Configuration Parameters dialog box, **PLC Code Generation** > **Identifiers** pane .

Settings

Default: off

☑ On

> Simulink PLC Coder autogenerates the enum type conversion code.

☐ Off

> Manually create a MATLAB function to convert the enum type value to an integer or to convert an integer to an enum type value.

Command-Line Information
Parameter: PLC_GenerateEnumCastFunction
Type: string

Value: `'on'` | `'off'`
Default: `'off'`

See Also

"Generate Structured Text from the Model Window" on page 1-7

Use the Same Reserved Names as Simulation Target

Specify whether to use the same reserved names as those specified in the **Reserved names** field of the **Simulation Target** pane in the Configuration Parameters dialog box. This option is available on the **PLC Code Generation** > **Identifiers** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

Uses the same reserved names as those specified in the **Reserved names** filed of the **Simulation Target** pane in the Configuration Parameters dialog box.

☐ Off

Does not use the same reserved names as those specified in the **Simulation Target** > **Identifiers pane** pane.

Command-Line Information
Parameter: `PLC_RTWUseSimReservedNames`
Type: string
Value: `'on'` | `'off'`
Default: `'off'`

See Also

"Generate Structured Text from the Model Window" on page 1-7

Reserved Names

Enter the names of variables or functions in the generated code that you do not want to be used. This option is available on the **PLC Code Generation** > **Identifiers** pane in the Configuration Parameters dialog box.

Settings

Default: ( )

Changes the names of variables or functions in the generated code to avoid name conflicts with identifiers in custom code. Reserved names must be fewer than 256 characters in length.

Tips

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

Command-Line Information
Parameter: PLC_RTWReservedNames
Type: string
Value: string
Default: ''

See Also

"Generate Structured Text from the Model Window" on page 1-7

Externally Defined Identifiers

Specify the names of identifiers for which you want to suppress definitions. This option is available on the **PLC Code Generation** > **Identifiers** pane in the Configuration Parameters dialog box.

Settings

Default: ( )

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

Tips

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

Command-Line Information
Parameter: PLC_ExternalDefinedNames
Type: string
Value: string
Default: ''

See Also

- "Generate Structured Text from the Model Window" on page 1-7
- "Integrate Externally Defined Identifiers" on page 9-2
- Integrating User Defined Function Blocks, Data Types, and Global Variables into Generated Structured Text

Preserve Alias Type Names for Data Types

Specify that the generated code must preserve alias data types from your model. This option is available on the **PLC Code Generation > Identifiers** pane in the Configuration Parameters dialog box.

Using the `Simulink.AliasType` class, you can create an alias for a built-in Simulink data type. If you assign an alias data type to signals and parameters in your model, when you use this option, the generated code uses your alias data type to define variables corresponding to the signals and parameters.

For instance, you can create an alias `SAFEBOOL` from the base data type `boolean`. If you assign the type `SAFEBOOL` to signals and parameters in your model, the variables in the generated code corresponding to those signals and parameters also have the type `SAFEBOOL`. Using this alias type `SAFEBOOL`, you can conform to PLCopen safety specifications that suggest using safe data types for differentiation between safety-relevant and standard signals.

Settings

Default: off

☑ On

> The generated code preserves alias data types from your model.
>
> For your generated code to be successfully imported to your target IDE, the IDE must support your alias names.

☐ Off

> The generated code does not preserve alias types from your model. Instead, the base type of the `Simulink.AliasType` class determines the variable data type in generated code.

Tips

The alias that you define for a Simulink type must have the same semantic meaning as the base Simulink type. It must not be a data type already supported in Structured Text and semantically different from the base Simulink type. For instance, `WORD` is a data type supported in Structured Text but is semantically different from an integer type. If you define an alias `WORD` for a Simulink built-in integer type, for instance `uint16`, and preserve the alias name, the type `WORD` that appears in your generated code is used semantically as a `WORD` and not as an `INT`. The generated code has a different meaning from the semantics of the model.

Command-Line Information
Parameter: `PLC_PreserveAliasType`
Type: string
Value: `'on'` | `'off'`
Default: `'off'`

Inline Enum Cast Function

Use this option to inline the generated enum-to-integer or integer-to-enum function. By default, the software generates an enum-to-integer or integer-to-enum function as a part of the generated code. This option is available in the Configuration Parameters dialog box, **PLC Code Generation > Identifiers** pane. Select the **Generate enum cast function** option.

Settings

Default: off

☑ On

Inline the generated enum cast function.

☐ Off

Do not inline the generated enum cast function.

Command-Line Information
Parameter: `PLC_InlineEnumCastFunction`
Type: string
Value: `'on'` | `'off'`
Default: `'off'`

PLC Coder: Report
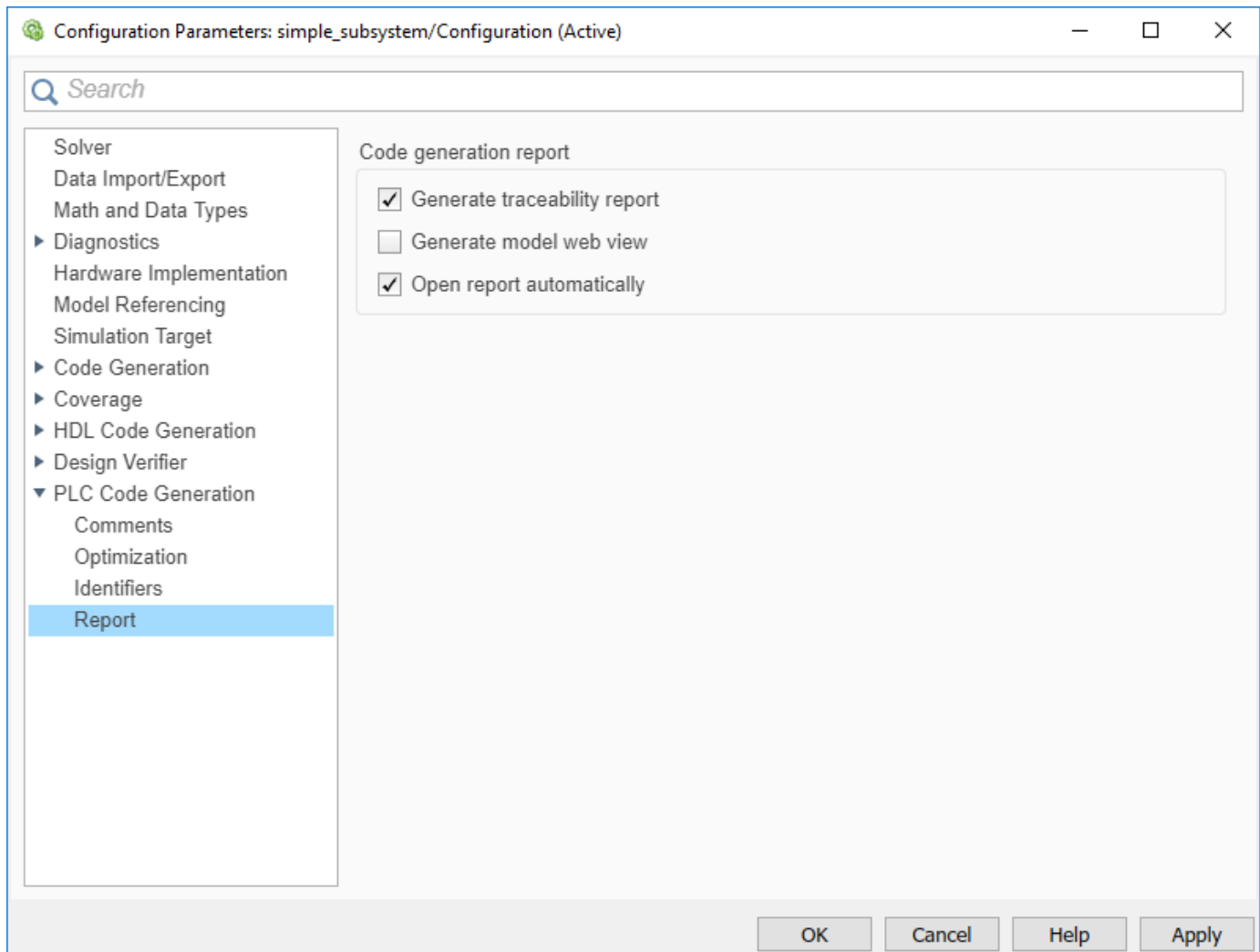
In this section...
"Report Overview" on page 13-32
"Generate Traceability Report" on page 13-33
"Generate Model Web View" on page 13-33
"Open Report Automatically" on page 13-34

Report Overview

After code generation, specify whether a report must be produced. Control the appearance and contents of the report.

The code generation report shows a mapping between Simulink model objects and locations in the generated code. The report also shows static code metrics about files, global variables, and function blocks.

See Also

"Generate Structured Text from the Model Window" on page 1-7

Generate Traceability Report

Specify whether to create a code generation report. This option is available on the **PLC Code Generation > Report** pane in the Configuration Parameters dialog box.

Settings

Default: on

☑ On

  Creates code generation report as an HTML file.

☐ Off

  Suppresses creation of code generation report.

Command-Line Information
Parameter: PLC_GenerateReport
Type: string
Value: 'on' | 'off'
Default: 'on'

See Also

"Generate Structured Text from the Model Window" on page 1-7

Traceability Report Limitations

Simulink PLC Coder does not generate a traceability report file when generating Ladder Diagrams from Stateflow charts. However, traceability report file is generated when generating Structured Text from Stateflow charts.

Ladder Diagrams. charts. However, traceability report file is generated when generating Structured Text from charts.

Generate Model Web View

To navigate between the code and the model within the same window, include the model web view in the code generation report. This option is available on the **PLC Code Generation > Report** pane in the Configuration Parameters dialog box.

You can share your model and generated code outside of the MATLAB environment. You must have a Simulink Report Generator to include a Web view (Simulink Report Generator) of the model in the code generation report.

Settings

Default: Off

☑ On

Includes model Web view in the code generation report.

☐ Off

Omits model Web view in the code generation report.

Command-Line Information
Parameter: PLC_GenerateWebView
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

"Generate Structured Text from the Model Window" on page 1-7

Open Report Automatically

Specify whether to open the code generation report automatically. This option is available on the **PLC Code Generation** > **Report** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

Opens the code generation report as an HTML file.

☐ Off

Suppresses opening of the code generation report.

Command-Line Information
Parameter: PLC_LaunchReport
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

"Generate Structured Text from the Model Window" on page 1-7

PLC Coder:Interface
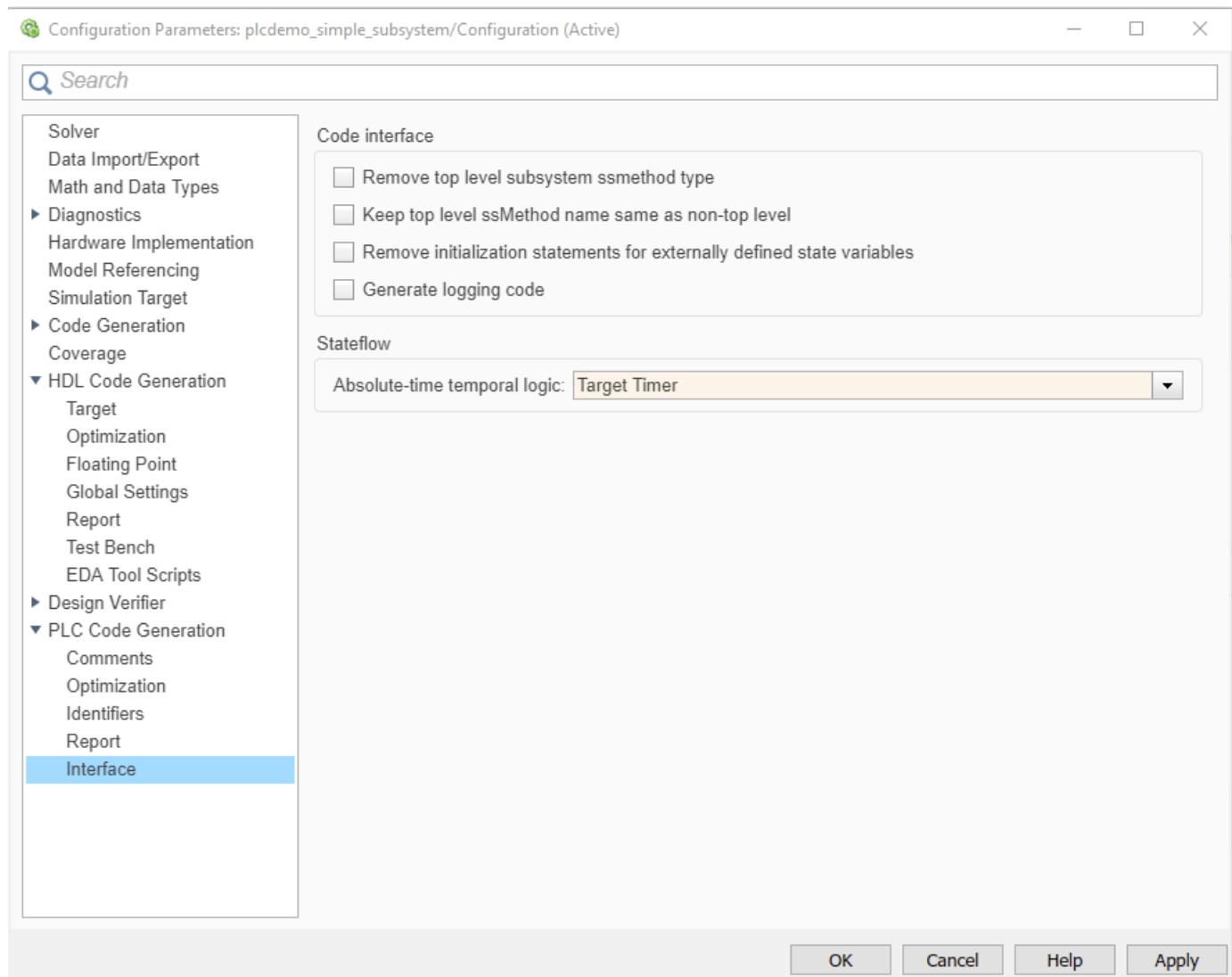
In this section...

Interface Overview

The **PLC Code Generation > Interface** category includes parameters for configuring the interface of the generated code.

See Also

"Generate Structured Text from the Model Window" on page 1-7

Generate Logging Code

With this option, you can generate code with logging instrumentation to collect run-time data on supported PLC targets. The PLC target IDEs must have support for `inout` variables. For Rockwell Automation targets, you can set up an Open Platform Communications (OPC) server and use the Simulation Data Inspector (SDI) in Simulink to visualize and monitor the logging data.

This option is available on the **PLC Code Generation > Interface** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

   Generate Function block logging code for supported targets.

☐ Off

   No logging instrumentation is included in the generated code.

Command-Line Information
Parameter: PLC_GenerateLoggingCode
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

"Generate Structured Text from the Model Window" on page 1-7

Keep Top-Level ssmethod Name the Same as the Non-Top Level Name

Prevent renaming the SS_OUTPUT type to SS_STEP type from the top-level subsystem argument interface. When you select this option, the software emits the same ssMethod type in the code generation for both top and non-top level blocks.

This option is available on the **PLC Code Generation > Interface** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

> Generated code for top-level block does not contain the SS_STEP type in generated code.

☐ Off

> Generated code contains SS_STEP AND SS_OUTPUT type function blocks.

Command-Line Information
Parameter: PLC_RemoveSSStep
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

- "Distributed Model Code Generation Options" on page 24-2
- "Generated Code Structure for PLC_RemoveSSStep" on page 24-3

Remove Top-level Subsystem Ssmethod Type

Use this option to remove the ssmethod type from the top-level subsystem argument interface. When this option is enabled, the software removes the ssmethod type and converts the subsystem initialization code from switch case statement to conditional if statement. As a result, the generated code has the same interface as the model subsystem.

This option is available on the **PLC Code Generation** > **Interface** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

> Remove top level function block ssmethod type in generated code.

☐ Off

> Generated code contains ssmethod type Function block and switch case statements.

Command-Line Information
Parameter: PLC_RemoveTopFBSSMethodType
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

"Generate Structured Text from the Model Window" on page 1-7

Remove Initialization Statements for Externally Defined State Variables

Use this option to remove initialization assignment statements for variables that have storage class `ImportedExtern` and `ExportedGlobal` from the generated code.

Mark `ExportedGlobal` variables as externally defined. For more information, see "Externally Defined Identifiers" on page 13-29

Settings

Default: off

☑ On

> Remove from the generated code initialization assignment statements for variables that have storage class `ImportedExtern` and `ExportedGlobal`.

☐ Off

> Generated code contains initialization assignment statements for variables that have storage class `ImportedExtern` and `ExportedGlobal`.

Command-Line Information
Parameter: `PLC_PreventExternalVarInitialization`
Type: string
Value: `'on'` | `'off'`
Default: `'off'`

See Also

- "Distributed Model Code Generation Options" on page 24-2
- "Generated Code Structure for PLC_PreventExternalVarInitialization" on page 24-5

Absolute-Time Temporal Logic

Use this option to specify if the generated code uses the target timer or a target-independent counter for Stateflow absolute-time temporal logic semantics implementation.

Settings

Default: `Target Timer`

`Target Timer`

> Generated code uses the target timer to implement Stateflow absolute-time temporal logic semantics.

`Target-independent Counter`

> Generated code contains a target-independent integer counter to implement Stateflow absolute-time temporal logic semantics.

Command-Line Information
Parameter: PLC_AbsTimeTemporalLogic
Type: character vector
Value: 'timer' | 'counter'
Default: 'timer'

Limitations

- Absolute-time temporal logic does not support stateflow chart using global clocks.
- Testbench code verification can fail for absolute-time temporal logic using floating-point comparison operations.

Exclude block definitions as Functions

Use this option to enter a list of full paths to model components whose definitions you want to exclude from the generated code.

Settings

Default: empty path

Suppresses the definition of model components in the generated code. When you import the generated code into the PLC IDE, you must provide definitions for the excluded modules, which are generated as functions. Specify the full paths to the model components that you want to exclude as a list of comma-separated values.

Limitations

- The model components that you exclude must be:

  - Subsystem blocks
  - MATLAB Function blocks
  - Stateflow charts

- Top-level subsystems marked as external are not allowed.
- The same block must not be included in the `Exclude block definitions as function blocks`.
- The model components that are marked for exclusion, must not:

  - Have cell array data types as inputs or outputs.
  - Contain virtual bus components.
  - Have any input or output ports that use array data types.

- The data types used in the model components to be excluded must be of only type numeric or struct.
- Model subsystem blocks that have a single output which is of struct or array data type cannot be marked for exclusion.

Command-Line Information
Parameter: PLC_ExcludeBlocksAsFunction
Type: string

Value: `string`
Default: `''`

Exclude block definitions as Function blocks

Use this option to enter a list of full paths to model components whose definitions you want to exclude from the generated code.

Settings

Default: empty path

Suppresses the definition of model components in the generated code. When you import the generated code into the PLC IDE, you must provide definitions for the excluded modules, which are generated as function blocks. Specify the full paths to the model components you want to exclude as a list of comma-separated values.

Limitations

- The model components that you exclude must be:

  - Subsystem blocks
  - MATLAB Function blocks
  - Stateflow charts

- Top-level subsystems marked as external are not allowed.
- The same block must not be included in the `Exclude block definitions as functions`.
- The model components that are marked for exclusion, must not:

  - Have cell array data types as inputs or outputs.
  - Contain virtual bus components.

- The data types used in the model components to be excluded must be of only type numeric or struct.

Command-Line Information
Parameter: `PLC_ExcludeBlocksAsFunctionBlock`
Type: string
Value: `string`
Default: `''`

External Mode

External Mode Logging

External mode logging uses specific code that Simulink PLC Coder™ generates. External mode logging can save system states, outputs, and simulation time at each model execution time step.

You can verify your generated code by collecting run-time data while executing the code on the target PLC IDE. Collect run-time data on PLC targets by using external mode logging and visualize the run-time data by using Simulation Data Inspector.

Generate code from Simulink models with external mode logging enabled by using the **Generate logging code** feature. To use external mode logging, the target PLC IDE must support `InOut` variables. These target PLC IDEs, support external mode logging:

- 3S-Smart Software Solutions CoDeSys Version 2.3
- 3S-Smart Software Solutions CoDeSys Version 3.5
- Rockwell Automation RSLogix 5000
- Rockwell Automation Studio 5000
- Beckhoff TwinCAT 2.11
- Beckhoff TwinCAT 3
- Generic
- PLCopen XML
- Rexroth IndraWorks
- OMRON Sysmac Studio

Visualize the logging data for Rockwell Automation targets, by using an Open Communications Platform (OPC) server and Simulation Data Inspector. The OPC Toolbox™ is required to run the external mode visualization.

See Also

More About

Generate Structured Text Code That Has Logging Instrumentation

This example shows how to generate code for the Rockwell Automation Studio 5000 IDE by using external mode logging.

1   Create a Simulink model `ext_demo1.slx` that has a top-level subsystem with two child subsystems, S1, S2, a MATLAB Function block, and a Stateflow chart.
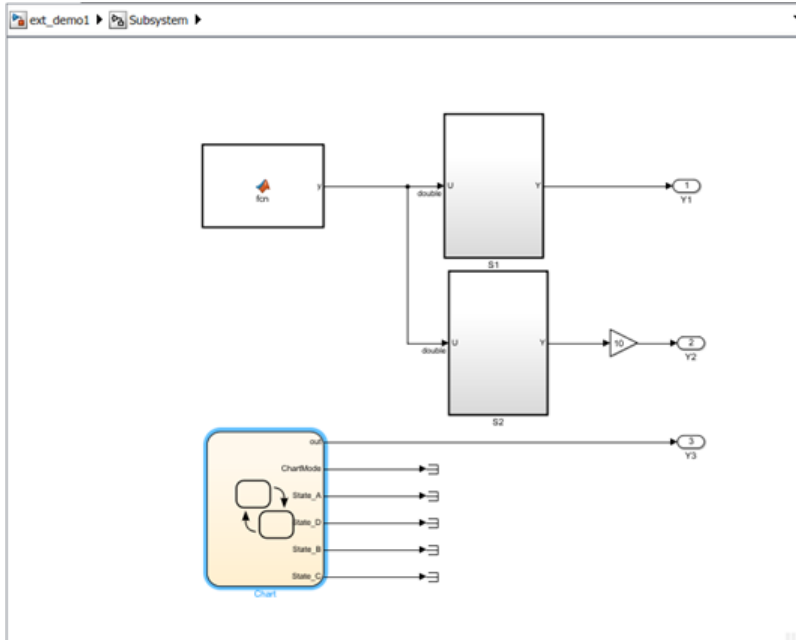
The S1 and S2 blocks are identical and contain a simple feedback loop.The Stateflow chart contains a simple state machine.
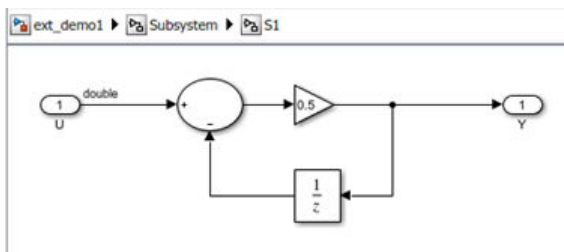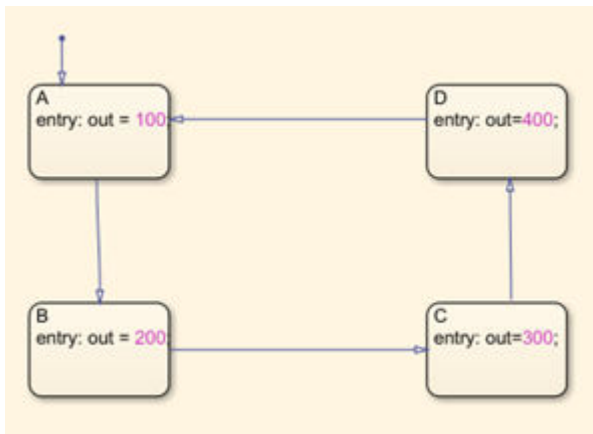
2 The MATLAB function block implements this code:

```
function y = fcn
persistent i;

if isempty(i)
    i=0;
end

if (i>20)
    i = 0;
else
    i=i+1;
end

y = sin(pi*i/10);
```

3 Select the top-level subsystem and open the **PLC Coder** app. On the **PLC Code** tab, click **Settings > PLC Code Generation** and select the **Target IDE** as Rockwell Studio 5000: AOI. On the **Interface** pane, select **Generate logging code**. Click **OK**.
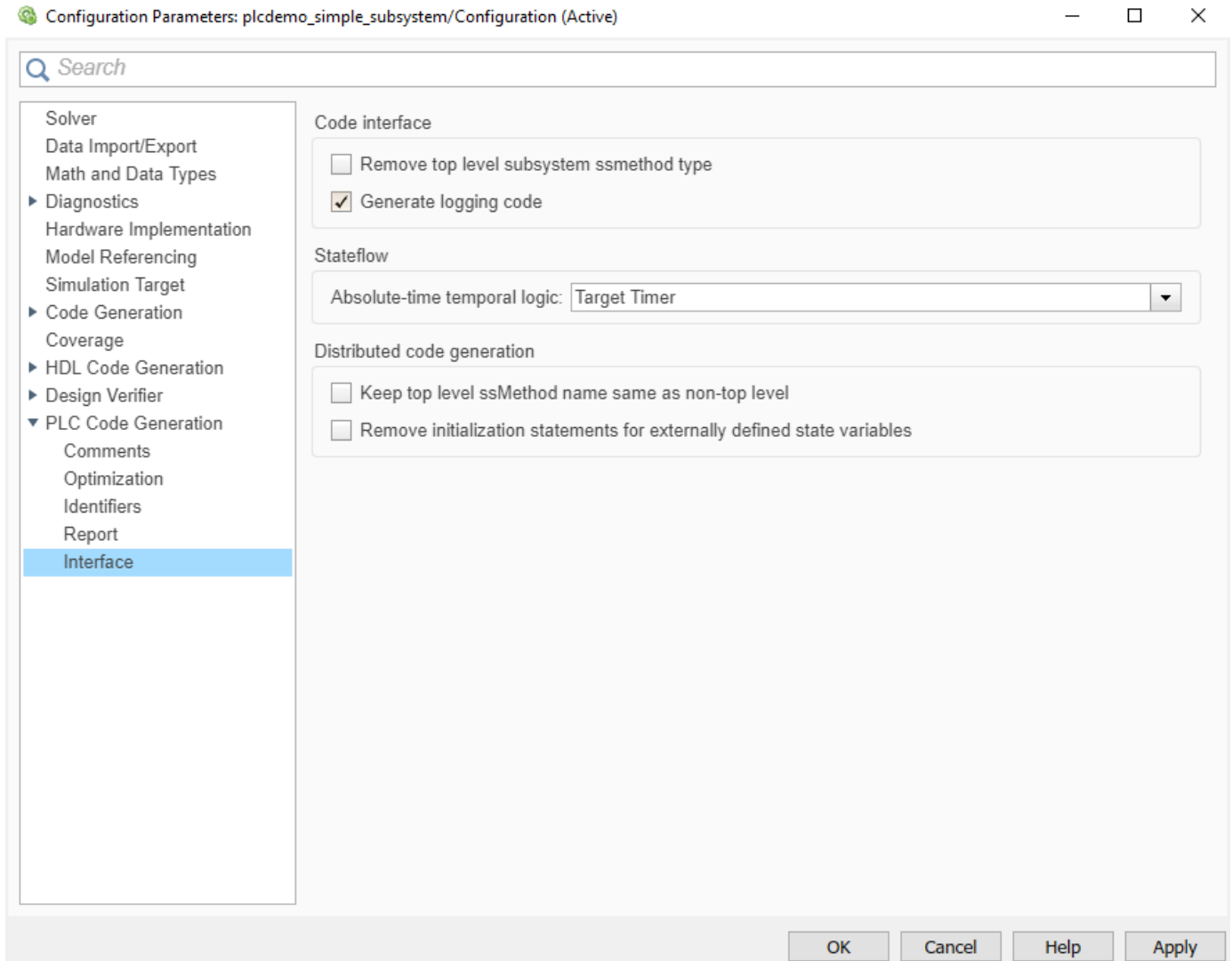
4   In the model, select the top subsystem block. On the **PLC Code** tab, click **Generate PLC Code**.

You generate the `ext_demo.L5X` code for the top subsystem block, the children S1, S2, the MATLAB function, and Stateflow chart blocks. Also generated is the `plc_log_data.mat`, which has the external logging data information.
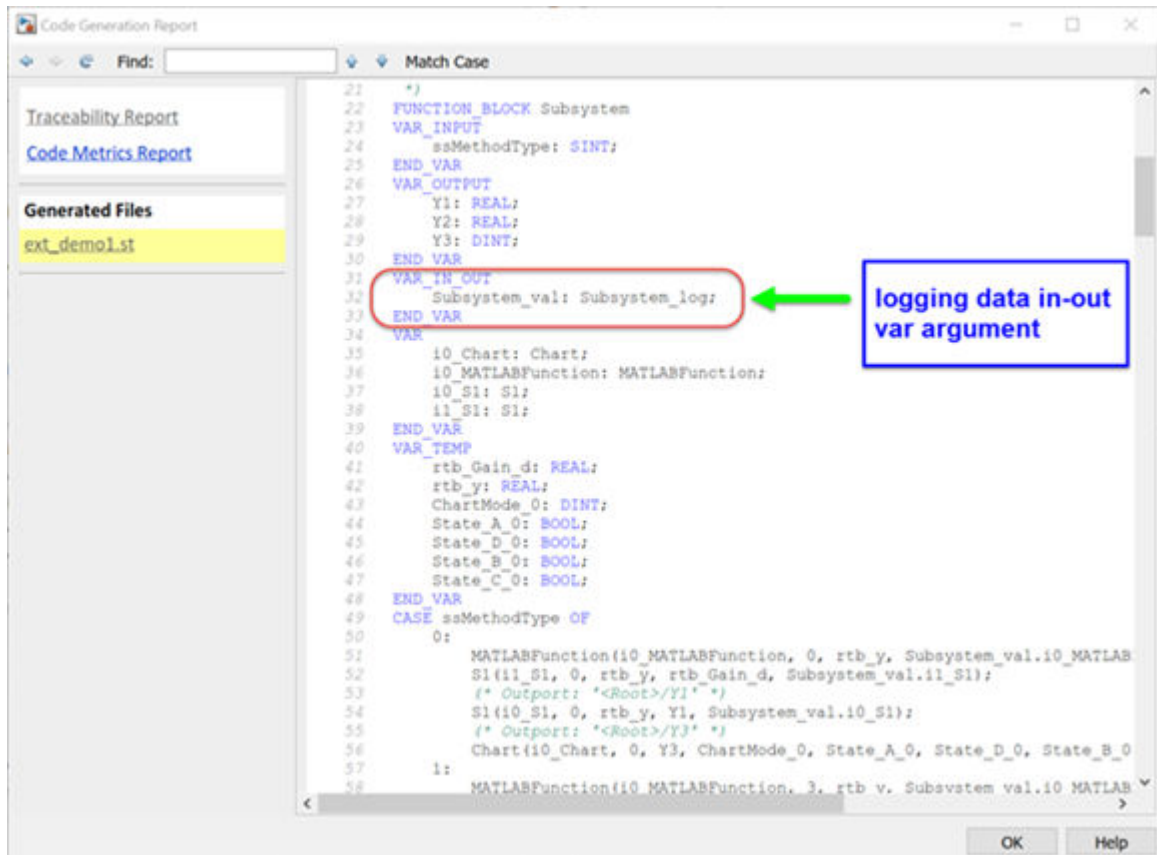
To run the `ext_demo.L5X` file in the Rockwell Automation Studio 5000 IDE, see "Visualize and Monitor Logging Data by using Simulation Data Inspector" on page 14-7.

See Also

More About

- "External Mode Logging" on page 14-2
- "Visualize and Monitor Logging Data by using Simulation Data Inspector" on page 14-7

Visualize and Monitor Logging Data by using Simulation Data Inspector

This example shows how to collect PLC run-time data for Rockwell Automation targets. Set up an Open Platform Communications (OPC) server. Use the Simulation Data Inspector in Simulink to visualize and monitor the logging data.

Set Up and Download Code to Studio 5000 IDE

1 Start the Studio 5000 IDE and create a project with the name `ext_demo1`.

2 Import the generated `ext_demo.L5X` to the Add-On Instructions tree node of the project. For more information, see "Generate Structured Text Code That Has Logging Instrumentation" on page 14-3.

3 In the `MainProgram` node, delete the ladder `MainRoutine` and create an ST `MainRoutine` node.
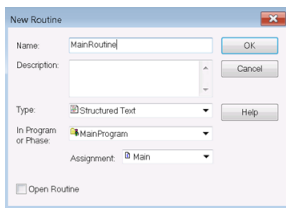
4 In ST `MainRoutine`, define the tags listed in this table.

Tag Name	Tag Type
`i0_Subsystem`	`Subsystem`
`i0_Subsystem_val`	`Subsystem_log`
`Init`	`BOOL`
`Y1`	`REAL`
`Y2`	`REAL`
`Y3`	`DINT`

5 In Studio 5000 IDE, `i0_Subsystem tag` is the instance of the top subsystem AOI and the `i0_Subsystem_val tag` is the logging data with structure type `Subsystem_log`. Set the initial value of `init` tag to 1.

6 Double-click the `MainRoutine` tree node and type in the code in the image. The statement `Subsystem(i0_Subsystem, 23, Y1, Y2, Y3, i0_Subsystem_val)` calls the logging method (`ssmethod value=23`) to log in data to the `i0_Subsystem_val` tag.
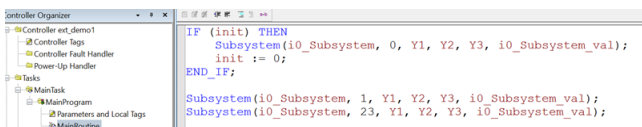
7 Compile the project in Studio 5000 IDE and download to the PLC target.

Configure RSLinx OPC Server

1 Start RSLinx Classic Gateway and select the menu item `DDE/OPC->Topic Configuration`.

2 In the dialog box, create a topic `ext_demo1` by clicking the `New` button. Select the target PLC from the PLC list.
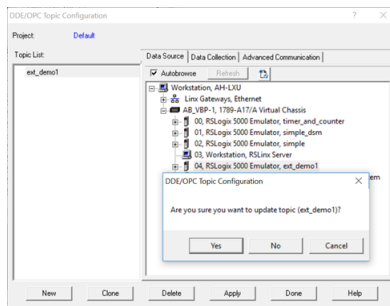
3 Click the `Yes` button to update the topic (`ext_demo1`).

4 To verify that the log data is set up on the OPC server, select the menu item `Edit->Copy DDE/OPC Link`. The `i0_Subsystem_val` tag for log data must be shown on the RSLinx OPC Server.
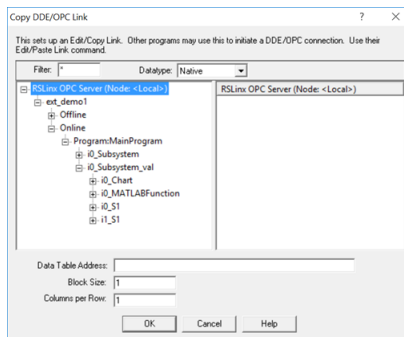
Stream and Display Live Log Data by Using PLC External Mode Commands

After the RSLinx OPC Server is configured, you can use the PLC external mode commands to connect to the server, stream the logging data, and display live logging data on the Simulation Data Inspector. The log data information is in the `plc_log_data.mat` file, which you can find in the `plcsrc` folder. You can use the `plcdispextmodedata` function to display the contents of the MAT-file. In the MATLAB Command Window, type:

```
>>cd plcsrc
>>plcdispextmodedata plc_log_data.mat

Log data:
#1: Y1: LREAL
#2: Y2: LREAL
#3: Y3: LREAL
#4: io_Chart.out: DINT
```

```
#5:  io_Chart.ChartMode: DINT
#6:  io_Chart.State_A: BOOL
#7:  io_Chart.State_B: BOOL
#8:  io_Chart.State_C: BOOL
#9:  io_Chart.State_D: BOOL
#10: io_Chart.is_active_c3_Subsystem: USINT
#11: io_MATLABFunction.y: LREAL
#12: io_MATLABFunction.i: LREAL
#13: io_S1.y: LREAL
#14: io_S1.UnitDelay_DSTATE: LREAL
#15: i1_S1.y: LREAL
#16: i1_S1.UnitDelay_DSTATE: LREAL
```

The format for the log data information is index number, name, and type. The log data for non-top subsystem function block output and state variables is named by using dot notation to represent the function block instances that own the data. You can use the index and name of the log data with the `plcrunextmode` command to specify a subset of log data for streaming and visualization.

To connect to the OPC server and stream log data, use the `plcrunextmode` function. For example, executing the `plcrunextmode ('localhost', 'studio5000', 'ext_demo1', 'plc_log_data.mat');` command streams live log data for the example model into Simulation Data Inspector.
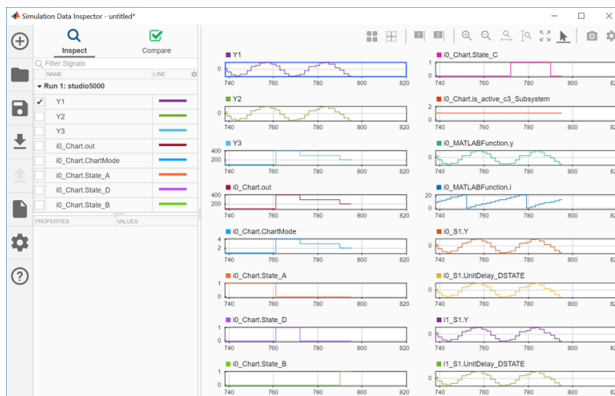
The `plcrunextmode` command continues to run and stream log data. To exit streaming, at the MATLAB command prompt, type Ctrl-C.

See Also
plcdispextmodedata | plcrunextmode

More About
- "External Mode Logging" on page 14-2
- "Generate Structured Text Code That Has Logging Instrumentation" on page 14-3

Ladder Diagram Instructions

Instructions Supported in Ladder Diagram

The supported ladder diagram instructions are useful while importing the ladder into Simulink. The instructions can be categorised into two:

- Instructions that are implemented in Simulink using ladder diagram blocks with same name
- Instructions that are implemented in Simulink using other ladder diagram blocks.

The table lists the instructions that map to blocks in Simulink

L5X Instructions	Ladder Model Blocks
ADD	ADD Block
AFI	AFI Block
AND	AND Block
CLR	CLR Block
COP	COP Block
CTD	CTD Block
CTU	CTU Block
DIV	DIV Block
EQU	EQU Block
FBC	FBC Block
FLL	FLL Block
GEQ	GEQ Block
GRT	GRT Block
JMP	JMP Block
LBL	LBL Block
LEQ	LEQ Block
LES	LES Block
MCR	MCR Block
MOV	MOV Block
MUL	MUL Block
NCP	NCP Block
NEQ	NEQ Block
NOT	NOT Block
OR	OR Block
OTE	OTE Block
OTL	OTL Block
OTU	OTU Block
RES	RES Block
RTO	RTO Block

L5X Instructions	Ladder Model Blocks
SUB	SUB Block
TND	TND Block
TOF	TOF Block
TON	TON Block
XIC	XIC Block
XIO	XIO Block

The special instructions that are implemented using another block in Simulink are:

• **JSR** instruction is implemented by using a **Subroutine block**.

• **AOI** call instruction is implemented by using an **Inline AOI block**

Ladder Diagram Blocks

Ladder Diagram Blocks

The Ladder Diagram Blocks that are a part of Ladder Diagram Library are listed.

XIC	XIO	OTE	OTL
OTU	TON	TOF	RTO
CTU	CTD	RES	JMP
LBL	TND	AFI	NOP
MCR	ADD	SUB	MUL
DIV	FRD	CPT	AND
OR	NOT	ONS	OSR
OSF		NEQ	EQU
LEQ	GEQ	LES	GRT
MOV	CLR	COP	FLL
Power Rail Start	Power Rail Terminal	RungTerminal	Junction
Variable Read	Variable Write	PLC Controller	Task
Program	Subroutine	Function Block (AOI)	

Fixed Point Code Generation

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**.

Model 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`.

Limitations

1 64-bit fixed-point data type is not supported.

Generating PLC Code for Multirate Models

Multirate Model Requirements for PLC Code Generation

In this section...
"Model Configuration Parameters" on page 18-2
"Limitations" on page 18-2

Model Configuration Parameters

Before generating Structured Text from a multirate model, you must configure the model as follows:

- **Solver** options that are recommended or required for PLC code generation:

  - **Type**:`Fixed-step`.
  - **Solver**:`Discrete(no continuous states)`. Other fixed-step solvers could be selected, but this option is usually best for simulating discrete systems.
  - **Tasking mode**: Must be explicitly set to `SingleTasking`. Do not set **Tasking mode** to `Auto`

- Change any continuous time signals in the top level subsystem to use discrete fixed sample times.

When you deploy code generated from a multirate model, you must run the code at the fundamental sample rate.

Limitations

These are the limitations when generating Structured Text from multirate models:

- The B&R Automation Studio IDE is not supported for multirate model code generation.

Generating PLC Code for MATLAB Function Block

Configuring the rand function for PLC Code generation

Simulink PLC Coder generates Structured Text code for MATLAB Function blocks and Stateflow charts that use `rand` functions from the library. The `rand` function is implemented using a pseudo random number generator that only works with PLC IDEs supporting the `uint32` data type. The software has conformance checks to report diagnostics for incompatible targets. Currently, the following targets have been tested for `rand` function support.

- 3S-Smart Software Solutions CODESYS Version 2.3 or 3.3 or 3.5 (SP4 or later)
- B&R Automation Studio 3.0 or 4.0
- Beckhoff TwinCAT 2.11 or 3
- OMRON Sysmac Studio Version 1.04, 1.05, 1.09 or 1.12
- Rexroth IndraWorks version 13V12 IDE
- Generic
- PLCopen XML

Width block requirements for PLC Code generation

Use a MATLAB Function block instead. In the MATLAB function on the block, use the `length` function to compute input vector width.

Workspace Parameter Data Type Limitations

If the data type of the MATLAB work space parameter value does not match that of the block parameter used in your model, the value of the variable in the generated code is set to zero.

If you specify the type of the `Simulink.Parameter` object by using the `DataType` property, use a typed expression when assigning a value to the parameter object. For example, if the `Simulink.Parameter` object `K1` is used to store a value of the type `single`, use a typed expression such as `single(0.3)` when assigning a value to `K1`.

```
K1 = Simulink.Parameter;
K1.Value = single(0.3);
K1.StorageClass = 'ExportedGlobal';
K1.DataType = 'single';
```

Limitations

These are the limitations when generating Structured Text from MATLAB Fubnction blocks :

- Cell arrays in MATLAB Function blocks
- In MATLAB Function blocks, only standard MATLAB functions are supported. Functions from toolboxes have not been tested and may result in issues during code generation or produce incorrect results. For a list of standard functions supported for code generation, see the items listed under the MATLAB category in the "Functions and Objects Supported for C/C++ Code Generation" table.

20

Model Architecture and Design

Fixed Point Simulink PLC Coder Structured Text Code Generation

In this section...
"Block Parameters" on page 20-2
"Model Parameters" on page 20-3
"Limitations" on page 20-4

Block Parameters

At the MATLAB command prompt type `plcdemo_fixed_point`. Once the example model opens , follow these instructions to configure the model for Structured Text code generation.

1. If the block in the subsystem has a **Signal Attributes** tab, navigate to that tab and jump to step 3.

2. If there are no blocks in the subsystem with a **Signal Attributes** tab use the Data Type Conversion block. Add the Data Type Conversionblock to the model and continue to the next step.

3. For the **Integer rounding mode** parameter, select `Round`.

4. Clear the **Saturate on integer overflow** check box.

5. For the **Output data type** parameter, select a fixed-point data type.

6. Click the **Data Type Assistant** button .

7. For the Word length parameter, enter 8, 16, or 32.

8. For the **Mode** parameter, select `Fixed point`.

9. For the **Scaling** parameter, select `Binary point`.

10 Click **OK**.

Model Parameters

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

2 For the **Device vendor** parameter, select `Generic` or `Custom Processor`. If you select `Custom Processor` proceed to step 4.

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.

Limitations

- 64 bit fixed-point data type not supported.

- The data type and value type must match for fixed-point tunable parameters of type `Simulink.Parameter`.

Simulink.Parameter: efi_param   ✕

Value:	`<1x1 embedded.fi>`
Data type:	`fixdt(1,16,15)` `>>`
Dimensions:	`[1 1]` Complexity: `real`
Minimum:	`[ ]` Maximum: `[ ]`

Stored integer values (derived)

Minimum:	`[ ]` Maximum: `[ ]`

Unit:

Code generation options

Storage class:	ExportedGlobal ▼
Identifier:	
Alignment:	`-1`

Description:

```
>> efi_var.Value

ans =

    0.2750

        DataTypeMode: Fixed-point: binary point scaling
         Signedness: Signed
         WordLength: 16
      FractionLength: 15
```

No mismatch between value and Data type fields.

OK   Cancel   Help   Apply

- **Scaling** parameter type `Slope and bias` is not supported for code generation.

You are now ready to:

- "Prepare Model for Structured Text Generation" on page 1-3
- "Verify System Compatibility for Structured Text Code Generation" on page 1-6

- "Generate and Examine Structured Text Code" on page 1-7

Generating Simulink PLC Coder Structured Text Code for Multirate Models

Multirate Model Requirements for PLC Code Generation

At the MATLAB command prompt type in `plcdemo_multirate`. Once the demo model opens up follow the instructions below to configure the model for Structured Text code generation:

Model Configuration Parameters

Before generating Structured Text from a multirate model, you must configure the model as follows:

- **Solver** options that are recommended or required for PLC code generation:

  - **Type**:`Fixed-step`.
  - **Solver**:`Discrete(no continuous states)`. Other fixed-step solvers could be selected, but this option is usually best for simulating discrete systems.
  - **Tasking mode**: Must be explicitly set to `Single Tasking`. Do not set **Tasking mode** to `Auto`

- Change any continuous time input signals in the top level subsystem to use discrete fixed sample times.

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

- In the resulting block dialog box, select **Treat as atomic unit**.
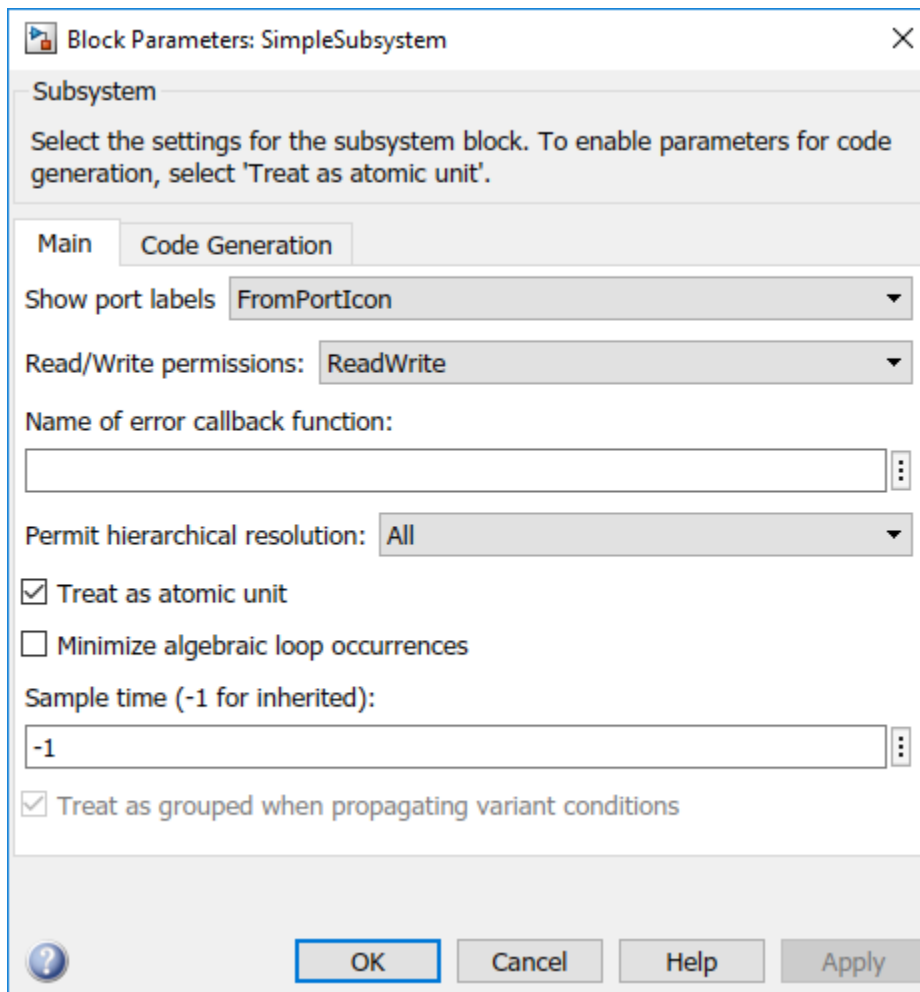
When you deploy code generated from a multirate model, you must run the code at the fundamental sample rate.

