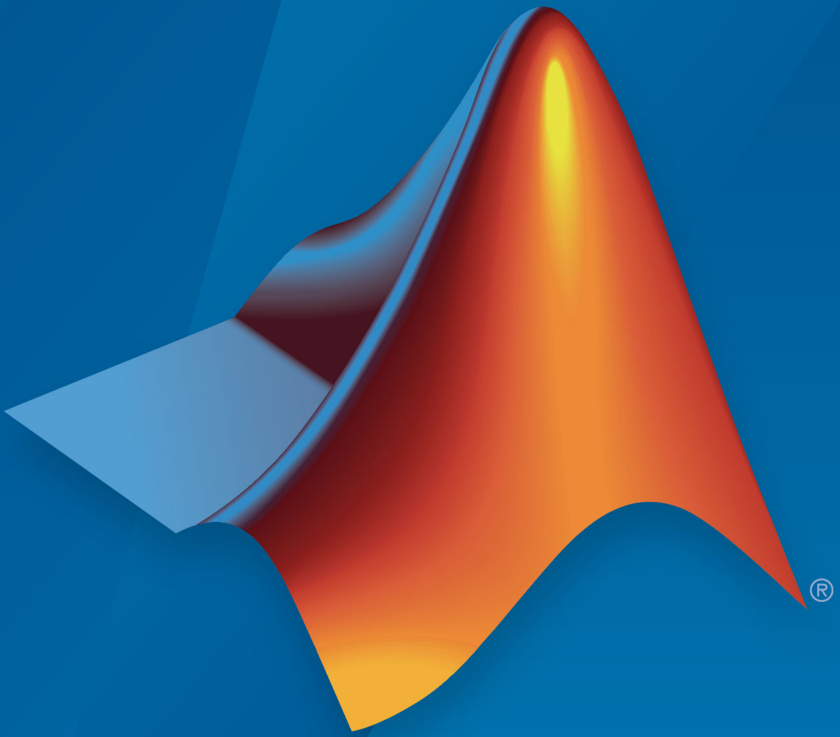
Simulink® PLC Coder™

User's Guide



MATLAB®&SIMULINK®

R2018**b**

MathWorks®

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)

Contents

1 Getting Started

3 Generating Ladder Diagram

4 Generating Test Bench Code

9 IDE-Specific Considerations

10 Supported Simulink and Stateflow Blocks

11 Limitations

12 Functions — Alphabetical List

13 Configuration Parameters for Simulink PLC Coder Models

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. The Structured Text and ladder diagrams are 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. 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).

Key Features

- Automatic generation of IEC 61131-3 Structured Text and ladder diagrams
- IDE support, including 3S-Smart Software Solutions CODESYS, Rockwell Automation Studio 5000, Siemens TIA Portal, OMRON Sysmac Studio, and PLCopen XML
- Simulink support, including reusable subsystems, PID controller blocks, and lookup tables
- Stateflow support, including state machines, graphical functions, and truth tables
- MATLAB support, including if-else statements, loop constructs, and math operations
- Support for multiple data types, including Boolean, integer, enumerated, and floating-point, as well as vectors, matrices, buses, and tunable parameters
- Test bench creation

PLC Code Generation in the Development Process

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

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

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

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

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

Using Simulink PLC Coder, you can also generate Ladder Diagram code for your applications from a Stateflow chart. The benefits are:

- You can design your application by using states and transitions in a Stateflow chart. Once you complete the design, you can generate Ladder Diagram code in XML or another format. You then import the generated code to an IDE such as CODESYS 3.5 or RSLogix™ AOI 5000 and view the ladder diagram.
- When you test your Stateflow chart by using a set of inputs, you can reuse these inputs to create a test bench for the Ladder Diagram code. You import the test bench to your PLC IDE and compare the results of simulation with the results of running the ladder diagram. If the results match, the original Stateflow chart is equivalent to the generated Ladder Diagram code.

Expected Users

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

- PLC manufacturing
- Machine manufacturing
- Systems integration

You must be familiar with:

- MATLAB and Simulink software and concepts
- PLCs
- Structured Text language

If you want to download generated code to a PLC IDE, you must also be familiar with your chosen PLC IDE platform. For a list of these platforms, see "Supported IDE Platforms" on page 1-6.

Glossary

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

System Requirements

For a list of related products, see System Requirements at the MathWorks® website.

Issues with Anti-Virus Software

The Simulink PLC Coder software ships with IDE-specific executables that are used in the "Import Structured Text Code Automatically" on page 1-27 workflows. Some anti-virus software identifies these files as malware. However, it has been determined that these cases are false positives and that the files are safe. You can mark these files as safe in your antivirus program.

Supported IDE Platforms

IDEs Supported for Structured Text Generation

The Simulink PLC Coder product is tested with the following IDE platforms:

- 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
- PHOENIX CONTACT Software MULTIPROG® 5.0 or 5.50.

  PHOENIX CONTACT Software GmbH was previously called KW-Software GmbH. The Simulink PLC Coder software supports only the English version of MULTIPROG target IDE.
- OMRON Sysmac Studio Version 1.04, 1.05, 1.09 or 1.12
- Phoenix Contact® PC WORX™ 6.0

  The Simulink PLC Coder software supports only the English version of Phoenix Contact PC WORX target IDE.
- Rexroth IndraWorks version 13V12 IDE
- Rockwell Automation RSLogix 5000 Series Version 17, 18, 19 or 20 and Rockwell Studio 5000 Logix Designer Version 21 or 24

  Simulink PLC Coder can generate code for Add-On instructions (AOIs) and routine code. The software supports automatic import and verification of generated code only for the RSLogix IDEs and not the Studio 5000 IDE.
- Siemens SIMATIC® STEP® 7 Version 5.3, 5.4 or 5.5

  The Simulink PLC Coder software assumes that English systems use English S7. It assumes that German systems use German S7.
- Siemens TIA Portal V13
- Generic
- PLCopen XML

For a list of supported IDEs and platforms, see Supported IDEs at the MathWorks website.

IDEs Supported for Ladder Diagram Code Generation

The Simulink PLC Coder product is tested with the following IDE platforms:

- 3S-Smart Software Solutions CODESYS Version 3.5 SP6
- Rockwell Automation RSLogix 5000 Series Version 20 and Rockwell Studio 5000 Logix Designer Version 24
- PLCopen XML

PLC Code Generation Workflow

Your basic Simulink PLC Coder workflow is:

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

Prepare Model for Structured Text Generation

In this section...

Tasking Mode

This step is only required if your Simulink model contains multi-rate signals. If your Simulink model does not contain multi-rate signals, you may proceed to solver selection.

Simulink PLC Coder only generates code for single-tasking subsystems. For multi-rate 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**.

Solvers