Limitations

The B&R Automation Studio does not support structured text code generation from multirate models.

You are now ready to:

- "Verify System Compatibility for Structured Text Code Generation" on page 1-6
- "Generate and Examine Structured Text Code" on page 1-7

MATLAB Function Block Simulink PLC Coder Structured Text Code Generation

In this section...
"Configuring the rand function for PLC Code Generation" on page 20-9
"Simulink Width Block Requirements for PLC Code generation" on page 20-9
"Workspace Parameter Data Type Limitations" on page 20-9
"Limitations" on page 20-9

Configuring the rand function for PLC Code Generation

Simulink PLC Coder generates structured text code for MATLAB Function blocks and Stateflow charts that use the MATLAB `rand` function. You implement the `rand` function by using a pseudo random number generator that works with PLC IDEs supporting the `uint32` data type. The software has conformance checks to report diagnostics for incompatible targets. These targets have been tested for `rand` function support.

- 3S-Smart Software Solutions CODESYS Version 2.3 or 3.3 or 3.5 (SP4 or later)
- B&R Automation Studio 3.0 or 4.0
- Beckhoff TwinCAT 2.11 or 3
- OMRON Sysmac Studio Version 1.04, 1.05, 1.09 or 1.12
- Rexroth IndraWorks version 13V12 IDE
- PLCopen XML

Simulink Width Block Requirements for PLC Code generation

Instead of using the Simulink Width block , inside the MATLAB Functionuse the MATLAB `length` function to compute the input vector width.

Workspace Parameter Data Type Limitations

If the data type of the MATLAB work space parameter value does not match that of the block parameter in your model, the value of the variable in the generated code is set to zero.

If you specify the type of the `Simulink.Parameter` object by using the `DataType` property, use a typed expression when assigning a value to the parameter object. For example, if the `Simulink.Parameter` object `K1` stores a value of the type `single`, use a typed expression such as `single(0.3)` when assigning a value to `K1`.

```
K1 = Simulink.Parameter;
K1.Value = single(0.3);
K1.StorageClass = 'ExportedGlobal';
K1.DataType = 'single';
```

Limitations

When generating structured text from MATLAB Function blocks, these are the limitations :

- Cell arrays in MATLAB Function blocks are not supported.
- If you want to use a function from a toolbox within the MATLAB Function block, you must check the toolbox function page to see if that block supports code generation from Simulink PLC Coder.
- When generating a testbench for models that use the `rand` function , different `rand` output values may be generated when gathering test vectors vs code generation, leading to testbench verification failures . To prevent these failures make sure that the `rand` output value remains constant across different model compilations.

PLC Coder Code Deployment

Deploy Structured Text

In this section...
"Learning Objectives" on page 21-2
"Prerequisites" on page 21-2
"Workflow" on page 21-2
"Importing Generated Structured Text Code Manually" on page 21-2

Using Simulink PLC Coder, you can generate structured text and test bench code, and then import the generated code into the target IDE.

Learning Objectives

In this tutorial you learn how to:

- Open the `plcdemo_simple_subsystem` model and prepare the model for code generation.
- Verify the code that you generated.
- Automatically or manually import your generated code into your target IDE.

Prerequisites

- Simulink PLC Coder
- Target IDE folder location (for automatic import).

Workflow

1 Open the `plcdemo_simple_subsystem` model.
2 Open the model settings and set **Solver Selection** to `Fixed-step` and **Solver** to `discrete(no continuous states)`.
3 If your target IDE is in the "PLC IDEs for Importing Code Automatically" on page 1-14 , see "Generate and Automatically Import Structured Text Code" on page 1-14. Otherwise , see "Importing Generated Structured Text Code Manually" on page 21-2

Importing Generated Structured Text Code Manually

If your target IDE does not automatically import generated code:

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

   The Configuration Parameters dialog box is displayed.
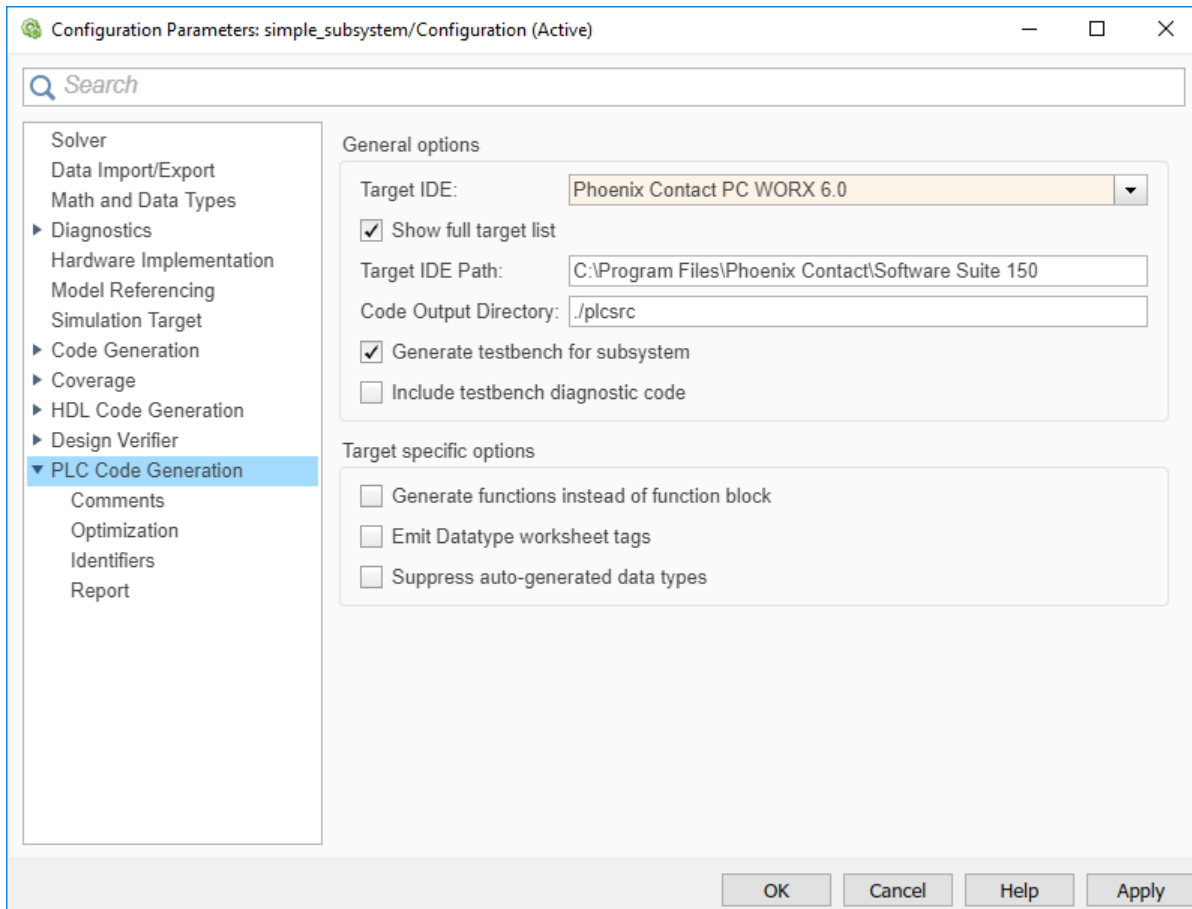
2 On the **PLC Code Generation** pane, select an option from the **Target IDE** list, for example, 3S CoDeSys 2.3.

The default **Target IDE** list displays the full set of supported IDEs. To see a reduced subset of the target IDEs supported by Simulink PLC Coder, disable the option **Show full target list**. To customize this list, use the plccoderpref function.

3 Click **Apply**.

4 Click **Generate code**.

This button:

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

When code generation is complete, a **View diagnostics** hyperlink appears at the bottom of the model window. Click this hyperlink to open the Diagnostic Viewer window.
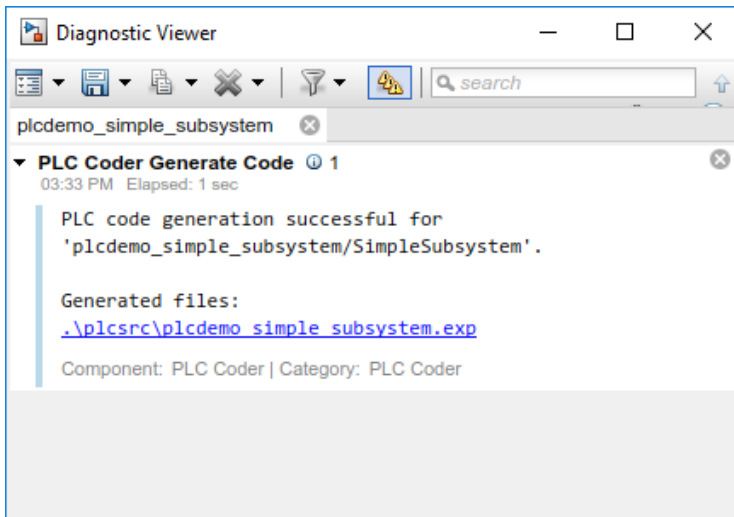
> This window has links that you can click to open the associated files. For more information, see "Files Generated by Simulink PLC Coder" on page 1-11.

5 To import generated code into your target IDE import the generated files manually into your target IDE.

Deploy Ladder Diagram

In this section...
"Learning Objectives" on page 21-5
"Prerequisites" on page 21-5
"Workflow" on page 21-5
"Importing Generated Ladder Diagram Code Manually" on page 21-5

Using Simulink PLC Coder you can generate Structured Text, along with test bench code and import the generated code into the target IDE.

Learning Objectives

In this tutorial you will learn how to:

- Open the `plcdemo_ladder_timers` model and prepare the model for code generation.
- Verify the code you generated.
- Have your generated code either automatically or manually imported into your target IDE.

Prerequisites

- Simulink PLC Coder
- You have access to either Rockwell Automation RSLogix 5000 or Studio 5000 IDE.

Workflow

1. Open the `plcdemo_ladder_timers` model.
2. Open the model settings and set **Solver Selection** to `Fixed-step` and **Solver** to `discrete(no continuous states)`.
3. See "Importing Generated Ladder Diagram Code Manually" on page 21-5

You can manually import the generated L5X file into RSLogix 5000 or Studio 5000 IDEs.

Importing Generated Ladder Diagram Code Manually

For L5X import file generation:

1. Right-click the `Motor Controller` block and select **PLC Code > Options**.

   This displays the **PLC Code Generation** configuration parameters window:
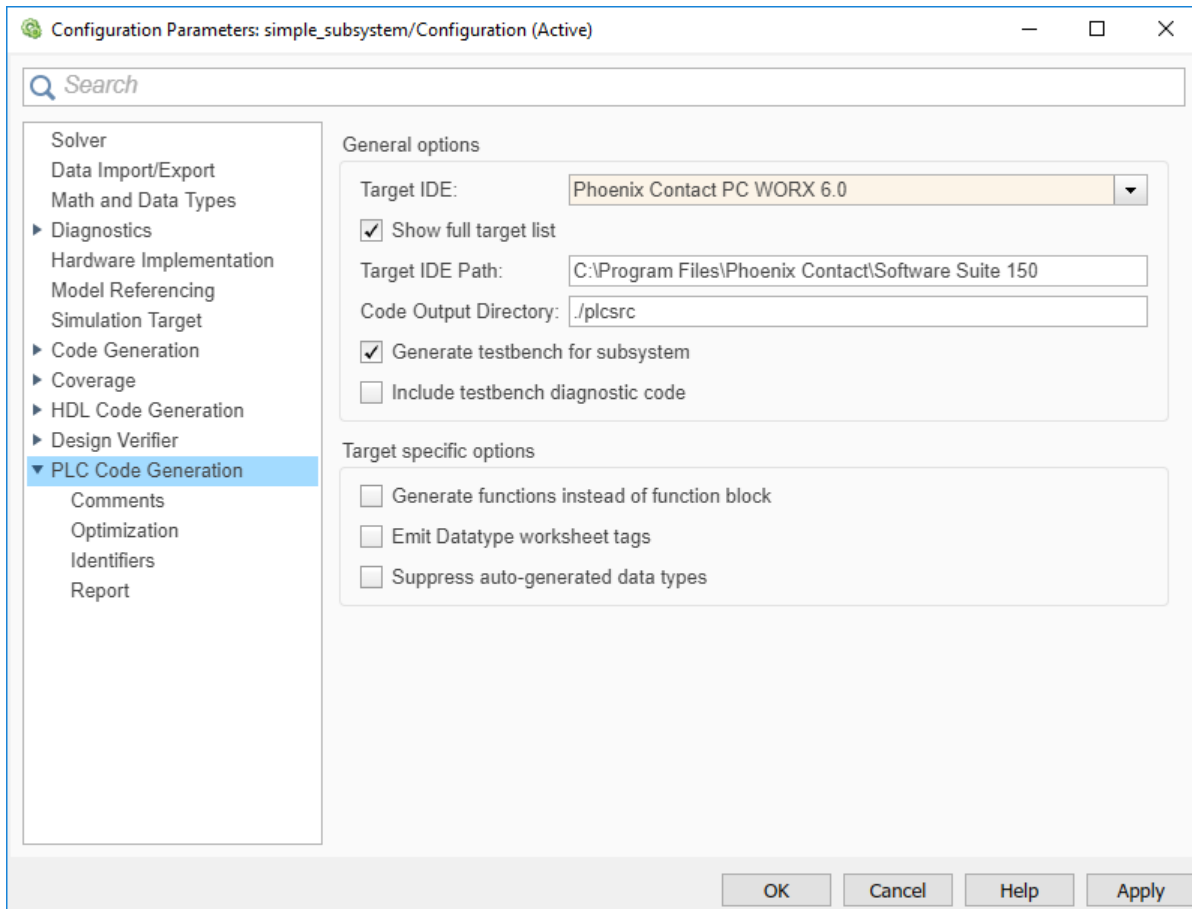
2   On the **PLC Code Generation** pane, from the **Target IDE** list, select either `Rockwell Studio 5000:AOI` or `Rockwell RSLogix5000:AOI`.

3   In **Target IDE Path**, enter the path to the folder where you want the generated L5X file to be saved. In, **Code Output Directory**, enter the name of the folder to save the generated L5X file.

4   Click **Apply**.

5   Right-click the `Motor Controller` block and select **PLC CodeGenerate Code for Subsystem** .

6   Upon, completion of code generation the Diagnostic window displays a message with the path to the generated L5X file.
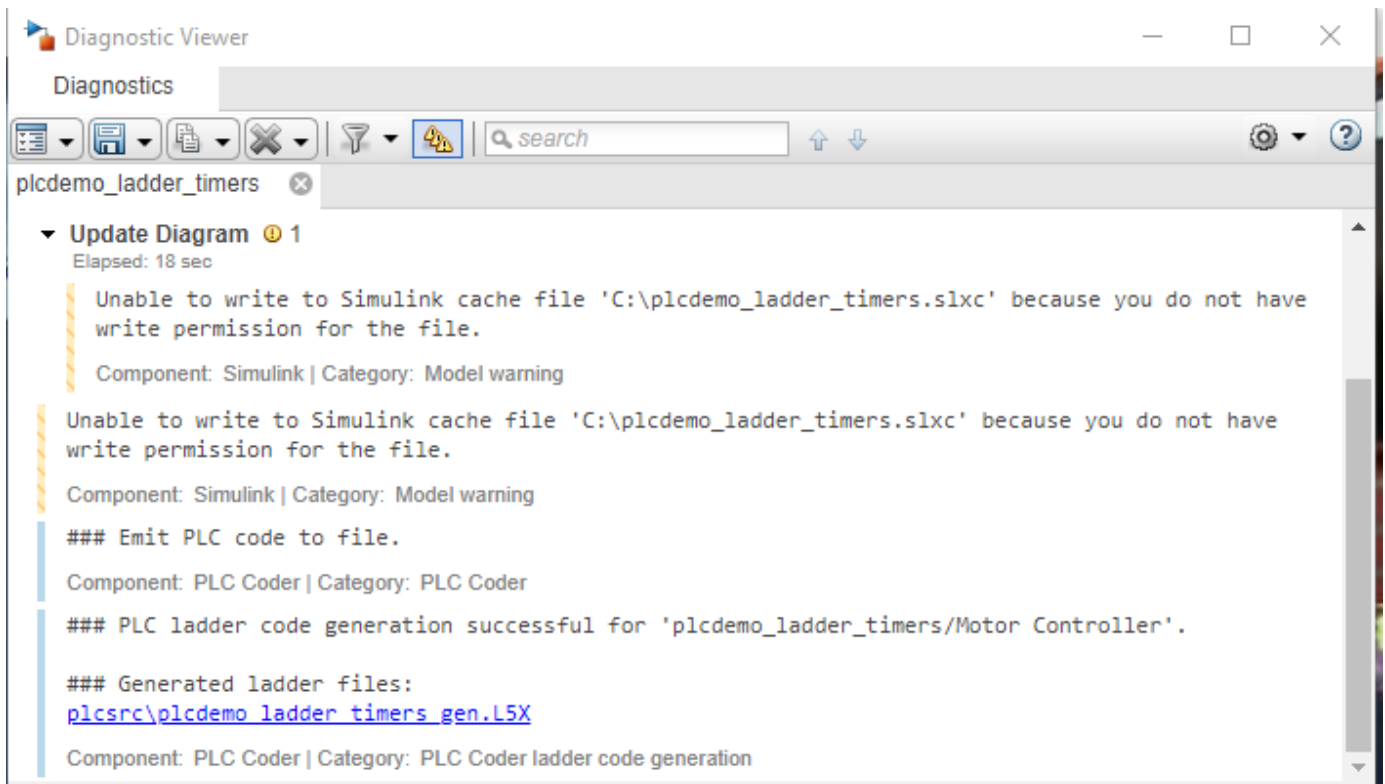
Simulink PLC Coder Structured Text Code Generation For Simulink Data Dictionary (SLDD)

Generate Structured Text Code For Simulink Data Dictionary Defined Model Parameters

In this section...
"Learning Objectives" on page 22-2
"Requirements" on page 22-2
"Workflow" on page 22-2

Simulink Data Dictionary (SLDD) is the preferred Model-Based-Design (MBD) data modeling and management tool. SLDD provides advantages such as data separation, logical partitioning, traceability, and so on. To achieve traceability between your generated code and model, and for code reusability and model and data sharing, use SLDD

Learning Objectives

In this tutorial, you learn how to:

- Open the `plcdemo_tunable_params` model and migrate the model to use Simulink Data Dictionary (SLDD).
- Generate code for the model.

Requirements

- Base workspace variable definition must match the variable definition in the SLDD file. If there is a mismatch, Simulink PLC Coder displays an error during the code generation process.
- If your model has a Data Store Memory(DSM) object, you must have a matching `Simulink.Signal` object in the SLDD file.

Workflow

Migrate the `plcdemo_tunable_params` model base workspace variables to an SLDD file for code generation:

Note Copy the `plcdemo_tunable_params` model to your current working directory prior to starting the workflow.

---

1 Open the `plcdemo_tunable_params` model .
2 From the Simulink Editor **Modeling** tab, click **Model Explorer**.
3 Under the **Model Hierarchy** pane, click **Base Workspace** . The **Contents** pane displays the base workspace variables.
4 Right-click K1, K2, and K3. Choose the `Convert to parameter object` option to convert them to the `Simulink.Parameter` type.
5 Right-click `plcdemo_tunable_params`, and then select `Properties`.
6 Select the **External Data** tab.

7   Click **New**. Enter the file name as `plcdemo_tunable_params`.

8   Click the **Migrate data** button. Then click **Apply** in response to the `Link Model to Data Dictionary`message and **Migrate** in response to the `Migrate Data` message.

9   Click **OK**.

10

To open the dictionary, in the Simulink Editor, click the model data badge   in the bottom left corner, then click the **External Data** link. To inspect the contents of the dictionary, in the Model Explorer **Model Hierarchy** pane, under the **External Data** node, expand the dictionary node.

To generate code for the model, see "Generate and Examine Structured Text Code" on page 1-7 .

See Also

More About

•   `Simulink.Parameter`

•   `Simulink.Signal`

•   Data Store Memory

•   "Migrate Models to Use Simulink Data Dictionary"

Simulink PLC Coder Structured Text Code Generation For Enumerated Data Type

Structured Text Code Generation for Enum To Integer Conversion

Autogenerate structured text code for enum to integer conversion model.

Load enum class

For this example, the `myEnum.m` script loads the enum class definition. Place this script file in the same project folder as the `plc_enum_to_int` model file.

Open the model

open_system('plc_enum_to_int.slx')

This model shows PLC code generation for enum to integer type conversion.
To generate PLC code, open PLC Coder App. Select Settings->PLC Code Generation->General options->Target IDE and choose Target IDE that supports enum type. Select Settings->PLC Code Generation->Identifiers->Generate enum cast function. Click the Subsystem block and click the Generate PLC Code button.

The Diagnostic Viewer displays hyperlinks to the generated code files, click the links to view the generated files.

Copyright 2009-2020 The MathWorks, Inc.

See Also

More About

- "Use Enumerated Data in Simulink Models"
- "Code Generation for Enumerations"

Distributed Code Generation with Simulink PLC Coder

Distributed Model Code Generation Options

Distributed models allow you to model complex systems as individual components and simulate the components at different sample times or cycle rates. The Simulink PLC Coder distributed model code generation options allow you to generate structured text code for individual components of the model and integrate the generated code externally. Use this table to decide the code generation option to use based on your distributed model design and requirements.

Goal	Option
Generate code for individual model subsystems and integrate the generated code externally.	"Keep Top-Level ssmethod Name the Same as the Non-Top Level Name" on page 13-36
Prevent initialization of externally defined variables.	"Remove Initialization Statements for Externally Defined State Variables" on page 13-38

The distributed code generation options are model-specific, and when selected at the top level, are enabled for all the model subsystems. Once you enable the option, it stays on. When generating code for individual subsystems, you might see unintended behavior in the generated code due to the option remaining on.

See Also

More About

- "Generated Code Structure for PLC_RemoveSSStep" on page 24-3
- "Generated Code Structure for PLC_PreventExternalVarInitialization" on page 24-5

Generated Code Structure for PLC_RemoveSSStep

The example shows you how to enable the `PLC_RemoveSSStep` option for your model, generate code and display the comparison between code generated with the `PLC_RemoveSSStep` option enabled and then disabled.

1 Open the `UsePLC_RemoveSSStepforDistributedCode GenerationExample` example:

```
openExample('plccoder/UseRemoveSSStepForDistributedCodeGenerationExample')
```

.

2 Copy all the model files to a folder of your choice.

3 Open the `mSystemIntegration` model.

4 Open the Simulink PLC Coder app, and then select the `Subsystem1` block. .

5 Click **Settings**. Navigate to **PLC Code Generation > Identifiers**. Select the **Keep top level ssMethod name same as non-top level** check box.

6 Click **OK**.

7 Click **Generate PLC Code**.

8 Select the `Subsystem1` block.

9 Click **Settings**. Navigate to **PLC Code Generation > Identifiers**. Clear the **Keep top level ssMethod name same as non-top level** check box.

10 Click **Generate PLC Code**.

This image shows a comparison between the code generated with `PLC_RemoveSSStep` enabled, and then disabled. Removing `SS_STEP` enables easier external code integration of the different subsystems because they all the same `ssMethodType`.

See Also

"Keep Top-Level ssmethod Name the Same as the Non-Top Level Name" on page 13-36

More About

- "Distributed Model Code Generation Options" on page 24-2

Generated Code Structure for PLC_PreventExternalVarInitialization

The example shows you how to enable the PLC_PreventExternalVarInitialization option for your model, generate code and display the comparison between code generated with the PLC_PreventExternalVarInitialization option disabled and then enabled.

1   Open the PLC_PreventExternalVarInitializationExample example:

    openExample('plccoder/PreventExternalVarInitializationExample')

    .

2   Copy all the model files to a folder of your choice.

3   Open the External_Var_Distributed_Codegen model.

4   Open the Simulink PLC Coder app, and select the Subsystem block.

5   Click **Settings**. Navigate to **PLC Code Generation > Interface**. Clear the **Remove initialization statements for externally defined state variables** check box.

6   Click **OK**.

7   Click **Generate PLC Code**.

8   Select the Subsystem block.

9   Click **Settings**. Navigate to **PLC Code Generation > Interface**. Set the **Remove initialization statements for externally defined state variables** check box.

10 Click **Generate PLC Code**.

This image shows a comparison between the code generated with PLC_PreventExternalVarInitialization disabled, and then enabled. Removing initialization statements for externally defined variables prevents the corruption of their data values.

See Also

"Remove Initialization Statements for Externally Defined State Variables" on page 13-38

More About

- "Distributed Model Code Generation Options" on page 24-2

PLC_RemoveSSStep for Distributed Code Generation

Generate structured text code for different components of your model.

Open model

Open the model by using the following command:

```
open_system('mSystemIntegration');
```

Note: Before code generation, copy mSubSystem1, mSubSystem2, mSubSystem3, and mSystemIntegration SLX files to the same folder location in your current working directory (CWD).

Configure Model Components for Distributed Code Generation

To autogenerate structured text code with the same `ssMethod` type for every component of your model for external code integration later on, use Keep Top-Level ssMethod Name the Same as the Non-Top Level Name. For more information, see "Keep Top-Level ssmethod Name the Same as the Non-Top Level Name" on page 13-36 function.

Mark Externally Defined Variables

1   Open the Simulink PLC Coder app. For more information, see Simulink PLC Coder.
2   Select the `TopSystem` block.
3   Click **Settings**. Navigate to **PLC Code Generation** > **Identifiers**. In the `Identifier Names` box enter `Subsystem1,Subsystem2,Subsystem3`.
4   Click **OK**.

Code Generation

1   Open the Simulink PLC Coder app. For more information, see Simulink PLC Coder.
2   Select the `Subsystem1` block.
3   Click **Settings**. Navigate to **PLC Code Generation** > **Identifiers**. Select the **Keep top level ssMethod name same as non-top level** check box.
4   Click **OK**.
5   Repeat steps 2 through 4 for `SubSystem2`, `SubSystem3`, and `TopSystem`.

Generate Code for the Subsystem

To generate code for the individual subsystem use the plcgenerate code function:

plcgeneratecode('mSystemIntegration/TopSystem/SubSystem1');

plcgeneratecode('mSystemIntegration/TopSystem/SubSystem2');

plcgeneratecode('mSystemIntegration/TopSystem/SubSystem3');

Generate Code for the Integrated Model

To generate code for the integrated model:

plcgeneratecode('mSystemIntegration/TopSystem');

Structured Text Code Generation for Subsystem Reference Blocks

This example shows how to autogenerate structured text code for subsystem reference blocks.

Open Simulink Model

To open the Simulink test bench model, use the following command.

```
open_system('mSubSysRefSystemIntegration');
```


Note: Before code generation, copy SSRefSubSystem1, SSRefSubSystem2, SSRefSubSystem3, refSubSystem1, refSubsystem2, refSubSystem3 and mSubSysRefSystemIntegration SLX files to the same folder location in your current working directory (CWD).

Generate Code for the Subsystem

To generate code for the subsystem use `plcgeneratecode`

```
generatedfiles = plcgeneratecode('mSubSysRefSystemIntegration/TopSystem');

### Generating PLC code for 'mSubSysRefSystemIntegration/TopSystem'.
### Using model settings from 'mSubSysRefSystemIntegration' for PLC code generation parameters.
### Begin code generation for IDE codesys23.
### Emit PLC code to file.
### Creating PLC code generation report mSubSysRefSystemIntegration_codegen_rpt.html.
### PLC code generation successful for 'mSubSysRefSystemIntegration/TopSystem'.
```

```
### Generated files:
plcsrc\mSubSysRefSystemIntegration.exp
```

See Also

More About

- "Subsystem Reference"

Examples Book

Generate Structured Text Code for a Simple Simulink Subsystem

This example shows how to select the target IDE for a Simulink® model, generate code, and view generated files.

1. Open model `plcdemo_simple_subsystem` and save a copy to a writable location.
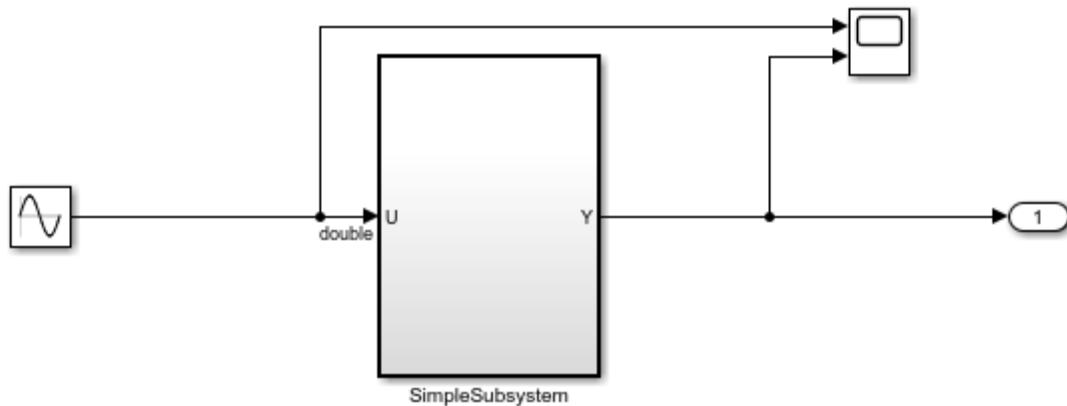
This example shows how to generate code for a simple subsystem block.
To generate structured text code for the subsystem, select the SimpleSubsystem block and right-click **PLC Code > Generate Code for Subsystem.**

Copyright 2009-2019 The MathWorks, Inc.

2. Open the **Simulink PLC Coder** app.

3. Open the PLC Code Generation dialog box.. In the Target IDE select **3S CoDeSys 2.3**.
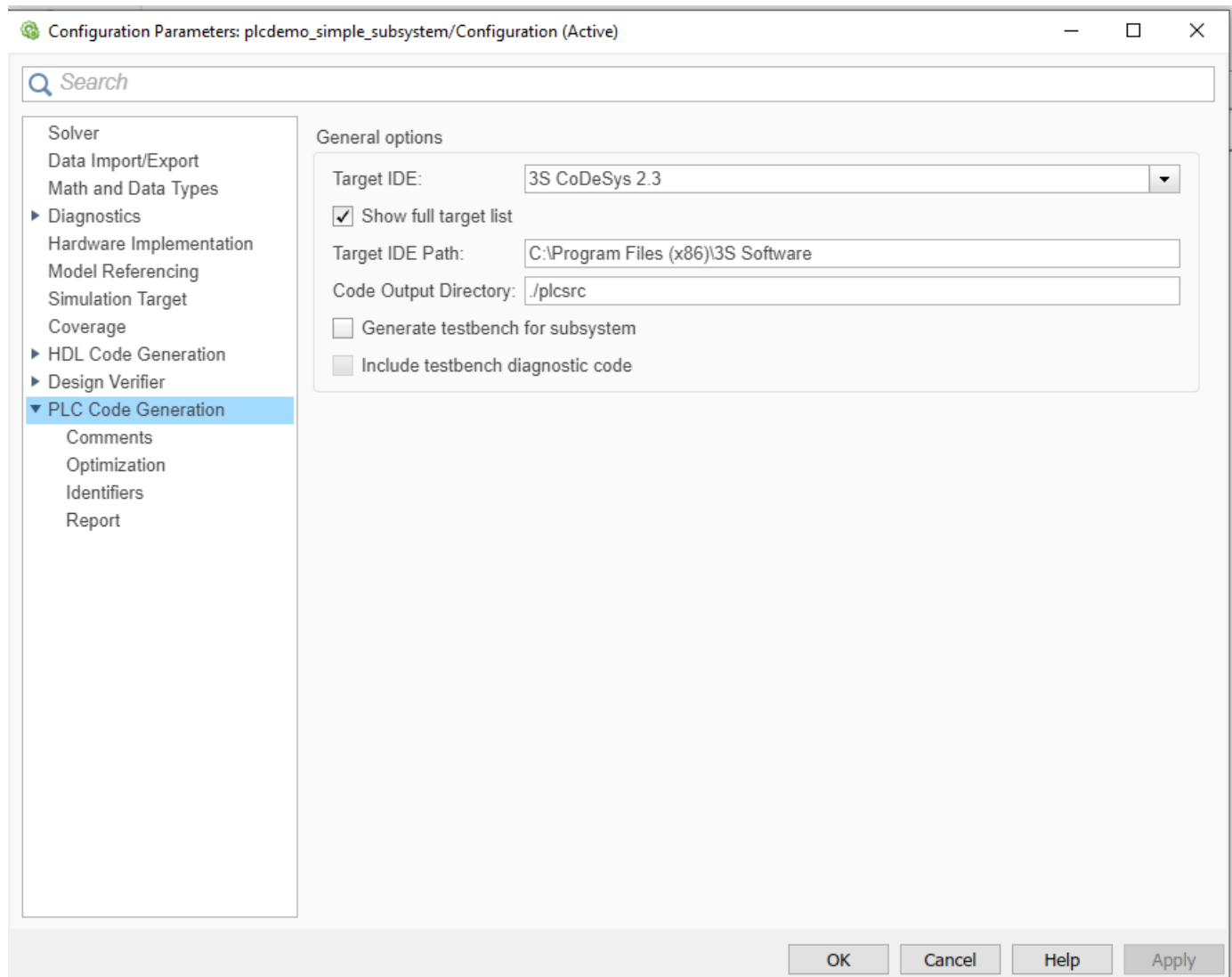
Click **OK**

4. Select the `SimpleSubsystem` block and click **Generate PLC Code**. Alternatively, from the command line, enter:

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

5. View the code generation report.

The report includes links to the generated code file `plcdemo_simple_subsystem.exp` and associated traceability and code metrics reports.
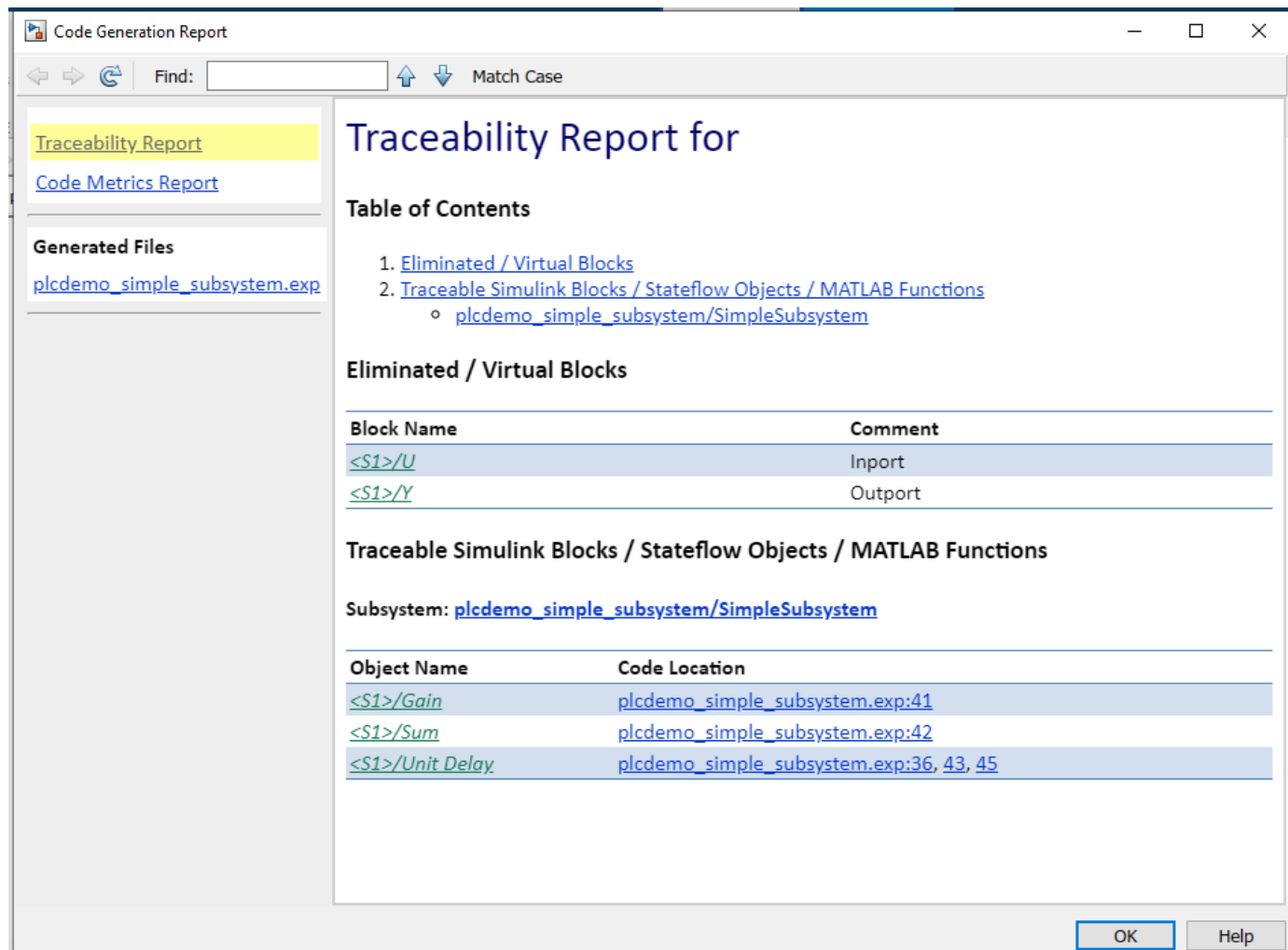
6. This figure contains the generated code `plcdemo_simple_subsystem.exp`.

File: **plcdemo_simple_subsystem.exp**

```
1   (*
2    *
3    * File: plcdemo_simple_subsystem.exp
4    *
5    * IEC 61131-3 Structured Text (ST) code generated for subsystem "plcdemo_simple_subsystem/SimpleSubsystem"
6    *
7    * Model name                  : plcdemo_simple_subsystem
8    * Model version               : 1.62
9    * Model creator               : The MathWorks, Inc.
10   * Model last modified by      : The MathWorks, Inc.
11   * Model last modified on      : Thu Dec 12 12:37:48 2019
12   * Model sample time           : 0.1s
13   * Subsystem name              : plcdemo_simple_subsystem/SimpleSubsystem
14   * Subsystem sample time       : 0.1s
15   * Simulink PLC Coder version  : 3.2 (R2020b) 27-Feb-2020
16   * ST code generated on        : Thu Mar 12 13:39:57 2020
17   *
18   * Target IDE selection        : 3S CoDeSys 2.3
19   * Test Bench included         : No
20   *
21   *)
22   FUNCTION_BLOCK SimpleSubsystem
23   VAR_INPUT
24       ssMethodType: SINT;
25       U: LREAL;
26   END_VAR
27   VAR_OUTPUT
28       Y: LREAL;
29   END_VAR
30   VAR
31       UnitDelay_DSTATE: LREAL;
32   END_VAR
33   CASE ssMethodType OF
34       SS_INITIALIZE:
35           (* SystemInitialize for Atomic SubSystem: '<Root>/SimpleSubsystem' *)
36           (* InitializeConditions for UnitDelay: '<S1>/Unit Delay' *)
37           UnitDelay_DSTATE := 0.0;
38           (* End of SystemInitialize for SubSystem: '<Root>/SimpleSubsystem' *)
39       SS_STEP:
40           (* Outputs for Atomic SubSystem: '<Root>/SimpleSubsystem' *)
41           (* Gain: '<S1>/Gain' incorporates:
42            *  Sum: '<S1>/Sum'
43            *  UnitDelay: '<S1>/Unit Delay' *)
44           Y := (U - UnitDelay_DSTATE) * 0.5;
45           (* Update for UnitDelay: '<S1>/Unit Delay' *)
46           UnitDelay_DSTATE := Y;
47           (* End of Outputs for SubSystem: '<Root>/SimpleSubsystem' *)
48   END_CASE;
49   END_FUNCTION_BLOCK
50   VAR_GLOBAL CONSTANT
51       SS_INITIALIZE: SINT := 0;
52       SS_STEP: SINT := 1;
53   END_VAR
54
```

Related Topics

- "Generate and Examine Structured Text Code" on page 1-7

Generating Structured Text for a Hierarchical Simulink Subsystem with Virtual Subsystems

This introductory model shows the code generated for a hierarchical subsystem consisting of other Simulink® subsystems.

HierarchicalSubsystem
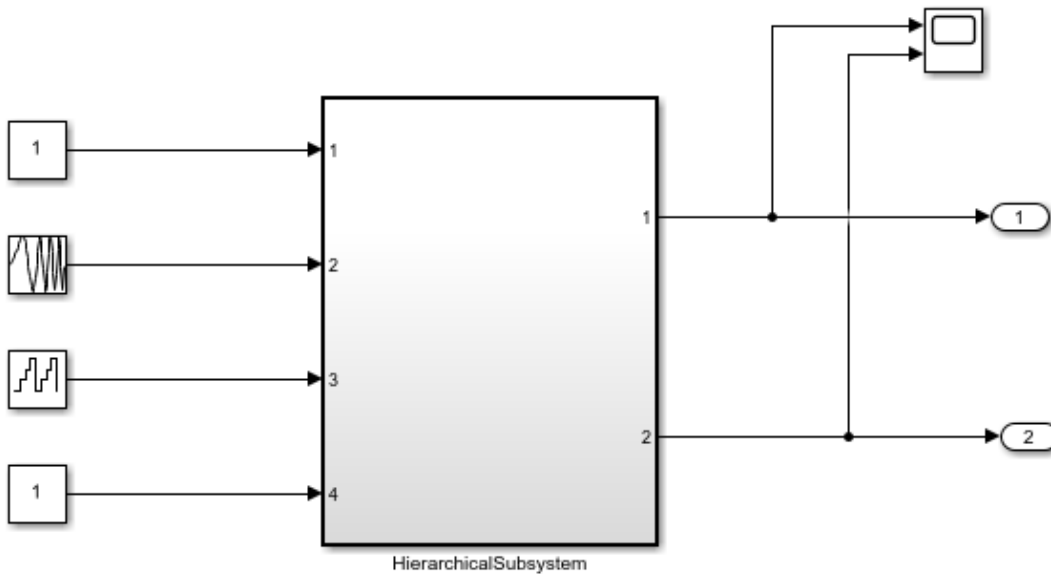
This introductory model shows the code generated for a hierarchical subsystem consisting of two other subsystems named "S1" and "S2". These subsystems are not marked atomic and hence they do not generate separate Function Blocks. The code for these subsystems gets inlined into the Function Block for the parent subsystem "HierarchicalSubsystem".

To build the subsystem, right-click on the subsystem block and select PLC Code > Generate Code for Subsystem.

The Diagnostic Viewer displays hyperlinks to the generated code files, click the links to view the generated files.

This model contains a hierarchical subsystem containing other virtual subsystems.
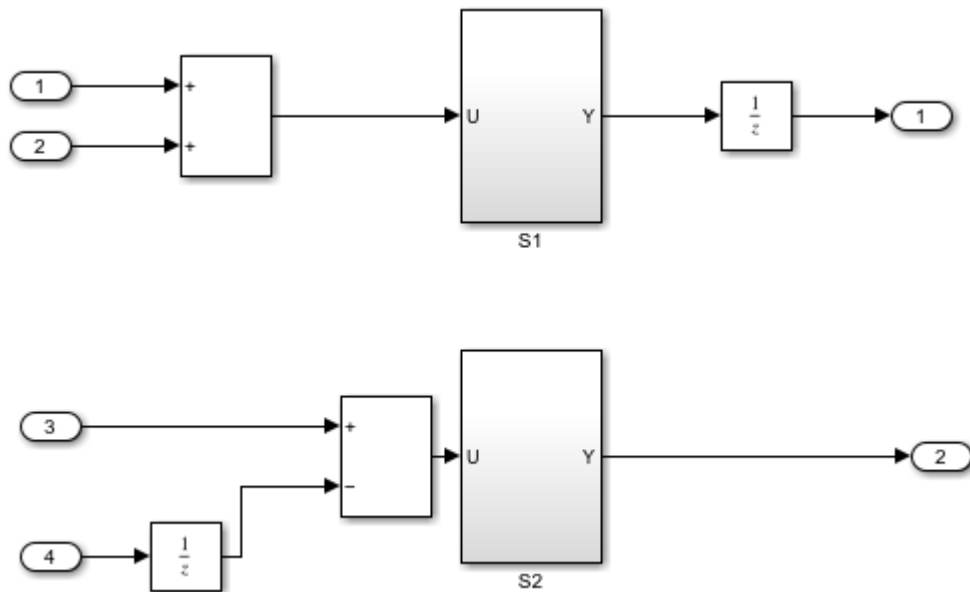
You can generate PLC Structured Text code for this subsystem by right-clicking on the subsystem block and select PLC Code -> Generate Code for Subsystem Alternatively, you can use the following command

```
generatedFiles = plcgeneratecode('plcdemo_hierarchical_virtual_subsystem/HierarchicalSubsystem')
```

After the code generation, the Diagnostic Viewer window is displayed with hyperlinks to the generated code files. You can open the generated files by clicking on the links.

Generating Structured Text for a Hierarchical Simulink Subsystem

This introductory model shows the code generated for a hierarchical subsystem consisting of other Simulink® subsystems.

HierarchicalSubsystem
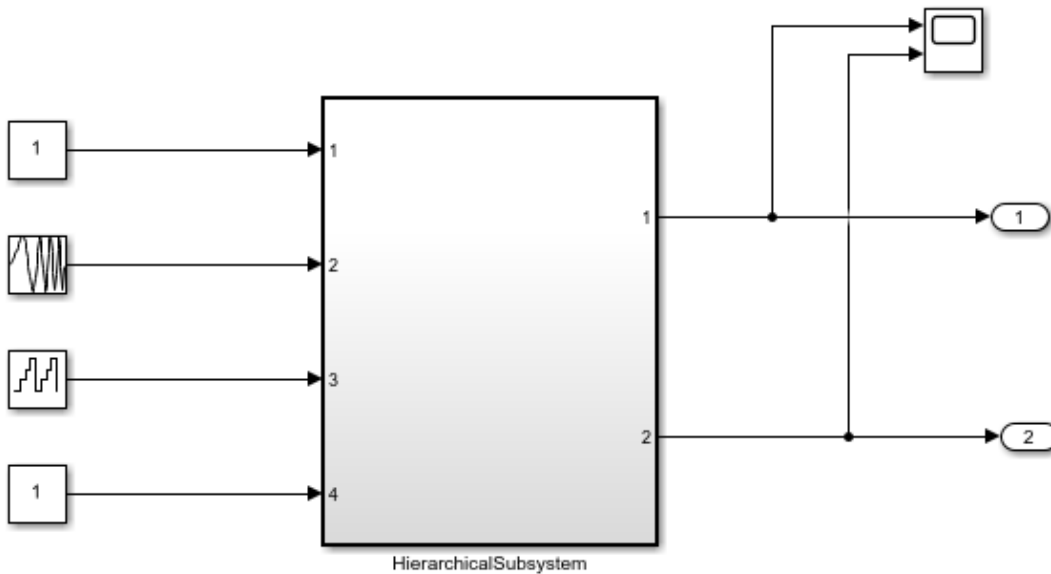
This introductory model shows the code generated for a hierarchical subsystem consisting of
two other subsystems named "S1" and "S2". Each of these subsystems generate a separate Function Block.
To build the subsystem, right-click on the subsystem block and
select PLC Code > Generate Code for Subsystem.

The Diagnostic Viewer displays hyperlinks to the generated code files, click the links to view the generated files.

This model contains a hierarchical subsystem containing other subsystems.
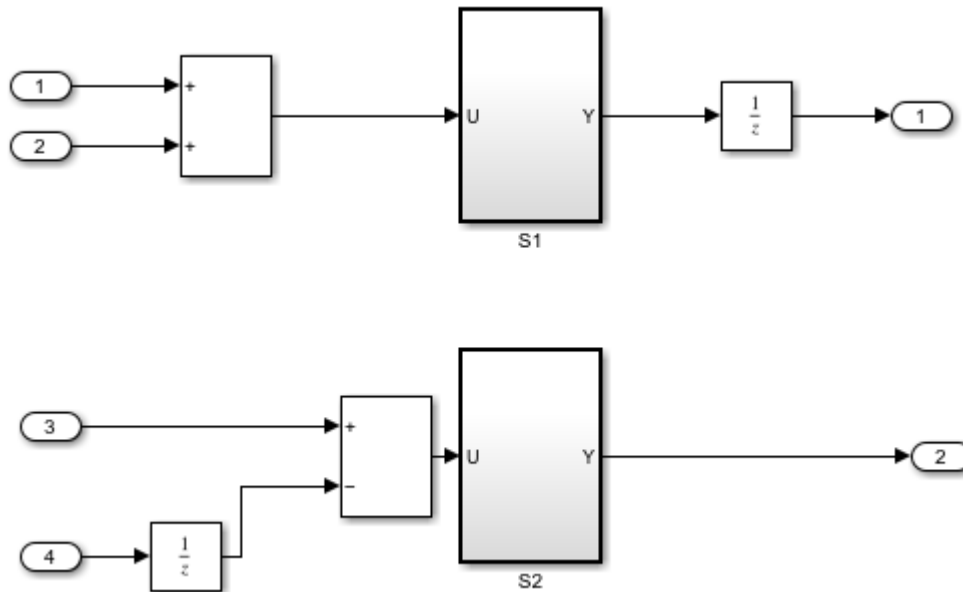
You can generate PLC Structured Text code for this subsystem by right-clicking on the subsystem block and select PLC Code -> Generate Code for Subsystem Alternatively, you can use the following command

```
generatedFiles = plcgeneratecode('plcdemo_hierarchical_subsystem/HierarchicalSubsystem');
```

After the code generation, the Diagnostic Viewer window is displayed with hyperlinks to the generated code files. You can open the generated files by clicking on the links.

Generate Structured Text Code for Reusable Subsystems

This example shows how to simulate and generate structured text code for an atomic reusable subsystem. Reusable subsystems enable you to model commonly used blocks as a subsystem and reuse the subsystems multiple times within a model. Use reusable subsystems in your model to :

- Facilitate team-based development.
- Optimize the time, to make model changes by making changes to one subsystem and copying the subsystem to all other locations in the model where the subsystem is used.
- Reduce complexity of the generated code by generating a single shared function block.

To generate code for the reusable subsystems, mark the child subsystems as atomic by right-clicking the subsystem blocks, selecting **Block Parameters (Subsystem),** and then selecting **Treat as atomic unit** under the **Main** tab.

Model Description

The model consists of a top-level atomic subsystem called `ReusableSubsystem`. Under the top-level subsystem there are two identical atomic subsystems `S1` and `S2`.
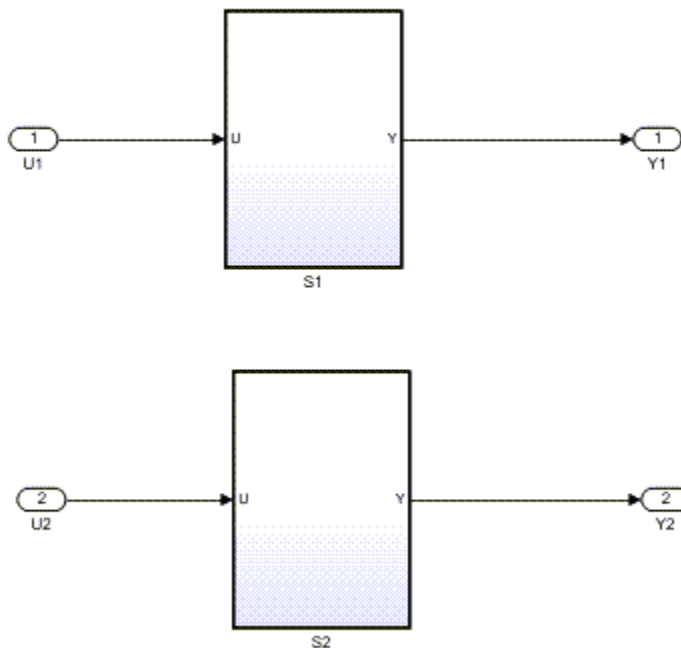
Open the model:

```
load_system('plcdemo_reusable_subsystem');
open_system('plcdemo_reusable_subsystem/ReusableSubsystem');
```

Generate Code

To generate structured text code, do one of the following:

- Open the PLC Coder app. Select the `ReusableSubsystem` block and click `Generate PLC Code`.
- Use the `plcgeneratecode` function:

```
plcgeneratecode('plcdemo_reusable_subsystem/ReusableSubsystem');

### Generating PLC code for 'plcdemo_reusable_subsystem/ReusableSubsystem'.
### Using model settings from 'plcdemo_reusable_subsystem' for PLC code generation parameters.
### Begin code generation for IDE codesys23.
### Emit PLC code to file.
### Creating PLC code generation report plcdemo_reusable_subsystem_codegen_rpt.html.
### PLC code generation successful for 'plcdemo_reusable_subsystem/ReusableSubsystem'.
### Generated files:
plcsrc\plcdemo_reusable_subsystem.exp
```

Structure of Generated Code

The generated structured text code consists of a single shared function block S1 as a result of code reuse optimizations. The generated structured text code for the reusable subsystem consists of these components:

- Subsystem instance variables: The subsystems S1 and S2 are declared as instances of S1. The code invokes these two instances separately by passing in different inputs to the function block.
- Function calls: This part of the code contains calls to the subsystem instance variable declarations with the different inputs required for each instance of S1.
- Reusable function block definition: This part of the code contains code for the components inside the S1 subsystem. This function block definition is called by the code in the function calls section.

This image shows the different components of the generated structured text code.

```
        i0_S1: S1;
        i1_S1: S1;              Subsystem instance variables
    END_VAR
    CASE ssMethodType OF
        SS_INITIALIZE:
            (* Outport: '<Root>/Y1' *)
            i0_S1(ssMethodType := SS_INITIALIZE, U := U1);
            (* Outport: '<Root>/Y2' *)
            i1_S1(ssMethodType := SS_INITIALIZE, U := U2);
        SS_STEP:
            (* Outport: '<Root>/Y1' *)
            i0_S1(ssMethodType := SS_OUTPUT, U := U1);
            Y1 := i0_S1.Y;                               Function
            (* Outport: '<Root>/Y2' *)                     calls
            i1_S1(ssMethodType := SS_OUTPUT, U := U2);
            Y2 := i1_S1.Y;
    END_CASE;
    END_FUNCTION_BLOCK
    FUNCTION_BLOCK S1
    VAR_INPUT
        ssMethodType: SINT;
        U: LREAL;
    END_VAR                                Reusable
    VAR_OUTPUT                        FUNCTION_BLOCK
        Y: LREAL;                          definition
    END_VAR
    VAR
        UnitDelay_DSTATE: LREAL;
    END_VAR
    CASE ssMethodType OF
        SS_INITIALIZE:
            (* InitializeConditions for UnitDelay: '<S2>/Unit Delay' *)
            UnitDelay_DSTATE := 0.0;
        SS_OUTPUT:
            (* Gain: '<S2>/Gain' incorporates:
             *   Sum: '<S2>/Sum'
             *   UnitDelay: '<S2>/Unit Delay' *)
            Y := (U - UnitDelay_DSTATE) * 0.5;
            (* Update for UnitDelay: '<S2>/Unit Delay' *)
            UnitDelay_DSTATE := Y;
    END_CASE;
    END_FUNCTION_BLOCK
    VAR_GLOBAL CONSTANT
        SS_OUTPUT: SINT := 3;
        SS_INITIALIZE: SINT := 0;
        SS_STEP: SINT := 1;
    END_VAR
```

Close the Model

To close the model:

```
close_system('plcdemo_reusable_subsystem');
```

Generate Structured Text Code for a Simulink Subsystem that Has Multirate Components

This example shows how to model, simulate, and generate code for a Simulink® subsystem that has components running at different sample times (multirate). Multirate systems are common in industrial applications and programable logic controllers(PLCs). For example, within a PLC there is a task to collect information from sensors to control a process that runs at a slower rate and the actual logic to control the process that runs at a faster rate. To setup your multirate model for simulation and code generation, see "Generating Simulink PLC Coder Structured Text Code for Multirate Models" on page 20-7.

Model Description

The model has a simple subsystem with components running at different sample rates. The input sine wave to U1 runs at a 0.1 second sample rate and the input sine wave to U2 runs at a 0.2 second sample rate. Output Y1 is calculated using U1, output Y2 is calculated using U2, and output Y3 is calculated using both U1 and U2. This image shows the components of the multirate subsystem. The blocks in red run at a 0.1 second sample rate and the blocks in green run at a 0.2 second sample rate. When using unit delay blocks in multirate models set the unit delay block Sample time parameter to -1.
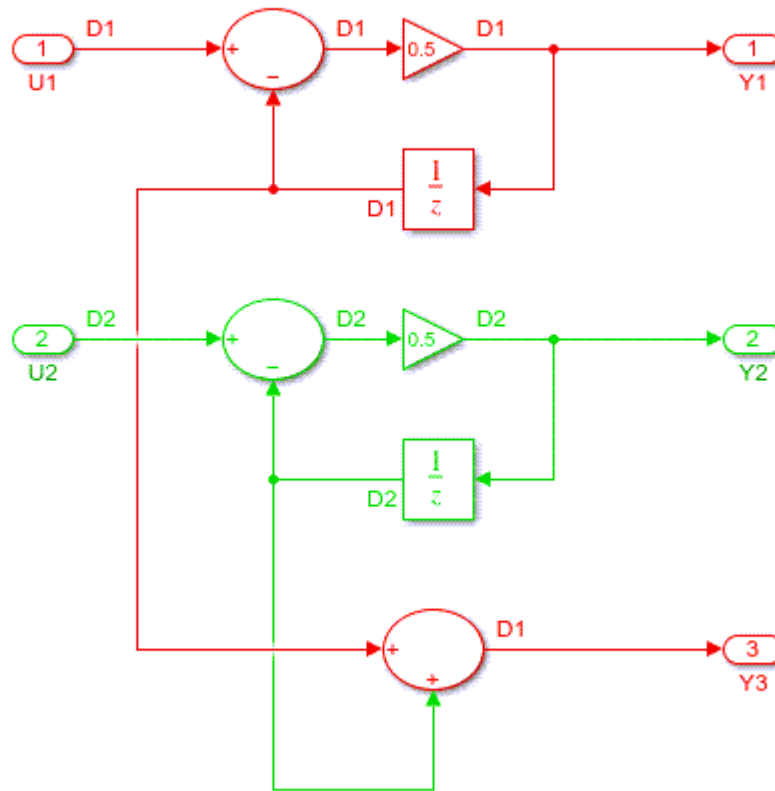
Open the model:

```
load_system('plcdemo_multirate');
open_system('plcdemo_multirate/SimpleSubsystem');
```

Generate Code

To generate structured text code, do one of the following:

- Open the PLC Coder app. Select the `SimpleSubsystem` block and click `Generate PLC Code`.
- Use the `plcgeneratecode` function:

```
plcgeneratecode('plcdemo_multirate/SimpleSubsystem');
```

```
### Generating PLC code for 'plcdemo_multirate/SimpleSubsystem'.
### Using model settings from 'plcdemo_multirate' for PLC code generation parameters.
### Begin code generation for IDE codesys23.
### Emit PLC code to file.
### Creating PLC code generation report plcdemo_multirate_codegen_rpt.html.
### PLC code generation successful for 'plcdemo_multirate/SimpleSubsystem'.
```

```
### Generated files:
plcsrc\plcdemo_multirate.exp
```

Structure of Generated Code

The generated structured text code for the multirate subsystem consists of these components:

- Time step counter variable: This portion of the code contains variable declarations for the time step counter that implements the different sample times. For subsystems that have n different sample times, the generated code has $n-1$ time step counter variables that correspond to the $n-1$ slower sample rates.

- Logic to implement time step counter: This portion of the code contains the code that implements the counter to implement the different sample times.

- Code blocks running at different sample rates: These are portions of the code that run at different sample rates. For example, in this image the blocks of code in green run at a 0.2 second sample rate and the blocks of code in red run at a 0.1 second sample rate.

This image shows the different components of the generated structured text code.

```
VAR
    UnitDelay_DSTATE: LREAL;
    UnitDelay1_DSTATE: LREAL;
    UnitDelay1: LREAL;
END_VAR
VAR_TEMP
    rtb_UnitDelay: LREAL;
    rtb_Gain1: LREAL;
END_VAR
CASE ssMethodType OF
    SS_INITIALIZE:
        plc_ts_counter1 := 0;
        (* SystemInitialize for Atomic SubSystem: '<Root>/SimpleSubsystem' *)
        (* InitializeConditions for UnitDelay: '<S1>/Unit Delay' *)
        UnitDelay_DSTATE := 0.0;
        (* InitializeConditions for UnitDelay: '<S1>/Unit Delay1' *)
        UnitDelay1_DSTATE := 0.0;
        (* End of SystemInitialize for SubSystem: '<Root>/SimpleSubsystem' *)
    SS_STEP:
        (* Outputs for Atomic SubSystem: '<Root>/SimpleSubsystem' *)
        (* UnitDelay: '<S1>/Unit Delay' *)
        rtb_UnitDelay := UnitDelay_DSTATE;
        (* Gain: '<S1>/Gain' incorporates:
         *  Sum: '<S1>/Sum' *)
        Y1 := (U1 - rtb_UnitDelay) * 0.5;
        IF plc_ts_counter1 = 0 THEN
            (* UnitDelay: '<S1>/Unit Delay1' *)
            UnitDelay1 := UnitDelay1_DSTATE;
            (* Gain: '<S1>/Gain1' incorporates:
             *  Sum: '<S1>/Sum1' *)
            rtb_Gain1 := (U2 - UnitDelay1) * 0.5;
            (* Update for UnitDelay: '<S1>/Unit Delay1' *)
            UnitDelay1_DSTATE := rtb_Gain1;
        END_IF;
        (* Update for UnitDelay: '<S1>/Unit Delay' *)
        UnitDelay_DSTATE := Y1;
        (* End of Outputs for SubSystem: '<Root>/SimpleSubsystem' *)
        IF plc_ts_counter1 = 0 THEN
            (* Outport: '<Root>/Y2' *)
            Y2 := rtb_Gain1;
        END_IF;
        (* Outputs for Atomic SubSystem: '<Root>/SimpleSubsystem' *)
        (* Outport: '<Root>/Y3' incorporates:
         *  Sum: '<S1>/Sum2' *)
        Y3 := rtb_UnitDelay + UnitDelay1;
        (* End of Outputs for SubSystem: '<Root>/SimpleSubsystem' *)
        IF plc_ts_counter1 >= 1 THEN
            plc_ts_counter1 := 0;
        ELSE
            plc_ts_counter1 := plc_ts_counter1 + 1;
        END_IF;
END_CASE;
END_FUNCTION_BLOCK
VAR_GLOBAL CONSTANT
    SS_INITIALIZE: SINT := 0;
    SS_STEP: SINT := 1;
END_VAR
VAR_GLOBAL
    plc_ts_counter1: DINT;
END_VAR
```

0.1 second sample rate

0.2 second sample rate

Logic to implement time step counter

Time step counter variable

When you deploy the generated code, you must run the code at the fastest sample rate. For example, when deploying the generated code for this model run the code at a 0.1 second sample rate.

See Also

- "Generating Simulink PLC Coder Structured Text Code for Multirate Models" on page 20-7
- "Sample Times in Subsystems"
- "Sample Times in Systems"

Simulate and Generate Structured Text Code for a Stateflow Chart

This example shows how to simulate and generate code for the ControlModule Stateflow® chart from the `plcdemo_stateflow_controller` model.

Open the Model

```
open_system('plcdemo_stateflow_controller')
```
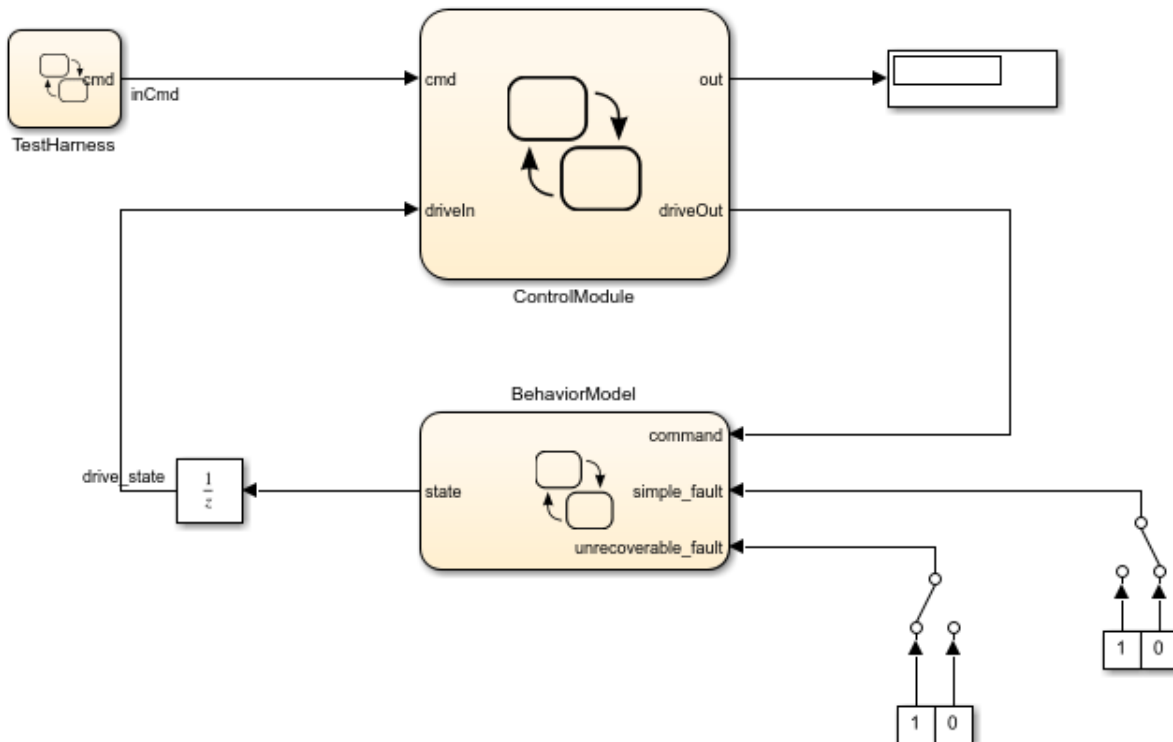
This example shows the use of a Stateflow chart to implement the control logic for a drive.
The TestHarness chart provides a test scenario of starting, holding, and resetting the drive.
The BehaviorMode chart provides a simple chart to test the ControlModule chart behavior by injecting faults.
The ControlModule chart performs the drive control logic.
To generate code for the subsystem, select the ControlModule chart and right-click **PLC Code > Generate Code for Subsystem**.

Copyright 2009-2019 The MathWorks, Inc.

To start the simulation, click **Run**.

Generate Code

To generate code for the ControlModule chart, use plcgeneratecode. For more information, see `plcgeneratecode`:

generatedfiles = plcgeneratecode('plcdemo_stateflow_controller/ControlModule');

The plcdemo_stateflow_controller consists of stateflow charts to simulate a drive module. The TestHarness chart provides a test scenario of starting, holding, and resetting the drive.

`open_system(`'plcdemo_stateflow_controller/TestHarness'`);`
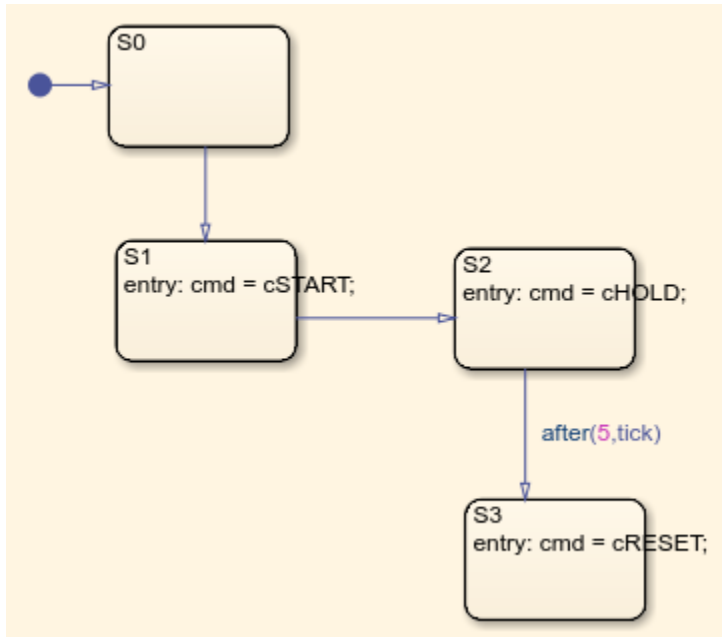
The BehaviorModel chart provides a simple chart to test the ControlModule chart behavior by injecting faults.

`open_system(`'plcdemo_stateflow_controller/BehaviorModel'`);`
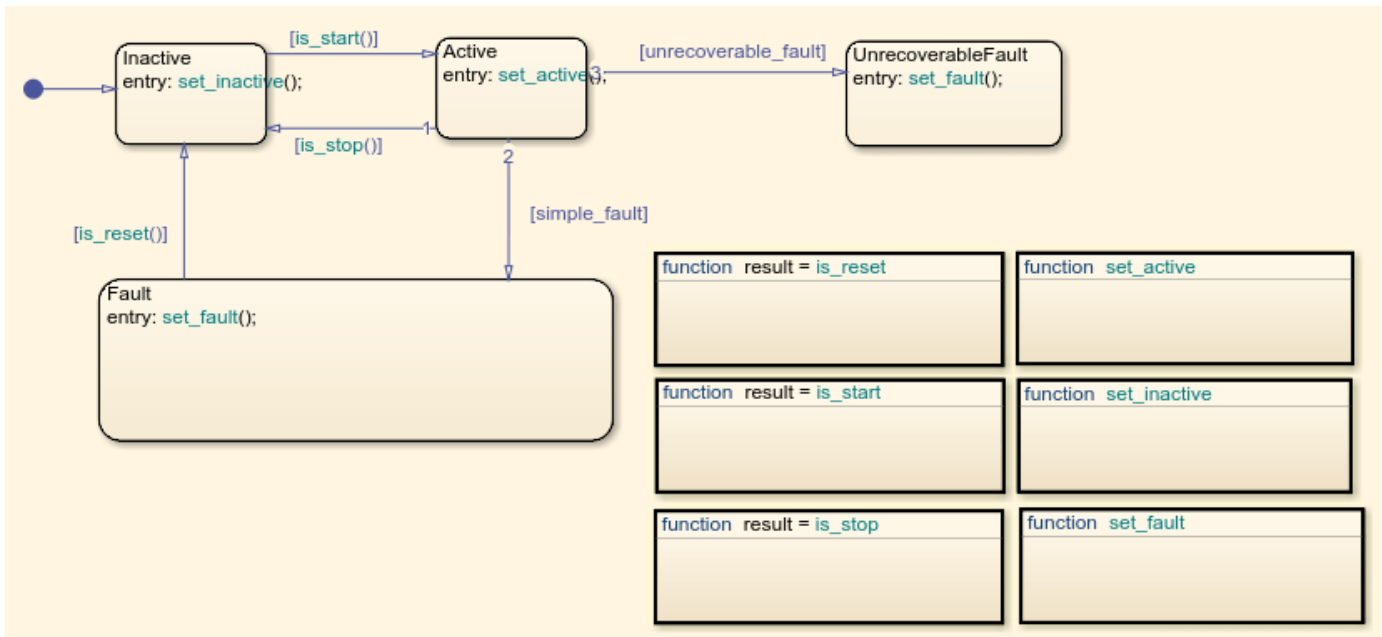
The ControlModule chart performs the drive control logic.

```
open_system('plcdemo_stateflow_controller/ControlModule');
```
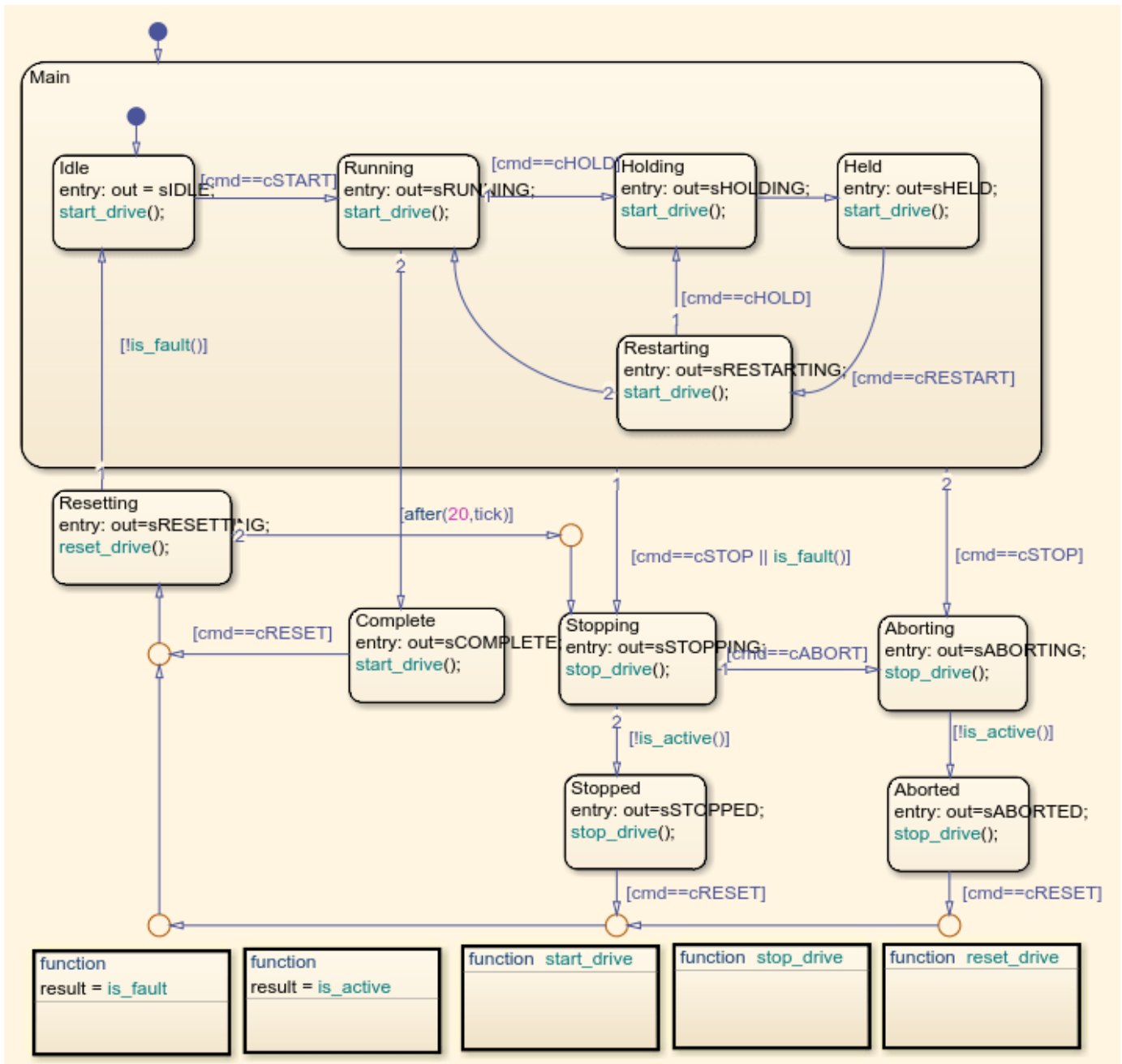
Generate Structured Text Code for a MATLAB Function Block

This example shows how to model, simulate, and generate code for a Simulink® subsystem that has a MATLAB® Function Block. Generate code for the MATLAB Function Block by using Simulink PLC Coder™.

Model Description

The model consists of these components:

- `TankControl` — MATLAB Function block that takes the tank height and valve command as inputs and generates the tank input flow rate, output flow rate, and stirrer speed.
- `Tank Dynamics` — Simulink subsystem that models the dynamics of the tank.
- `Signal Builder` — `Signal Builder` block that models the valve command to the `TankControl` block.

Open the model.

```
open_system('plcdemo_eml_tankcontrol');
```
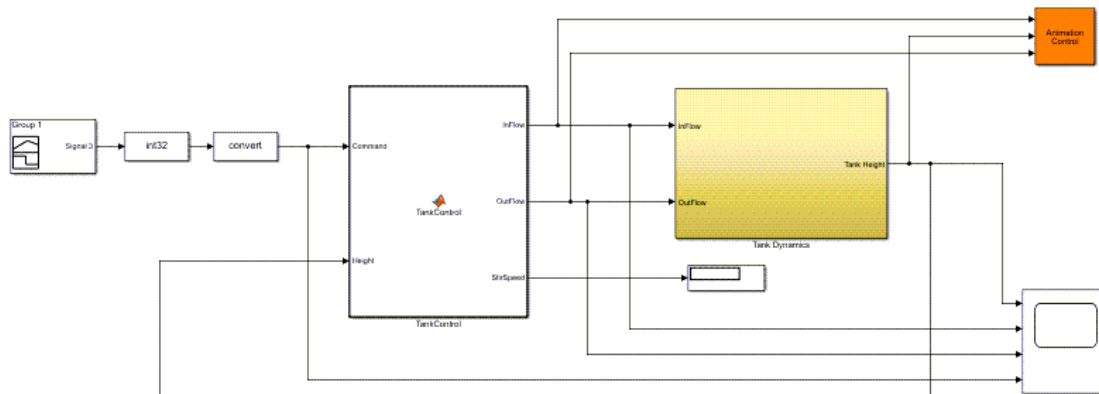

Copyright 2009-2021 The MathWorks, Inc.

Simulate the model and verify the valve operation by changing the tank height, flow rates, and other parameters. After verifying that the `TankControl` block functions according to your requirements, generate code for the `TankControl` block.
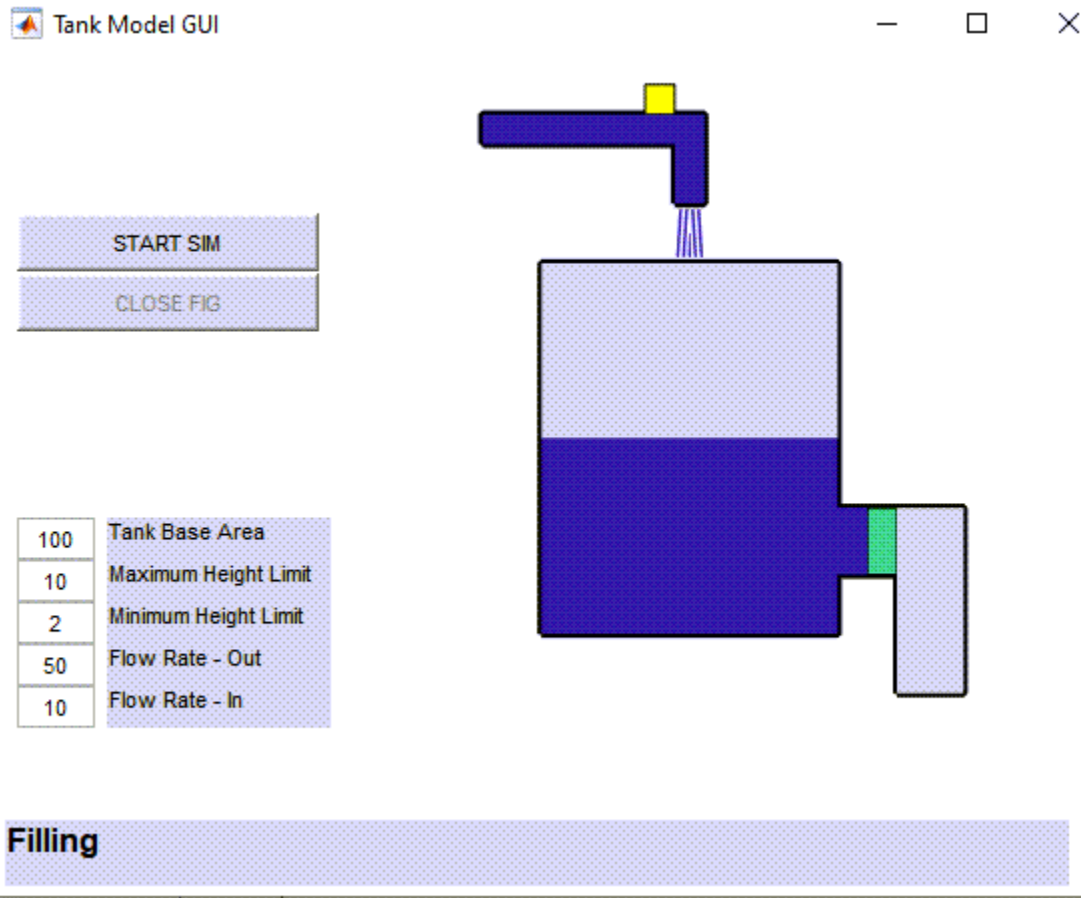
Generate Code

To generate structured text code, do one of the following:

- Open the PLC Coder app. Select the `SimpleSubsystem` block and click `Generate PLC Code`.
- Use the `plcgeneratecode` function:

```
plcgeneratecode('plcdemo_eml_tankcontrol/TankControl');
```

```
### Generating PLC code for 'plcdemo_eml_tankcontrol/TankControl'.
### Using model settings from 'plcdemo_eml_tankcontrol' for PLC code generation parameters.
### Begin code generation for IDE codesys23.
### Emit PLC code to file.
### PLC code generation successful for 'plcdemo_eml_tankcontrol/TankControl'.
### Generated files:
.\plcsrc\plcdemo_eml_tankcontrol.exp
```

See Also

- "MATLAB Function Block Simulink PLC Coder Structured Text Code Generation" on page 20-9
- `plcgeneratecode`

Generating Structured Text for a Feedforward PID Controller

This model shows the code generated for a Feedforward PID Controller implemented using Simulink® library blocks.

```
mdl = 'plcdemo_pid_feedforward';
open_system(mdl);
```


This model contains the following subsystem which implements the Feedforward controller.

```
open_system('plcdemo_pid_feedforward/pid_feedforward');
```
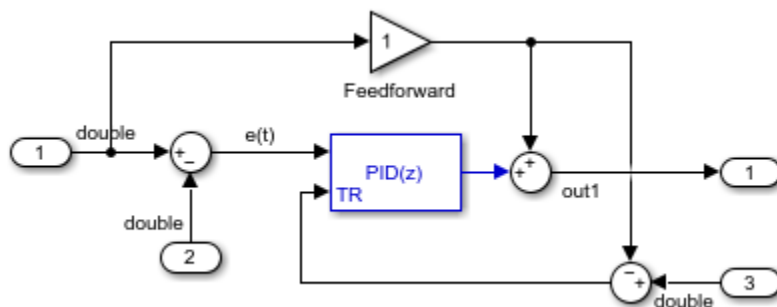
You can generate PLC Structured Text code for this subsystem by right-clicking on the subsystem block and select PLC Code -> Generate Code for Subsystem Alternatively, you can use the following command

```
generatedFiles = plcgeneratecode('plcdemo_pid_feedforward/pid_feedforward');
```

After the code generation, the Diagnostic Viewer window is displayed with hyperlinks to the generated code files. You can open the generated files by clicking on the links.

Mapping Tunable Parameters to Structured Text

This model shows how to map tunable parameters from the Simulink® model to the generated Structured Text code.
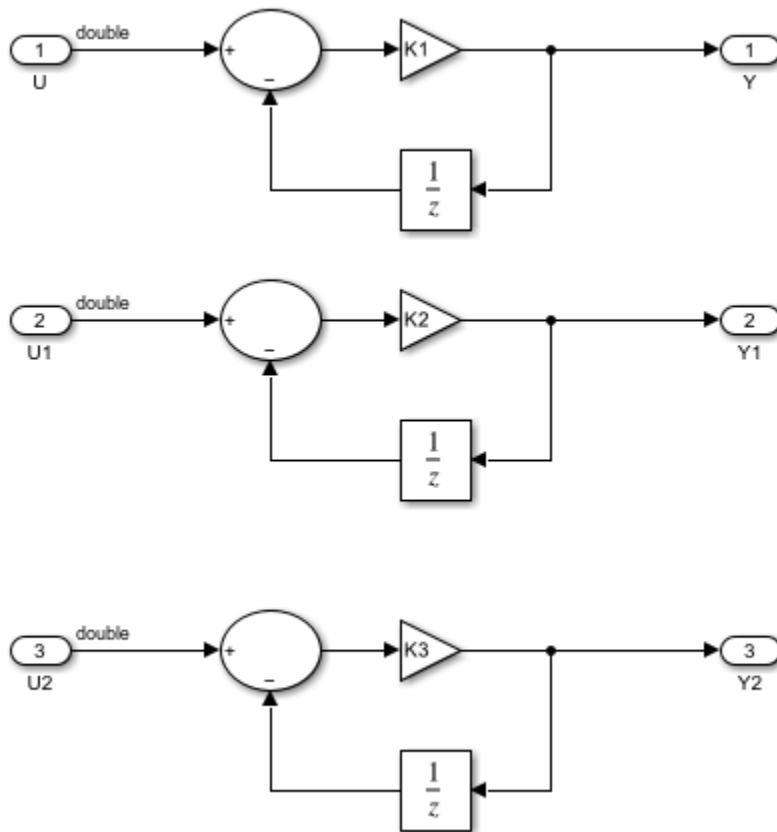
This model shows the different implementations of tunable parameters in the generated code. It makes use of three parameters K1, K2, and K3 defined in the MATLAB base workspace. To build the subsystem, right-click on the subsystem block and select PLC Code Generation > Generate Code for Subsystem. The Diagnostic Viewer with hyperlinks to the generated code is displayed automatically.

In this model:

- K1 is set to "Auto" storage class
- K2 is set to "ExportedGlobal" storage class
- K3 is set to "ExportedGlobal" constant storage class

In the generated Structured Text code for compatible IDE targets:

- K1 is mapped to a Function Block local variable
- K2 is mapped to a global variable
- K3 is mapped to a global constant

For the RSLogix 5000 Add On Instruction (AOI) format:

- K1 is mapped to an AOI local tag
- K2 and K3 are mapped to AOI input tags

For the RSLogix 5000 Routine format:

- K1 is mapped to routine instance tag
- K2 and K3 are mapped to global program tags

See the Simulink PLC Coder documentation on tunable parameter code generation for more information.

You can generate PLC Structured Text code for this subsystem by right-clicking on the subsystem block and select PLC Code -> Generate Code for Subsystem Alternatively, you can use the following command generatedFiles = plcgeneratecode('plcdemo_tunable_params/SimpleSubsystem');

After the code generation, the Diagnostic Viewer window is displayed with hyperlinks to the generated code files. You can open the generated files by clicking on the links.

Simulation and Code Generation for Tunable Parameters

This example shows how to map tunable parameters that are defined as Simulink.Parameter objects in the MATLAB® workspace to structured text code.

Open Model
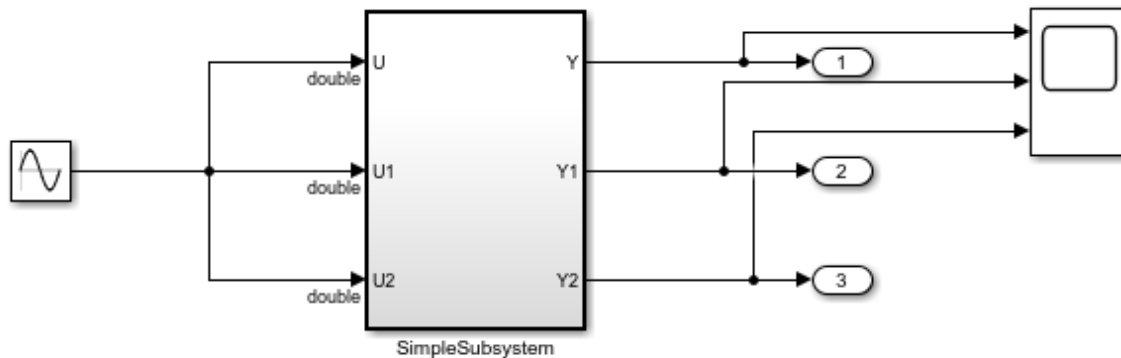
This model shows the usage of tunable parameters by specifying them as Simulink.Parameter objects in MATLAB base workspace.This model makes use of three parameters K1, K2 and K3 defined in the MATLAB base workspace as Simulink.Parameter objects (please see 'setup_tunable_params.m' in the same directory as this model).
To build the subsystem, open the PLC Coder app, select the subsystem block, and in the PLC Code tab click Generate PLC Code right-click on the subsystem block and select PLC Code > Generate Code for Subsystem. The Diagnostic Viewer displays hyperlinks to the generated code files, click the links to view the generated files.

In this model:
* K1 has 'Model default' storage class
* K2 has 'ExportedGlobal' storage class
* K3 has 'Custom' storage class and 'Const' custom storage class

In the generated Structured Text code for compatible IDE targets:
* K1 is mapped to a Function Block local variable
* K2 is mapped to a global variable
* K3 is mapped to a global constant

For the Step7 and TIA Portal targets:
* K1 is mapped to a Function Block local variable
* K2 and K3 are mapped to fields of DATA_BLOCK PLCGlobalVar

For the Studio 5000 and RSLogix 5000 Add On Instruction (AOI) format:
* K1 is mapped to an AOI local tag
* K2 and K3 are mapped to fields in an AOI InOut tag, *top_subsystem_name_*Gvar, of struct type *top_subsystem_name_*GVarUDT. At initialization, the top subsystem AOI calls the PLC_INIT_PARAMETERS
AOI to initialize the tag with values of K2 and K3. Besides the generated code L5X file, a separate L5X file has definition of the tag. You can import the tag through this file.

For the Studio 5000 and RSLogix 5000 Routine format:
* K1 is mapped to routine instance tag
* K2 and K3 are mapped to global program tags

In this model:

- K1 has 'Model default' storage class
- K2 has 'ExportedGlobal' storage class
- K3 has 'ExportedGlobal' storage class and 'Const' custom storage class

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

%   Copyright 2011-2019 The MathWorks, Inc.

% tunable parameter mapped to local variable
K1 = Simulink.Parameter;
K1.Value = 0.1;
K1.CoderInfo.StorageClass = 'Model default';

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

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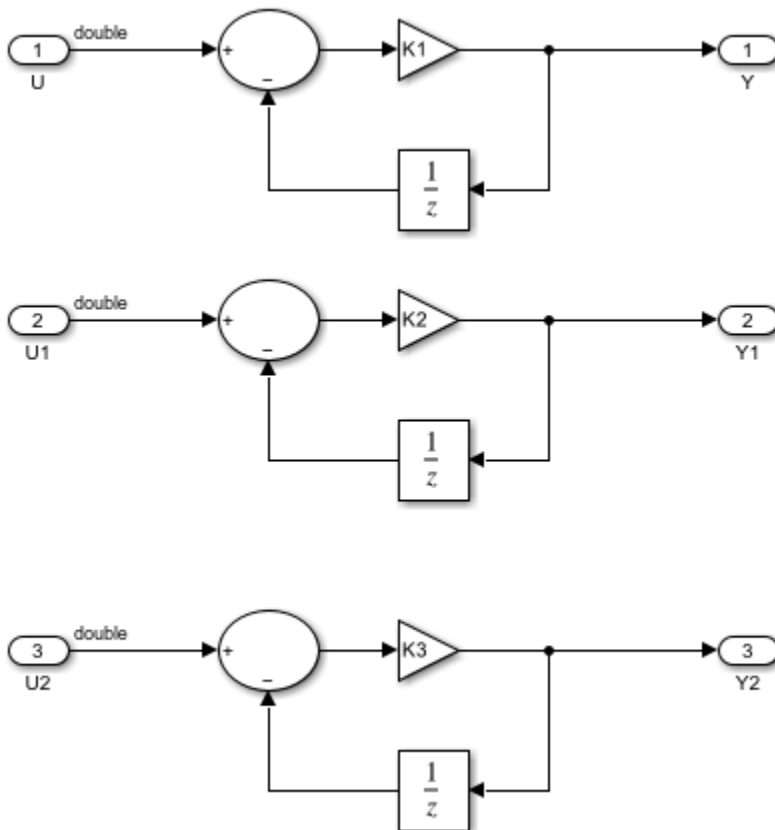
Generate PLC Code

Open the **PLC Coder** app, select the `SimpleSubsystem` block, and on the **PLC Code** tab click **Generate PLC Code**.

To generate code using the MATLAB command line, enter:

```
generatedFiles = plcgeneratecode('plcdemo_tunable_params_slparamobj/SimpleSubsystem');
```

After the code generation, the Diagnostic Viewer window is displayed with hyperlinks to the generated code files. You can open the generated files by clicking on the links.

Tunable Parameter Mapping in Generated Code

In the generated Structured Text code for compatible IDE targets:

- K1 is mapped to a Function Block local variable
- K2 is mapped to a global variable
- K3 is mapped to a global constant

For the RSLogix 5000 Add On Instruction (AOI) format:

- K1 is mapped to an AOI local tag
- K2 and K3 are mapped to AOI input tags

For the RSLogix 5000 Routine format:

- K1 is mapped to routine instance tag
- K2 and K3 are mapped to global program tags

See Also

- "Block Parameters in Generated Code" on page 7-2
- "Control Appearance of Block Parameters in Generated Code" on page 7-4

Simulate and Generate Code for Speed Cruise Control System

This example shows how to simulate and generate code for the Controller subsystem from a speed cruise control model by using Simulink® and Stateflow®.

Open the Model

```
open_system('plcdemo_cruise_control')
```

Speed Cruise Control System Using Simulink and Stateflow
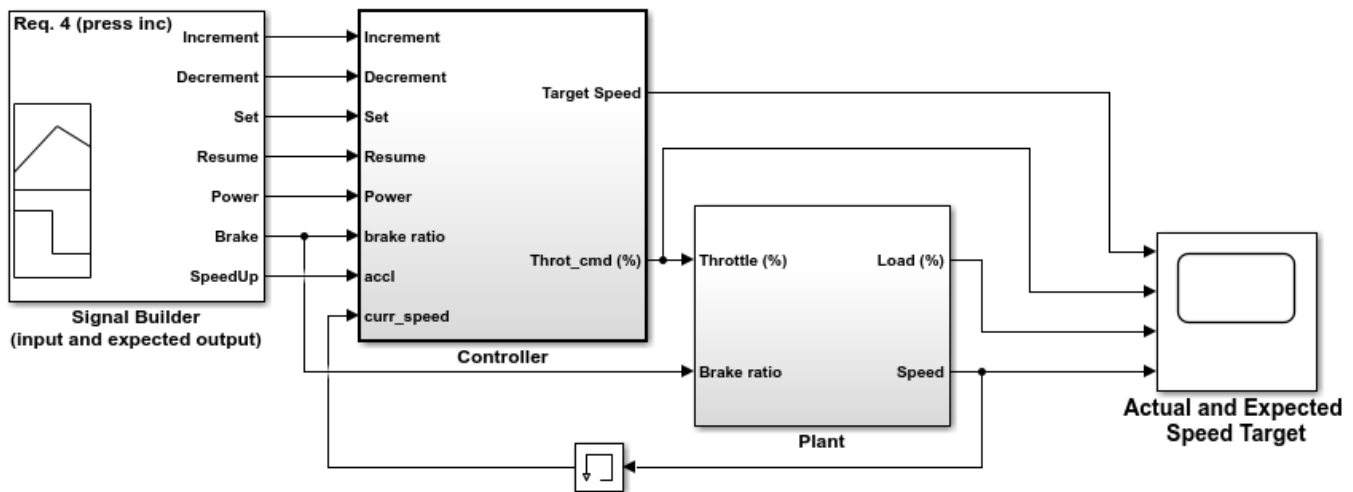
This model shows the code generation for the Speed Cruise Control Controller subsystem. Open the Controller subsystem. This model uses a Triggered Stateflow Chart for the Enable and Setpoint calculations. It uses a discrete PID Controller to compute the Throttle Command. Click the scopes to observe the Target and Actual speeds.

Copyright 2002-2019 The MathWorks, Inc.

To start the simulation, click **Run**.

Generate Code

To generate code for the Controller subsystem, use `plcgeneratecode`:

```
generatedfiles = plcgeneratecode('plcdemo_cruise_control/Controller');
```

The Controller subsystem performs the Enable and Setpoint calculations by using a Triggered Stateflow® chart. The Throttle Command is computed by using a discrete PID controller.

Variable Step Speed Cruise Control System

This example shows how to simulate and generate code for the Controller subsystem from a speed cruise control model by using variable continuous step solver.

Open the Model

open_system('plcdemo_cruise_control_continuous');

This model shows PLC code generation using variable-step continuous solver for a Speed Cruise Control controller subsystem. PLC Coder supports code generation of model controller subsystem block with fixed sample time. To satisfy this requirement, user can
 1. Choose fixed-step solver so the controller subsystem block runs at fixed sample time as demonstrated in the plcdemo_cruise_control demo;
 2. Choose variable-step continuous solver and set explicit sample time for the controller block as demonstrated in this demo.

In this model, the model solver is set to variable-step continuous (ode45); the sample time of the controller subsystem is set to 0.1. To set the controller sample time, right click on the controller block and select Subsystem Parameters, on the Main tab, set the Sample time parameter. This method allows combined modeling of discrete-time controller and continuous-time plant in the same model with PLC code generation support.

To build code for the subsystem, right-click on the "Controller" subsystem block and select PLC Code > Generate Code for Subsystem.

The Diagnostic Viewer displays hyperlinks to the generated code files, click the links to view the generated files.

Copyright 2002-2019 The MathWorks, Inc.

To start the simulation, click **Run**

Generate Code

To generate code for the Controller subsystem, use `plcgeneratecode`:

```
generatedfiles =
plcgeneratecode('plcdemo_cruise_control_continuous/Controller');
```

In this model, the model solver is set to variable-step continuous (ode45); the sample time of the controller subsystem is set to 0.05. To set the controller sample time, right click on the controller block and select Subsystem Parameters, on the Main tab, set the Sample time parameter. This method allows combined modeling of discrete-time controller and continuous-time plant in the same model with PLC code generation support.

Simulate and Generate Code for Airport Conveyor Belt Control System

This example shows how to simulate and generate code for the Controller subsystem from an airport conveyor belt model.

Open the Model
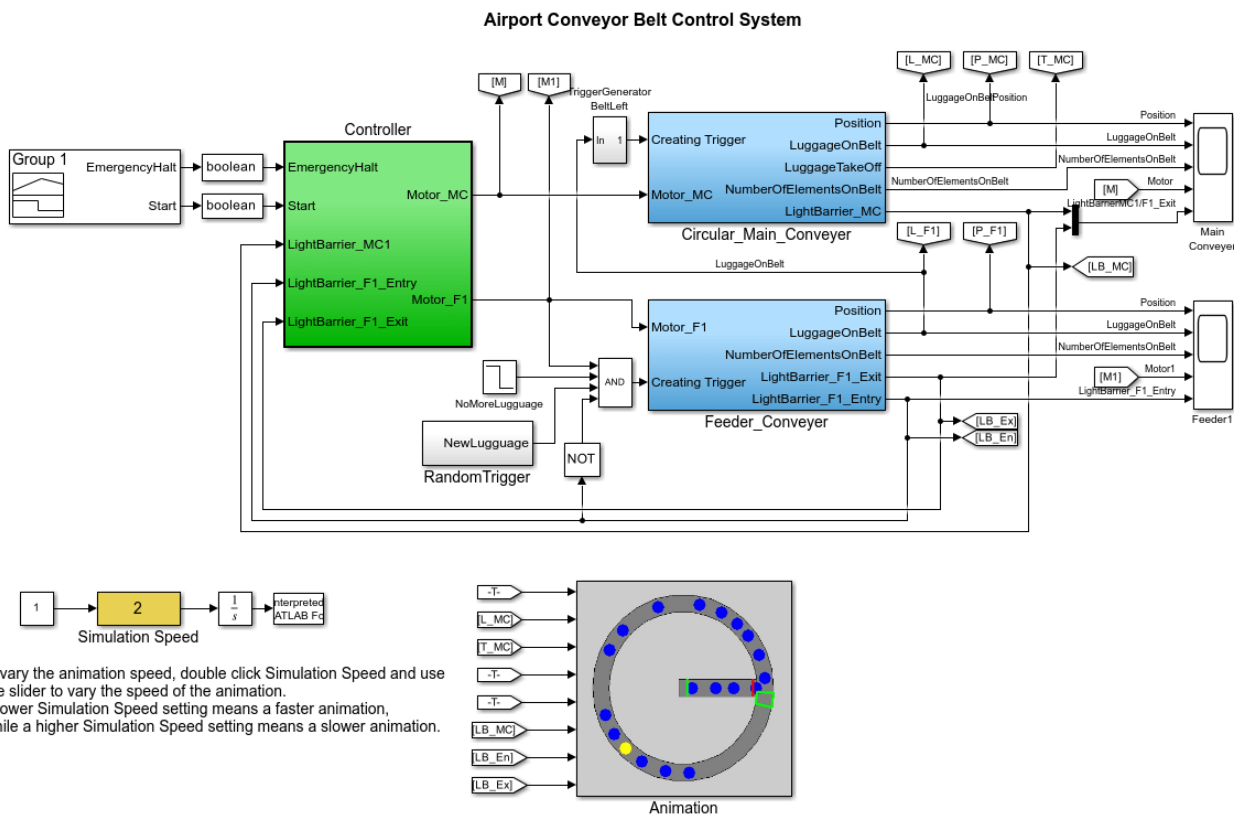
```
open_system('plcdemo_airport_conveyor')
```


To start the simulation, click **Run**. Observe the conveyor belt animation.

Generate Code

To generate code for the Controller subsystem, use `plcgeneratecode`:

generatedfiles = plcgeneratecode('plcdemo_airport_conveyor/Controller')

Generate Structured Text Code for Simulink Model That Has Fixed-Point Data Types

This example shows how to generate structured text code for a Simulink® model that has fixed-point data types by using Simulink® PLC Coder™.

Model Description

The model consists of a subsystem block that performs mathematical operations and delays on the inputs to the subsystem block. Open the model:

```
load_system('plcdemo_fixed_point');
open_system('plcdemo_fixed_point/Subsystem');
```
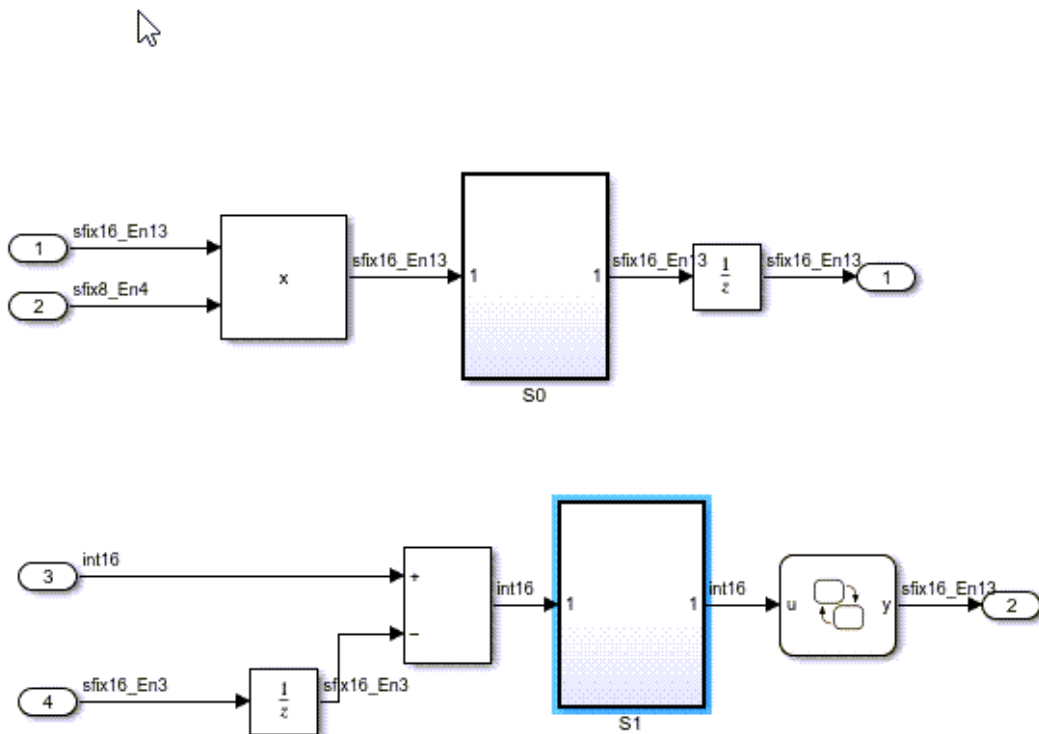

Generate Code

To generate structured text code for the subsystem, do one of the following:

- Open the PLC Coder app. Select the `Subsystem` block and click **Generate PLC Code**.
- Use the `plcgeneratecode` function:

```
plcgeneratecode('plcdemo_fixed_point/Subsystem');
```

```
### Generating PLC code for 'plcdemo_fixed_point/Subsystem'.
### Using model settings from 'plcdemo_fixed_point' for PLC code generation parameters.
### Begin code generation for IDE step7.
### Emit PLC code to file.
### Creating PLC code generation report plcdemo_fixed_point_codegen_rpt.html.
### PLC code generation successful for 'plcdemo_fixed_point/Subsystem'.
### Generated files:
plcsrc\plcdemo_fixed_point\plcdemo_fixed_point.scl
```

See Also

- "Fixed Point Simulink PLC Coder Structured Text Code Generation" on page 20-2

Generate Structured Text Code for a Stateflow Chart That Uses Absolute-Time Temporal Logic

Model, simulate, and generate code for a Stateflow® chart that uses absolute-time temporal logic. Absolute-time temporal logic tracks elapsed time since the state became active. Absolute-time temporal logic is used in industrial applications on programmable logic controllers (PLCs) to implement logic for processes that transition between process steps by using a combination of logic- and time-based transitions. After verifying that the model functions for your requirements, generate code for the Stateflow® chart by using Simulink® PLC Coder™.

Model Description

The model has a simple Stateflow® chart called `Temporal`. The chart transitions between four states that use absolute-time temporal logic. Verify that the transitions occur at the designed times by simulating the model and opening the Scope block. This image shows the absolute-time temporal logic based transitions inside the `Temporal` chart.
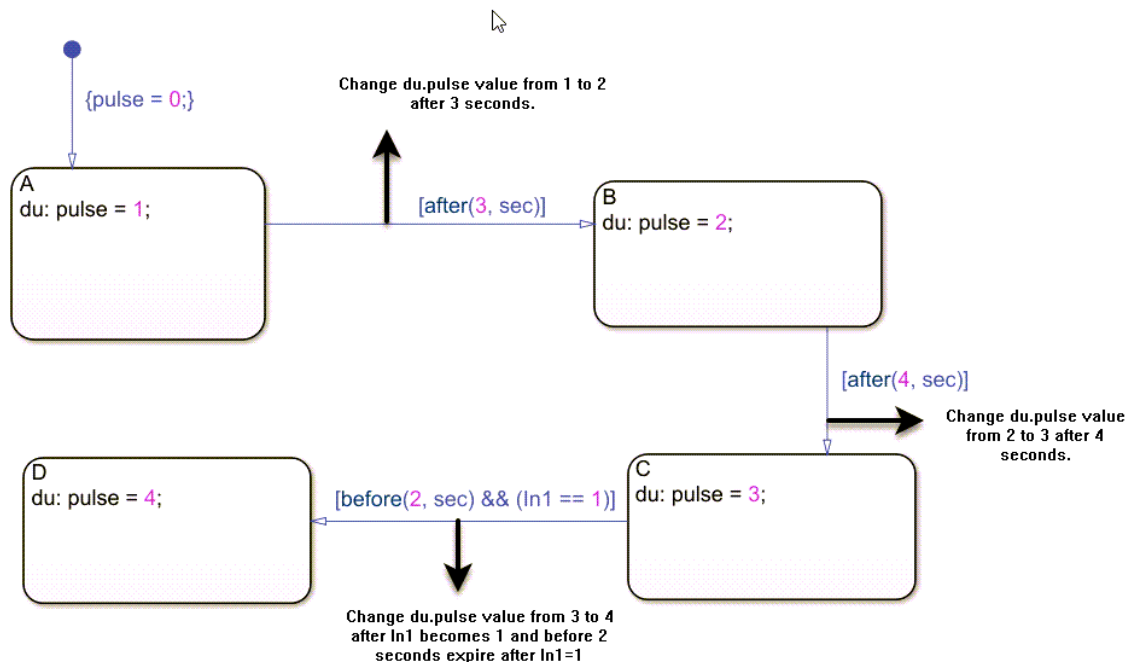
Open the model:

```
load_system('plcdemo_sf_abs_time');
open_system('plcdemo_sf_abs_time/Temporal');
```

Generate Code

To generate structured text code, do one of the following:

- Open the PLC Coder app. Click `Settings > PLC Code Generation > Interface > Absolute-time temporal logic`. Set `Absolute-time temporal logic` to either `Target timer` or `Target-independent counter`. For more information on which option to set, see "Absolute-Time Temporal Logic" on page 13-38.

- Open the PLC Coder app. Select the `Temporal` chart and click `Generate PLC Code`.

- Use the `plcgeneratecode` function:

```
warning('OFF', 'plccoder:plccheckreport:UnsupportedType');
plcgeneratecode('plcdemo_sf_abs_time/Temporal');
```

```
### Generating PLC code for 'plcdemo_sf_abs_time/Temporal'.
### Using model settings from 'plcdemo_sf_abs_time' for PLC code generation parameters.
### Begin code generation for IDE rslogix5000.
Creating PLC Code Generation Check Report file://C:\TEMP\Bdoc22b_2054784_6060\ibB18F8B\28\tp81b09
PLC check for 'plcdemo_sf_abs_time' complete with 0 errors, 4 warnings, and 0 messages.
### Emit PLC code to file.
### Creating PLC code generation report plcdemo_sf_abs_time_codegen_rpt.html.
### PLC code generation successful for 'plcdemo_sf_abs_time/Temporal'.
### Generated files:
plcsrc\plcdemo_sf_abs_time.L5X
```

```
warning('ON', 'plccoder:plccheckreport:UnsupportedType');
```

Structure of Generated Code

The generated structured text code components depend on the setting of the `Absolute-time temporal logic` parameter. For more information, see "Absolute-Time Temporal Logic" on page 13-38.

To deploy the generated code to a target PLC hardware that is supported by the target IDE, set `Absolute-time temporal logic` to `Target timer`. When you set `Absolute-time temporal logic` to `Target timer`, the generated code has a function block that implements the target integrated development environment (IDE) specific timer semantics. This image shows the function block generated for the Codesys version 2.3 target IDE.

```
FUNCTION_BLOCK PLC_CODER_TIMER
VAR_INPUT
    timerAction: INT;
    maxTime: DINT;
END_VAR
VAR_OUTPUT
    done: BOOL;
END_VAR
VAR
    plcTimer: TON;
    plcTimerExpired: BOOL;
END_VAR
CASE timerAction OF
    1:
        (* RESET *)
        plcTimer(IN:=FALSE, PT:=T#0ms);
        plcTimerExpired := FALSE;
        done := FALSE;
    2:
        (* AFTER *)
        IF (NOT(plcTimerExpired)) THEN
            plcTimer(IN:=TRUE, PT:=DINT_TO_TIME(maxTime));
        END_IF;
        plcTimerExpired := plcTimer.Q;
        done := plcTimerExpired;
    3:
        (* BEFORE *)
        IF (NOT(plcTimerExpired)) THEN
            plcTimer(IN:=TRUE, PT:=DINT_TO_TIME(maxTime));
        END_IF;
        plcTimerExpired := plcTimer.Q;
        done := NOT(plcTimerExpired);
END_CASE;
END_FUNCTION_BLOCK
VAR_GLOBAL CONSTANT
    Temporal_IN_NO_ACTIVE_CHILD: USINT := 0;
    Temporal_IN_A: USINT := 1;
    Temporal_IN_B: USINT := 2;
    Temporal_IN_C: USINT := 3;
    Temporal_IN_D: USINT := 4;
    SS_INITIALIZE: SINT := 0;
    SS_STEP: SINT := 1;
END_VAR
```

To verify your generated code by generating a testbench, set `Absolute-time temporal logic` to `Target-independent counter`. When you set `Absolute-time temporal logic` to `Target-independent counter` the generated code implements the timer by using a target IDE independent counter block.

See Also

- "Absolute-Time Temporal Logic" on page 13-38
- "Control Chart Execution by Using Temporal Logic" (Stateflow)

Integrating User Defined Function Blocks, Data Types, and Global Variables into Generated Structured Text

This model shows how to integrate user defined function blocks, data types and global variables into generated structured text.
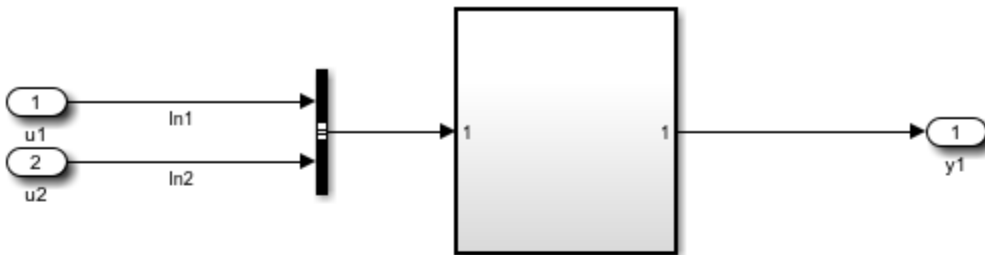
This model shows how to integrate user defined function blocks, data types and global variables into generated structured text.

Right click on the top level subsystem 'Subsystem' and select 'PLC Code > Options ...' to bring up the 'Configuration Parameters' dialog box. In the dialog box, select 'Symbols' pane under 'PLC Code Generation'. You will see the following symbols in the 'Externally Defined Symbols' field:

  ExternallyDefinedBlock InBus K1

Specifying these symbols here would respectively omit the function block, bus data type and global variable with the same names in the generated structured text.

To build the subsystem, right-click on the subsystem block and select PLC Code > Generate Code for Subsystem.

The Diagnostic Viewer displays hyperlinks to the generated code files, click the links to view the generated files. In the generated code, you will notice that there are calls to function block 'ExternallyDefinedBlock', however the definition of the function block has been omitted. Similarly, the definition of bus type 'InBus1' and global variable 'K1' have also been omitted.

Open the top level subsystem 'Subsystem' by double clicking on it. You will notice it has a bunch of blocks including the block 'ExternallyDefinedBlock'. The user would like to replace this with an externally defined block in the PLC IDE in the generated code. In this model 'ExternallyDefinedBlock' is a MATLAB® block. This could be any other Simulink® block or subsystem as well. The input port 'In1' is a bus input of 'InBus' data type. The user would like to provide the definition of 'InBus' externally in the PLC IDE. Similarly, the user would like to provide the definition of the 'K1' global tunable parameter of the Gain block externally.
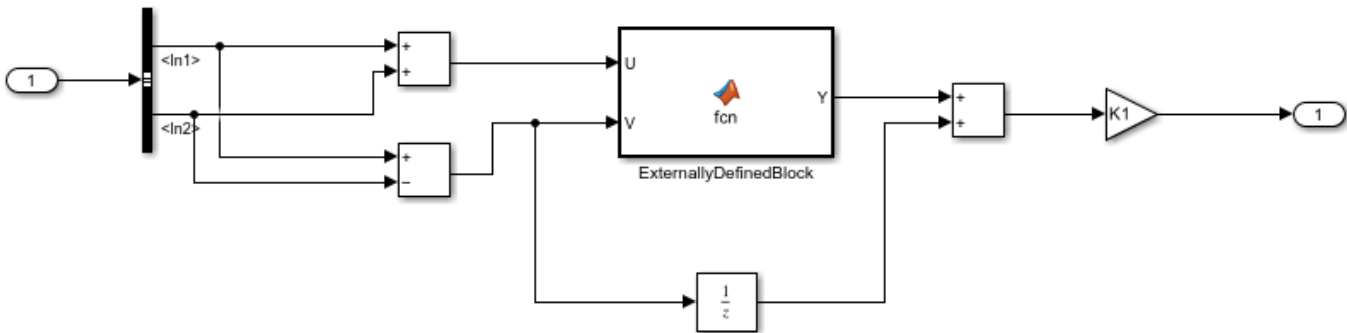
To do this, right click on the top level subsystem 'Subsystem' and select 'PLC Code -> Options ...' to bring up the 'Configuration Parameters' dialog box. In the dialog box, select 'Symbols' pane under 'PLC Code Generation'. You will see the following symbols in the 'Externally Defined Symbols' field:

```
ExternallyDefinedBlock InBus K1
```

Specifying these symbols here would respectively omit the function block, bus data type and global variable with the same names in the generated structured text.

Now you can generate PLC Structured Text code for this subsystem by right-clicking on the subsystem block and select PLC Code -> Generate Code for Subsystem Alternatively, you can use the following command generatedFiles = plcgeneratecode('plcdemo_external_symbols/Subsystem');

After the code generation, the Diagnostic Viewer window is displayed with hyperlinks to the generated code files. You can open the generated files by clicking on the links.

In the generated code, you will notice that there are calls to function block 'ExternallyDefinedBlock', however the definition of the function block has been omitted. Similarly, the definition of bus type 'InBus1' and global variable 'K1' have also been omitted.

Simulate and Generate Structured Text Code for Rockwell Automation Motion Instructions

This example shows how to model Rockwell Automation® motion instructions in Stateflow®. Generate structured text code for the modeled motion instructions by using Simulink® PLC Coder™.

Model Description

The model consists of a Stateflow® chart called `MotionController`. The `MotionController` chart consists of these components:

- `Controller1` and `Controller2` subcharts that simulate the motion instruction.
- `Drive1` and `Drive2` subcharts that simulate the motion of a drive and the axis on which motion control is performed.
- `MSF`, `MSO`, and `MAM` graphical functions that define the respective motion instructions.
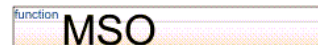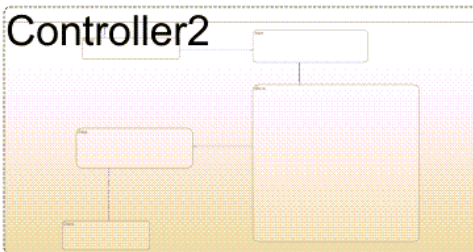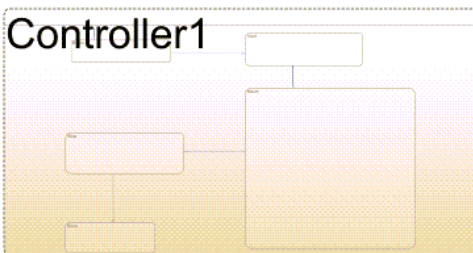
Open the model:

```
load_system('MotionControllerExample');
open_system('MotionControllerExample/MotionController/Chart');
```

Generate Code

Generate structured text code for the chart by using the `plcgeneratemotionapicode` function:

```
warning('OFF', 'plccoder:plccg_ext:AutomaticTypeConversions');
plcgeneratemotionapicode('MotionControllerExample/MotionController');

Created temporary model for code generation :MotionController
### Generating PLC code for 'MotionController/MotionController'.
### Using model settings from 'MotionController' for PLC code generation parameters.
### Begin code generation for IDE studio5000.
### Emit PLC code to file.
### PLC code generation successful for 'MotionController/MotionController'.
### Generated files:
plcsrc\MotionController.L5X

warning('ON', 'plccoder:plccg_ext:AutomaticTypeConversions');
```

Clean Up Generated Files

To close the model and clean up the generated files:

```
close_system('MotionControllerExample');
```

See Also

- "Simulation and Code Generation of Motion Instructions" on page 1-18

Tank Control Simulation and Code Generation by Using Ladder Logic

This example shows how to simulate ladder logic and generate code from the ladder tank controller model.

Import, Simulate,and Generate Code

1. Create a folder with write permission and copy the files `plcdemo_ladder_tankcontrol_template.slx` and `TankControl.L5X` into that folder.

2. Change the current folder to the newly created folder and rename `plcdemo_ladder_tankcontrol_template.slx` to `plcdemo_ladder_tankcontrol.slx`.

3. In MATLAB, run the plcimportladder command. for more information, see `plcimportladder` command:

```
plcimportladder('TankControl','TopAOI','TankControl');
```

4. Open the generated model `TankControl_runner_TankControl.slx` and select and copy the `TankControl_runner` block. Open `plcdemo_ladder_tankcontrol`, and replace `Controller/TankControl_runner` with the copied block.

5. To start the simulation, click **Run**. Open the `Tank HMI` block and use the Control Command rotary switch to set controller command input.

Set the Control Command Input

- Set the Control Command switch to the 'Fill' position to fill the tank.
- Set the Control Command switch to the 'Hold' position to hold the current tank state.
- Set the Control Command switch to the 'Empty' position to empty the tank.
- Set the Control Command switch to the 'Stir' position to activate the tank stir state.

The tank enters the Stir state only when the fluid level is full. Otherwise the Stir command has no effect. If the tank is in the Stir state, the Stir indicator lamp is on. Otherwise, it is off. The numeric value of the tank command is:

- Fill -- 0
- Hold -- 1
- Empty -- 2
- Stir --3

The tank animation UI shows the tank status as the simulation runs.

The completed simulink model should resemble

```
open_system('plcdemo_ladder_tankcontrol_complete');
```
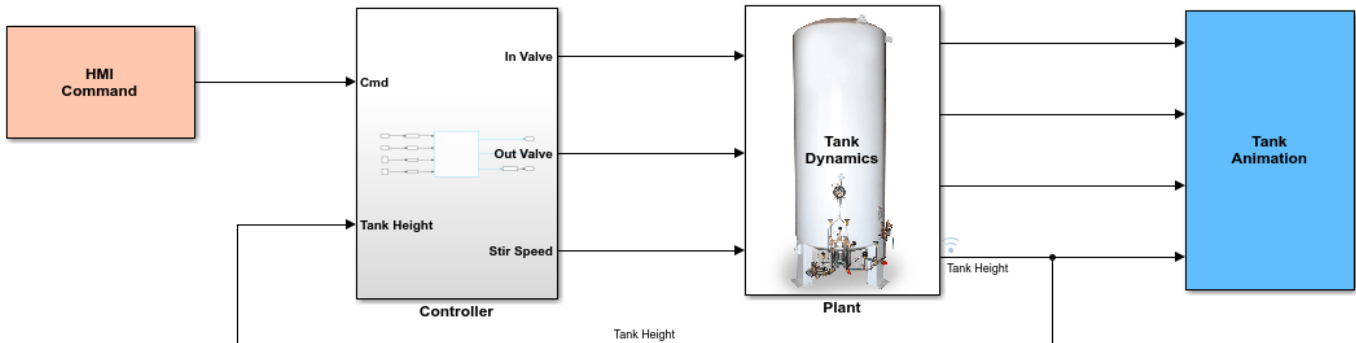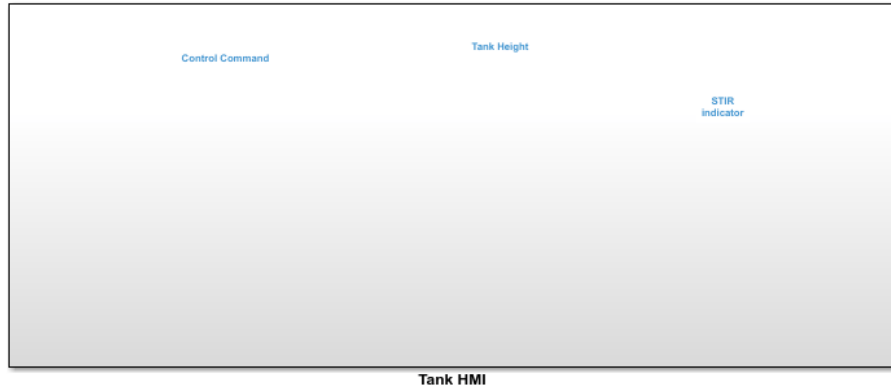
This model is the completed model for the ladder tank controller example. The controller is auto-generated from the ladder file TankControl.L5X by using the plcimportladder command.

To start the simulation, click **Run.** Use the Tank **HMI Control Command** rotary switch to set the controller command input.

The Tank Model UI displays the tank animation and status. The states of the ladder rungs and the data values are displayed in the ladder block window.

Copyright 2018-2019 The MathWorks, Inc.

6. To generate code for the subsystem, use plcgeneratecode. for more information, see `plcgeneratecode`:

```
generatedfiles = plcgeneratecode('plcdemo_ladder_tankcontrol/Controller')
```

7. To generate a testbench, open the ladder tank control testbench model:

```
open_system('plcdemo_ladder_tankcontrol_tb');
```
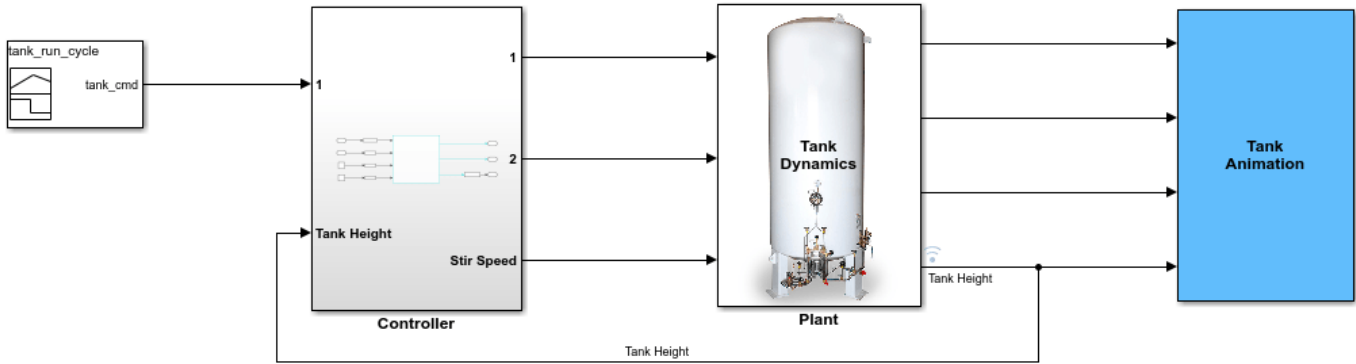
This model shows the ladder code generation from the ladder tank controller example.

To generate ladder code, select the Controller/TankControl_runner block and right-click **PLC Coder > Generate Code for Subsystem**.

To generate the testbench, in the PLC Configuration Parameters dialog box, select the **Generate testbench for subsystem** option, and then generate code.

Simulate, Model, and Generate Code for Timer-Based Ladder Logic

Model and simulate ladder logic for a simple timer-based motor controller. The ladder logic based controller uses an on-delay timer to delay the start of the motor and an off-delay timer to delay the stopping of the motor. After verifying that the controller and timers function for your requirements generate code for the controller.

Model Description

The model consists of a `Motor Controller` block that implements the ladder logic used to control the motor. The model contains a Human Machine Interface (`HMI`) block that enables you to interact with the model. Open the model:

```
open_system('plcdemo_ladder_timers');
```

HMI Block

The `HMI` block consists of inputs and outputs that you use to interact with the model simulation. Open the `HMI` block by double-clicking the block. This image shows the components of the HMI block.
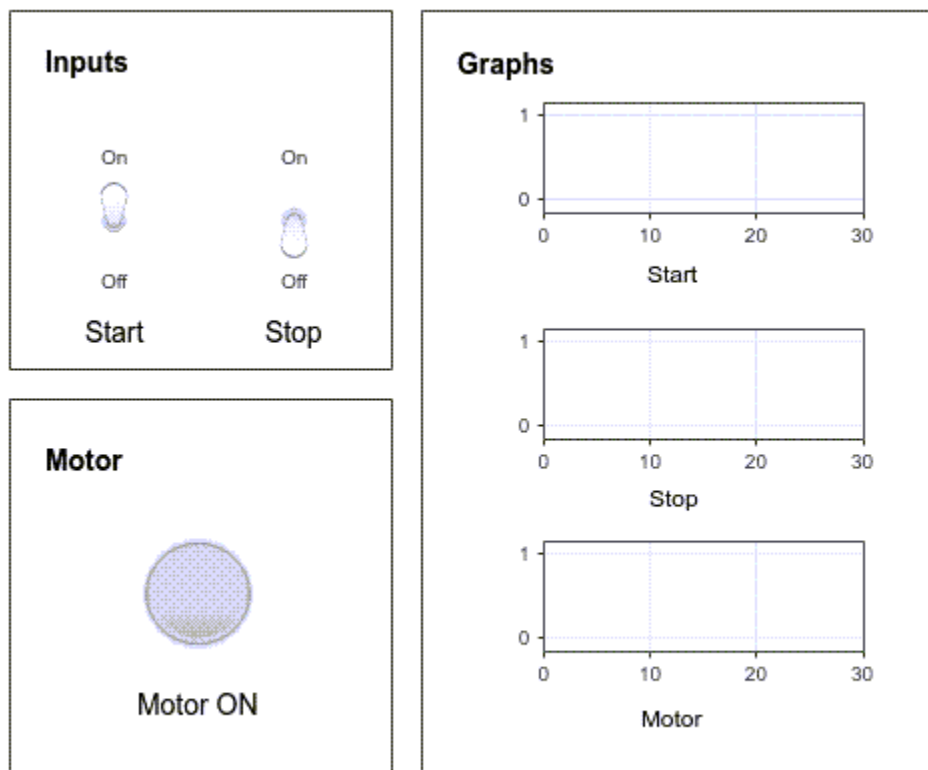
This HMI block contains:

- **Inputs:** You use the `Start` and `Stop` toggle switches to change the value of the respective inputs. When the toggle switch is in the `On` position, the value of the corresponding input is 1.

25-51

- **Motor:** Indicates motor status. The green-colored `Motor ON` indicator means that the motor is running. The gray-colored `Motor ON` indicator means that the motor is stopped.
- **Graphs:** Displays the status of `Start`, `Stop,` and `Motor` against time as the model simulation progresses.

Motor Controller

The `Motor Controller` block is a PLC Controller block. It contains a Ladder Program block that houses the ladder logic. To view the ladder logic for the controller, open the `Motor Controller` block, and then open the `Ladder Diagram Program` block.

Using Timers in Ladder Logic

This image shows the ladder logic implementation of the timer-based motor controller inside the `Ladder Diagram Program` block.
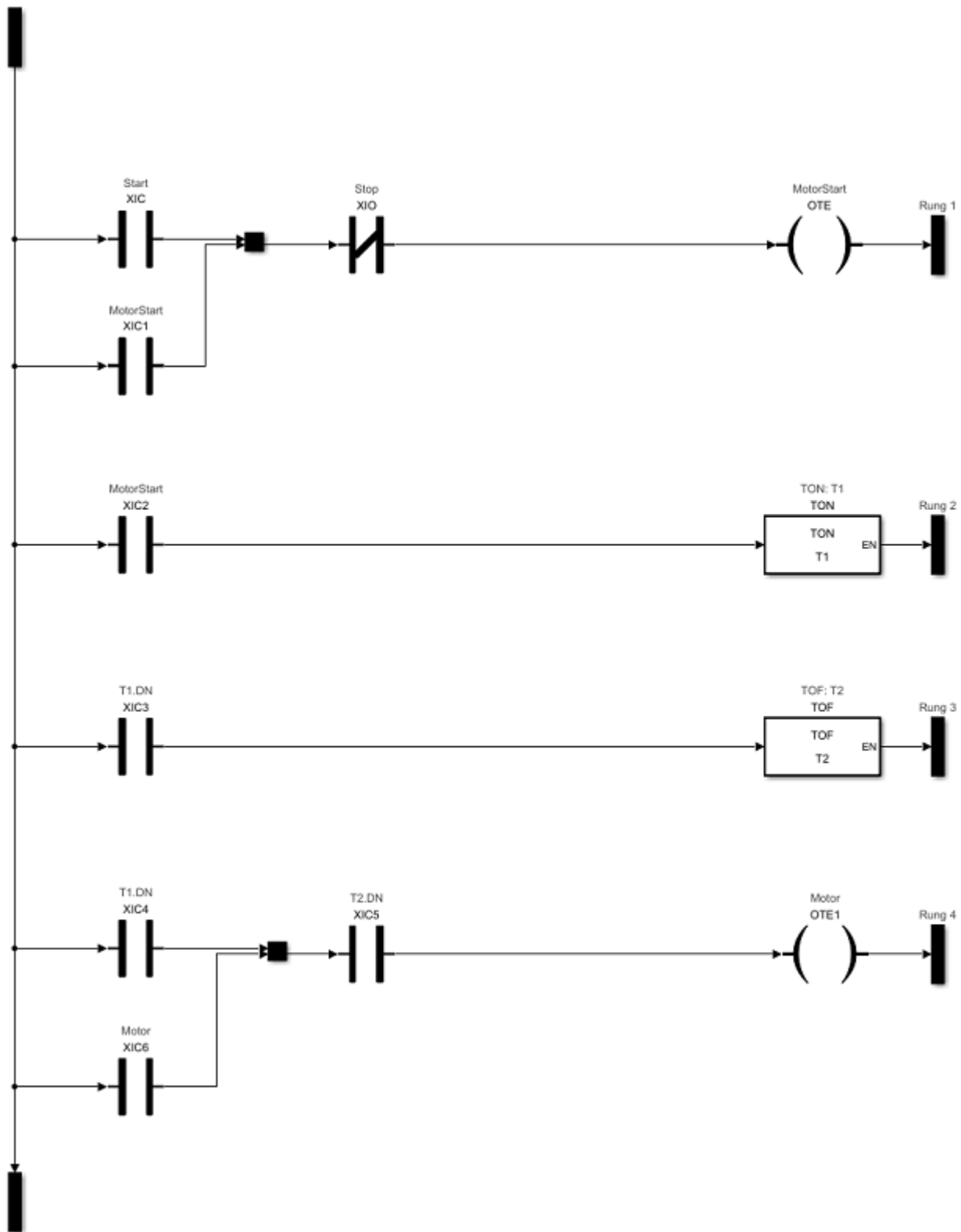
This ladder logic has a `TON` timer named `T1` that is responsible for the delay when starting the motor. The ladder logic has a `TOF` timer named `T2` that is responsible for the delay when stopping the motor. `T1` has a preset of 5 seconds and `T2` has a preset of 10 seconds.

When the `Start` input is toggled to `1`, the `MotorStart` output in the first rung is activated which starts the timer `T1` counting operation. The `T1.DN` bit is set when `T1` finishes counting to 5 seconds. The third rung with timer `T2` is then activated. Because T2 is a `TOF` timer, the `T2.DN` bit is set. The timer starts the counting operation only when this rung becomes false. Both the inputs to the lowermost rung are true and the `Motor` output is activated.

When the `Stop` input is toggled to `1`, the `MotorStart` coil is deactivated and the `T1.DN` bit is reset. The timer T2 starts counting. Once T2 finishes counting to 10 seconds, the `T2.DN` bit is reset and the `Motor` output is deactivated.

Configure Timer Delay Values

You can configure the timer delay value for T1 and T2 by specifying their delay values inside the `InitFcn` callback property for the model. To access this setting, click **Modeling > Model Settings > Model Properties > Callbacks >InitFcn.**
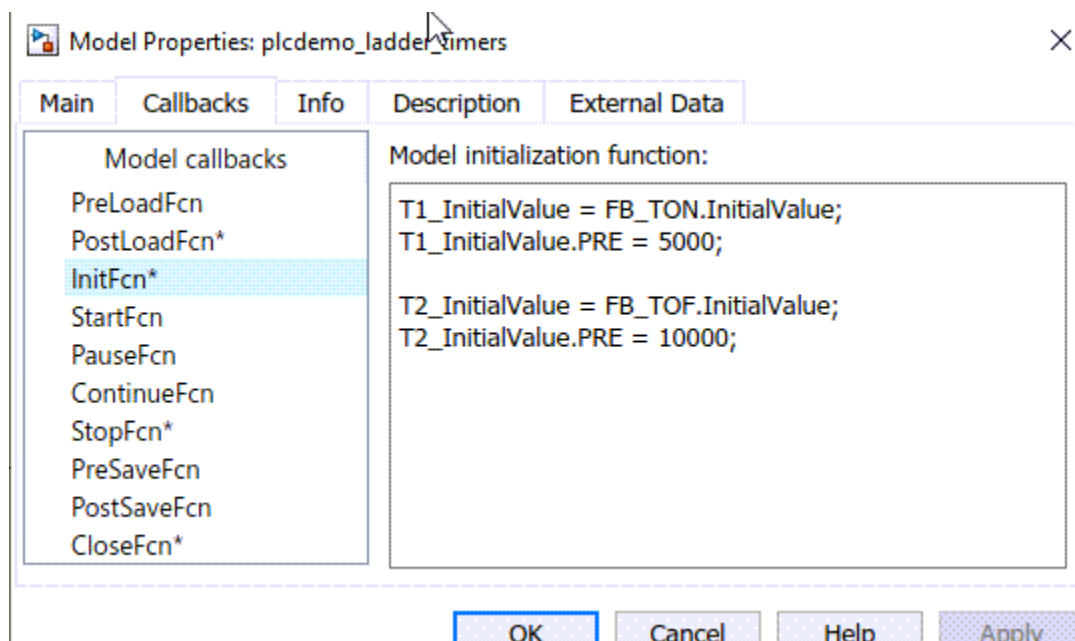
To specify a preset delay, enter the value in milliseconds into the `TimerName_InitialValue.PRE`. For example, `T1_InitialValue.PRE` is set to 5000, which means a time delay of 5 seconds and `T2_InitialValue.PRE` is set to 10,000, which means a time delay of 10 seconds. You can simulate different start and stop delays by changing these timer presets to different values.

Simulate Model and Generate Code

Simulate the model and observe the graphs in the `HMI` block. Turn on the `Start` button in the `HMI`. The `Motor ON` light turns green after five seconds.
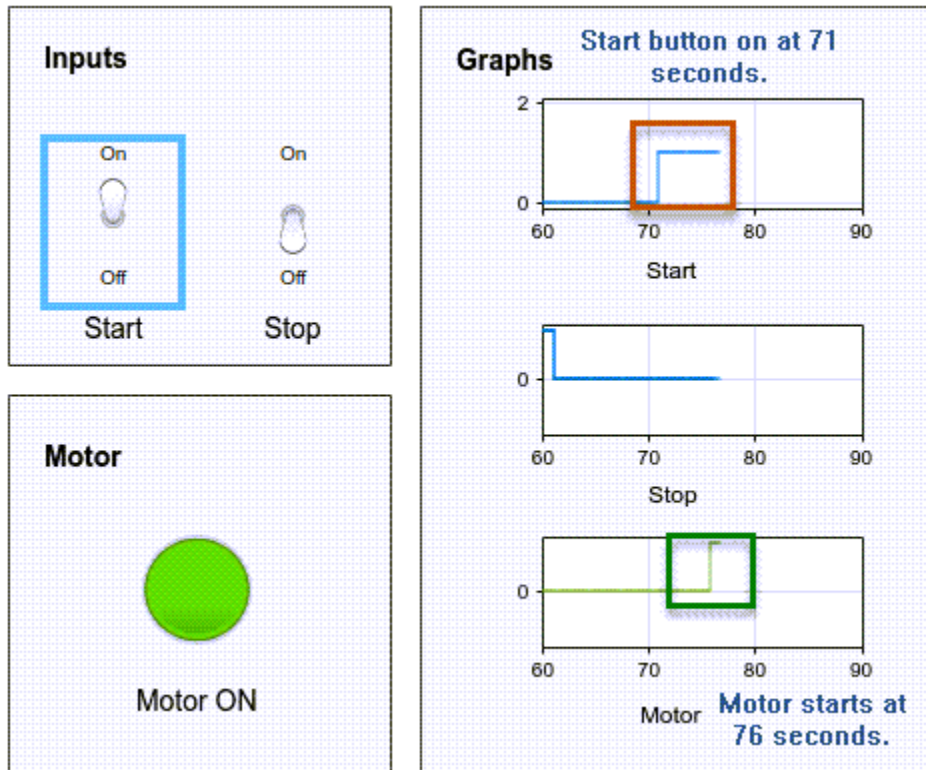
Turn on the Stop button. The Motor ON light turns gray after 10 seconds.
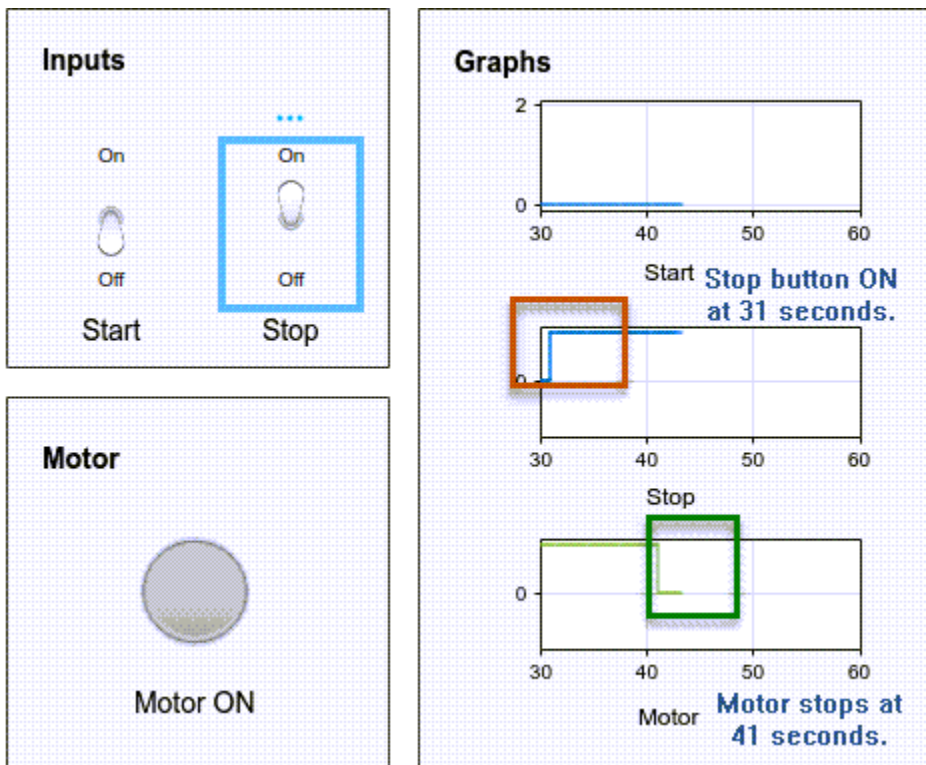
After verifying that the timers function for your requirements, generate code for the `Motor Controller` block

```
plcgeneratecode('plcdemo_ladder_timers/Motor Controller')
```

```
### Generating PLC code for 'plcdemo_ladder_timers/Motor Controller'.
### Using model settings from 'plcdemo_ladder_timers' for PLC code generation parameters.
### Begin code generation for IDE studio5000.
### Emit PLC code to file.
### PLC ladder code generation successful for 'plcdemo_ladder_timers/Motor Controller'.

### Generated ladder files:
plcsrc\plcdemo_ladder_timers_gen.L5X
```

Alternatively generate code by selecting the `Motor Controller` block, and then select **APPS > PLC Coder.** On the **PLC Code** tab, click **Generate PLC Code**.

See Also

- TOF
- TON
- `plcgeneratecode`
- Simulink PLC Coder
- "Model and Simulate Ladder Diagrams in Simulink" on page 3-8

Model, Simulate, and Generate Code for a Ladder Logic-Based Temperature Controller

Model and simulate a ladder logic-based temperature controller of a home. The ladder logic controls the heater based on the thermostat settings and the outside ambient temperature. After verifying that the controller functions for your requirements, generate code for your controller.

Model Description

The model consists of a `House` block that models the thermal characteristics of the house and the heating system. The `Temperature Controller` block implements the ladder logic that controls the heater. The `Human Machine Interface (HMI)` block enables you to interact with the model.

Open the model:

`open_system('plcdemo_ladder_househeat_complete');`

Initialize Model

This model calculates heating costs for a generic house. Opening the model loads the information about the house from the plcdemo_ladder_househeat_data.m file. The file:

- Defines the house geometry (size, number of windows)
- Specifies the thermal properties of house materials
- Calculates the thermal resistance of the house
- Provides the heater characteristics (temperature of the hot air, flow rate)
- Defines the cost of electricity (0.09$/kWhr)
- Specifies the initial room temperature (20°C = 68° F)

Note: Time is given in units of hours. Certain quantities, like air flow rate, are expressed per hour (not per second).

Model Components

Set Point

`Set Point` specifies the temperature that must be maintained indoors. It is 70° Fahrenheit by default. Temperatures are given in °Fahrenheit. The model converts the temperature to °Celsius.

Range

`Range` is a constant block. This specifies the range around the set point for the room temperature to fluctuate. It is 5° Fahrenheit by default. The room temperature varies between *Set_L* and *Set_H* where:

   Set_L = SetPoint − Range

   Set_H = SetPoint + Range

Model the Environment

The model uses a heat sink with infinite heat capacity and time varying temperature `Tout` to simulate the environment. The constant block `Avg. Outdoor Temp` specifies the average air temperature

outdoors. The `Daily Temp Variation` Sine Wave block generates daily outdoor temperature fluctuations. You can vary these parameters to see how they affect the heating costs.

HMI Block

The `HMI` block consists of inputs and outputs that you use to interact with the model simulation. Open the `HMI` block by double-clicking the block. This image shows the components of the HMI block.
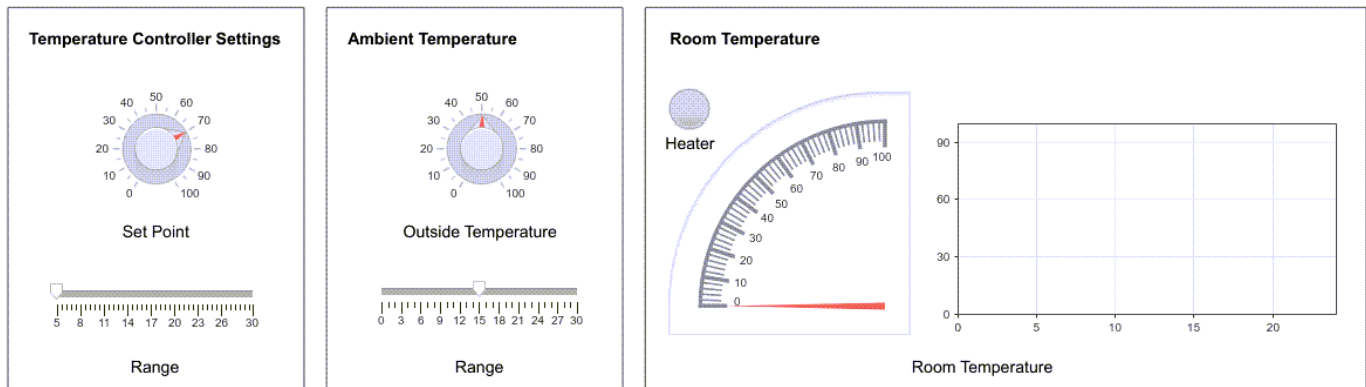
This HMI block contains:

*   **Temperature Controller Settings**: Use the `Set Point` and `Range` to specify the inputs to the temperature controller. The values are specified in ° Fahrenheit.
*   **Ambient Temperature:** Use the `Outside Temperature` to set `Avg.Outdoor Temp`. Use the `Range` slider to set the amplitude of the `Daily Temp Variation` Sine Wave block. The values are specified in ° Fahrenheit.
*   **Room Temperature:** The graph and dial indicate the room temperature in ° Fahrenheit. The red-colored `Heater` indicator means that the heater is on. The gray-colored `Heater` indicator means that the heater is off.

Temperature Controller

The `Temperature Controller` block is an Add-On Instruction (AOI) runner block. The AOI runner block contains a ladder logic implementation of the heater controls. To view the ladder logic open:

*   The `Temperature Controller` block,
*   The `Temperature Controller Runner` block
*   The `Function Block`

Then, select `Logic Routine`.

This image shows the ladder logic-based temperature controller inside the `Logic Routine` block.
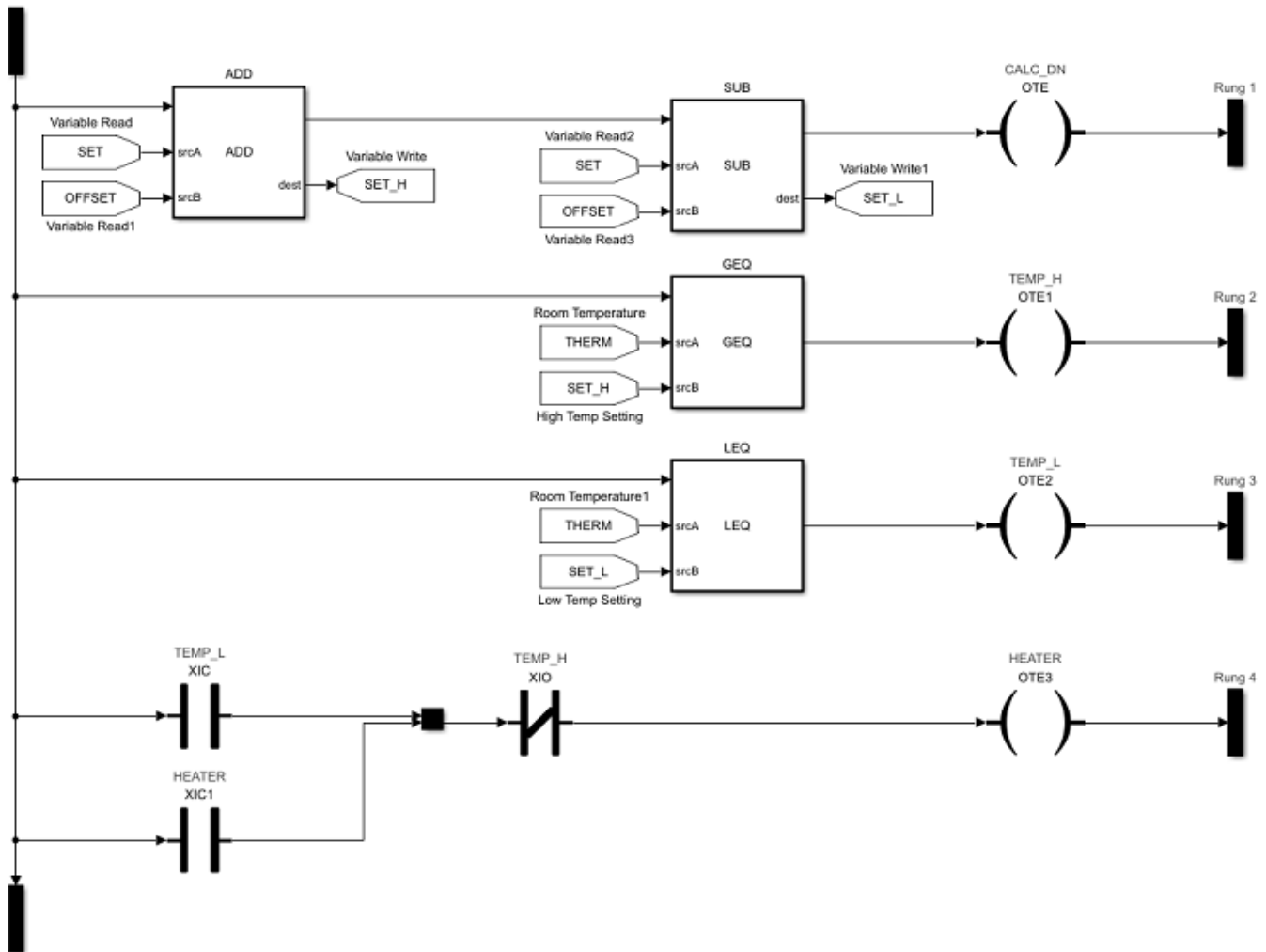
The first rung calculates SET_H and SET_L. On the second rung, when the room temperature is greater than or equal to SET_H, OTE1 is on. On the third rung when the room temperature is lesser than or equal to SET_L, OTE2 is on. When OTE2 is on and OTE1 is off, the heater is on.

Simulate Model and Generate Code

Simulate the model and observe the graphs in the HMI block. Vary the Set Point, Range, Avg.Outdoor Temp to see how the heater turns on and off. In this image, the set point is 70° Fahrenheit, the range is set to 5° Fahrenheit, Ambient temperature is set to 50° Fahrenheit, and the amplitude of the disturbance is set to 15° Fahrenheit.
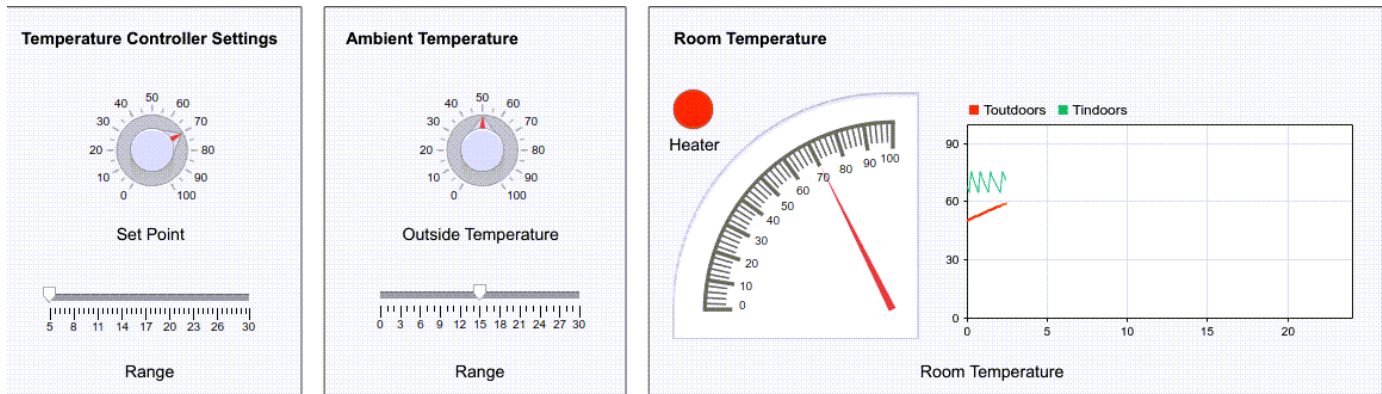
After verifying that the controller functions for your requirements, generate code for the `Temperature Controller` block.

```
plcgeneratecode('plcdemo_ladder_househeat_complete/Temperature Controller/Temperature Controller
```

```
### Generating PLC code for 'plcdemo_ladder_househeat_complete/Temperature Controller/Temperature
### Using model settings from 'plcdemo_ladder_househeat_complete' for PLC code generation parame
### Begin code generation for IDE studio5000.
### Emit PLC code to file.
### PLC ladder code generation successful for 'plcdemo_ladder_househeat_complete/Temperature Con

### Generated ladder files:
plcsrc\plcdemo_ladder_househeat_complete_gen.L5X
```

Alternatively generate code by selecting the `Temperature Controller` block, `Temperature Controller Runner` block, and then select **APPS > PLC Coder**. On the **PLC Code** tab, click **Generate PLC Code**.

To generate a test bench, select the `Temperature Controller Runner` block. On the **PLC Code** tab, click **Settings > PLC Code Generation > Generate testbench for subsystem.** Click **Generate PLC Code.**

Generate Simulink Design Verifier™ Test Cases

Execute this command:

```
plcladderoptions(gcs,'FastSim','on');
```

Open the `Temperature Controller` block, right-click **AOI Runner** block, and select `Design Verifier > Generate test case for subsystem`.

See Also

- AOI Runner
- "Thermal Model of a House"
- "Model and Simulate Ladder Diagrams in Simulink" on page 3-8

Model, Simulate, and Generate Code for Ladder Logic-Based Elevator Controller

Rapidly prototype the ladder logic-based controller for a single car elevator by using Simulink® PLC Coder™ to model and simulate the ladder logic. After verifying that the controller works for your requirements, generate code for the controller.

Model Description

The model consists of a `Elevator Control System` block, `Elevator Block`, and a Human Machine Interface (`HMI`) block. Open the model:

```
open_system('plcdemo_ladder_elevator');
```
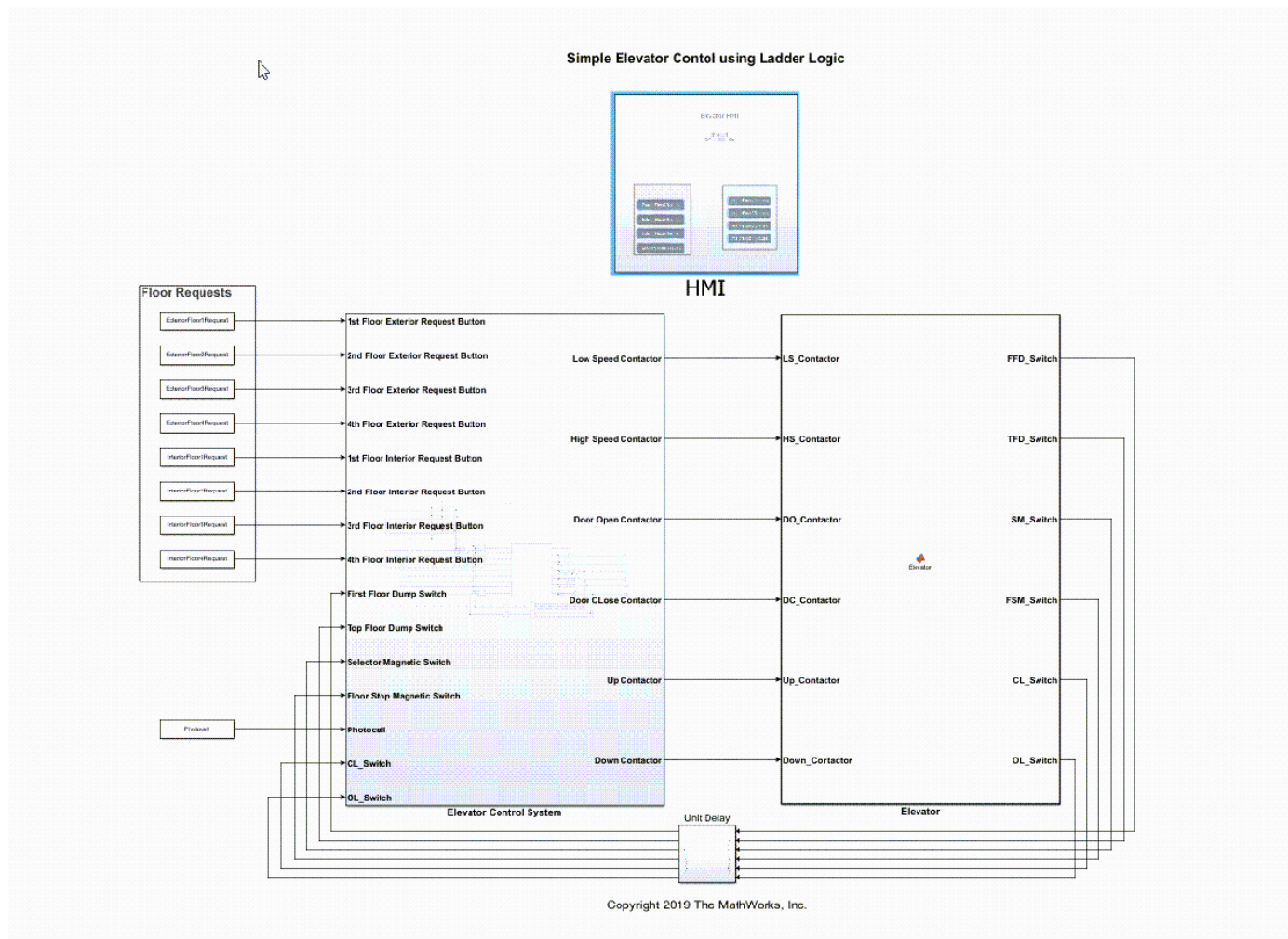

Opening the model loads the timer configuration, model data types, and initializes the model by using information from the `plcdemo_ladder_elevator_init.m` file.

```
type plcdemo_ladder_elevator_init
```

```
% sample time
Ts = 0.1;

% load PLC builtin types
plcloadtypes;

% define timer initial values
T37_InitialValue = FB_TON.InitialValue;
T37_InitialValue.PRE = int32(300*100);

T38_InitialValue = FB_TOF.InitialValue;
T38_InitialValue.PRE = int32(100*100);

T39_InitialValue = FB_TON.InitialValue;
T39_InitialValue.PRE = int32(12*100);

T40_InitialValue = FB_TON.InitialValue;
T40_InitialValue.PRE = int32(10*100);

T41_InitialValue = FB_TOF.InitialValue;
T41_InitialValue.PRE = int32(1*100);

% define dashboard parameters
InteriorFloor1Request = false;
InteriorFloor2Request = false;
InteriorFloor3Request = false;
InteriorFloor4Request = false;

ExteriorFloor1Request = false;
ExteriorFloor2Request = false;
ExteriorFloor3Request = false;
ExteriorFloor4Request = false;

Photocell  = false;
```

Elevator Control System Block

The `Elevator Control System` block is a PLC Controller block. It contains a Ladder Program block that contains the ladder logic. To view the ladder logic for the controller, open:

- The `Elevator Control System` block
- The `Elevator PLC Ladder Diagram System` block
- The `Elevator Controller` block
- The `Task` block
- The `Program` block

This image shows a section of the ladder diagram logic inside the `Program` block.
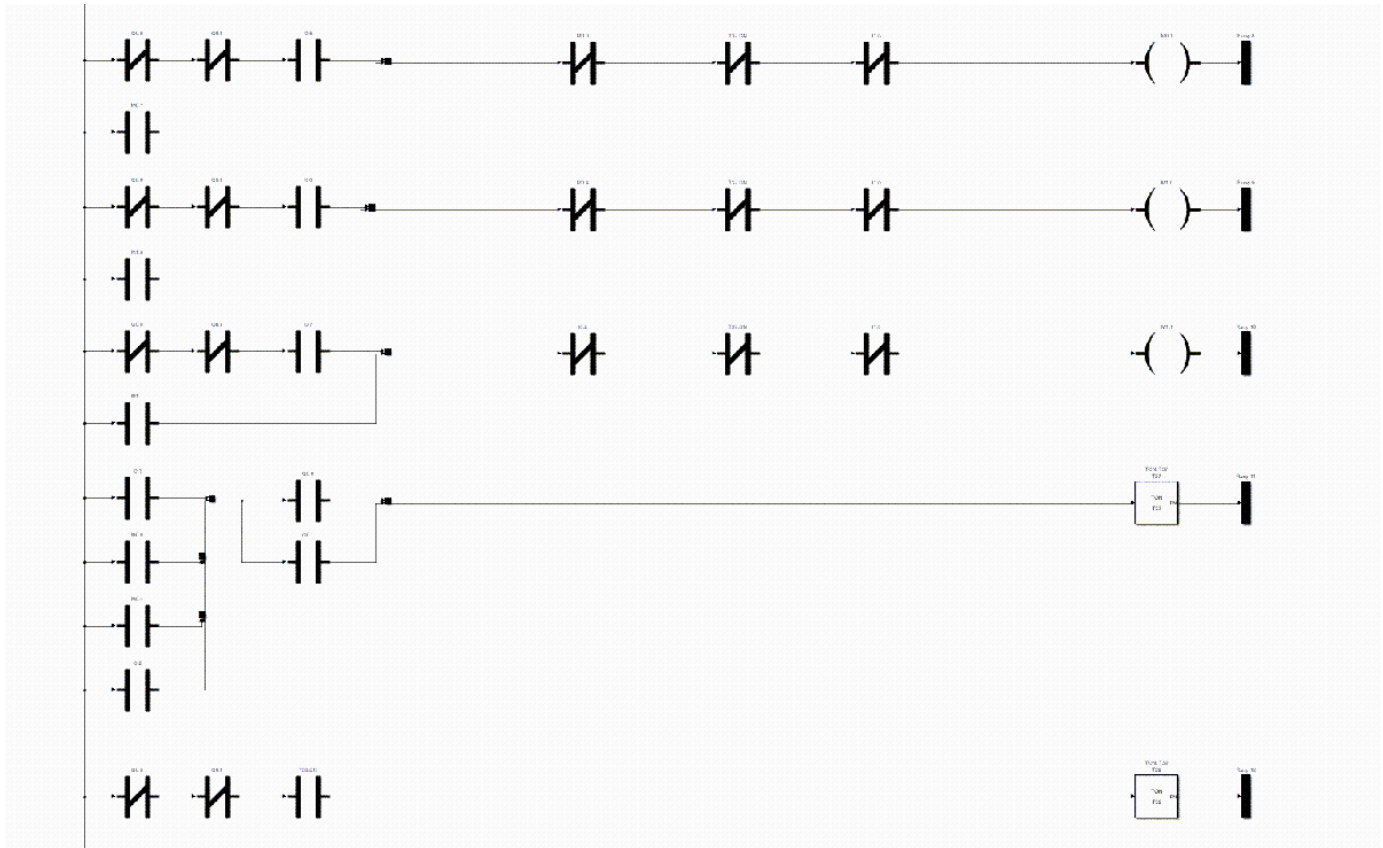
HMI Block

The `HMI Block` enables you to interact with the model simulation. Open the `HMI` block by double-clicking the block. This image shows the components of the `HMI` block.

Photocell

Off ⬤ On
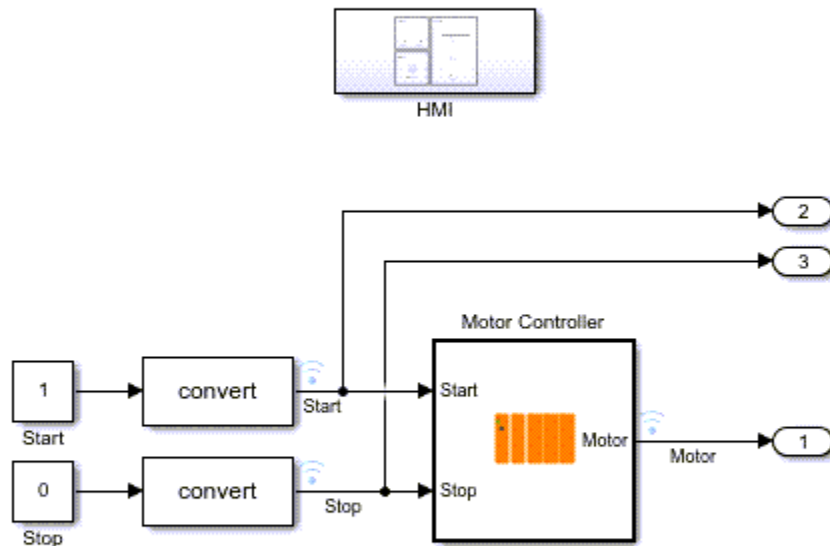
Exterior Buttons

ExteriorFloor4Request

ExteriorFloor3Request

ExteriorFloor2Request

ExteriorFloor1Request

Interior Buttons

InteriorFloor4Request

InteriorFloor3Request

InteriorFloor2Request

InteriorFloor1Request
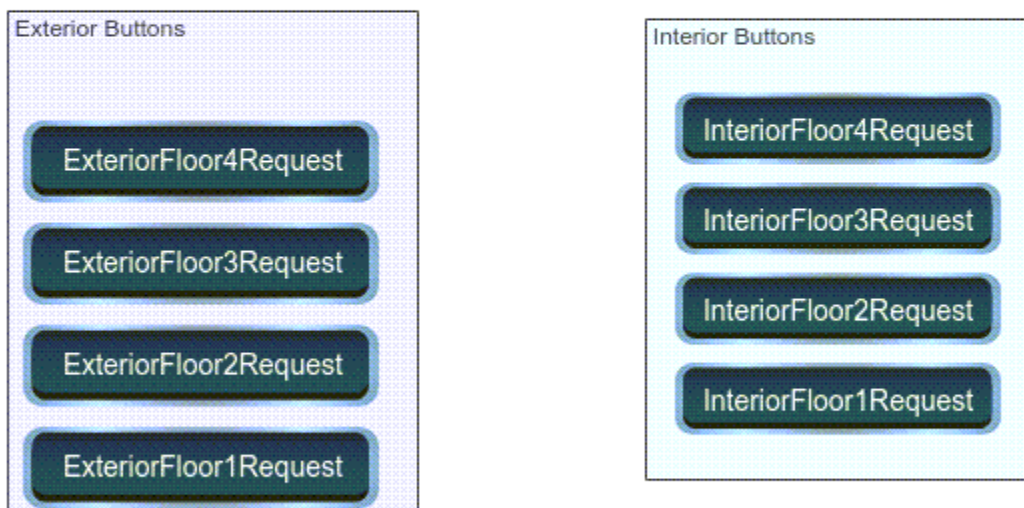
The HMI block contains:

- **Exterior Buttons:** Represents the exterior floor request buttons.
- **Interior Buttons:** Represents the interior floor request buttons.
- **Photocell:** Represents the door sensor.

Elevator Block

The Elevator block is a MATLAB Function Block that contains a mathematical model representing the elevator system. The calculations from the Elevator block are used as inputs to the graphics function to simulate the elevator movement.

Simulate Model and Generate Code

Simulate the model and observe the elevator animation. In this image, the second floor external request button was pressed.
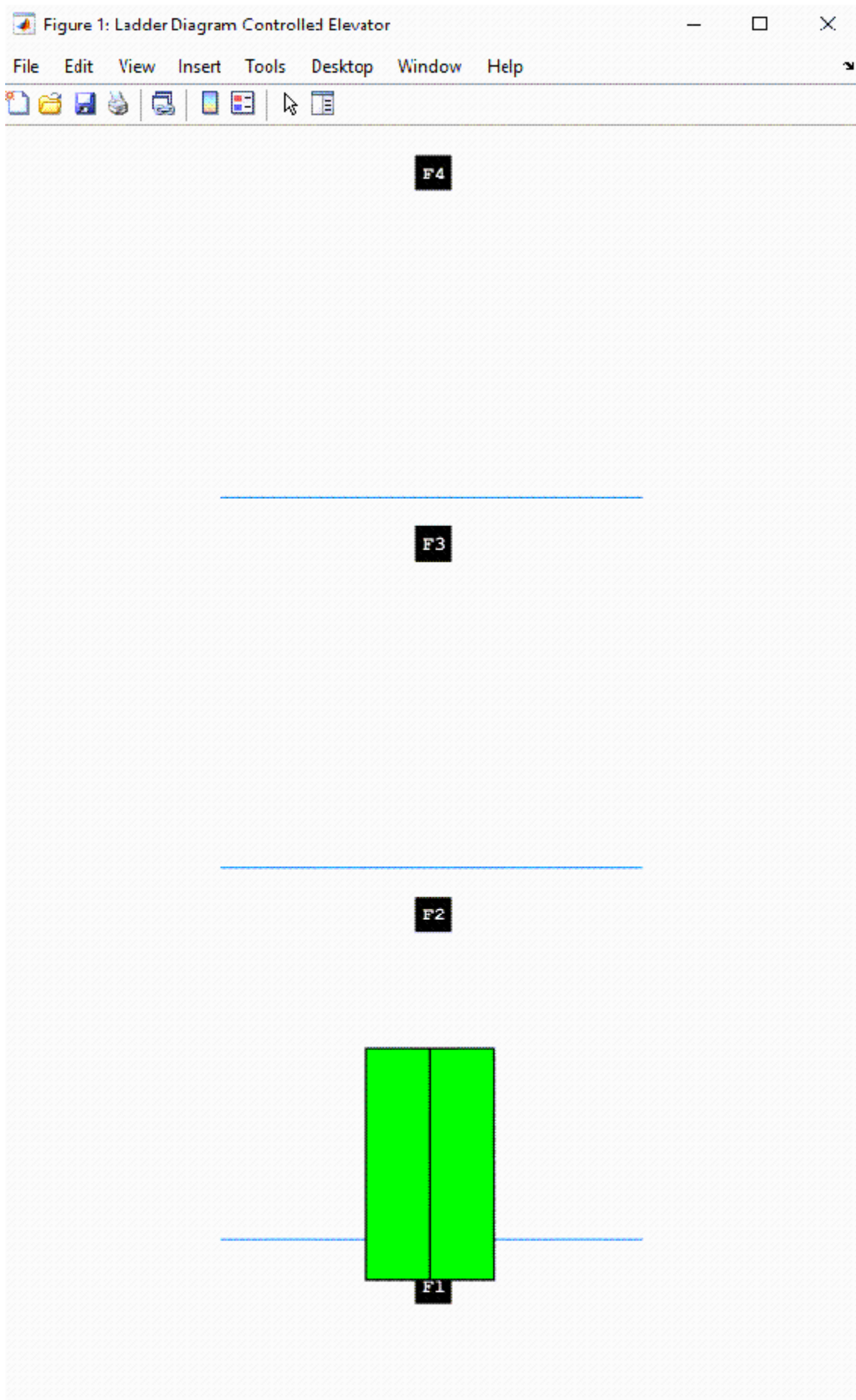
After verifying that the controller functions for your requirements, generate code for the `Elevator Controller` block

`plcgeneratecode('``plcdemo_ladder_elevator/Elevator Control System/Elevator PLC Ladder Diagram Sys`

Alternatively, generate code by selecting the `Elevator Controller` block, and then selecting **APPS > PLC Coder**. On the **PLC Code** tab, click **Generate PLC Code.**

See Also

- TON
- TOF
- `plcgeneratecode`
- "Model and Simulate Ladder Diagrams in Simulink" on page 3-8
- "Simulate, Model, and Generate Code for Timer-Based Ladder Logic" on page 25-51

Structured Text Code Generation for Simulink Data Dictionary

This example shows how to autogenerate structured text code for a model with a Simulink® Data Dictionary Component.

Prerequisites

Copy **plc_sldd_ex.slx** and **plc_sldd_ex.sldd** to the same folder in your current working directory (CWD)

Open the model

```
open_system('plc_sldd_ex')
```
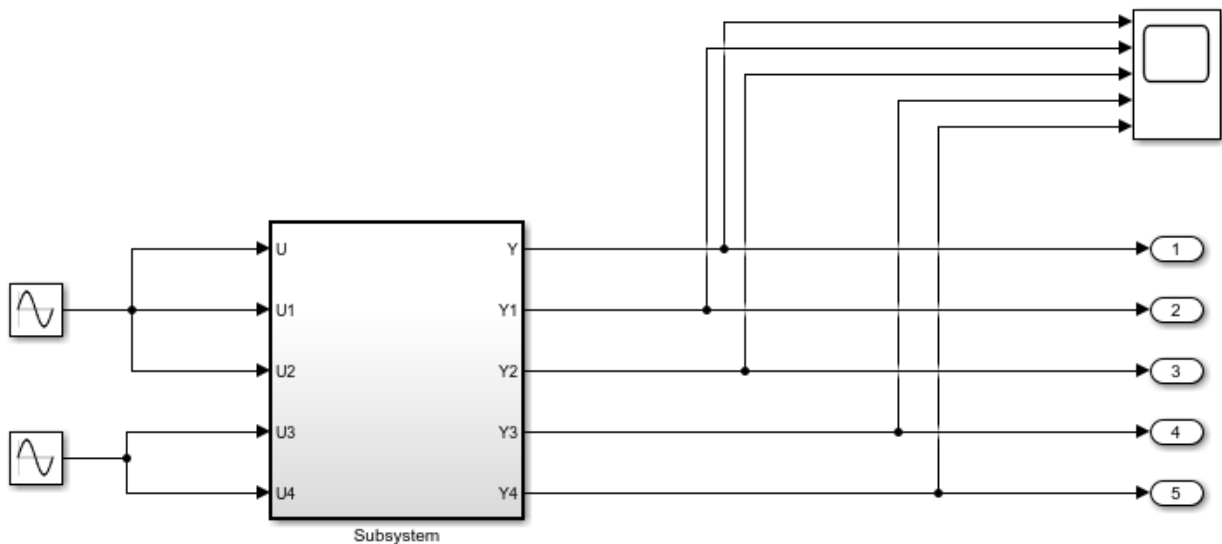

This model shows the use of Simulink Data Dictionary to specify model parameter and signal data and generate PLC code. The parameters K1, K2, K3, and signals dsm1, dsm2 are defined in data dictionary file plc_sldd_ex.sldd.
To generate PLC code, open PLC Coder App. Select the Subsystem block and click the Generate PLC Code button.

The Diagnostic Viewer displays hyperlinks to the generated code files, click the links to view the generated files.

Copyright 2009-2021 The MathWorks, Inc.

Structured Text Code Generation for Subsystem Reference Blocks

This example shows how to autogenerate structured text code for subsystem reference blocks.

Open Simulink Model

To open the Simulink test bench model, use the following command.

```
open_system('mSubSysRefSystemIntegration');
```


Note: Before code generation, copy SSRefSubSystem1, SSRefSubSystem2, SSRefSubSystem3, refSubSystem1, refSubsystem2, refSubSystem3 and mSubSysRefSystemIntegration SLX files to the same folder location in your current working directory (CWD).

Generate Code for the Subsystem

To generate code for the subsystem use `plcgeneratecode`

```
generatedfiles = plcgeneratecode('mSubSysRefSystemIntegration/TopSystem');

### Generating PLC code for 'mSubSysRefSystemIntegration/TopSystem'.
### Using model settings from 'mSubSysRefSystemIntegration' for PLC code generation parameters.
### Begin code generation for IDE codesys23.
### Emit PLC code to file.
### Creating PLC code generation report mSubSysRefSystemIntegration_codegen_rpt.html.
### PLC code generation successful for 'mSubSysRefSystemIntegration/TopSystem'.
### Generated files:
plcsrc\mSubSysRefSystemIntegration.exp
```

PLC_RemoveSSStep for Distributed Code Generation

Generate structured text code for different components of your model.

Open model

Open the model by using the following command:

```
open_system('mSystemIntegration');
```

Note: Before code generation, copy mSubSystem1, mSubSystem2, mSubSystem3, and mSystemIntegration SLX files to the same folder location in your current working directory (CWD).

Configure Model Components for Distributed Code Generation

To autogenerate structured text code with the same `ssMethod` type for every component of your model for external code integration later on, use Keep Top-Level ssMethod Name the Same as the Non-Top Level Name. For more information, see "Keep Top-Level ssmethod Name the Same as the Non-Top Level Name" on page 13-36 function.

Mark Externally Defined Variables

1   Open the Simulink PLC Coder app. For more information, see Simulink PLC Coder.

2   Select the `TopSystem` block.

3   Click **Settings**. Navigate to **PLC Code Generation** > **Identifiers**. In the `Identifier Names` box enter `Subsystem1,Subsystem2,Subsystem3`.

4   Click **OK**.

Code Generation

1   Open the Simulink PLC Coder app. For more information, see Simulink PLC Coder.

2   Select the `Subsystem1` block.

3   Click **Settings**. Navigate to **PLC Code Generation** > **Identifiers**. Select the **Keep top level ssMethod name same as non-top level** check box.

4   Click **OK**.

5   Repeat steps 2 through 4 for `SubSystem2`, `SubSystem3`, and `TopSystem`.

Generate Code for the Subsystem

To generate code for the individual subsystem use the plcgenerate code function:

plcgeneratecode('mSystemIntegration/TopSystem/SubSystem1');

plcgeneratecode('mSystemIntegration/TopSystem/SubSystem2');

plcgeneratecode('mSystemIntegration/TopSystem/SubSystem3');

Generate Code for the Integrated Model

To generate code for the integrated model:

plcgeneratecode('mSystemIntegration/TopSystem');

See Also

More About

*   "Generated Code Structure for PLC_RemoveSSStep" on page 24-3
*   "Distributed Model Code Generation Options" on page 24-2

Structured Text Code Generation for Enum To Integer Conversion

Autogenerate structured text code for enum to integer conversion model.

Load enum class

For this example, the myEnum.m script loads the enum class definition. Place this script file in the same project folder as the plc_enum_to_int model file.

Open the model

```
open_system('plc_enum_to_int.slx')
```


This model shows PLC code generation for enum to integer type conversion.
To generate PLC code, open PLC Coder App. Select Settings->PLC Code Generation->General options->Target IDE and choose Target IDE that supports enum type. Select Settings->PLC Code Generation->Identifiers->Generate enum cast function. Click the Subsystem block and click the Generate PLC Code button.

The Diagnostic Viewer displays hyperlinks to the generated code files, click the links to view the generated files.

Copyright 2009-2020 The MathWorks, Inc.

Structured Text Code Generation for Integer To Enum Conversion

This example shows how to autogenerate structured text code for integer to enum conversion model.

Load enum class

For this example, the `myColor.m` script loads the enum class definition. Place this script file in the same project folder as the `plc_int_to_enum` model file.

Open the model

```
open_system('plc_int_to_enum.slx')
```
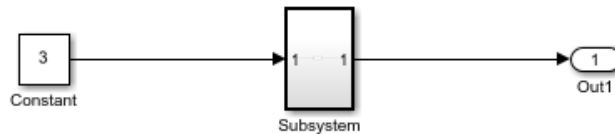

This model shows PLC code generation for integer to enum type conversion.
To generate PLC code, open PLC Coder App. Select Settings->PLC Code Generation->General options->Target IDE and choose Target IDE that supports enum type. Select Settings->PLC Code Generation->Identifiers->Generate enum cast function. Click the Subsystem block and click the Generate PLC Code button.

The Diagnostic Viewer displays hyperlinks to the generated code files, click the links to view the generated files.

Copyright 2009-2020 The MathWorks, Inc.

Prevent External Variable Initialization for Distributed Code Generation

Generate structured text code for different components of your model.

Open model

Open the model by using the following command:

```
open_system('External_Var_Distributed_Codegen');
```
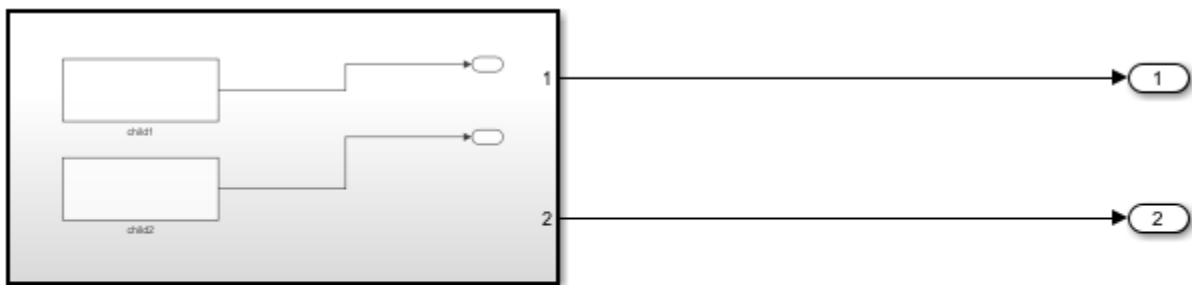

Copyright 2020 The MathWorks, Inc.

Configure Model Components for Distributed Code Generation

To autogenerate structured text code by preventing initialization statements for externally defined variables for external code integration later on, use remove Initialization Statements for Externally Defined State Variables. for more information, see "Remove Initialization Statements for Externally Defined State Variables" on page 13-38.

Mark Externally Defined Variables

1   Open the Simulink PLC Coder app. For more information, see Simulink PLC Coder.
2   Select the `Subsystem` block.
3   Click **Settings**. Navigate to **PLC Code Generation > Identifiers**. In the `Identifier Names` box enter `child1,child2,DSExportedGlobal`.
4   Click **OK**.

Code Generation

1   Open the Simulink PLC Coder app. For more information, see Simulink PLC Coder.
2   Select the `Subsystem` block.
3   Click **Settings**. Navigate to **PLC Code Generation > Interface**. Select the **Remove Initialization Statements for externally defined state variables** check box.
4   Click **OK**.

Generate Code for the Subsystem

To generate code for the individual subsystem use the plcgenerate code function:

plcgeneratecode('External_Var_Distributed_Codegen/Subsystem');

Related Topics

"Generated Code Structure for PLC_PreventExternalVarInitialization" on page 24-5.

See Also

More About

- "Generated Code Structure for PLC_PreventExternalVarInitialization" on page 24-5
- "Distributed Model Code Generation Options" on page 24-2

Simulation and Structured Text Generation for MPC Controller Block

This example shows how to simulate and generate structured text for an MPC Controller block by using Simulink® PLC Coder™ software. The generated code uses single-precision data.

Required Products

To run this example, Model Predictive Control Toolbox™, Simulink®, and Simulink PLC Coder™ are required.

```
if ~mpcchecktoolboxinstalled('simulink')
    disp('Simulink is required to run this example.')
    return
end
if ~mpcchecktoolboxinstalled('plccoder')
    disp('Simulink PLC Coder is required to run this example.');
    return
end
if ~mpcchecktoolboxinstalled('mpc')
    disp('MPC Toolbox is required to run this example.');
    return
end
```

Define Plant Model and MPC Controller

Define a SISO plant.

```
plant = ss(tf([3 1],[1 0.6 1]));
```

Define the MPC controller for the plant.

```
Ts = 0.1;    %Sample time
p = 10;      %Prediction horizon
m = 2;       %Control horizon
Weights = struct('MV',0,'MVRate',0.01,'OV',1); % Weights
MV = struct('Min',-Inf,'Max',Inf,'RateMin',-100,'RateMax',100); % Input constraints
OV = struct('Min',-2,'Max',2); % Output constraints
mpcobj = mpc(plant,Ts,p,m,Weights,MV,OV);
```

Simulate and Generate Structured Text

Open the Simulink model.

```
mdl = 'mpc_plcdemo';
open_system(mdl)
```
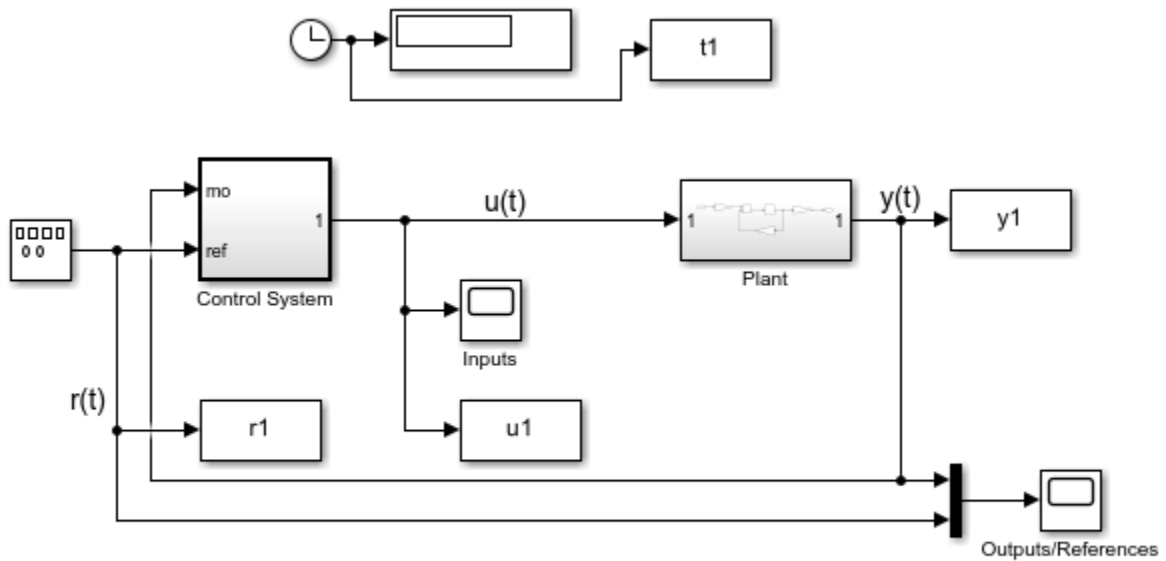
Copyright 1990-2021 The MathWorks, Inc.
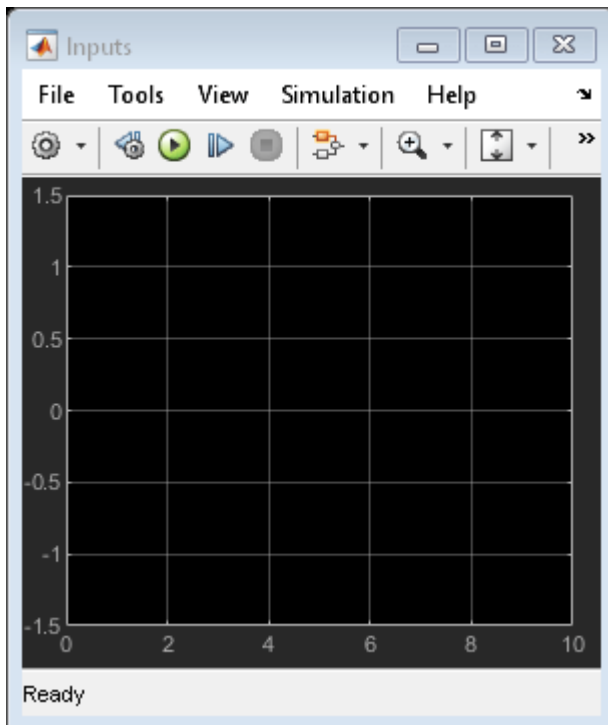
To generate structured text for the MPC Controller block, put the MPC block inside a subsystem block and treat the subsystem block as an atomic unit. Select the **Treat as atomic unit** property of the subsystem block.

Simulate the model in Simulink.

```
close_system([mdl '/Control System/MPC Controller'])
open_system([mdl '/Outputs//References'])
open_system([mdl '/Inputs'])
sim(mdl)
```

-->Converting model to discrete time.
-->Assuming output disturbance added to measured output channel #1 is integrated white noise.
-->The "Model.Noise" property is empty. Assuming white noise on each measured output.
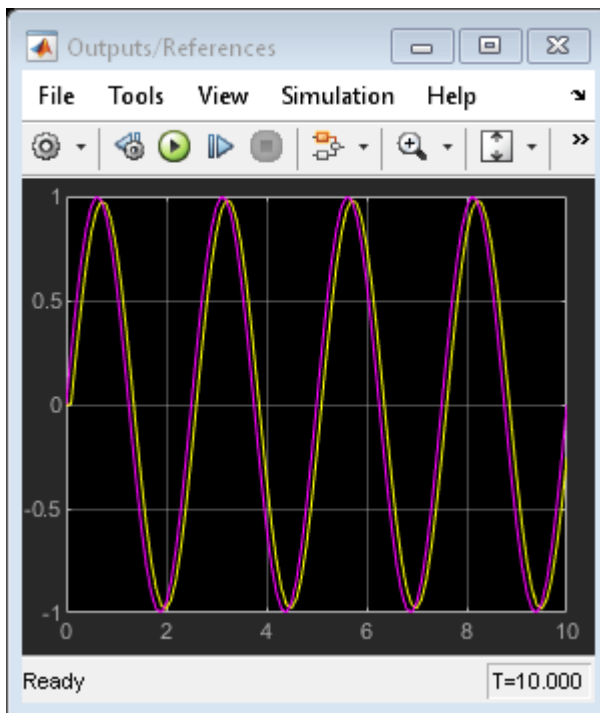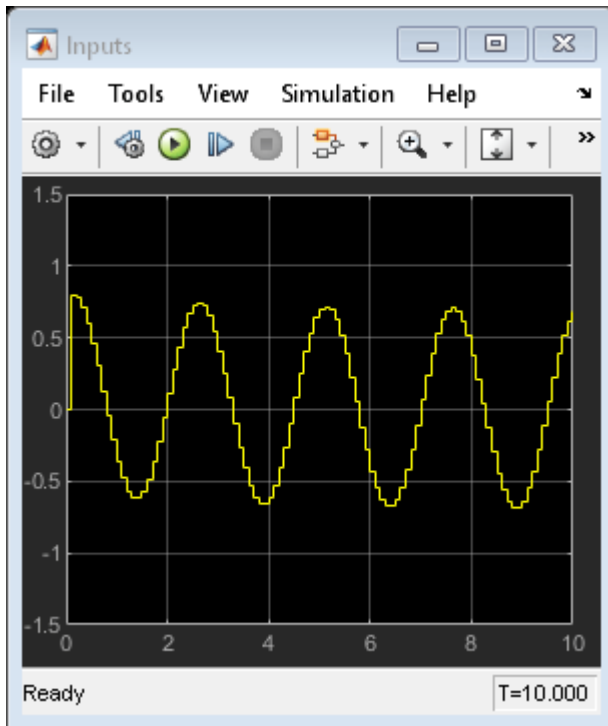
To generate code by using the Simulink PLC Coder, use the `plcgeneratecode` command.

```
disp('Generating PLC structure text... Please wait until it finishes.')
%
%    plcgeneratecode([mdl '/Control System']);
```

```
Generating PLC structure text... Please wait until it finishes.
```

The Message Viewer dialog box shows that PLC code generation was successful.

Close the Simulink model and return to the original directory.

```
bdclose(mdl)
```

View Requirement Links from Generated Code

View requirements linked to a model by using the Requirements Manager app and model object context menu.

Open Model

This model shows how to generate code for an airport conveyer belt controller. Open the Controller subsystem, then open the Stateflow chart inside it. This chart implements the control logic for starting and stopping the conveyer belt motor, depending on sensor inputs.

```
open_system('plcdemo_airport_conveyor_requirements');
```
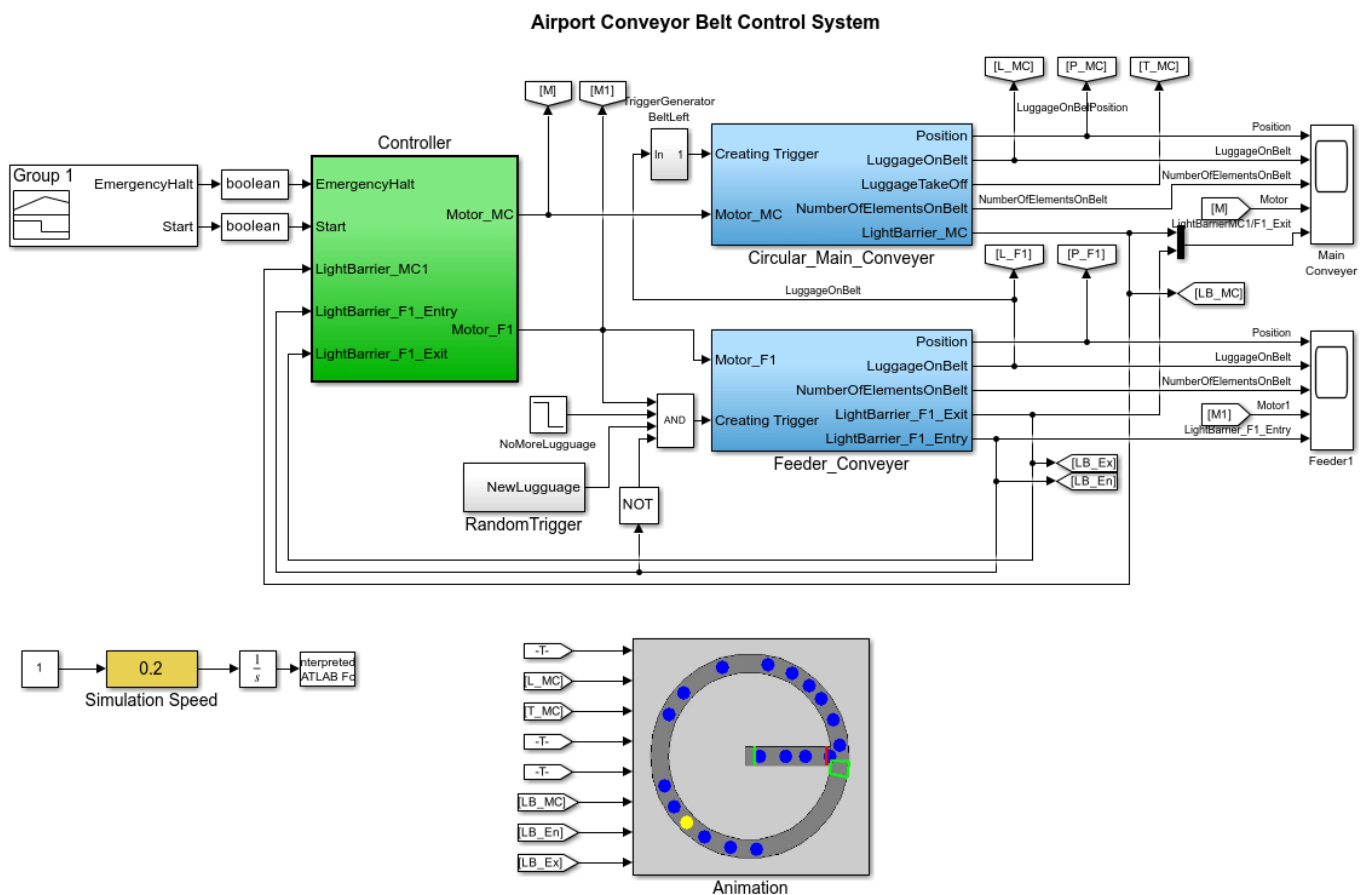

Copyright 2010-2019 The MathWorks, Inc.

View Requirements

You can view requirements linked to the model by using the * Requirements Manager* app and by using the object context menu on specific model.

- **Requirements Manager** app

Open the **Requirements Manager** app. Click a requirement in the Requirements Browser. The corresponding model element is highlighted.

- Model Object Context Menu

To view the requirement for a model-specific object, right-click an element and select **Requirements > Open Outgoing Links**. For example, to view the requirements for the `Controller` subsystem block, right-click the `Controller` block and select **Requirements > Open Outgoing Links**.
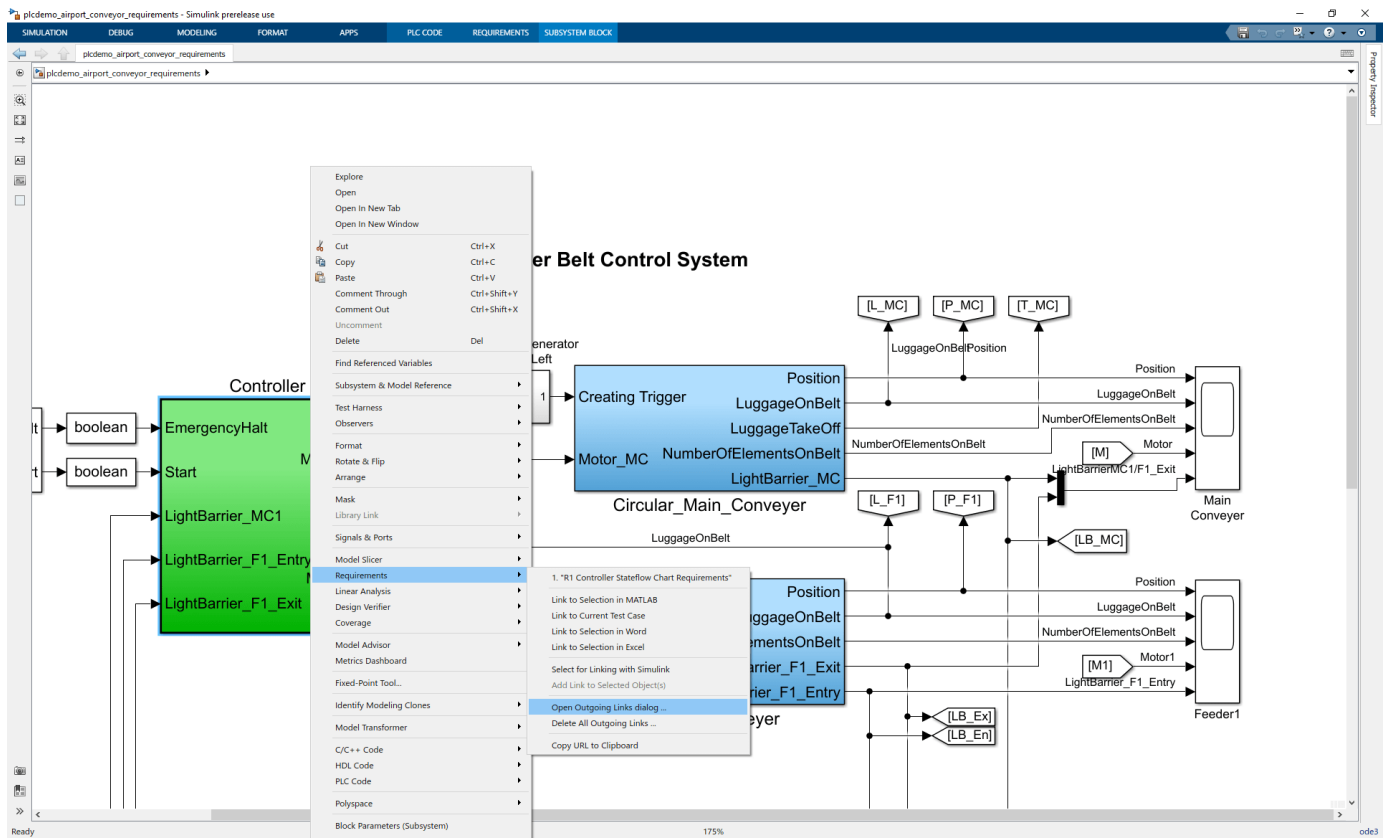
See Also

- "Link Model Objects" (Requirements Toolbox)
- "Requirements Management Interface" (Requirements Toolbox)

Run-Time Data Collection by Using External Mode Logging

Generate code by using external mode logging. Download the generated code with the external mode logging function to the target Programmable Logic Controller (PLC) and collect run-time data. Visualize and monitor the collected run-time data by using Simulation Data Inspector and an Open Platform Communications (OPC) server.

Target Integrated Development Environments (IDEs)

- Rockwell Automation® Studio 5000® IDE
- Rockwell Automation® RSLinx® Classic

Open Model

Open the `ext_demo1.slx` model. The model consists of two children subsystems `S1` and `S2`, a MATLAB® Function block, and a Stateflow® chart.

```
uiopen('ext_demo1.slx',1);
```
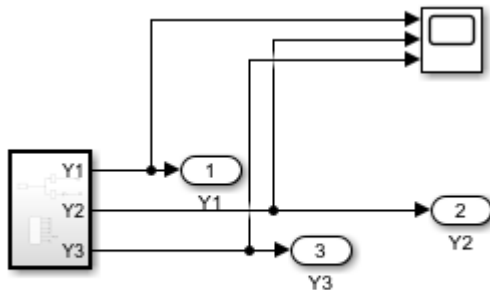

Copyright 2020 The MathWorks, Inc.

The `S1` and `S2` children subsystems are identical and contain a simple feedback loop.

```
mdl_1 = 'ext_demo1/Subsystem/S1';
open_system(mdl_1);
```
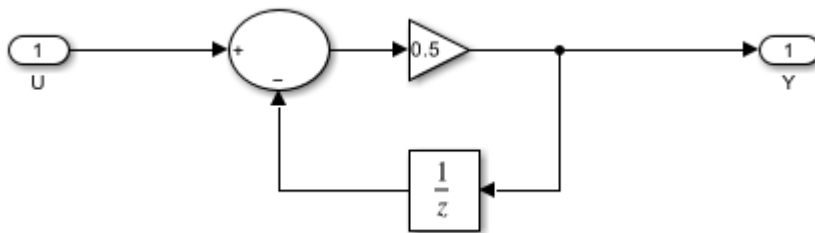

The Stateflow® chart is a simple state machine that has four states. The states change the value of the variable `out` during every simulation timestamp.

```
mdl_2 = 'ext_demo1/Subsystem/Chart';
open_system(mdl_2);
```

25-83
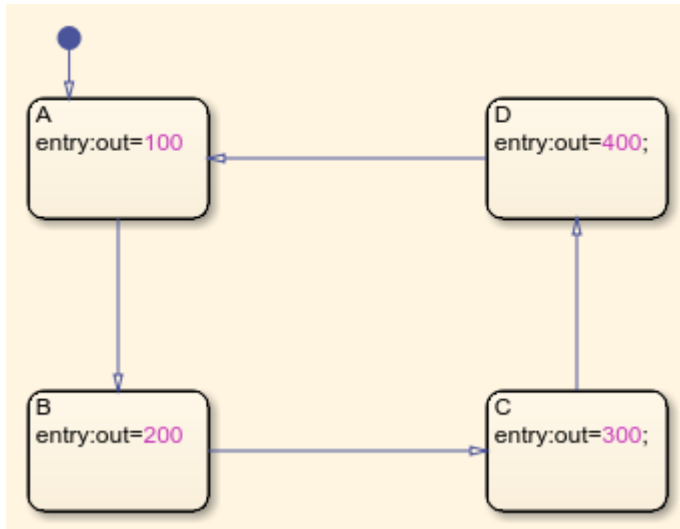
The MATLAB® Function block produces code to generate a sine wave. The sine wave is the input to the S1 and S2 subsystems.

```
mdl_3 = 'ext_demo1/Subsystem/MATLAB Function';
open_system(mdl_3);
```

```
1      function y = fcn
2  -      persistent i;
3
4  -      if isempty(i)
5  -          i=0;
6        end
7
8  -      if (i>20)
9  -          i = 0;
10       else
11 -          i=i+1;
12       end
13
14 -      y = sin(pi*i/10);
```

External Mode Logging and Code Generation

External mode logging can save system states, outputs, and simulation time at each model execution step. The data is written to a MAT-file. Collect run-time data for the variables in the MAT-file by running the generated code, which contains the logging function in a target IDE.

To enable external mode logging and generate code:

1   Open the Simulink® PLC Coder™ app.

2   Select the Subsystem block. In the **PLC Code** tab, click **Settings**.

3   On the **PLC Code Generation** pane, set **Target IDE** to Rockwell Studio 5000: AOI.

4 On the **Interface** pane, select **Generate Logging Code**. Click **OK**.

5 In the **PLC Code** tab, click **Generate PLC Code**.

The software also generates a `plc_log_data.mat` file during code generation.

Download Code and Configure RSLinx® OPC Server

To download and set up the OPC server:

1 Open the `ext_demo1.ACD` file by using the Studio 5000® IDE. Compile the file and download it to your target PLC.

2 Start RSLinx® and select **DDE/OPC > Topic Configuration**. Click **New**, and in the dialog box, enter `ext_demo1` as the topic name. On the **Data Source** tab, select your target PLC. Click **Yes**.
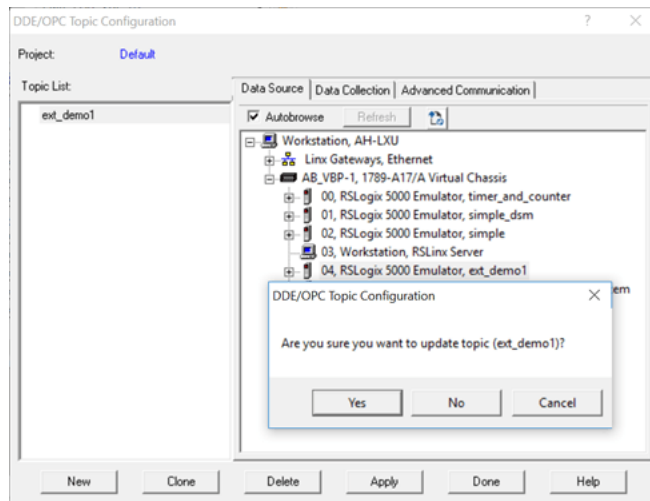
To verify the OPC server setup, in RSLinx® select **Edit > Copy DDE/OPC Link**. If `i0_Subsystem_val` is present, the server configuration is complete.
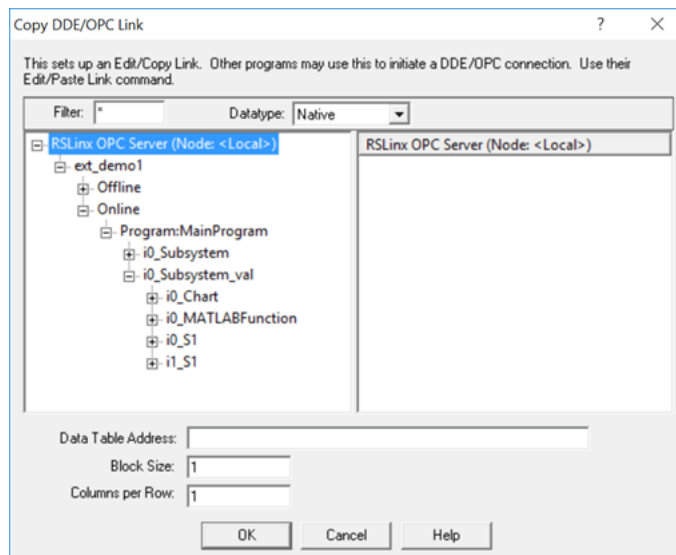
Stream and Display Run-Time Data

You can stream and display the logging data through Simulation Data Inspector by using Simulink PLC Coder™ external mode commands. Use **plcdispextmodedata** to display the contents of the plc_log_data MAT-file.

```
cd plcsrc
plcdispextmodedata plc_log_data.mat
```

Connect to the OPC server and stream logging data by using the **plcrunextmode** function.

```
plcrunextmode('localhost','studio5000','ext_demo1','plc_log_data.mat')
```

You must have the RSLinx® classic version to copy the DDE/OPC link. The RSLinx® Classic Lite version does not work.

See Also

- plcdispextmodedata
- plcrunextmode

Verify Generated Code by Using Cosimulation

Verify your generated code by using cosimulation. The generated code and testbench run on a Codesys software-based programable logic controller (soft PLC). The Simulink ™ model uses Open Platform Communication Unified Architecture (OPC UA) to communicate with and retrieve the cosimulated data from the soft PLC. The Simulink model verifies the generated code by comparing the model simulation results to the cosimulated soft PLC results.

Prerequisites

On your system, you must have installed:

- Codesys V3.5.SP16 + IDE or Codesys V3.5 SP16 Patch 1 + IDE
- Industrial Communication Toolbox™
- Simulink® PLC Coder™

Model Description

The simple_cosim model is made up of the SimpleSubsystem block and the MATLAB System (PLCCOSIM) block. During simulation, the model compares values from the SimpleSubsystem block to the values from the soft plc which are retrieved by using the MATLAB System (PLCCOSIM) block.
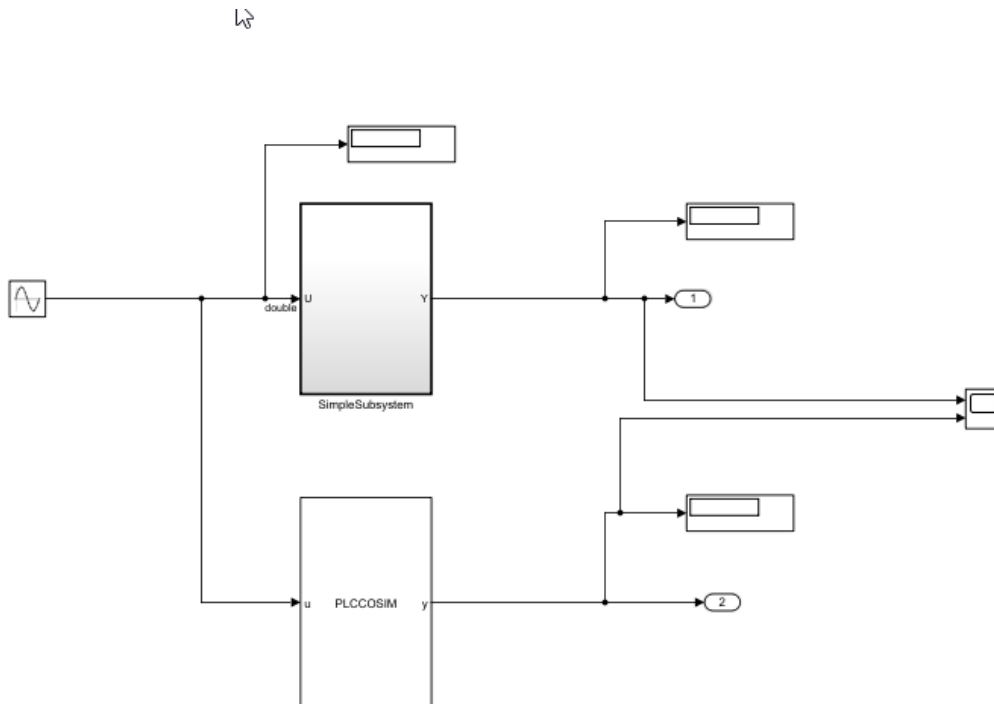
The SimpleSubsystem block contains a simple feedback loop.
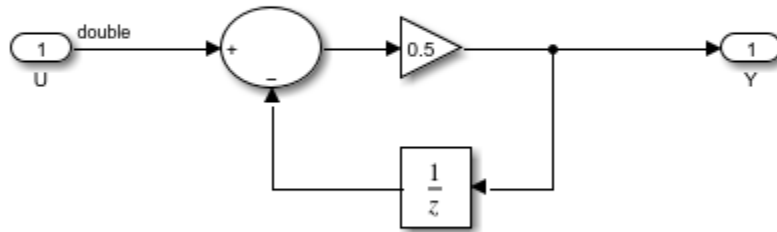
The MATLAB System block is set up to run the PLCCOSIM.m file. The file contains the code to set up OPC UA communications and retrieve the cosimulated data from the soft PLC.

```
type PLCCOSIM.m
```

```
classdef PLCCOSIM < matlab.System
    % PLCCOSIM Add summary here
    %
    % This template includes the minimum set of functions required
    % to define a System object with discrete state.

    % Public, tunable properties
    properties

    end

    properties(DiscreteState)
        CycleNum;
    end

    % Pre-computed constants
    properties(Access = private)
        UAObj;
        DeviceNode;
        Cycle_U;
        Cycle_Y;
        TestCycleNum;
        PreviousCycleNum;
    end

    methods(Access = protected)
        function setupImpl(obj)
            % Perform one-time calculations, such as computing constants
            % init opc UA server connection
            obj.UAObj = opcua('opc.tcp://localhost:4840');
```

```
            connect(obj.UAObj);
            obj.DeviceNode = findNodeByName(obj.UAObj.Namespace,'DeviceSet','-once');
            obj.Cycle_U = findNodeByName(obj.DeviceNode,'cycle_U');
            obj.Cycle_Y = findNodeByName(obj.DeviceNode,'cycle_Y');
            obj.TestCycleNum = findNodeByName(obj.DeviceNode,'testCycleNum');
            obj.PreviousCycleNum = findNodeByName(obj.DeviceNode,'previousCycleNum');
        end

        function y = stepImpl(obj,u)
            % Implement algorithm. Calculate y as a function of input u and
            % discrete states.
            obj.CycleNum = obj.CycleNum+1;
            writeValue(obj.UAObj, obj.Cycle_U, u);
            writeValue(obj.UAObj, obj.TestCycleNum, obj.CycleNum);

            valueUpdated = false;
            for rct = 1:100
                previousCycleNumValue = readValue(obj.UAObj, obj.PreviousCycleNum);
                if previousCycleNumValue == obj.CycleNum
                    valueUpdated = true;
                    y = readValue(obj.UAObj, obj.Cycle_Y);
                    break
                end
                pause(0.001)
            end

            if ~valueUpdated
                error('not get the value for cycle number %d', obj.CycleNum);
            end
        end

        function resetImpl(obj)
            % Initialize / reset discrete-state properties
            obj.CycleNum = 0;
            writeValue(obj.UAObj, obj.TestCycleNum, obj.CycleNum);
            writeValue(obj.UAObj, obj.PreviousCycleNum, obj.CycleNum);
        end

        function out = getOutputSizeImpl(obj)
            out = propagatedInputSize(obj,1);
        end

        function out = getOutputDataTypeImpl(obj)
            out = propagatedInputDataType(obj,1);
        end

        function c1 = isOutputComplexImpl(obj)
            c1 = false;
        end

        function c1 = isOutputFixedSizeImpl(obj)
            c1 = true;
        end

        function s = getDiscreteStateImpl(obj)
            s = obj.CycleNum;
        end
```

```
        end
    end
```

Compile, and Simulate the Codesys Project

The Codesys project file runs the generated code on the soft PLC. If you have the Codesys V3.5.SP16 + IDE, use the `plc-cosim_v3sp16.project` as the project file. If you have the Codesys V3.5.SP16 Patch 1+ IDE, use the `plc-cosim_v3sp16patch1.project` as the project file.

1   In the Windows ® system tray, select CODESYS Control Win SysTray -x64, right-click, and select **Start PLC.**

2   Open the project file based on the Codesys Target IDE version.

3   Double-click `CODESYS_Control_Win_V3_x64` and select **Scan Network.**

CODESYS_Control_Win_V3_x64 ✕

Communication Settings	Scan Network	Gateway ▾	Device ▾
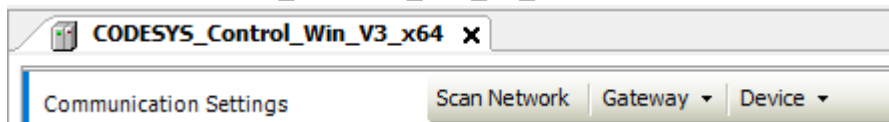
4   In the **Scan Network tab,** select your device in the **Select Device** window. Click **OK.**

Select Device   ✕

Select the Network Path to the Controller

```
⊟ 🖧● Gateway-1 (Scanning...)
     🖥 AH-SKAPALI[0001]
```

Device Name:
Gateway-1

Driver:
TCP/IP

IP-Address:
localhost

Port:
1217

Scan Network

Wink

OK   Cancel
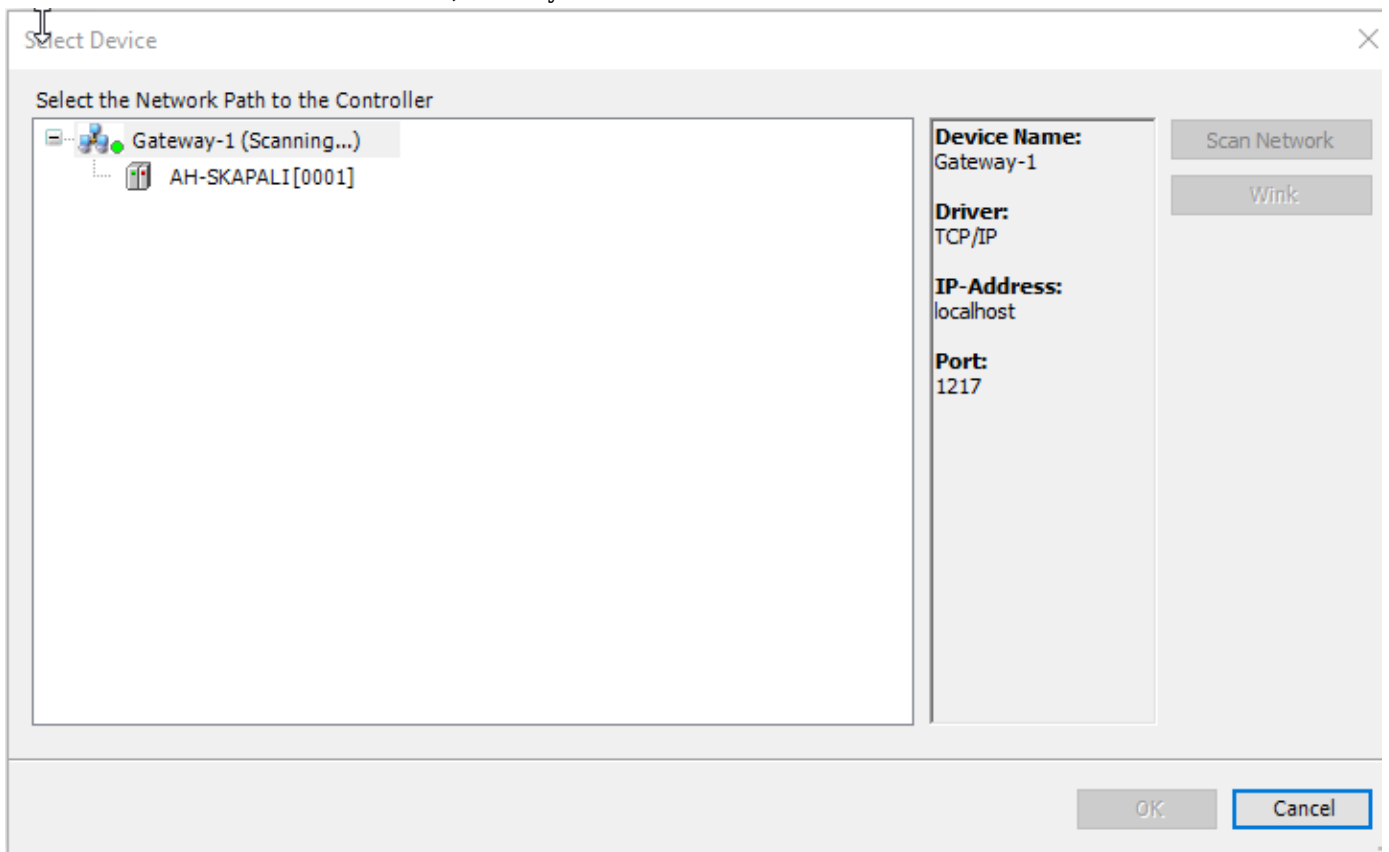
5   Select **Build > Build.**
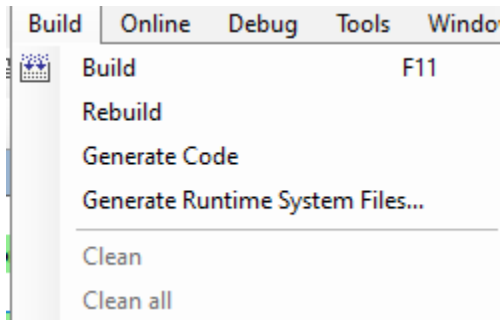
6.Select **Online > Login.**
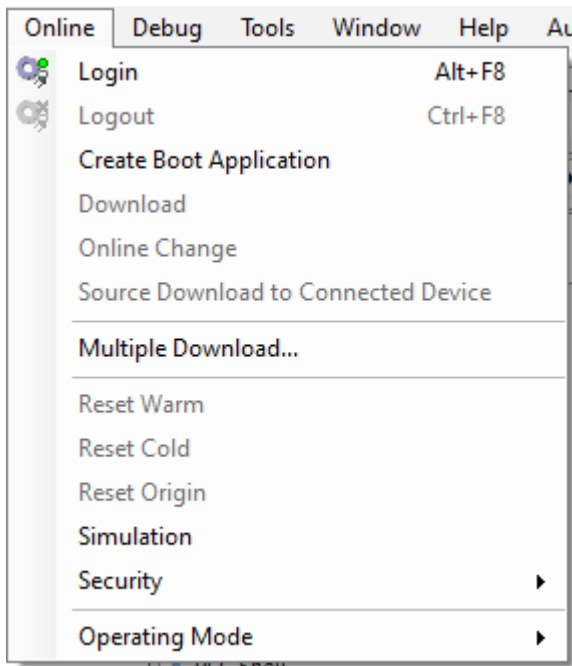
7. Select **Debug > Start.**
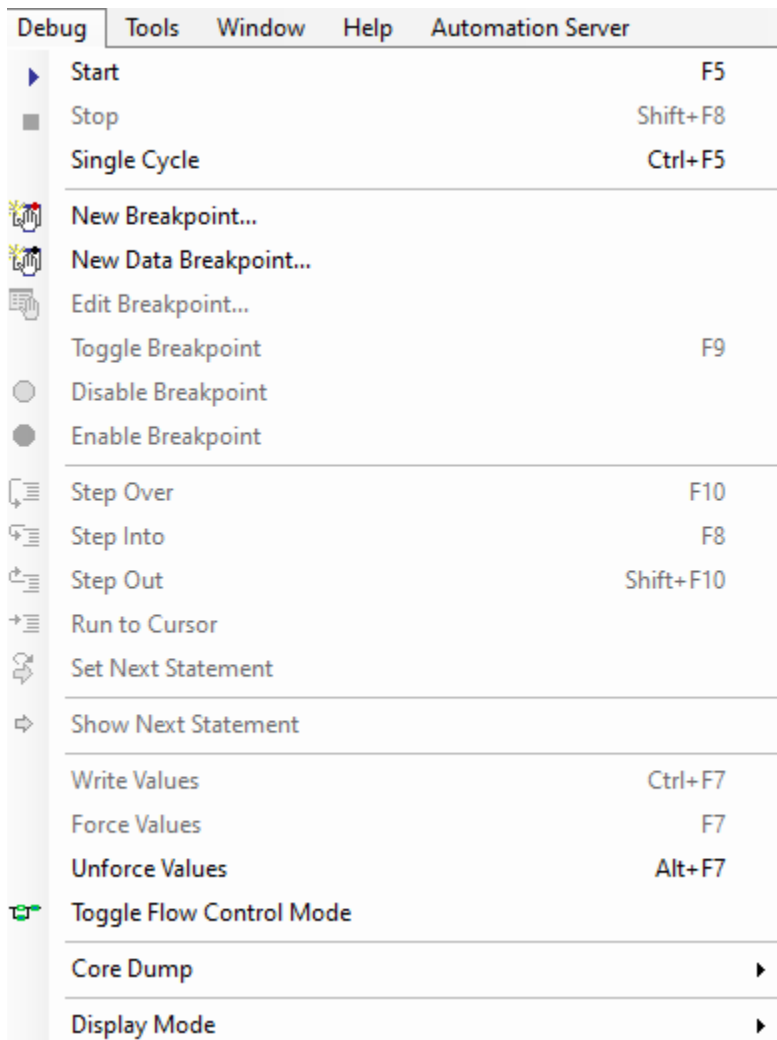
The soft PLC is now running the generated code.

Simulate the Model and Verify Generated Code

Simulate the Simulink ™ model and verify the generated code by comparing the simulation model results to the soft PLC results.The soft PLC results are retrieved by using the OPC UA connection.

1   Open the `simple_cosim.slx` file for the Simulink model.

```
% open_system('simple_cosim')
```

2. Simulate the model. Verify the generated code by comparing the model simulation results to the soft PLC cosimulation results.
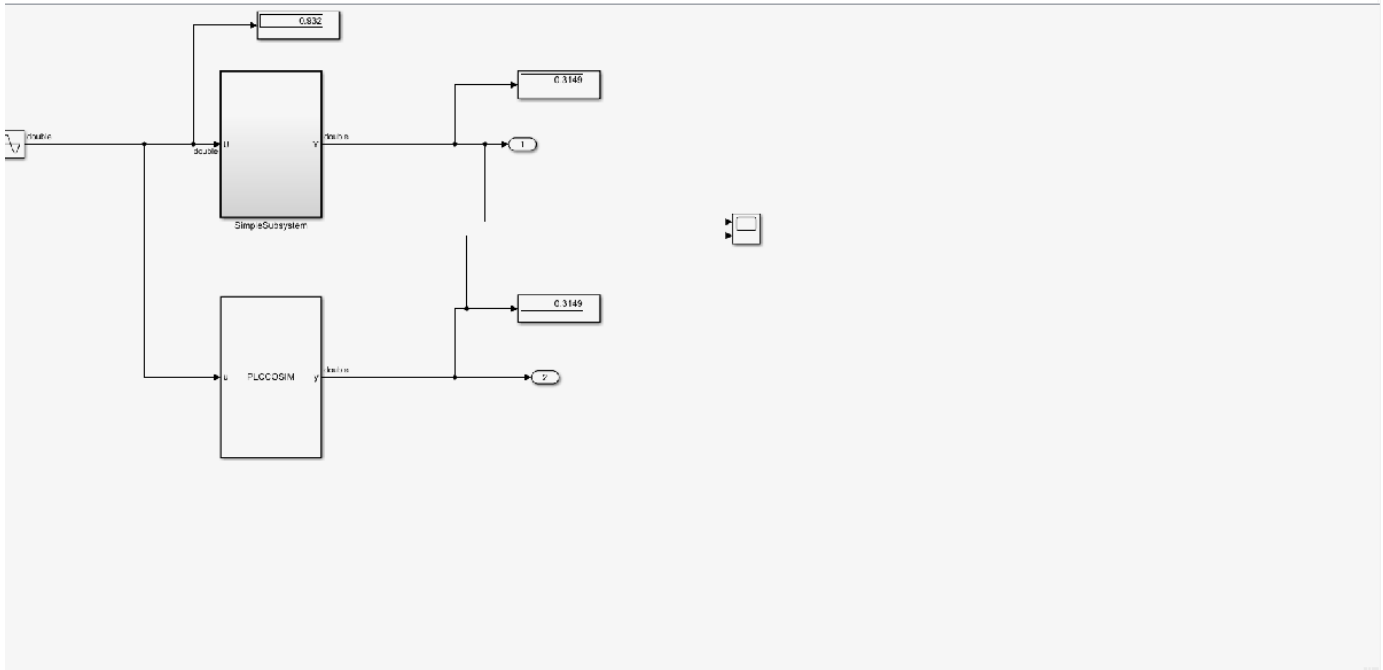
See Also

- "Test Bench Verification" on page 4-2
- "External Mode Logging" on page 14-2

Generate Structured Text Code for Variable-Size Signals

Deploy applications such as machine learning models to your target PLC by generating code for variable-size signals. Verify the generated structured text code by generating test bench code and importing the generated test bench code into your target IDE.

Model Description

The model consists of a simple MATLAB® Function Block, Stateflow® chart and a Simulink® subsystem. The MATLAB® Function Block generates a variable one-dimensional array based on whether the input signal u is greater than or less than zero. The Stateflow® chart has two states that transition between assigning the output variable y to the input variable u and multiplying y by two. The Simulink® subsystem has a `Sum of Elements` block that sums all the elements in y.

Simulate and Generate Structured Text Code

Open the Simulink model.

```
mdl = 'vardim_ex';
open_system(mdl)
```
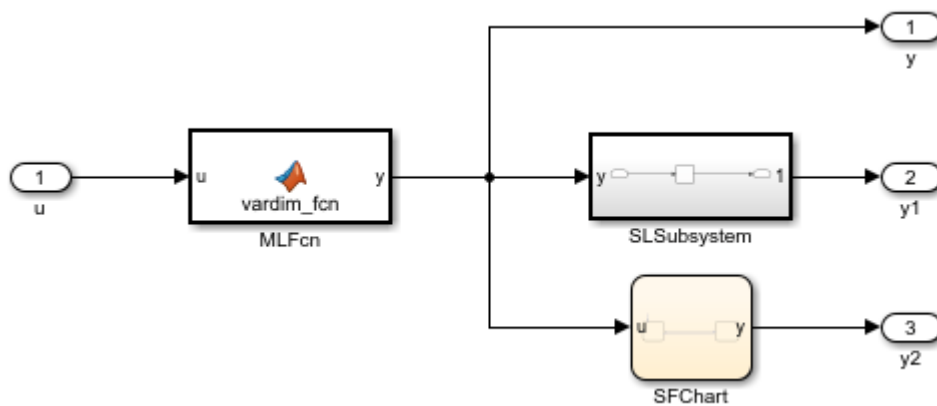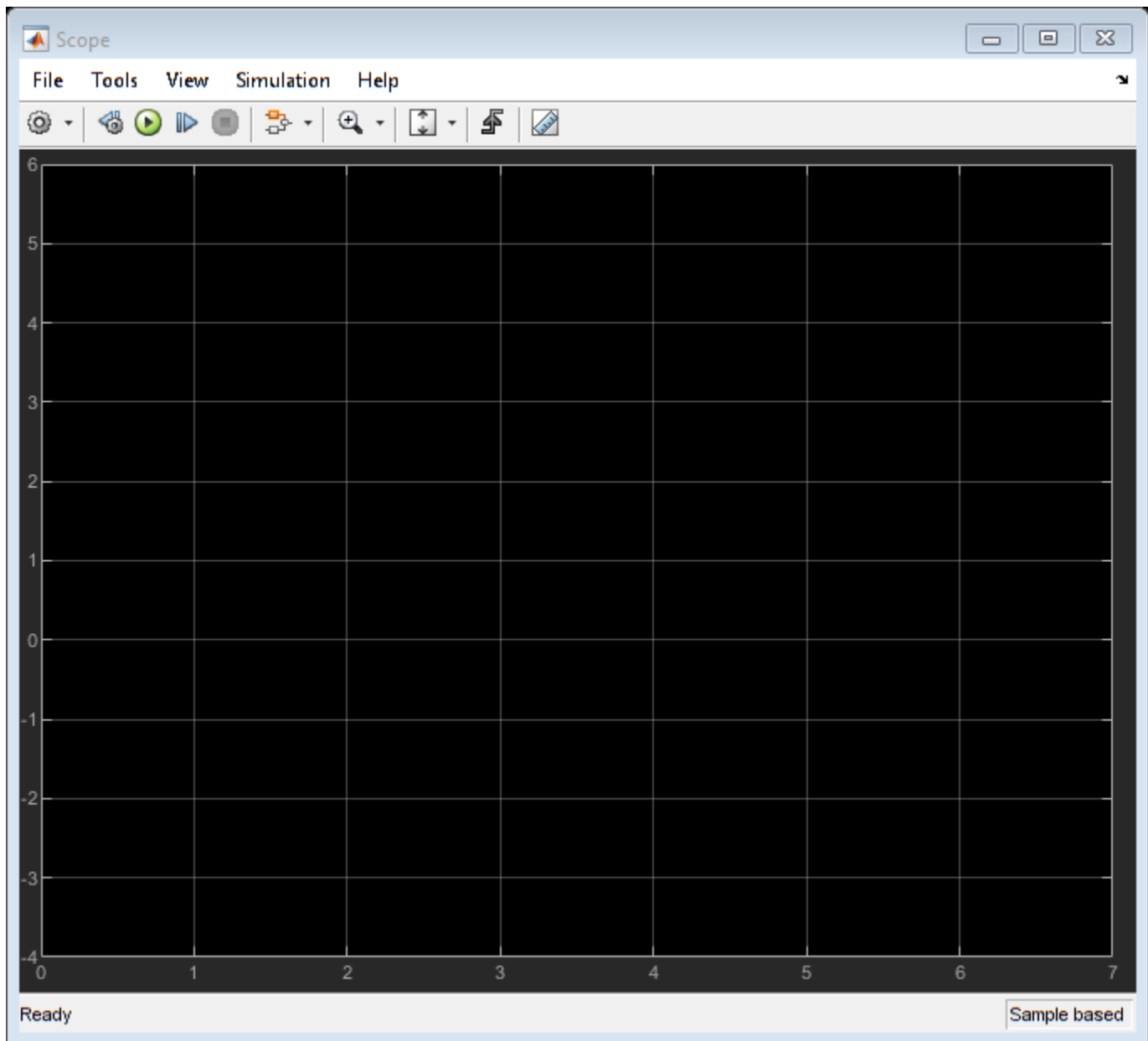
To generate structured text code for the `Subsystem` block, use the `plcgeneratecode` command.

plcgeneratecode([mdl '/Subsystem']);

Close the model

close_system(mdl)

See Also
"Variable-Size Signal Code Generation" on page 30-2

Add Subsystem Port and Bus Descriptions in Generated Code

This example shows how to add descriptions for subsystem input ports, output ports, and buses. The generated structured text code then includes these port and bus descriptions.

Model Description

The model consists of two buses: aBus that has three elements and bBus that has two elements. The model also consists of a Subsystem block that performs operations on the individual elements of the buses.

Open the model.

```
open_system('GenerateComments');
```
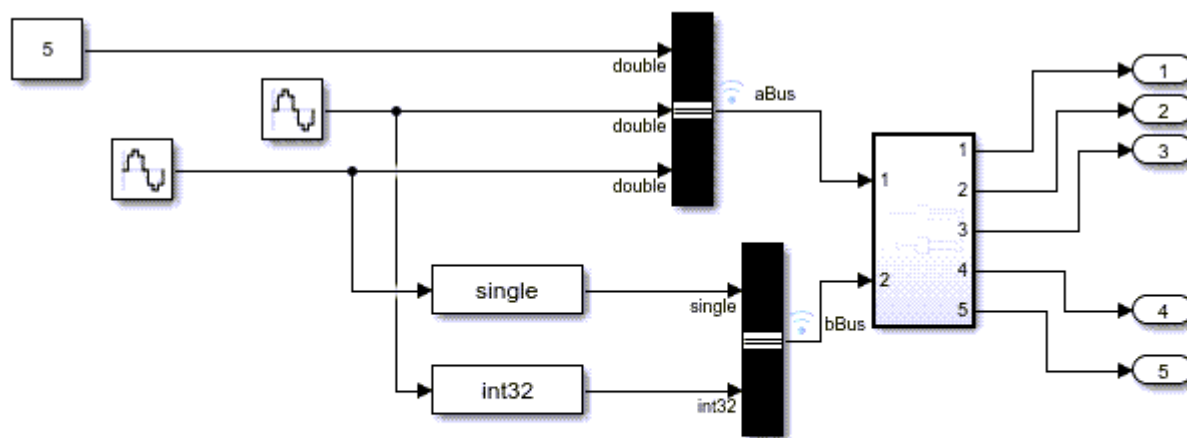

Copyright 2021 The MathWorks, Inc.

Input and Output Port Descriptions

The descriptions for your input and output ports appear in the generated structured text code. The input and output ports must be inside the Subsystem block for their descriptions to appear in the generated code. To add a description to an input or output port block, right-click the block, select **Properties,** and enter a description in the **Description** field. This image shows the port description for input port In1.
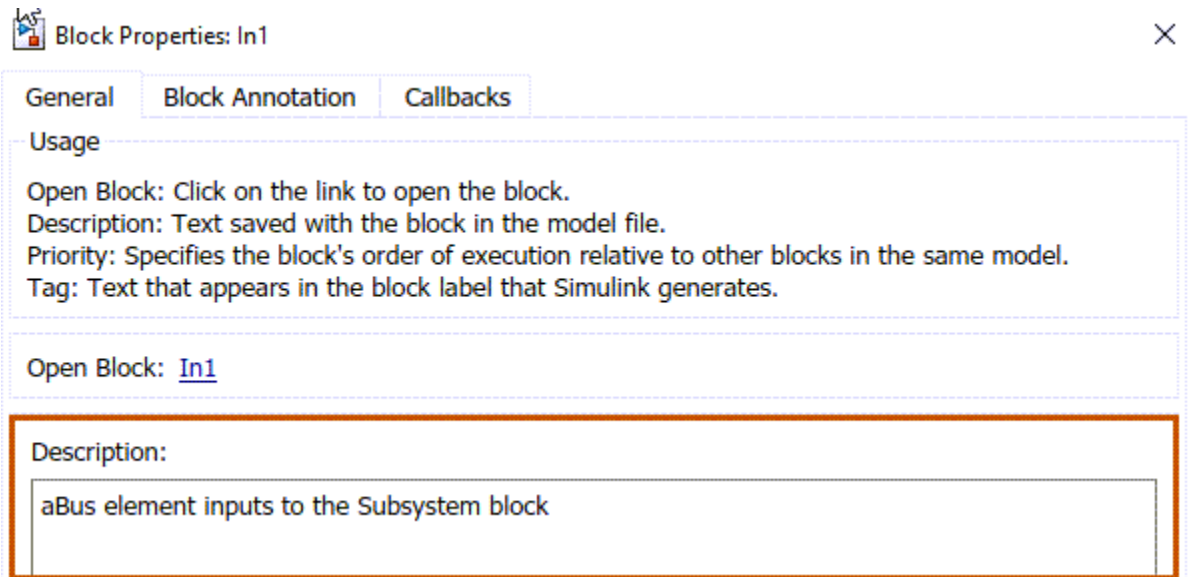
Bus Descriptions

The descriptions for your buses appear in the generated code. The descriptions for buses appear as struct data types. To add a description for a bus, open the **Type Editor**. In the MATLAB Command Window, enter:

```
typeeditor
```

In the Type Editor, select the bus object and enter the description in the **Description** field. This image shows the description for the bus object named bBus.
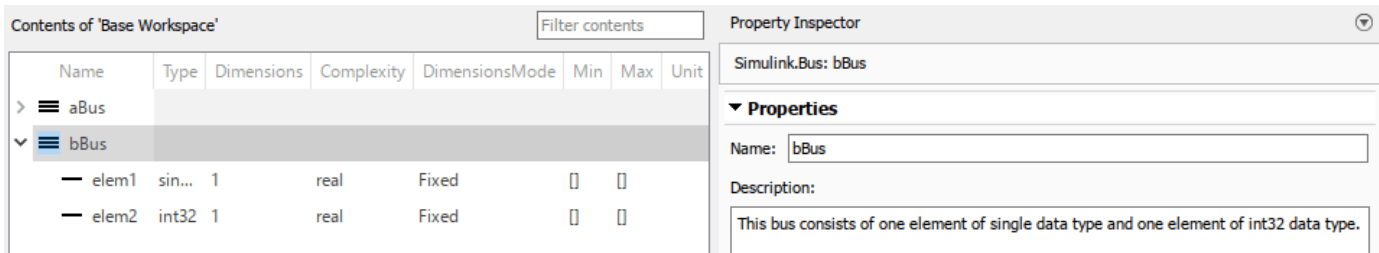
Generate Code

To generate structured text code, do one of the following:

- Open the PLC Coder app. Select the SimpleSubsystem block and click Generate PLC Code.
- Use the plcgeneratecode function:

```
plcgeneratecode('GenerateComments/Subsystem');

### Generating PLC code for 'GenerateComments/Subsystem'.
### Using model settings from 'GenerateComments' for PLC code generation parameters.
### Gathering test vectors for PLC testbench.
### Begin code generation for IDE codesys23.
### Emit PLC code to file.
### PLC code generation successful for 'GenerateComments/Subsystem'.
```

```
### Generated files:
plcsrc\GenerateComments.exp
```

Descriptions in Generated Code

The descriptions for the subsystem ports and buses appear in the generated code.

This image shows the input port descriptions in the generated code.

```
VAR_INPUT
     (* Description: aBus element inputs to the Subsystem block
     *)
     In1: aBus;
     (* Description: bBus element inputs to the Subsystem block
     *)
     In2: bBus;
END VAR
```

This image shows the bus descriptions in the generated code.

```
TYPE bBus:
     (* Description: This bus consists of one element of single data type and one element of int32 data type
     *)
     STRUCT
          elem1: REAL;
          elem2: DINT;
     END_STRUCT
END_TYPE
TYPE aBus:
     (* Description: This bus consists of three elements of double data type.
     *)
     STRUCT
```

Close the Model

```
close_system('GenerateComments');
```

Generate Structured Text Code By Using a Custom Supported Function List

This example shows how to generate structured text code by using a custom supported function list. Simulink® PLC Coder™ returns an error when generating code for unsupported blocks or functions. Generate structured text code for the unsupported blocks or functions by creating a custom hook file function. You must validate the generated code when using a custom hook file.

Model Description

The model consists of a subsystem block called `Subsystem` that accepts two integers as inputs to a MATLAB Function block. The MATLAB Function block adds the two integers together along with a double precision number generated by using the `str2double` function inside the function block. The `str2double` function is currently unsupported for PLC code generation.

```
open_system("CustomizedMLFcn.slx");
```
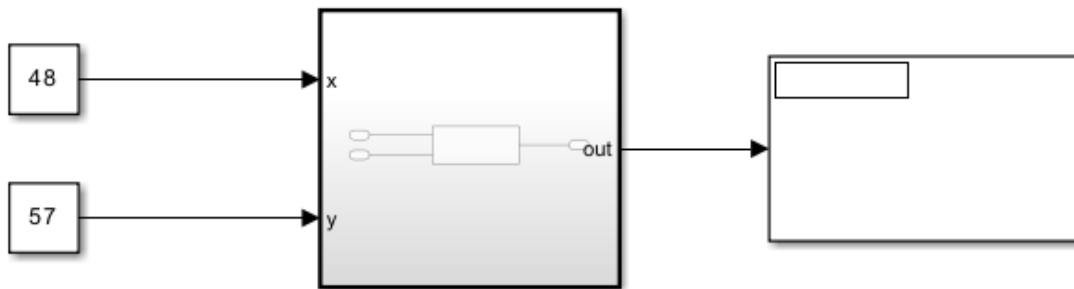

Copyright 2022 The MathWorks, Inc.

Create Custom Function List

Create a custom function list by using this template file. The function name must be `plc_library_function_hook`. Inside the function, list the path to the function or block for which you want to generate code. To list all blocks or functions inside a toolbox or folder, use an asterisk.

```
function lib_map = plc_library_function_hook(lib_map)
lib_map('toolbox/tbx_foo/tbx_foo/lib_bar/fcn_xyz.m') = {'fcn_xyz'}; % register toolbox/tbx_foo/
lib_map('toolbox/tbx_foo1/tbx_foo1/lib_bar') = {'*'}; % register all functions or blocks under
end
```

This custom hook file enables code generation for the `str2double` function.

```
type plc_library_function_hook.m

function lib_map = plc_library_function_hook(lib_map)
    lib_map('toolbox/eml/lib/matlab/strfun/str2double.m') = {'str2double'};
    lib_map('toolbox/eml/lib/matlab/strfun/char.m') = {'char'};
end
```

Place the custom function list on the MATLAB path.

Generate Structured Text Code

To generate structured text code using the Simulink® PLC Coder™ app:

1 Open the **PLC Coder** app. Click **Settings > PLC Code Generation.** Change the **Target IDE** to `3S CodeSys 2.3`. Click **OK**.

2 Select the `Subsystem` block. In the **PLC Code** tab, click **Generate PLC Code**.

Alternatively, to generate structured text code from the MATLAB command line, use the `plcgeneratecode` function.

```
generatedfiles = plcgeneratecode('CustomizedMLFcn/Subsystem');

### Generating PLC code for 'CustomizedMLFcn/Subsystem'.
### Using model settings from 'CustomizedMLFcn' for PLC code generation parameters.
### Gathering test vectors for PLC testbench.
### Begin code generation for IDE codesys23.
### Emit PLC code to file.
### Creating PLC code generation report CustomizedMLFcn_codegen_rpt.html.
### PLC code generation successful for 'CustomizedMLFcn/Subsystem'.
### Generated files:
plcsrc\CustomizedMLFcn.exp
```

See Also

More About

- "Test Bench Verification" on page 4-2

Generate Optimized Structured Text Code by Folding Expressions

This example shows how to generate optimized structured text code by optimizing the output variables of function blocks generated from reusable Simulink® subsystems. Optimize the output variables by using expression folding to minimize the computation of intermediate results at block outputs and the storage of these results in temporary buffers or variables. When you enable the **Fold temporary variable using Function block instance output variable** optimization, Simulink® PLC Coder™ folds the function block computations into a single expression, instead of generating separate code statements and storage declarations for each function block.

Open the Model

`open_system('Calculator.slx')`

The model consists of a subsystem at the top level that accepts the inputs `a`, `b`, and `c`. The subsystem contains a reusable masked subsystem, `fb_Calculate`, that houses an enabled subsystem consisting of:

- `fb_Subtract` — A reusable function block that subtracts `b` from `a`.
- `fb_Divide` — A reusable function block that divides the output of `fb_Subtract` by `c`.

The output of the subsystem is a switched output signal that outputs the result of `fb_Subtract` if the difference is greater than ten and the output of `fb_Divide` when the difference is less than ten.
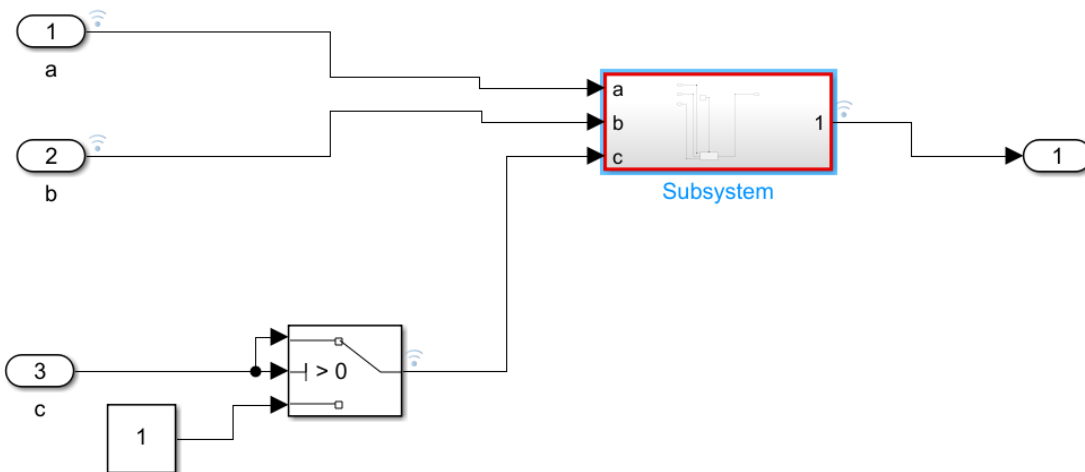
Copyright 2022 The MathWorks, Inc.

Enable Optimization

To enable the **Fold temporary variable using function block instance output variable** optimization using the Simulink® PLC Coder™ app:

1 In the apps tab, click **PLC Coder**.

2   In the **PLC Coder** tab, click **Settings > PLC Code Generation Settings**. In the **Configuration Parameters** window, click **Optimizations.**

3   Select **Fold temporary variable using Function Block instance output variable.**

4   Click **OK**.

Alternatively, to enable the optimization by using the MATLAB® command line, enter:

```
set_param('Calculator','PLC_FoldFBCallOutputVar','on')
```

Generate Structured Text Code

To generate structured text code using the Simulink® PLC Coder™ app:

1   In the **PLC Coder** tab, click **Settings > PLC Code Generation Settings.** Change the **Target IDE** to 3S CodeSys 2.3. Click **OK**.

2   Select the Subsystem block. In the **PLC Code** tab, click **Generate PLC Code**.

Alternatively, to generate structured text code from the MATLAB command line, use the plcgeneratecode function.

```
generatedfiles = plcgeneratecode('Calculator/Subsystem');

### Generating PLC code for 'Calculator/Subsystem'.
### Using model settings from 'Calculator' for PLC code generation parameters.
### Gathering test vectors for PLC testbench.
### Begin code generation for IDE codesys23.
### Emit PLC code to file.
### PLC code generation successful for 'Calculator/Subsystem'.
### Generated files:
tb\Calculator.exp
```

Non-Optimized Code and Optimized Code

This image shows the comparison between the non-optimized code where expression folding is disabled and optimized code where expression folding is enabled.

Non-Optimized   **Optimized**

When you enable the optimization, Simulink® PLC Coder™:

- Removes temporary variables and buffer declarations.
- Folds the function block output expressions into a single line. For example, `i0_Divide(ssMethodType :=SS_INITIALIZE b_d :=i0_Subtract.Difference, c :=c)`
- Replaces temporary variable calls with folded expression calls.

See Also

"Fold temporary variable using function block instance output variable" on page 13-22

Generate Structured Text Code with VAR_IN_OUT Variables

This example shows how to model and generate structured text code that has VAR_IN_OUT variables. A VAR_IN_OUT variable is an input and output variable that is a part of the function block interface. Use VAR_IN_OUT variables to pass values to the function block by using the pass by reference method.

Open the Model

```
open_system('GenerateVarInOutVariables.slx')
```
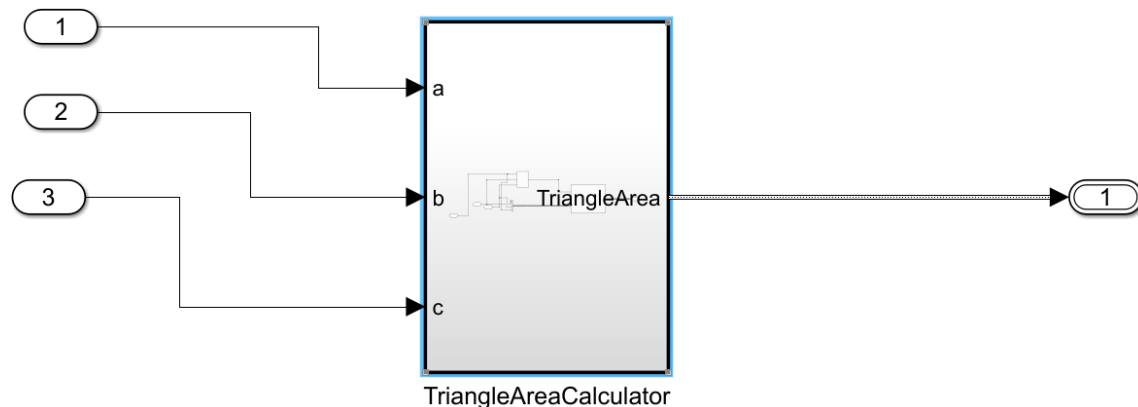

Copyright 2022 The MathWorks, Inc.

The model consists of a subsystem `TriangleAreaCalculator` at the top level that accepts three inputs `a`, `b`, and `c`. The subsystem contains :

- A subsystem block `CalculateTriangleSemiPerimeter` that accepts `a`, `b`, and `c` as an input and generates an output `s`.
- A MATLAB® Function block `CalculateTriangleArea` that accepts `TriangleArea` and `s` as inputs and generates `TriangleArea` as the output.

Model to Generate VAR_IN_OUT Variables

Generate VAR_IN_OUT variables by using a MATLAB Function block and declaring a function of type:

```
function x = function_name(x)
% body of function
```

Simulink® PLC Coder supports VAR_IN_OUT generation only from MATLAB Function blocks. This is the structure of the MATLAB Function Block:

```
function TriangleArea = fcn(s,TriangleArea)
intermediate_area = (s-TriangleArea.a)*(s-TriangleArea.b)*(s-TriangleArea.c);
TriangleArea.area = sqrt(s*intermediate_area);
```

Generate Structured Text Code

To generate structured text code,

1 In the **Apps** tab, click **PLC Coder**.

2 In the **PLC Coder** tab, clock **Settings > PLC Code Generation Settings**. Change the **Target IDE** to 3S CodeSys 2.3. Click **OK**.

3 Select the subsystem block. In the **PLC Code** tab, click **Generate PLC Code**.

Alternatively, to generate structured text code from the MATLAB command line, use the plcgeneratecode function.

```
generatedfiles = plcgeneratecode('GenerateVarInOutVariables/TriangleAreaCalculator');

### Generating PLC code for 'GenerateVarInOutVariables/TriangleAreaCalculator'.
### Using model settings from 'GenerateVarInOutVariables' for PLC code generation parameters.
### Begin code generation for IDE codesys23.
### Emit PLC code to file.
### Creating PLC code generation report GenerateVarInOutVariables_codegen_rpt.html.
### PLC code generation successful for 'GenerateVarInOutVariables/TriangleAreaCalculator'.
### Generated files:
plcsrc\GenerateVarInOutVariables.exp
```

The generated structured text code contains VAR_IN_OUT variable declarations. The MATLAB Function block takes the variable TriangleArea as an input and generates the same variable as the output.

```
FUNCTION_BLOCK TriangleAreaCalculator
VAR_INPUT
    a: LREAL;
    b: LREAL;
    c: LREAL;
END_VAR
VAR_OUTPUT
    TriangleArea_e: TriangleArea;
END_VAR
VAR
    i0_CalculateTriangleArea: CalculateTriangleArea;
END_VAR
(* Outputs for Atomic SubSystem: '<Root>/TriangleAreaCalculator' *)
(* BusCreator: '<S1>/Bus Creator' incorporates:
 *  MATLAB Function: '<S1>/CalculateTriangleArea'
 *  Outport: '<Root>/TriangleArea' *)
TriangleArea_e.area := 0.0;
TriangleArea_e.a := a;
TriangleArea_e.b := b;
TriangleArea_e.c := c;
(* Product: '<S3>/Divide' incorporates:
 *  Constant: '<S3>/Constant'
 *  Outport: '<Root>/TriangleArea'
 *  Sum: '<S3>/Add' *)
i0_CalculateTriangleArea(b_s := ((a + b) + c) / 2.0, TriangleArea_i := TriangleArea_e);
(* End of Outputs for SubSystem: '<Root>/TriangleAreaCalculator' *)
END_FUNCTION_BLOCK
FUNCTION_BLOCK CalculateTriangleArea
VAR_INPUT
    b_s: LREAL;
END_VAR
VAR_IN_OUT
    TriangleArea_i: TriangleArea;
END_VAR
```

25-105

```
(* MATLAB Function 'TriangleAreaCalculator/CalculateTriangleArea': '<S2>:1' *)
(* '<S2>:1:2' intermediate_area = (s-TriangleArea.a)*(s-TriangleArea.b)*(s-TriangleArea.c); *)
(* '<S2>:1:3' TriangleArea.area = sqrt(s*intermediate_area); *)
TriangleArea_i.area := SQRT((((b_s - TriangleArea_i.a) * (b_s - TriangleArea_i.b)) * (b_s - Trian
END_FUNCTION_BLOCK
TYPE TriangleArea:
    STRUCT
        area: LREAL;
        b: LREAL;
        c: LREAL;
        a: LREAL;
    END_STRUCT
END_TYPE
```

See Also

More About

- "MATLAB Function Block Simulink PLC Coder Structured Text Code Generation" on page 20-9

PLC Coder Model Advisor

PLC Coder Checks in Model Advisor Overview

The **Simulink PLC Coder** checks in the Model Advisor verify your Simulink model or subsystem for compatibility with PLC Coder code generation. Update suboptimal conditions or settings identified by using the report generated by running the model advisor checks. To learn about Model Advisor, see "Check Your Model Using the Model Advisor".

The left pane displays folders that perform various checks:

- **Model configuration checks**: Prepare your model for compatibility with PLC code generation. This folder contains checks that verify whether model parameters are PLC code generation compatible, whether the masked subsystem parameters are configured correctly, and so on.
- **Checks for blocks and block settings**. Verify whether your model has blocks and block settings that are compatible for PLC code generation. This folder contains checks that verify whether the model uses event based blocks, probe blocks, contains Stateflow charts with continuous update rates, and so on.
- **Industry standard checks**. This folder contains checks that verify whether model contains reserved keyword names, names that exceed a maximum defined length, and so on.

T learn more about each individual check, right-click that check, and select **What's This?**
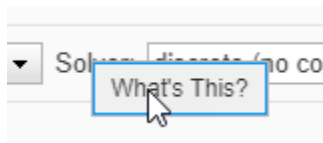
Model Configuration Checks

To prepare your model for compatibility with PLC code generation, use the checks in this folder. This folder contains checks that verify whether:

- The components in data store memory blocks resolve to `Simulink.Signal` objects.
- The model uses Stateflow messages.
- The model uses row-major algorithms.
- The model mask parameters use the `Inf` value.
- The model tunable parameters uses the `Inf` value.
- The model uses machine-parented data.
- The model has signal lines that resolve to `Simulink.Signal` objects.

Check Data Store Memory blocks

Check ID:mathworks.PLC.DSMResolveToSLSignal

Check that the Data Store Memory block has the option to resolve the data store memory object to `Simulink.Signal` object enabled.

Available with Simulink PLC Coder.

Description

This check displays a warning when Data Store Memory blocks do not resolve to Simulink signal objects.

Results and Recommended Actions

Condition	Recommended Action
Data Store Memory blocks have unsupported settings	Create a `Simulink.Signal` object. In the data store memory object enable the **Data store name must resolve to Simulink signal object**.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Check model for Stateflow messages

Check ID:mathworks.PLC.SFMessagesCheck

Check that the model does not use Stateflow messages.

Description

This check displays a warning when the model uses Stateflow messages.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Stateflow messages are not supported by Simulink PLC Coder	Avoid Stateflow messages in the model.

Capabilities and Limitations

Exclusion of blocks and charts is not supported.

Check if signal lines are configured properly

Check ID:mathworks.PLC.LineResolveToSLSignalCheck

Check that the model does not have signal lines that resolve to `Simulink.Signal` objects.

Description

This check displays a warning when the model uses signal lines that resolve to `Simulink.Signal` objects.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Signal lines resolve to `Simulink.Signal` object.	Disable the `Resolve to Signal Object` property.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Check if model uses row-major algorithms

Check ID:mathworks.PLC.RowMajorCheck

Check that the model does not use row-major algorithms.

Description

This check displays a warning when the model uses row-major algorithms.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
`UseRowMajorAlgorithm` parameter is set to `on`. Simulink PLC Coder does not support this setting.	Configure the model to avoid using row major algorithms.

Capabilities and Limitations

Exclusion of blocks and charts is not supported.

Check model mask parameters

Check ID:mathworks.PLC.MaskParamInfCheck

Check that the model does not contain masked parameters that use the Inf value.

Description

This check displays a warning when the model contains masked parameters that use the Inf value.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Mask parameters have Inf elements.	Check the mask parameters settings to ensure that there are no Inf values.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Check if model uses machine parented data

Check ID:mathworks.PLC.MachineParentedDataCheck

Check that the model does not contain blocks or events that use machine-parented data.

Description

This check displays a warning when the model contains blocks that use machine-parented data.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
These blocks and events are not supported by Simulink PLC Coder	Design the model to avoid using machine-parented data.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Check if model uses custom code

Check ID:mathworks.PLC.CustomCodeCheck

Check that the model does not use custom code.

Description

This check displays a warning when the model contains blocks that use custom code.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Issues found by Simulink PLC Coder.	Remove the use of custom code from the **Simulation Target** pane of the model configuration settings.

Capabilities and Limitations

Exclusion of blocks and charts is not supported.

Check model tunable parameters

Check ID:mathworks.PLC.TunableParamInfCheck

Check that the model does not have tunable parameters that use the Inf values.

Description

This check displays a warning when the model contains tunable parameters that use the Inf values.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Tunable parameters have Inf elements.	Check tunable parameter settings to ensure that they do not have Inf elements.

Capabilities and Limitations

Exclusion of blocks and charts is not supported.

Check for blocks and block settings overview

These checks verify whether blocks in your model are supported for PLC code generation and whether the supported blocks have PLC coder-compatible settings. You can verify whether:

- The models use Event-Based blocks, Probe blocks, or Environment Controller blocks.
- The model Stateflow chart has a `Continuous` chart update rate.
- The model has Discrete Integrator blocks that have settings not supported for PLC code generation.
- The model contains blocks that are not supported for PLC code generation.
- The top-level subsystem has inputs and outputs when testbench generation is enabled.
- The subsystem block parameter function packaging is not set to `Nonreusable function`.
- The model does not contain Trigonometric Function blocks that have settings not supported for PLC code generation.

Check if model uses event based blocks

Check ID: `mathworks.PLC.EventBlockCheck`

Check that the model does not use Event-Based blocks.

Description

This check displays a warning when the model uses Event-Based blocks.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Blocks not supported by Simulink PLC Coder.	Design your model to avoid using blocks that are not supported.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Check if model uses probe blocks

Check ID: `mathworks.PLC.ProbeBlockCheck`

Check that the model does not use the Probe block.

Description

This check displays a warning when the model uses the Probe block.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Blocks not supported by Simulink PLC Coder.	Design your model to avoid using blocks that are not supported.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Check if model uses environment controller blocks

Check ID: `mathworks.PLC.EnvControllerBlockCheck`

Check that the model does not use the Environment Controller block.

Description

This check displays a warning when the model uses the Environment Controller block.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Blocks not supported by Simulink PLC Coder.	Design your model to avoid using blocks that are not supported.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Check Stateflow chart update

Check ID: `mathworks.PLC.ChartUpdateCheck`

Check that the model does not have Stateflow charts containing update rates set to `Continuous`.

Description

Simulink PLC Coder does not support code generation for Stateflow charts that have continuous update rates.

This check generates a warning when the model contains Stateflow charts that have continuous update rate.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Stateflow charts not supported by Simulink PLC Coder.	Change the update method of the Stateflow charts to `Discrete`.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Check issues with integrator blocks

Check ID: `mathworks.PLC.IntegratorBlockCheck`

Check that the model does not have Discrete-Time Integrator blocks that have conditions not supported for PLC code generation.

Description

This check generates a warning when the model contains Discrete-Time Integrator blocks that have conditions not supported for PLC code generation.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Blocks not supported by Simulink PLC Coder.	Modify the conditions of the Discrete-Time Integrator block to conditions that are supported by Simulink PLC Coder for code generation.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Check if model uses unsupported blocks

Check ID: `mathworks.PLC.UnsupportedBlockCheck`

Check that the model does not have blocks that are not supported for PLC code generation.

Description

This check generates a warning when the model contains blocks that are not supported for PLC code generation.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Blocks not supported by Simulink PLC Coder.	Remove blocks from your model that are not supported for PLC code generation.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Check if model can generate testbench

Check ID: `mathworks.PLC.TestBenchWithoutIOCheck`

Check that the model top-level subsystem has inputs and outputs when the generate testbench option is enabled.

Description

This check generates a warning when the generate testbench option is enabled and the top-level subsystem has no inputs and outputs.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
The top-level subsystem has no inputs and outputs. Testbench generation is not supported.	Do not enable testbench generation when the top-level subsystem has no inputs and outputs.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Check function packaging configuration

Check ID: mathworks.PLC.NonReusableSubSystemCheck

Check that the subsystem block **Function packaging** parameter is not set to `Nonreusable function`.

Description

This check generates a warning when the subsystem block **Function packaging** parameter is set to `Nonreusable function`.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Subsystem **Function packaging** is set to `Nonreusable function`, which is not supported by Simulink PLC Coder .	Change the subsystem **Function packaging** parameter to `Auto`, `Inline`, or `Reusable function`.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Check trigonometric blocks

Check ID: `mathworks.PLC.TrigonometricBlockCheck`

Check that the trigonometric functions in the model do not have settings that are not supported for PLC code generation.

Description

This check generates a warning when the trigonometric functions in the model do not have settings that are not supported for PLC code generation.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Trigonometric function blocks have settings that are not supported.	Change the trigonometric function block settings to settings that are supported by Simulink PLC Coder.

Capabilities and Limitations

- Analyzes content in the selected subsystem only.
- Exclusion of blocks and charts is not supported.

Industry standard checks overview

These checks verify whether your Simulink model conforms to the coding best practices as described in the `PLCOpen` standard and other industry standards. Use the checks in this folder to verify whether:

- Names in your model are not reserved keyword names.
- Names in your model have consistent upper case or lower case utilization.
- Subsystem names, top-level subsystem and port names, and signal and port names have the recommended number of characters in length.
- Your model has comments that describe the role of the subsystem, functions, and so on.
- Your model does not have nested comments.
- Your model subsystem inputs and outputs do not exceed the user defined maximum input and output variables.
- Your model variables use prefixes defined as part of the model PLC code generation configuration settings.

Define names to avoid

Check ID:mathworks.PLC.NamesToAvoid

Check that model does not contain names that are reserved keywords.

Description

This check displays a warning when names in the model match reserved names defined in a keywords list file.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
The model contains names that must be avoided.	Update the names displayed in the result window so that they do not match any names in the reserved keywords file and rerun the check.

Define use of case (capitals)

Check ID:`mathworks.PLC.UseOfCase`

Check that model consistently uses capital letters.

Description

This check displays a warning when names across the model use case (capitals) inconsistently.

Available with Simulink PLC Coder.

Input Parameters

To run this check, set the case style to any of these options:

- `alllowercase`
- `ALLUPPERCASE`
- `UpperCamelCase`
- `lowerCamelCase`

Results and Recommended Actions

Condition	Recommended Action
The names across the model are not consistent.	Update the names displayed in the result window to match the case style set in input parameters and rerun the check.

Define maximum variable name length

Check ID:mathworks.PLC.AcceptableNameLength

Check that a model contains names that do not exceed a predefined length.

Description

This check displays a warning when a model has names that exceed a predefined length.

Available with Simulink PLC Coder.

Input Parameters

To run this check, set the maximum variable name length in **Maximum acceptable length**.

Results and Recommended Actions

Condition	Recommended Action
This model contains names longer than the **Maximum acceptable length**.	Update the names displayed in the result window so that they do not exceed the maximum variable length set in **Maximum acceptable length** and rerun the check.

Comments must describe purpose of component

Check ID:mathworks.PLC.Comments

Check that a model component for code generation contains comments describing the purpose of the component.

Description

This check displays a warning when a model description has no comments describing its functionality.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
The function block has no comment.	Add a description to the model and rerun the check.

Avoid nested comments

Check ID:mathworks.PLC.NestedComments

Check that a model component for code generation does not contain nested comments.

Description

This check displays a warning when a model description has nested comments.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
Avoid nesting of multi-line comments.	Remove nested comments from the model description and rerun the check.

Define maximum number of input/output/in-out variables of a Program Organization Unit (POU)

Check ID:`mathworks.PLC.MaxInOut`

Check that model's input variables, output variables, and in-out variables are within a predefined limit.

Description

This check displays a warning when the number of input variables, output variables, and in-out variables of a model exceed the number of predefined maximum variables.

Available with Simulink PLC Coder.

Input Parameters

To run this check, set the maximum number of variables in **Maximum number of I/O variables**.

Results and Recommended Actions

Condition	Recommended Action
The function block has more than **Maximum number of I/O variables** variables.	Reduce the number of I/O variables and rerun the check.

Define type prefixes for variables (if used)

Check ID:mathworks.PLC.TypePrefixCheck

Check that model's data types use a predefined prefix.

Description

This check displays a warning when the model contains names that have invalid prefixes.

Available with Simulink PLC Coder.

Results and Recommended Actions

Condition	Recommended Action
The model contains names that have invalid prefixes.	Add the prefixes displayed in the results window to the variable names and rerun the check.

Using the PLC Coder Model Advisor

PLC Model Advisor Checks

In this section...
"Model configuration checks" on page 27-2
"Checks for blocks and block settings" on page 27-3
"Industry standard checks" on page 27-4

The PLC Model Advisor checks in Simulink PLC Coder verify and update your Simulink model or subsystem for compatibility with PLC code generation. The Model Advisor has checks for:

- Model Configuration settings
- Blocks and block settings
- Industry standard guidelines

When you run a check, the Model Advisor displays the result as a pass or failure. You can fix warnings or failures by using the Model Advisor recommended settings.

Model configuration checks

To prepare your model for compatibility with PLC code generation, use the checks in this folder. This folder contains checks that verify whether:

- The components in data store memory blocks resolve to `Simulink.Signal` objects.
- The model uses Stateflow messages.
- The model uses row-major algorithms.
- The model mask parameters use the `Inf` value.
- The model tunable parameters uses the `Inf` value.
- The model uses machine-parented data.
- The model has signal lines that resolve to `Simulink.Signal` objects.

Check Name	Description
"Check Data Store Memory blocks" on page 26-4	Check that the Data Store Memory block has the option to resolve the data store memory object to `Simulink.Signal` object enabled.
"Check model for Stateflow messages" on page 26-5	Check that the model does not use Stateflow messages.
"Check if signal lines are configured properly" on page 26-6	Check that the model does not have signal lines that resolve to `Simulink.Signal` objects.
"Check if model uses row-major algorithms" on page 26-7	Check that the model does not use row-major algorithms.
"Check model mask parameters" on page 26-8	Check that the model does not contain masked parameters that use the `Inf` value.
"Check if model uses machine parented data" on page 26-9	Check that the model does not contain blocks or events that use machine-parented data.
"Check if model uses custom code" on page 26-10	Check that the model does not use custom code.

"Check model tunable parameters" on page 26-11	Check that the model does not have tunable parameters that use the `Inf` values.

Checks for blocks and block settings

These checks verify whether blocks in your model are supported for PLC code generation and whether the supported blocks have PLC coder-compatible settings. You can verify whether:

- The models use Event-Based blocks, Probe blocks, or Environment Controller blocks.
- The model Stateflow chart has a `Continuous` chart update rate.
- The model has Discrete Integrator blocks that have settings not supported for PLC code generation.
- The model contains blocks that are not supported for PLC code generation.
- The top-level subsystem has inputs and outputs when testbench generation is enabled.
- The subsystem block parameter function packaging is not set to `Nonreusable function`.
- The model does not contain Trigonometric Function blocks that have settings not supported for PLC code generation.

Check Name	Description
"Check if model uses event based blocks" on page 26-13	Check that the model does not use Event-Based blocks.
"Check if model uses probe blocks" on page 26-14	Check that the model does not use the Probe block.
"Check if model uses environment controller blocks" on page 26-15	Check that the model does not use the Environment Controller block.
"Check Stateflow chart update" on page 26-16	Check that the model does not have Stateflow charts containing update rates set to `Continuous`.
"Check issues with integrator blocks" on page 26-17	Check that the model does not have Discrete-Time Integrator blocks that have conditions not supported for PLC code generation.
"Check if model uses unsupported blocks" on page 26-18	Check that the model does not have blocks that are not supported for PLC code generation.
"Check if model can generate testbench" on page 26-19	Check that the model top-level subsystem has inputs and outputs when the generate testbench option is enabled.
"Check function packaging configuration" on page 26-20	Check that the subsystem block **Function packaging** parameter is not set to `Nonreusable function`.
"Check trigonometric blocks" on page 26-21	Check that the trigonometric functions in the model do not have settings that are not supported for PLC code generation.

Industry standard checks

These checks verify whether your Simulink model conforms to the coding best practices as described in the `PLCOpen` standard and other industry standards. Use the checks in this folder to verify whether:

- Names in your model are not reserved keyword names.
- Names in your model have consistent upper case or lower case utilization.
- Subsystem names, top-level subsystem and port names, and signal and port names have the recommended number of characters in length.
- Your model has comments that describe the role of the subsystem, functions, and so on.
- Your model does not have nested comments.
- Your model subsystem inputs and outputs do not exceed the user defined maximum input and output variables.
- Your model variables use prefixes defined as part of the model PLC code generation configuration settings.

Check Name	Description
"Define names to avoid" on page 26-23	Check that model does not contain names that are reserved keywords.
"Define use of case (capitals)" on page 26-24	Check that model consistently uses capital letters.
"Define maximum variable name length" on page 26-25	Check that a model contains names that do not exceed a predefined length.
"Comments must describe purpose of component" on page 26-26	Check that a model component for code generation contains comments describing the purpose of the component.
"Avoid nested comments" on page 26-27	Check that a model component for code generation does not contain nested comments.
"Define maximum number of input/output/in-out variables of a Program Organization Unit (POU)" on page 26-28	Check that model's input variables, output variables, and in-out variables are within a predefined limit.
"Define type prefixes for variables (if used)" on page 26-29	Check that model's data types use a predefined prefix.

See Also

More About

- "Check Your Model Using the Model Advisor"

Custom Keyword List

Create Custom Target-Based Keyword List

In this section...
"Custom Keyword File Template" on page 28-2
"Custom Keyword File Usage Workflow" on page 28-19
"Verify Custom Keyword Name Changes in Generated Code" on page 28-20
"Limitations" on page 28-22

During code generation, Simulink PLC Coder uses a hook file to modify the target IDE default keyword list. Simulink PLC Coder uses keywords from the modified keyword list to check for and modify model component names that match any keywords. To create a callback hook file that contains a target IDE-specific custom keyword list, use MATLAB.

Custom Keyword File Template

The name of the custom keyword file must be `plc_custom_keyword.m`. To create a custom keyword file for a single target IDE, use this template:

```matlab
function keyword_list = plc_custom_keyword(keyword_list)
%

%    Copyright 2020 The MathWorks, Inc.

    add_list = { 'state', ...
                 'test',...
                 'control',...
               };

    delete_list = { 'jmp', ...
                    'method', ...
                    'transition', ...
                  };


    keyword_list = union(keyword_list, add_list);
    keyword_list = setdiff(keyword_list, delete_list);
end
```

To create a custom keyword file for multiple target IDEs, use this template:

```matlab
function keyword_list = plc_custom_keyword(keyword_list)
%

%    Copyright 2020 The MathWorks, Inc.

target = get_param(gcs,'PLC_TargetIDE');

    switch target
        case 'codesys23'
            add_list = {'state'};
            delete_list = {'jmp'};
        case 'pcworx60'
            add_list = {'control'};
            delete_list = {'method'};
```

```
        case 'codesys35'
            add_list = {'mykeyword3'};
            delete_list = {'time'};
        case 'omron'
            add_list = {'mykeyword'};
            delete_list = {'reset'};
        case 'rslogix5000'
            add_list = {'mykeyword1'};
            delete_list = {'retain'};
        case 'tiaportal'
            dd_list = {'mykeyword2'};
              delete_list = {'sint'};
        otherwise
            add_list = {'test'};
            delete_list = {'transition'};
    end
    keyword_list = union(keyword_list, add_list);
    keyword_list = setdiff(keyword_list, delete_list);
end
```

The input argument `keyword_list` is the default keyword list for the selected target. Modify the target IDE specific keyword list, by using the template to create `add_list` and `delete_list` lists to modify the default `keyword_list`. The keywords from the output `keyword_list` are used to match and rename model components during code generation. Refer to these default keyword lists to decide which keywords to add or remove to your custom keyword list.

These lists are the target IDE-specific default `keyword_list` lists.

Generic and PLCOpen ST Keyword List

```
keyword_list = { ...
'abs', ...
'acos', ...
'action', ...
'add', ...
'adr', ...
'adrinst', ...
'and', ...
'andn', ...
'any', ...
'array', ...
'asin', ...
'at', ...
'atan', ...
'begin', ...
'bitadr', ...
'bool', ...
'by', ...
'byte', ...
'cal', ...
'calc', ...
'calcn', ...
'case', ...
'configuration', ...
'const', ...
'constant', ...
'continue', ...
'cos', ...
```

```
'counter', ...
'date', ...
'data_and_time', ...
'dint', ...
'div', ...
'd', ...
'do', ...
'ds', ...
'dt', ...
'dword', ...
'else', ...
'elsif', ...
'en', ...
'end', ...
'end_action', ...
'end_case', ...
'end_const', ...
'end_for', ...
'end_function', ...
'end_function_block', ...
'end_if', ...
'end_program', ...
'enf_configuration', ...
'end_repeat', ...
'end_step', ...
'end_struct', ...
'end_type', ...
'end_var', ...
'end_while', ...
'eno', ...
'eq', ...
'exit', ...
'exp', ...
'expt', ...
'f_edge', ...
'false', ...
'for', ...
'function', ...
'function_block', ...
'from', ...
'ge', ...
'gt', ...
'if', ...
'indexof', ...
'ini', ...
'initial_step', ...
'int', ...
'jmp', ...
'jmpc', ...
'jmpcn', ...
'l', ...
'ld', ...
'ldn', ...
'le', ...
'lint', ...
'limit', ...
'ln', ...
'log', ...
```

```
'lreal', ...
'lt', ...
'lword', ...
'max', ...
'method', ...
'min', ...
'mod', ...
'move', ...
'mul', ...
'mux', ...
'n', ...
'ne', ...
'non_retain', ...
'not', ...
'of', ...
'on', ...
'or', ...
'orn', ...
'p', ...
'persistent', ...
'pointer', ...
'program', ...
'r', ...
'r_edge', ...
'read_only', ...
'read_write', ...
'real', ...
'repeat', ...
'reset', ...
'resource', ...
'ret', ...
'retain', ...
'retc', ...
'retcn', ...
'return', ...
'rol', ...
'ror', ...
's', ...
'sd', ...
'sel', ...
'shl', ...
'shr', ...
'sin', ...
'sint', ...
'sizeof', ...
'sl', ...
'sqrt', ...
'st', ...
'step', ...
'stn', ...
'string', ...
'struct', ...
'sub', ...
'tan', ...
'task', ...
'then', ...
'time', ...
'timer', ...
```

```
    'time_of_day', ...
    'to', ...
    'tod', ...
    'transition', ...
    'true', ...
    'trunc', ...
    'type', ...
    'udint', ...
    'uint', ...
    'ulint', ...
    'until', ...
    'usint', ...
    'var', ...
    'var_access', ...
    'var_config', ...
    'var_constant', ...
    'var_external', ...
    'var_global', ...
    'var_in_out', ...
    'var_input', ...
    'var_output', ...
    'var_temp', ...
    'while', ...
    'with', ...
    'word', ...
    'wstring', ...
    'xor', ...
    'xorn', ...
      };
```

Omron Keyword List

```
omron_list = { ...
'np', ...
'up',...
};
```

Rockwell Keyword List

```
rockwell_list = { ...
'control',...
};
```

Selectron Keyword List

```
selectron_list = { ...
'&' ...
'(' ...
')' ...
'*' ...
'**' ...
'+' ...
'-' ...
'/' ...
'<' ...
'<=' ...
'<>' ...
'=' ...
```

```
'>' ...
'>=' ...
'ACTION' ...
'ADD' ...
'AND' ...
'ANDN' ...
'ANY' ...
'ANY_BIT' ...
'ANY_DATE' ...
'ANY_DUT' ...
'ANY_FB' ...
'ANY_INT' ...
'ANY_NUM' ...
'ANY_REAL' ...
'ARRAY' ...
'AT' ...
'BODY' ...
'BOOL' ...
'BY' ...
'BYTE' ...
'CAL' ...
'CALC' ...
'CALCN' ...
'CASE' ...
'CONFIGURATION' ...
'CONSTANT' ...
'DATE' ...
'DATE_AND_TIME' ...
'DINT' ...
'DIV' ...
'DO' ...
'DT' ...
'DWORD' ...
'ELSE' ...
'ELSIF' ...
'EN' ...
'END_ACTION' ...
'END_BODY' ...
'END_CASE' ...
'END_CONFIGURATION' ...
'END_FOR' ...
'END_FUNCTION' ...
'END_FUNCTION_BLOCK' ...
'END_IF' ...
'END_PLC_CONFIG' ...
'END_PROGRAM' ...
'END_REPEAT' ...
'END_RESOURCE' ...
'END_STEP' ...
'END_STRUCT' ...
'END_TRANSITION' ...
'END_TYPE' ...
'END_VAR' ...
'END_WHILE' ...
'ENO' ...
'EQ' ...
'EXIT' ...
'FALSE' ...
```

```
'FOR' ...
'FROM' ...
'FUNCTION' ...
'FUNCTION_BLOCK' ...
'F_EDGE' ...
'GE' ...
'GT' ...
'IF' ...
'INITIAL_STEP' ...
'INT' ...
'INTERVAL' ...
'JMP' ...
'JMPC' ...
'JMPCN' ...
'LD' ...
'LDN' ...
'LE' ...
'LINT' ...
'LREAL' ...
'LT' ...
'LWORD' ...
'MOD' ...
'MUL' ...
'NE' ...
'NOT' ...
'OF' ...
'OFFSETOF' ...
'ON' ...
'OR' ...
'ORN' ...
'PLC_CONFIG' ...
'PRIORITY' ...
'PROGRAM' ...
'R' ...
'R1' ...
'READ_ONLY' ...
'READ_WRITE' ...
'REAL' ...
'REPEAT' ...
'RESOURCE' ...
'RET' ...
'RETAIN' ...
'RETC' ...
'RETCN' ...
'RETURN' ...
'R_EDGE' ...
'S' ...
'S1' ...
'SINGLE' ...
'SINT' ...
'SIZEOF' ...
'ST' ...
'STEP' ...
'STN' ...
'STRING' ...
'STRUCT' ...
'SUB' ...
'TASK' ...
```

```
'THEN' ...
'TIME' ...
'TIME_OF_DAY' ...
'TO' ...
'TOD' ...
'TRANSITION' ...
'TRUE' ...
'TYPE' ...
'UDINT' ...
'UINT' ...
'ULINT' ...
'UNTIL' ...
'USINT' ...
'VAR' ...
'VAR_ACCESS' ...
'VAR_CONSTANT' ...
'VAR_CONSTANT_RETAIN' ...
'VAR_EXTERNAL' ...
'VAR_EXTERNAL_CONSTANT' ...
'VAR_EXTERNAL_CONSTANT_RETAIN' ...
'VAR_EXTERNAL_RETAIN' ...
'VAR_GLOBAL' ...
'VAR_GLOBAL_CONSTANT' ...
'VAR_GLOBAL_CONSTANT_RETAIN' ...
'VAR_GLOBAL_RETAIN' ...
'VAR_INPUT' ...
'VAR_INPUT_RETAIN' ...
'VAR_IN_EXT' ...
'VAR_IN_OUT' ...
'VAR_IN_OUT_CONSTANT' ...
'VAR_OUTPUT' ...
'VAR_OUTPUT_RETAIN' ...
'VAR_RETAIN' ...
'VAR_TEMP' ...
'WHILE' ...
'WITH' ...
'WORD' ...
'XOR' ...
'XORN' ...
'_ACTION' ...
'_SFC_DEBUG' ...
'_STEP' ...
'auto' ...
'break' ...
'char' ...
'const' ...
'continue' ...
'default' ...
'double' ...
'enum' ...
'extern' ...
'float' ...
'goto' ...
'if' ...
'inline' ...
'long' ...
'register' ...
'restrict' ...
```

```
    'short' ...
    'signed' ...
    'static' ...
    'switch' ...
    'typedef' ...
    'union' ...
    'unsigned' ...
    'void' ...
    'volatile' ...
};
```

Siemens STEP7 Keyword List

```
step7_keyword_list = { ...
    'fb', ...
    'db', ...
    'ob', ...
    'fc', ...
    'ib', ...
    'mb', ...
    'udt', ...
    'di', ...
    'scale', ...
    'B', ...
    'ref', ...
    'switch', ...
    'norm', ...
    'set', ...
    'ss'
};
```

Siemens TIA Portal Keyword List

```
tia_portal_keyword_list = { ...
'a_dead_b', ...
'abs_ctrl', ...
'abs_diag', ...
'abs_init', ...
'act_tint', ...
'ag_cntex', ...
'ag_cntrl', ...
'ag_lock', ...
'ag_recv', ...
'ag_send', ...
'ag_unlock', ...
'alarm_d', ...
'alarm_dq', ...
'alarm_s', ...
'alarm_sc', ...
'alarm_sq', ...
'analog', ...
'as_dial', ...
'asi_3422', ...
'asi_ctrl', ...
'asin', ...
'atan', ...
'ath', ...
'att', ...
```

```
'attach', ...
'attr_db', ...
'bcdcpl', ...
'bitsum', ...
'blkmov', ...
'brcv', ...
'bsend', ...
'bt_lt', ...
'by', ...
'c_cntrl', ...
'cam_ctrl', ...
'cam_diag', ...
'cam_init', ...
'can_dint', ...
'can_tint', ...
'cdt', ...
'ceil', ...
'ch_diag', ...
'chars_to_strg', ...
'cir', ...
'cj_t_par', ...
'cnt2_ctr', ...
'cnt2rdpn', ...
'cnt2wrpn', ...
'cnt_ctl1', ...
'cnt_ctl2', ...
'cnt_ctrl', ...
'compress', ...
'concat', ...
'concat_date_ltod', ...
'concat_date_tod', ...
'cont_c', ...
'cont_s', ...
'convert', ...
'count', ...
'countofelements', ...
'crea_dbl', ...
'creat_db', ...
'create_db', ...
'crp_in', ...
'crp_out', ...
'ctd', ...
'ctrl_rtm', ...
'ctu', ...
'ctud', ...
'd_act_dp', ...
'datalogclear', ...
'datalogclose', ...
'datalogcreate', ...
'datalogdelete', ...
'datalognewfile', ...
'datalogopen', ...
'datalogwrite', ...
'db', ...
'db_any_to_variant', ...
'dcat', ...
'dead_t', ...
'deadband', ...
```

```
            'deco', ...
            'del_db', ...
            'del_si', ...
            'delete', ...
            'delete_db', ...
            'demux', ...
            'deserialize', ...
            'detach', ...
            'dev', ...
            'devicestates', ...
            'diag_inf', ...
            'diag_rd', ...
            'dif', ...
            'digital', ...
            'dis_airt', ...
            'dis_irt', ...
            'div', ...
            'dmsk_flt', ...
            'do', ...
            'dp_ctrl', ...
            'dp_diag', ...
            'dp_recv', ...
            'dp_send', ...
            'dp_topol', ...
            'dpnrm_dg', ...
            'dprd_dat', ...
            'dpsyc_fr', ...
            'dpwr_dat', ...
            'drum', ...
            'drum_x', ...
            'eb', ...
            'en', ...
            'en_airt', ...
            'en_irt', ...
            'enco', ...
            'encoderabssensordp', ...
            'encoderet200s1count', ...
            'encoderet200s1ssi', ...
            'encoderfm350', ...
            'encoderim174', ...
            'encoderim178', ...
            'encodersinamics', ...
            'encodersm338', ...
            'encoderuniversal', ...
            'endis_pw', ...
            'eno', ...
            'err_mon', ...
            'exit', ...
            'fb', ...
            'f_trig', ...
            'failsafe_protect', ...
            'fifo', ...
            'fill', ...
            'fill_blk', ...
            'find', ...
            'floor', ...
            'fmt_cj_t', ...
            'fmt_ds1', ...
```

```
'fmt_par', ...
'fmt_pid', ...
'fmt_pv', ...
'fmt_tun', ...
'force_355', ...
'frac', ...
'frequenc', ...
  'ftp_cmd', ...
'fuz_355', ...
'gb', ...
'gadr_lgc', ...
'gen_diag', ...
'gen_usrmsg', ...
'geo2log', ...
'geo_log', ...
'get', ...
'get_alarmstate', ...
'get_diag', ...
'get_err_id', ...
'get_error', ...
'get_features', ...
'get_im_data', ...
'get_name', ...
'get_s', ...
'getblockname', ...
'getinstancename', ...
'getinstancepath', ...
'getio', ...
'getio_part', ...
'getstationinfo', ...
'getsymbolname', ...
'getsymbolpath', ...
'high_speed_counter', ...
'hta', ...
'i_abort', ...
'i_get', ...
'i_put', ...
'imc', ...
'init_rd', ...
'insert', ...
'integ', ...
'inventory', ...
'io2mod', ...
'ip_config', ...
'is_array', ...
'join', ...
'lag1st', ...
'lag2nd', ...
'lead_lag', ...
'led', ...
'left', ...
'len', ...
'lgc_gadr', ...
'lifo', ...
'limalarm', ...
'limit', ...
'limiter', ...
'ln', ...
```

```
'loc_time', ...
'log2geo', ...
'log2mod', ...
'log_geo', ...
'logical_trigger', ...
'lp_sched', ...
'lp_sched_m', ...
'lt_bt', ...
'max', ...
'max_len', ...
'mb_client', ...
'mb_server', ...
'mc_control', ...
'mc_gearin', ...
'mc_halt', ...
'mc_home', ...
'mc_init', ...
'mc_moveabsolute', ...
'mc_movejog', ...
'mc_moverelaive', ...
'mc_moverelative', ...
'mc_movevelocity', ...
'mc_power', ...
'mc_reset', ...
'mc_simulation', ...
'mc_stopmotion', ...
'mcat', ...
'md', ...
'mid', ...
'mn', ...
'mod', ...
'modb_341', ...
'modbus_comm_load', ...
'modbus_master', ...
'modbus_slave', ...
'modulestates', ...
'move_blk', ...
'move_blk_variant', ...
'msk_flt', ...
'mux', ...
'none', ...
'nonlin', ...
'norm', ...
'norm_x', ...
'not', ...
'notify', ...
'null', ...
'ob', ...
'of', ...
'or', ...
'outputet200s2ao', ...
'outputim174', ...
'outputim178', ...
'outputmm4_dp', ...
'outputsinamics', ...
'outputsm332', ...
'outputuniversal', ...
'override', ...
```

```
'pb', ...
'p3964_config', ...
'p_print', ...
'p_prt341', ...
'p_rcv', ...
'p_rcv_rk', ...
'p_reset', ...
'p_send', ...
'p_snd_rk', ...
'pack', ...
'para_ctl', ...
'parm_mod', ...
'pe_cmd', ...
'pe_cmd_cp', ...
'pe_ds3_write_et200s', ...
'pe_ds3_write_et200s_cp', ...
'pe_end_rsp', ...
'pe_error_rsp', ...
'pe_get_mode_rsp', ...
'pe_i_dev', ...
'pe_i_dev_cp', ...
'pe_identify_rsp', ...
'pe_list_modes_rsp', ...
'pe_measurement_list_rsp', ...
'pe_measurement_value_rsp', ...
'pe_pem_status_rsp', ...
'pe_start_end', ...
'pe_start_end_cp', ...
'pe_start_rsp', ...
'pe_wol', ...
'peek', ...
'peek_bool', ...
'pg_dial', ...
'pid', ...
'pid_3step', ...
'pid_compact', ...
'pid_cp', ...
'pid_es', ...
'pid_fm', ...
'pid_par', ...
'pid_temp', ...
'pip', ...
'pnio_alarm', ...
'pnio_recv', ...
'pnio_rw_rec', ...
'pnio_send', ...
'poke', ...
'poke_blk', ...
'poke_bool', ...
'port', ...
'port_config', ...
'preset_timer', ...
'program_alarm', ...
'protect', ...
'prvrec', ...
'pulse', ...
'pulsegen', ...
'pulsegen_m', ...
```

```
'put', ...
'qry_cint', ...
'qry_dint', ...
'qry_tint', ...
'r_trig', ...
'ralrm', ...
'rcvrec', ...
'rd_addr', ...
'rd_dpar', ...
'rd_dpara', ...
'rd_dparm', ...
'rd_lgadr', ...
'rd_loc_t', ...
'rd_rec', ...
'rd_sinfo', ...
'rd_sys_t', ...
'rdrec', ...
'rdsysst', ...
're_trigr', ...
'read', ...
'read_355', ...
'read_big', ...
'read_dbl', ...
'read_err', ...
'read_little', ...
'read_rtm', ...
'read_si', ...
'readfromarraydb', ...
'readfromarraydbl', ...
'receive_config', ...
'receive_p2p', ...
'receive_reset', ...
'recipeexport', ...
'recipeimport', ...
'reconfigiosystem', ...
'repl_val', ...
'replace', ...
'reset', ...
'reset_timer', ...
'reseti', ...
'return', ...
'right', ...
'rmp_soak', ...
'roc_lim', ...
'rol', ...
'ror', ...
'round', ...
'rt_info', ...
'rtm', ...
'runtime', ...
's_cd', ...
's_conv', ...
's_cu', ...
's_cud', ...
's_ltint', ...
's_modb', ...
's_odt', ...
's_odts', ...
```

```
's_offdt', ...
's_pext', ...
's_pulse', ...
's_rcv', ...
's_rts', ...
's_send', ...
's_ussi', ...
's_ussr', ...
's_usst', ...
's_v24', ...
's_vset', ...
's_vstat', ...
's_xon', ...
'scale', ...
'scale_m', ...
'scale_x', ...
'seg', ...
'sel', ...
'send_config', ...
'send_p2p', ...
'serialize', ...
'set', ...
'set_addr', ...
'set_cint', ...
'set_clks', ...
'set_features', ...
'set_param', ...
'set_rtm', ...
'set_sw', ...
'set_sw_s', ...
'set_timezone', ...
'set_tint', ...
'set_tintl', ...
'seti', ...
'setio', ...
'setio_part', ...
'shl', ...
'shr', ...
'shrb', ...
'signal_get', ...
'signal_set', ...
'sin', ...
'smc', ...
'sms_send', ...
'snc_rtcb', ...
'sp_gen', ...
'split', ...
'splt_ran', ...
'sqr', ...
'sqrt', ...
'srt_dint', ...
'stp', ...
'strg_to_chars', ...
'swap', ...
'switch', ...
'sync_pi', ...
'sync_po', ...
't_add', ...
```

```
't_combine', ...
't_comp', ...
't_config', ...
't_conv', ...
't_diag', ...
't_diff', ...
't_reset', ...
't_sub', ...
'tbl', ...
'tbl_find', ...
'tbl_tbl', ...
'tbl_wrd', ...
'tcon', ...
'tcont_cp', ...
'tcont_s', ...
'tdiscon', ...
'test_db', ...
'this', ...
'time_tck', ...
'timestmp', ...
'tmail_c', ...
'to', ...
'tof', ...
'ton', ...
'tonr', ...
'tonr_x', ...
'tp', ...
'trcv', ...
'trcv_c', ...
'true', ...
'trunc', ...
'tsend', ...
'tsend_c', ...
'tun_ec', ...
'tun_es', ...
'turcv', ...
'tusend', ...
'typeof', ...
'typeofelements', ...
'ublkmov', ...
'ufill_blk', ...
'umove_blk', ...
'unscale', ...
'until', ...
'updat_pi', ...
'updat_po', ...
'urcv', ...
'urcv_s', ...
'usend', ...
'usend_s', ...
'uss_drive_control', ...
'uss_port_scan', ...
'uss_read_param', ...
'uss_write_param', ...
'v24_set', ...
'v24_set_340', ...
'v24_stat', ...
'v24_stat_340', ...
```

```
'variant_to_db_any', ...
'variantget', ...
'variantput', ...
'wait', ...
'wr_dparm', ...
'wr_loc_t', ...
'wr_parm', ...
'wr_rec', ...
'wr_sys_t', ...
'wr_usmsg', ...
'wrd_tbl', ...
'writ_dbl', ...
'write', ...
'write_big', ...
'write_little', ...
'writetoarraydb', ...
'writetoarraydbl', ...
'wrrec', ...
'wsr', ...
'www', ...
'x_abort', ...
'x_get', ...
'x_put', ...
'x_rcv', ...
'x_send', ...
'B', ...
'ref', ...
'ss', ...
};
```

Custom Keyword File Usage Workflow

This flowchart displays the process of using the custom keyword file.
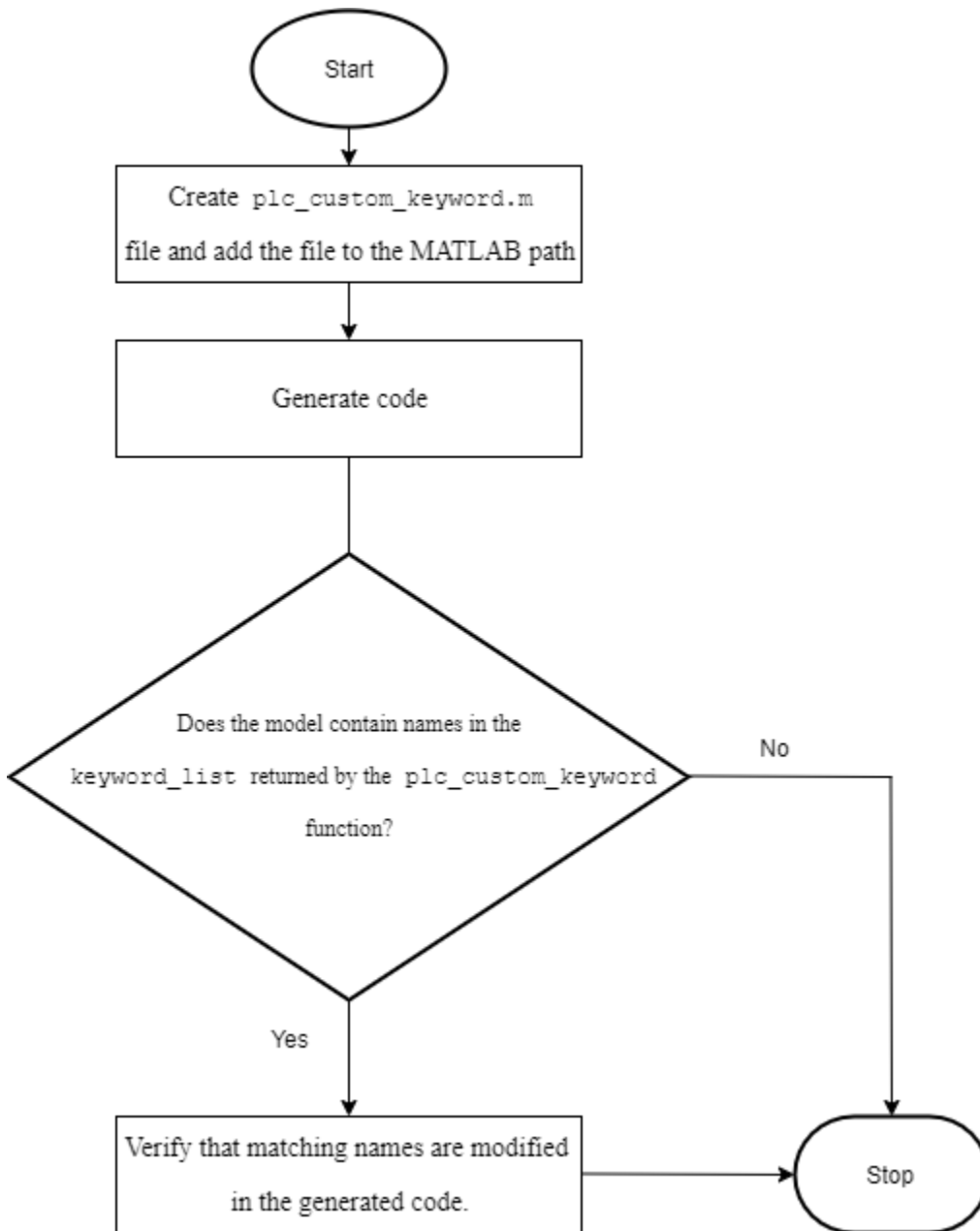
You must add the `plc_custom_keyword.m` file to the MATLAB path for custom keyword checks to work.

Verify Custom Keyword Name Changes in Generated Code

This example shows how to create a custom keyword file and verify that the generated code contains variables with names that have changed because they matched names in the keyword list.

1   Open the `plcdemo_simple_subsystem` example.

2   Open the `SimpleSubsystem` block. Change the input variable name from `U` to `Controller`.
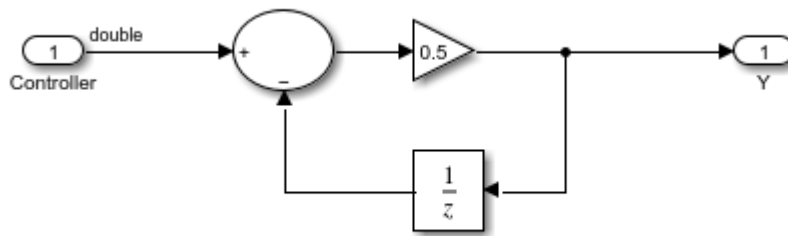
3 Create the `plc_custom_keyword.m` file by using this code:

```
function keyword_list = plc_custom_keyword(keyword_list)
%

%    Copyright 2020 The MathWorks, Inc.

    add_list = { 'state', ...
                 'test',...
                 'controller',...
               };

    delete_list = { 'jmp', ...
                    'method', ...
                    'transition', ...
                  };
    keyword_list = union(keyword_list, add_list);
    keyword_list = setdiff(keyword_list, delete_list);
end
```

4 Add the `custom_plc_keyword.m` file to the MATLAB path. Open the PLC Coder app. On the **PLC Code** tab, click **Generate PLC Code**.

5 Open the generated code file. Verify that `Controller` is changed to `b_Controller`.

```
1    (*
2     *
3     * File: plcdemo_simple_subsystem.exp
4     *
5     * IEC 61131-3 Structured Text (ST) code generated for subsystem "plcdemo_simple_subsystem/SimpleSubsystem"
6     *
7     * Model name                    : plcdemo_simple_subsystem
8     * Model version                 : 4.1
9     * Model creator                 : The MathWorks, Inc.
10    * Model last modified by        : skapali
11    * Model last modified on        : Sat Oct 31 16:50:18 2020
12    * Model sample time             : 0.1s
13    * Subsystem name                : plcdemo_simple_subsystem/SimpleSubsystem
14    * Subsystem sample time         : 0.1s
15    * Simulink PLC Coder version    : 3.4 (R2021a) 27-Oct-2020
16    * ST code generated on          : Sat Oct 31 16:52:47 2020
17    *
18    * Target IDE selection          : 3S CoDeSys 2.3
19    * Test Bench included           : No
20    *
21    *)
22    FUNCTION_BLOCK SimpleSubsystem
23    VAR_INPUT
24        ssMethodType: SINT;
25        b_Controller: LREAL;
26    END_VAR
27    VAR_OUTPUT
28        Y: LREAL;
29    END_VAR
30    VAR
31        UnitDelay_DSTATE: LREAL;
32    END_VAR
33    CASE ssMethodType OF
34        SS_INITIALIZE:
35            (* SystemInitialize for Atomic SubSystem: '<Root>/SimpleSubsystem' *)
36            (* InitializeConditions for UnitDelay: '<S1>/Unit Delay' *)
37            UnitDelay_DSTATE := 0.0;
38            (* End of SystemInitialize for SubSystem: '<Root>/SimpleSubsystem' *)
39        SS_STEP:
40            (* Outputs for Atomic SubSystem: '<Root>/SimpleSubsystem' *)
41            (* Gain: '<S1>/Gain' incorporates:
42             *  Sum: '<S1>/Sum'
43             *  UnitDelay: '<S1>/Unit Delay' *)
44            Y := (b_Controller - UnitDelay_DSTATE) * 0.5;
45            (* Update for UnitDelay: '<S1>/Unit Delay' *)
46            UnitDelay_DSTATE := Y;
47            (* End of Outputs for SubSystem: '<Root>/SimpleSubsystem' *)
48    END_CASE;
49    END_FUNCTION_BLOCK
50    VAR_GLOBAL CONSTANT
51        SS_INITIALIZE: SINT := 0;
52        SS_STEP: SINT := 1;
53    END_VAR
54
```

Limitations

The output of the `custom_plc_keyword.m` file must be of the data type cell array of character vectors.

Plugin Based Targets

- "Create Custom Target IDE for Code Generation" on page 29-2
- "Generate Custom Code by Using IDE—Specific Callback Functions" on page 29-18

Create Custom Target IDE for Code Generation

In this section...
"Plugin-Based Code Generation Workflow" on page 29-2
"Plugin Options" on page 29-5
"Generate Code by Using Plugin-Based Target IDE" on page 29-15

Simulink PLC Coder currently supports plugin-based target IDEs such as Selectron CAP1131, Omron Sysmac Studio, and so on. For your plugin-based custom target IDEs that are not supported, generate PLCOpen XML-or ASCII-compliant structured text code. Customize the generated code to meet your target IDE requirements, by leveraging the built-in plugin options in Simulink PLC Coder.

Plugin-Based Code Generation Workflow

To generate code for your custom plugin-based target IDEs:

```
        ┌─────────┐
        │  Start  │
        └────┬────┘
             │
             ▼
      ╱Create a custom╲      No
      ╲   target?    ╱ ──────────────────────────────►
             │ Yes
             ▼
  ┌──────────────────────────┐
  │ Create a folder with the │
  │ target IDE name on the   │
  │ MATLAB search path       │
  └──────────┬───────────────┘
             ▼
  ┌──────────────────────────┐
  │ Create plc_custom_ide.m  │
  └──────────┬───────────────┘
             ▼
  ┌──────────────────────────┐
  │ Optimize generated code  │
  │ by using built-in plugin │
  │ options                  │
  └──────────┬───────────────┘
```

Start

Create a custom target? — No

Yes

Create a folder with the target IDE name on the MATLAB search path

Create plc_custom_ide.m

Optimize generated code by using built-in plugin options

Need for preprocessing?   — No

Need for post processing?   — No

Yes

Yes

Add
plc_precg_callback_targetIDE.m

Add
plc_postcg_callback_targetIDE.m

Stop

Follow the flowchart to create the `plc_custom_ide.m` and if required the `plc_precg_callback_IDEname.m`, and `plc_postcg_callback_IDEname.m` files. To use these files to create and generate code for your custom target IDE, see "Generate Code by Using Plugin-Based Target IDE" on page 29-15.

Create plc_custom_ide.m file

Create `plc_custom_ide.m` by using this template:

```
function plc_ide_list = plc_custom_ide
%   Copyright 2012-2021 The MathWorks, Inc.
    plc_ide_list(1) = get_ide_info_myplcopen;
end

function ide_info = get_ide_info_myplcopen
    ide_info.name = 'myplcopen';
    ide_info.description = 'My PLCopen XML';
    ide_info.path = ''; % IDE path
    ide_info.format = 'xml'; % generic|xml
    ide_info.fileExtension = 'xml';
    ide_info.cfg = get_ide_cfg_myplcopen;
    ide_info.precg_callback = 'plc_precg_callback_myplcopen';
    ide_info.postcg_callback = 'plc_postcg_callback_myplcopen';
    ide_info.xmltree_callback = PLCCoder.PLCCGMgr.PLC_PLUGIN_CG_CALLBACK_EMPTY;
    ide_info.pluginVersion = 2.2;
    ide_info.compatibleBuildVersion = 1.6;
end

function cfg = get_ide_cfg_myplcopen
    cfg.fConvertDoubleToSingle = true;
    cfg.fConvertNamedConstantToInteger = true;
    cfg.fConvertEnumToInteger = true;
    cfg.fConvertOutputInitValueToAssignment = true;
    cfg.fConvertTunableParamToInputVariable = true;
    cfg.fSimplifyFunctionCallExpr = true;
    cfg.fConvertOutputInitValueToAssignment = true;
end
```

- Set the name of your target IDE, by using `ide_info.name`. The **Target IDE** configuration setting displays the name set in `ide_info.description`.

- If your target IDE is compliant with Generic ST standards, set `ide_info.format = 'generic'`. If your target IDE is compliant with PLCOpen XML, set `ide_info.format = 'xml'`.

- Set where the generated code files are placed by using `ide_info.path`.

- Set the extension for your target IDE files by using `ide_info.fileExtension`.

In the `function_cfg` section of the file, set your plugin options. To enable the plugin, set the plugin option to `true`. For example, `cfg.fArrayInitialValueBrackets = true;` enables the plugin. To disable the plugin, set the plugin option to `false`. To decide which plugin options you need in the `plc_custom_ide.m` file, see "Plugin Options" on page 29-5.

Create Callback Files

If your custom IDE requires pre- and post-code generation processing, create `plc_precg_callback_IDEname.m` and `plc_postcg_callback_IDEname.m` files.

Preprocessing Callback File

The `plc_precg_callback_IDEname.m` file provides access to the intermediate generated code representation. The intermediate code is stored in the `controller` struct data type, which contains information such as name of the subsystem block, inputs, outputs, generated code body, and so on. This callback is executed before the target-specific emitter function is called. To create a `plc_precg_callback_IDEname.m` file, use this template:

```
function controller = ...
plc_precg_callback_myplcopen(controller)
```

```
%   Copyright 2012-2020 The MathWorks, Inc.

    % do modifications to the controller struct here, f.ex.:
    for i = 1:length(controller.components)
        controller.components(i).body = sprintf('<<header_placeholder>>\r\n%s',controller.components(i).body);
    end

end
```

Postprocessing Callback File

The `plc_postcg_callback_IDEname.m` file provides access to the generated code file. Use this file to make the generated code match the syntax requirements for your custom target IDE. This file reads in and makes changes to the generated code file, which is read in as a string file. To create a `plc_postcg_callback_IDEname.m` file, use this template:

```
function generatedFiles = plc_postcg_callback_myplcopen(fileNames)
%   Copyright 2012-2020 The MathWorks, Inc.

    fileName = fileNames{1};
    str = fileread(fileName);
%   do modifications to str here, f.ex.:
%   str = regexprep(str,'BOOL_TO_LREAL','BOOL_TO_INT');
%   str = regexprep(str,'<USINT/>','<INT/>');
%   str = regexprep(str, 'END_STRUCT','END_STRUCT;');

    [sHeader,eHeader] = regexp(str,'\(\(*.*?\)\*\)');
    header = str(sHeader:eHeader);

    str = regexprep(str,'<<header_placeholder>>',header);

    sfprivate ('str2file', str, fileName);
    generatedFiles = {fileName};
end
```

Plugin Options

Generate custom code for your custom target IDE, by selecting from the plugin options listed in the table.

Plugins for Data Type Transformation

Plugin Name	Plugin Purpose	When to Use Plugin	Effect of Plugin on Generated Code

`fConvertBooleanCast`	Convert boolean type cast function to an if-else assignment.	If your target IDE does not support boolean type cast function or operator.	Generated code with plugin disabled:
			
			`Out1 := DINT_TO_INT(BOOL_TO_DINT(In1) +... BOOL_TO_DINT(In2));`
			
			Generated code with plugin enabled:
			
			`IF In1 THEN`
			`    temp1 := DINT#1;`
			`ELSE`
			`    temp1 := DINT#0;`
			`END_IF;`
			
			`IF In2 THEN`
			`    temp2 := DINT#1;`
			`ELSE`
			`    temp2 := DINT#0;`
			`END_IF;`
			
			`Out1 := DINT_TO_INT(temp1 + temp2);`

| fConvertDoubleToSingle | Convert double data type to single data type. | If your target IDE does not support double data types.

The generated code double data type variables are converted to single data types. A warning message is generated during code generation that data types that are not supported have been found and converted. LREAL data types are replaced by REAL data types.

The values in the generated code may be different from the simulation values. | Generated code with plugin disabled:

```VAR_INPUT\n   ssMethodType: SINT;\n   U: LREAL;\nEND_VAR\nVAR_OUTPUT\n   Y: LREAL;\nEND_VAR```

Generated code with plugin enabled:

```VAR_INPUT\n   ssMethodType: SINT;\n   U: REAL;\nEND_VAR\nVAR_OUTPUT\n   Y: REAL;\nEND_VAR``` |

`fConvertDoubleToSingleEmitter`	Convert double data types to single data types.	If your target IDE does not support double data types and you want to preserve values after conversion.
		
		The generated code double data type variables are converted to single data types. A warning message is generated during code generation that data types that are not supported have been found and converted. LREAL data types are replaced by REAL data types.
		
		The values in the generated code match the simulation values unless they exceed the REALMAX bounds.
			Generated code with plugin disabled:
			
			`VAR_INPUT`
			`    ssMethodType: SINT;`
			`    U: LREAL;`
			`END_VAR`
			`VAR_OUTPUT`
			`    Y: LREAL;`
			`END_VAR`
			
			Generated code with plugin enabled:
			
			`VAR_INPUT`
			`    ssMethodType: SINT;`
			`    U: REAL;`
			`END_VAR`
			`VAR_OUTPUT`
			`    Y: REAL;`
			`END_VAR`

| fConvertEnumInteger | Convert enum data types to integer data types. | If your target IDE does not support enum data types. | Generated code with plugin disabled for a target IDE that supports enum data type:<br><br>```
VAR_TEMP
    rtb_Switch: myEnum;
    in: myEnum;
END_VAR
```<br><br>Generated code with plugin enabled:<br><br>```
VAR_TEMP
    rtb_Switch: DINT;
    in: DINT;
END_VAR
``` |
|---|---|---|---|
| fConvertUnsignedIntToSignedInt | Converts unsigned integer to signed integer. | If your target does not support unsigned integer data type. | Generated code with plugin disabled:<br><br>```
FUNCTION_BLOCK Subsystem
VAR_INPUT
    In1: UDINT;
    In2: UDINT;
END_VAR
VAR_OUTPUT
    Out1: UDINT;
END_VAR
```<br><br>Generated code with plugin enabled:<br><br>```
FUNCTION_BLOCK Subsystem
VAR_INPUT
    In1: DINT;
    In2: DINT;
END_VAR
VAR_OUTPUT
    Out1: DINT;
END_VAR
``` |
| fInt32AsBaseInt | Sets int32 data type as the default integer data type. | Setting int32 as the default internal integer data type might reduce the number of type cast operations in the generated code. | Generated code with plugin disabled:<br><br>```
FUNCTION_BLOCK Subsystem
VAR_INPUT
    In1: SINT;
    In2: SINT;
END_VAR
VAR_OUTPUT
    Out1: SINT;
END_VAR
Out1 := DINT_TO_SINT(SINT_TO_DINT(In1) +...
 SINT_TO_DINT(In2));
END_FUNCTION_BLOCK
```<br><br>The generated code contains additional data type conversion code.<br><br>Generated code with plugin enabled:<br><br>```
FUNCTION_BLOCK Subsystem
VAR_INPUT
    In1: DINT;
    In2: DINT;
END_VAR
VAR_OUTPUT
    Out1: DINT;
END_VAR
Out1 := In1 + In2));
END_FUNCTION_BLOCK
```<br><br>The generated code does not contain additional data type conversion code. |

`fEmitEnumTypeInte` `gerValue`	Displays the enum value and corresponding integer value in the generated code.	To display the enum values and their matching integer values in the generated code.	Generated code with plugin disabled: `TYPE PLCCommandState:` `    (FILL, HOLD, EMPTY, ACTIVATE);` `END_TYPE` `TYPE PLCVesselState:` `    (EMPTIED, NOT_FULL, FULL);` `END_TYPE` `TYPE PLCValveState:` `    (SHUT, OPEN);` `END_TYPE` Generated code with plugin enabled: `TYPE PLCCommandState:` `    (FILL:=0, HOLD:=1, EMPTY:=2, ACTIVATE:=3);` `END_TYPE` `TYPE PLCVesselState:` `    (EMPTIED:=0, NOT_FULL:=1, FULL:=2);` `END_TYPE` `TYPE PLCValveState:` `    (SHUT:=0, OPEN:=1);` `END_TYPE`

Plugins for Syntax Change

Plugin Name	Plugin Purpose	When to Use Plugin	Effect of Plugin on Generated Code
`fArrayInitialValu` `eBrackets`	Encloses array initialization in the declaration area within brackets.	If your target IDE requires enclosing array initialization in the declaration area in brackets.	Generated code with plugin disabled: `EnableSetpoint_ZCE: ARRAY [0..2] OF USINT:=3,3,3` Generated code with plugin enabled:: `EnableSetpoint_ZCE: ARRAY [0..2] OF USINT:=[3,3,3]`
`fConvertAggregate` `InitValueToAssign` `ment`	Converts initial values for aggregate data types to an assignment statement.	Target IDE does not support array initialization in the declaration area.	Generated code with plugin disabled: `ARRAY [0..1] OF LREAL := LREAL#0.0,LREAL#0.0998` Generated code with plugin enabled: `tb_U[0] := 0.0;` `tb_U[1] := 0.0998;`
`fConvertAggregate` `TypeFunctionToFB`	Converts functions with aggregate data types to a function block (FB).	Target IDE supports aggregate data types for function blocks only.	Generated code with plugin disabled: `function foo(...):ARRAY [0..10] OF LREAL` Generated code with plugin enabled: `FUNCTION_BLOCK foo` `VAR_OUTPUT` `out1: ARRAY [0..10] OF LREAL;` `END_VAR`

fConvertFunctionT oFB	Convert function to function block (FB).	Target IDE does not support FUN notation but supports FB notation.	

fConvertOutputIni tValueToAssignmen t	Convert output variable initialization to an assignment.	Target IDE does not allow initial value definition and requires an assignment statement.	Generated code with plugin disabled:
			
			`FUNCTION_BLOCK foo`
			`VAR_OUTPUT`
			`somevalue: DINT := 100;`
			`END_VAR`
			
			Generated code with plugin enabled:
			
			`FUNCTION_BLOCK foo`
			`VAR_OUTPUT`
			`somevalue: DINT;`
			`END_VAR`
			`somevalue := 100;`

fEmitVarDeclarati onBeforeDescripti on	Toggles whether the variable description appears before or after the variable declaration.	Target IDE requires variable declaration before the variable description.	Generated code with plugin disabled:
			
			`VAR_GLOBAL`
			`    (* Description: Gain value two`
			`    *)`
			`    K2: REAL := 0.2;`
			`END_VAR`
			
			Generated code with plugin enabled:
			
			`VAR_GLOBAL`
			`    K2: REAL := 0.2;`
			`    (* Description: Gain value two`
			`    *)`
			`END_VAR`

fErrorOnTrailingU S	Code generation fails when it encounters variable names with a trailing underscore.	Target IDE does not support names with a trailing underscore.	Code generation fails with this message `$name$ has a trailing '_'. 'IDE name' names must not end with '_'.`

fHoistArrayIndexE xprs	Moves expressions out of array indices and creates a temporary variable for the expression.	Target IDE does not support expressions for array indices.	Generated code with plugin disabled:
			`EnvCur[TRUNC(j) - 1]`
			
			Generated code with plugin enabled:
			`temp1 := TRUNC(j) - 1;`
			`EnvCur[temp1]`

fSimplifyFunction CallExpr	Simplifies the function call for simple functions.	Target IDE does not allow assignment expressions in function calls.	Generated code with plugin disabled:
			
			`y := simplefunction(u_0 := u);`
			
			Generated code with plugin enabled:
			
			`y := simplefunction(u);`

fUseQualifiedType Constant	Appends data type to constant declaration.	Target IDE requires data type for constants.	Generated code with plugin disabled:
			
			`a := 11;`
			
			Generated code with plugin enabled:
			
			`a := DINT#11;`

Plugins for Interface Changes

Plugin Name	Plugin Purpose	When to Use Plugin	Effect of Plugin on Generated Code

| fConvertTunablePa ramToInputVariabl e | Converts tunable parameters to function block (FB) input variables. | You want to convert tunable parameters to function block inputs. This allows you to call a POU with different parameter sets. | Generated code with plugin disabled:

```FUNCTION_BLOCK Tunable_Param_to_Input   24   VAR_INPUT   25   ssMethodType: SINT;   26   Input1: REAL;   27   END_VAR   28   VAR_OUTPUT   29   Output1: REAL;   30   END_VAR   31   VAR   32   DSTATE: REAL;   33   END_VAR   34   'ST'   35   BODY   36   CASE ssMethodType OF   37   0:   40   UnitDelay_DSTATE := 0.0;   41   42   1:   43   Output1 := (Input1 - DSTATE) *   TunableParam;   48   DSTATE := Output1;   50   END_CASE;   52   END_BODY   53   END_FUNCTION_BLOCK   54```

The `TunableParam` variable is not declared as an input to the function block.

Generated code with plugin enabled:

```FUNCTION_BLOCK Tunable_Param_to_Input   24   VAR_INPUT   25   TunableParam: LREAL;   26   ssMethodType: SINT;   27   Input1: REAL;   28   END_VAR   29   VAR_OUTPUT   30   Output1: REAL;   31   END_VAR   32   VAR   33   DSTATE: REAL;   34   END_VAR   35   'ST'   36   BODY   37   CASE ssMethodType OF   38   0:   39   DSTATE := 0.0;   42   43   1:   44   Output1 := (Input1 - DSTATE) *   TunableParam;   49   DSTATE := Output1;   51   52   END_CASE;   53   END_BODY   54   END_FUNCTION_BLOCK   55```

The `TunableParam` variable is declared as an input to the function block. |

`fDefineFBExternalConstVariable`	Defines external variables in `VAR_GLOBAL CONSTANT`.	Target IDE requires declaration of external constant variables in `VAR_GLOBAL CONSTANT`.	Generated code with plugin disabled:
			
			```VAR_GLOBAL CONSTANT\n SS_INITIALIZE: SINT := 0;\n K3: LREAL := 0.3;\n SS_STEP: SINT := 1;\nEND_VAR```
			
			Generated code with plugin enabled:
			
			```VAR_GLOBAL CONSTANT\n K3: REAL := 0.3;```

`fDefineFBExternalVariable`	Defines external constants in `VAR_EXTERNAL`.	Target IDE requires declaration of external constants in `VAR_EXTERNAL`.	Generated code with plugin disabled:
			
			```VAR\n UnitDelay_DSTATE: LREAL;\n i0_ExternallyDefinedBlock: ExternallyDefinedBlock;\nEND_VAR```
			
			Generated code with plugin enabled:
			
			```VAR_EXTERNAL\n K1: REAL;\nEND_VAR```

`fReplaceShiftFunctions`	Replaces target IDE shift functions with functions to match Simulink shift function behavior.	When the number of shifts is greater than the length of the data type, there is a mismatch between output of Simulink shift block and target IDE shift block. Use this plugin to replace target IDE shift blocks with Simulink shift blocks in the generated code.	Generated code with plugin disabled:
			
			```Out10 := WORD_TO_INT(SHL(IN:=INT_TO_WORD(In1),...\n N:=Out2_tmp));```
			
			The generated code uses the target IDE `SHL` shift function.
			
			Generated code with plugin enabled:
			
			```Out10 := WORD_TO_INT(PLC_SHL(INT_TO_WORD(In1),...\n DINT_TO_USINT(Out2_tmp)));\nFUNCTION PLC_SHL: WORD\nVAR_INPUT\n in1: WORD;\n in2: USINT;\nEND_VAR\n'ST'\nBODY\nIF in2 > 16 THEN\n PLC_SHL := 16#0;\nELSE\n PLC_SHL := SHL(in1, in2);\nEND_IF;\nEND_BODY\nEND_FUNCTION```
			
			The generated code contains the function `PLC_SHL`, which replicates the Simulink shift block in the target IDE.

Plugins for Intrinsic Transformation Functions

Plugin Name	Plugin Purpose	When to Use Plugin	Effect of Plugin on Generated Code

`fSimplifyAllIntrinsicFcn`	Simplifies the inputs of all intrinsic functions.	Target IDE does not allow compound expressions as a part of intrinsic function arguments.	Generated code with plugin disabled:
			
			`a := SQRT(x*y);`
			
			Generated code with plugin enabled:
			
			`t1 := x*y;`
			`a := SQRT(t1);`

`fSimplifyIntrinsicFcn`	Simplifies intrinsic functions that are arguments of `fSimplifyIntrinsicFcnNameList`.	Target IDE does not allow compound expressions as a part of intrinsic function arguments.	Generated code with plugin disabled:
			
			`a := SQRT(x*y);`
			
			Generated code with plugin enabled:
			
			`cfg.fSimplifyIntrinsicFcnNameList = { 'SQRT' \}`
			`cfg.fSimplifyIntrinsicFcn = true;`
			`t1 := x*y;`
			`a := SQRT(t1);`

`fSimplifyIntrinsicFcnNameList`	Creates a list of intrinsic functions. Inputs to these intrinsic functions are simplified by using `fSimplifyIntrinsicFcn`.	Target IDE does not allow compound expressions as a part of intrinsic function arguments.	Generated code with plugin disabled:
			
			`a := SQRT(x*y);`
			
			Generated code with plugin enabled:
			
			`cfg.fSimplifyIntrinsicFcnNameList = { 'SQRT' \}`
			`cfg.fSimplifyIntrinsicFcn = true;`
			`t1 := x*y;`
			`a := SQRT(t1);`

`fSimplifyOperator`	Simplifies inputs of operator functions listed by using `fSimplifyOperatorNameList`.	Target IDE does not allow compound expressions as a part of operator function arguments.	Generated code with plugin disabled:
			
			`a := SHL(x*y);`
			
			Generated code with plugin enabled:
			
			`cfg.fSimplifyOperatorNameList = { 'SHL' \}`
			`cfg.fSimplifyOperator = true;`
			`t1 := x*y;`
			`a := SHL(t1);`

`fSimplifyOperatorNameList`	Creates a list of operator functions. Inputs to these operator functions are simplified by using `fSimplifyOperator`.	Target IDE does not allow compound expressions as a part of operator function arguments.	Generated code with plugin disabled:
			
			`a := SHL(x*y);`
			
			Generated code with plugin enabled:
			
			`cfg.fSimplifyOperatorNameList = { 'SHL' \}`
			`cfg.fSimplifyOperator = true;`
			`t1 := x*y;`
			`a := SHL(t1);`

`fSimplifyTrunc`	Simplifies inputs of the `TRUNC` function.	Target IDE does not allow compound expressions as arguments for the `TRUNC` function.	Generated code with plugin disabled:
			
			`a := TRUNC(x*y);`
			
			Generated code with plugin enabled:
			
			`cfg.fSimplifyTrunc = true;`
			`t1 := x*y;`
			`a := SHL(t1);`

Plugins for XSD Schema Modification

Plugin Name	Plugin Purpose	When to Use Plugin	Effect of Plugin on Generated Code

`fEmitXsdSchema`	Generates information related to XSD schema version `1.0`.	Target IDE requires XSD schema information as a part of the generated PLCOpen XML file.
		
		To generate schema set `fEmitXsdSchema` to `'v1'`.
			Generated code with plugin set to `'None'`:
			
			`<?xml version="1.0" encoding="UTF-8"?>`
			
			Generated code with plugin set to `'v1'`:
			
			`<?xml version="1.0" encoding="UTF-8"?>`
			`<project xmlns="http://www.plcopen.org/xml/tc6.xsd" xmlns:xhtml="http:/`

Generate Code by Using Plugin-Based Target IDE

This example shows how to generate code for a custom target IDE called my PLCopen XMLby using plugins.

1 Create a folder called `myplcopen`. Create a `plc_custom_ide.m` file in the folder by using this template:

```
function plc_ide_list = plc_custom_ide
%   Copyright 2012-2021 The MathWorks, Inc.
    plc_ide_list(1) = get_ide_info_myplcopen;
end

function ide_info = get_ide_info_myplcopen
    ide_info.name = 'myplcopen';
    ide_info.description = 'My PLCopen XML';
    ide_info.path = ''; % IDE path
    ide_info.format = 'xml'; % generic|xml
    ide_info.fileExtension = 'xml';
    ide_info.cfg = get_ide_cfg_myplcopen;
    ide_info.precg_callback = 'plc_precg_callback_myplcopen';
    ide_info.postcg_callback = 'plc_postcg_callback_myplcopen';
    ide_info.xmltree_callback = PLCCoder.PLCCGMgr.PLC_PLUGIN_CG_CALLBACK_EMPTY;
    ide_info.pluginVersion = 2.2;
    ide_info.compatibleBuildVersion = 1.6;
end

function cfg = get_ide_cfg_myplcopen
    cfg.fConvertDoubleToSingle = true;
    cfg.fConvertNamedConstantToInteger = true;
    cfg.fConvertEnumToInteger = true;
    cfg.fConvertOutputInitValueToAssignment = true;
    cfg.fConvertTunableParamToInputVariable = true;
```

29-15

```
        cfg.fSimplifyFunctionCallExpr = true;
        cfg.fConvertOutputInitValueToAssignment = true;
end
```

Set your plugin options in the `function_cfg` section of the file. To enable the plugin set the plugin option to `true`. For example, `cfg.fArrayInitialValueBrackets = true;` enables the plugin. To disable the plugin, set the plugin option to `false`.

2   Create `plc_precg_callback_IDEname.m` and `plc_postcg_callback_IDEname.m` files by using these templates:

```
function controller = plc_precg_callback_myplcopen(controller)
%  Copyright 2012-2020 The MathWorks, Inc.

    % do modifications to the controller struct here, f.ex.:
    for i = 1:length(controller.components)
        controller.components(i).body = sprintf('<<header_placeholder>>\r\n%s',controller.components(i).body);
    end

end

function generatedFiles = plc_postcg_callback_myplcopen(fileNames)
%  Copyright 2012-2020 The MathWorks, Inc.

    fileName = fileNames{1};
    str = fileread(fileName);
%  do modifications to str here, f.ex.:
%  str = regexprep(str,'BOOL_TO_LREAL','BOOL_TO_INT');
%  str = regexprep(str,'<USINT/>','<INT/>');
%  str = regexprep(str, 'END_STRUCT','END_STRUCT;');

    [sHeader,eHeader] = regexp(str,'\(\(\*.*?\)\*\)');
    header = str(sHeader:eHeader);

    str = regexprep(str,'<<header_placeholder>>',header);

    sfprivate ('str2file', str, fileName);
    generatedFiles = {fileName};
end
```

3   Create a `plc_header_hook.m` file by using this template:

```
function headerCommentText = plc_header_hook(filePath, blockH, headerCommentText)

headerCommentText = [headerCommentText(1:end-7) ...
    sprintf([' * Plugin Header Copy           : Yes \n']) ...
    headerCommentText(end-6:end)];

end
```

The `plc_header_hook.m` file copies the header information at the beginning of the generated code file to every function block instance.

4   Add the new folder and files to the MATLAB path.

- Right-click the folder and select **Add to Path** > **Selected Folders and Subfolders**.

- Use the `addpath` function. For example, `addpath(genpath('path to your folder'))`.

5   Run this command:

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

Restart your MATLAB session.

6   Open your model and select the model component for code generation. Open the **PLC Coder** app. Click **Settings**. On the **PLC Code Generation** pane, in **Target IDE**, select `My PLCopen XML`. Click **OK**.

7   In the **PLC Coder** app, **PLC Code** tab, click **Generate PLC Code** to generate code for your custom target IDE. The generated code files are placed in the path specified in `ide_info.path = ''; % IDE path`.

See Also

More About

- "Import and Verify Structured Text Code" on page 4-4
- "Generate Custom Code by Using IDE—Specific Callback Functions" on page 29-18
- "Videos"

Generate Custom Code by Using IDE—Specific Callback Functions

Replace strings, add custom headers, print a list of variables, and so on by generating custom code. Customize the code generated for supported target IDEs by creating IDE-specific custom callback functions. The custom callback functions can modify the generated code by using post-code generation callback functions or the intermediate controller cell array structure by using pre-code generation callback functions.

Custom Code Generation Workflow

To generate custom code for your target IDEs by using custom callback functions:

1 Create a custom pre- or post- code generation callback function. Place the callback functions on the MATLAB path.

2 Generate custom code by using Simulink PLC Coder.

Create a Custom Callback Function

Create a custom pre-code generation callback function by using this template. Replace the `Pre-Code Generation Callback File Name` with the name of the pre-code generation callback file for your target IDE from the table. The pre-code generation function operates on the `controller` structure which contains information such as variable names, data types, and so on.

```
function controller = Pre-Code Generation Callback File Name(controller)
% add your custom actions here
end
```

Create a custom post-code generation callback function by using this template. Replace the `Post-Code Generation Callback File Name` with the name of the post-code generation callback file for the target IDE from the table. The post-code generation function performs operations such as replacing string values, adding a header to every code section, and so on.

```
function generatedFiles = Post-Code Generation Callback File Name(fileNames)
fileName = fileNames{1};
str = fileread(fileName); % reading the file contents%   do modifications to str here, f.ex.:
%    str = regexprep(str,'BOOL_TO_LREAL','BOOL_TO_INT');
%    str = regexprep(str,'<USINT/>','<INT/>');
%    str = regexprep(str, 'END_STRUCT','END_STRUCT;');
sfprivate ('str2file', str, fileName);
generatedFiles = {fileName};
end
```

Target IDE Name	Pre-Code Generation Callback File Name	Post-Code Generation Callback File Name
3S-Smart Software Solutions CODESYS Version 2.3	`plc_precg_callback_codesys23`	`plc_postcg_callback_codesys23`
3S-Smart Software Solutions CODESYS Version 3.3	`plc_precg_callback_codesys33`	`plc_postcg_callback_codesys33`
3S-Smart Software Solutions CODESYS Version 3.5	`plc_precg_callback_codesys35`	`plc_postcg_callback_codesys35`
B&R Automation Studio Version 3.0	`plc_precg_callback_brautomation30`	`plc_postcg_callback_brautomation30`

Target IDE Name	Pre-Code Generation Callback File Name	Post-Code Generation Callback File Name
B&R Automation Studio Version 4.0	`plc_precg_callback_braut omation40`	`plc_postcg_callback_brau tomation40`
Beckhoff TwinCAT 2.11	`plc_precg_callback_twinc at211`	`plc_postcg_callback_twin cat211`
Beckhoff TwinCAT 3.0	`plc_precg_callback_twinc at3`	`plc_postcg_callback_twin cat3`
PHOENIX CONTACT Software MULTIPROG 5.0	`plc_precg_callback_multi prog50`	`plc_postcg_callback_mult iprog50`
Phoenix Contact PC WORX 6.0	`plc_precg_callback_pcwor x60`	`plc_postcg_callback_pcwo rx60`
Rockwell Automation RSLogix 5000	`plc_precg_callback_rslog ix5000`	`plc_postcg_callback_rslo gix5000`
Rockwell Automation Studio 5000	`plc_precg_callback_studi o5000`	`plc_postcg_callback_stud io5000`
Siemens SIMATIC STEP 7	`plc_precg_callback_step7`	`plc_postcg_callback_step 7`
Siemens TIA Portal	`plc_precg_callback_tiapo rtal`	`plc_postcg_callback_tiap ortal`
Siemens TIA Portal: Double Precision	`plc_precg_callback_tiapo rtal_double`	`plc_postcg_callback_tiap ortal_double`
Generic	`plc_precg_callback_gener ic`	`plc_postcg_callback_gene ric`
PLCopen XML	`plc_precg_callback_plcop en`	`plc_postcg_callback_plco pen`
Rexroth IndraWorks	`plc_precg_callback_indra works`	`plc_postcg_callback_indr aworks`
OMRON Sysmac Studio	`plc_precg_callback_omron`	`plc_postcg_callback_omro n`

Create a custom pre-code generation callback function by using this template. Replace the `Pre-Code Generation Callback File Name` with the name of the pre-code generation callback function from the table. The pre-code generation function operates on the `controller` structure which contains information such as variable names, data types, and so on.

```
function controller = Pre-Code Generation Callback File Name(controller)
% add your custom actions here
end
```

Create a custom post-code generation callback function by using this template. Replace the `Post-Code Generation Callback File Name` with the name of the post-code generation callback function from the table. The post-code generation function performs operations such as replacing string values, adding a header to every code section, and so on.

```
function generatedFiles = Post-Code Generation Callback File Name(fileNames)
fileName = fileNames{1};
str = fileread(fileName); % reading the file contents%   do modifications to str here, f.ex.:
%   str = regexprep(str,'BOOL_TO_LREAL','BOOL_TO_INT');
%   str = regexprep(str,'<USINT/>','<INT/>');
```

```
%   str = regexprep(str, 'END_STRUCT','END_STRUCT;');
sfprivate ('str2file', str, fileName);
generatedFiles = {fileName};
end
```

Generate Custom Code

1   To create a custom pre—code generation callback function file called
`plc_precg_callback_codesys23.m`, copy and paste this code into a MATLAB script and save
the file. Place the file on the MATLAB path.

```
function controller = plc_precg_callback_codesys23(controller)

fprintf(1, 'processing by plc_postcg_callback_codesys23\n');

end
```

2   To create a custom post—code generation callback function file called
`plc_postcg_callback_codesys23.m`, copy and paste this code into a MATLAB script and
save the file. Place the file on the MATLAB path. This callback function:

a   Reads the generated code file. For example, `plcdemo_simple_subsystem.exp`.

b   Scans the read string and finds the string `S1` and replaces it with the string `s1`.

c   Writes the modified string back to the generated file.

```
function generatedFiles = plc_postcg_callback_codesys23(fileNames)
fileName = fileNames{1};
str = fileread(fileName); % reading the file contents%   do modifications to str here, f.ex.:
%   str = regexprep(str,'BOOL_TO_LREAL','BOOL_TO_INT');
%   str = regexprep(str, 'END_STRUCT','END_STRUCT;');
str = regexprep(str,'<S1>','<s1>');
sfprivate ('str2file', str, fileName);
generatedFiles = {fileName};
end
```

3   Open and run the `Generate Structured Text Code for a Simple Simulink
Subsystem`.

```
openExample('plccoder/GenerateStructuredTextCodeForASimpleSimulinkRSubsystemExample')
```

4   Open the PLC Coder app, click **Settings** > **PLC Code Generation**. Change **Target IDE** to `3S
CoDeSys 2.3`. Click **OK**.

5   Select the `SimpleSubsystem` block. In the **PLC CODE** tab, click **Generate PLC Code**.

Observe the MATLAB command line output, the string `processing by
plc_postcg_callback_codesys23` is printed during code generation. In the generated code, the
string `S1` is replaced with the string `s1`. This image shows the generated custom code with the
replaced string.

```
CASE ssMethodType OF
    SS_INITIALIZE:
        (* SystemInitialize for Atomic SubSystem: '<Root>/SimpleSubsystem' *)
        (* InitializeConditions for UnitDelay: '<s1>/Unit Delay' *)
        UnitDelay_DSTATE := 0.0;
        (* End of SystemInitialize for SubSystem: '<Root>/SimpleSubsystem' *)
    SS_STEP:
        (* Outputs for Atomic SubSystem: '<Root>/SimpleSubsystem' *)
        (* Gain: '<s1>/Gain' incorporates:
         *  Sum: '<s1>/Sum'
         *  UnitDelay: '<s1>/Unit Delay' *)
        Y := (U - UnitDelay_DSTATE) * 0.5;
        (* Update for UnitDelay: '<s1>/Unit Delay' *)
        UnitDelay_DSTATE := Y;
        (* End of Outputs for SubSystem: '<Root>/SimpleSubsystem' *)
END_CASE;
END_FUNCTION_BLOCK
VAR_GLOBAL CONSTANT
    SS_INITIALIZE: SINT := 0;
    SS_STEP: SINT := 1;
END_VAR
```

See Also

More About

- "Import and Verify Structured Text Code" on page 4-4
- "Create Custom Target IDE for Code Generation" on page 29-2
- "Videos"

Variable-Size Code Generation

Variable-Size Signal Code Generation

To generate code for your Simulink subsystems, Stateflow charts, and MATLAB Function Blocks that have variable-size signals, use Simulink PLC Coder. An array that has at least one variable dimension is called a variable-size array. Variable-size arrays are commonly found in applications such as machine learning and deep learning. Construct variable-size signals in your model by using Simulink blocks, MATLAB Function Blocks, and Stateflow charts. For more information, see "Variable-Size Signal Basics".

Limitations

Simulink PLC Coder does not support variable-size code generation for:

- Subsystems that do not have the **Function Packaging** set to `Inline`.
- Do not use one-dimensional variable-size inputs and outputs in the top-level subsystem.
- Selector blocks that have variable-size data.

Simulink PLC Coder supports only these model components for variable-size code generation:

- Simulink subsystems
- MATLAB Function Blocks
- Stateflow charts

Variable-Size Code Generation Example

To generate code for a model with variable-size signals, see "Generate Structured Text Code for Variable-Size Signals" on page 25-94.

Generated Code Structure for Variable-Size Signals

For the example "Generate Structured Text Code for Variable-Size Signals" on page 25-94, these images show the structure of the code generated for the model.

This image shows the highlighted output variables for the variable-size data MATLAB Function Block.

This image shows the highlighted output variables for the Stateflow chart.

This image shows the highlighted generated variable-size code for the generated test bench.

```
    out_y2: ARRAY [0..4] OF LREAL;
    tb_y2_sz: ARRAY [0..70] OF DINT := 3,3,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,3,3,3,3,3,3,
        3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,5,5,5,5,5,5,5;
    cycle_y2_sz: DINT;
    out_y2_sz: DINT;
    i: DINT;
    i2: DINT;
    i3: DINT;
    i4: DINT;
    i0_Subsystem: Subsystem;
END_VAR
IF testVerify THEN
    IF testCycleNum < 71 THEN
        (* TEST CYCLE SETUP  *)
        cycle_u := tb_u[testCycleNum];
        FOR i := 0 TO 4 DO
            cycle_y[i] := tb_y[(testCycleNum * 5) + i];
        END_FOR;
        cycle_y_sz := tb_y_sz[testCycleNum];
        cycle_y1 := tb_y1[testCycleNum];
        FOR i2 := 0 TO 4 DO
            cycle_y2[i2] := tb_y2[(testCycleNum * 5) + i2];
        END_FOR;
        cycle_y2_sz := tb_y2_sz[testCycleNum];
        IF testCycleNum = 0 THEN
            (* INIT  *)
            i0_Subsystem(ssMethodType := SS_INITIALIZE, u := cycle_u);
        END_IF;
        (* STEP  *)
        i0_Subsystem(ssMethodType := SS_STEP, u := cycle_u);
        out_y := i0_Subsystem.y;
        out_y_sz := i0_Subsystem.y_sz;
        out_y1 := i0_Subsystem.y1;
        out_y2 := i0_Subsystem.y2;
        out_y2_sz := i0_Subsystem.y2_sz;
        (* VERIFY  *)
        IF testVerify AND (out_y_sz <> cycle_y_sz) THEN
            testVerify := FALSE;
        END_IF;
        FOR i3 := 0 TO cycle_y_sz - 1 DO
            IF testVerify THEN
                IF ABS(cycle_y[i3]) < 1.0E-12 THEN
                    IF ABS(out_y[i3]) > 1.0E-5 THEN
                        testVerify := FALSE;
                    END_IF;
                ELSIF ABS(out_y[i3] - cycle_y[i3]) > (1.0E-5 * ABS(cycle_y[i3])) THEN
                    testVerify := FALSE;
                END_IF;
            END_IF;
        END_FOR;
        IF testVerify AND (out_y2_sz <> cycle_y2_sz) THEN
            testVerify := FALSE;
        END_IF;
        FOR i4 := 0 TO cycle_y2_sz - 1 DO
```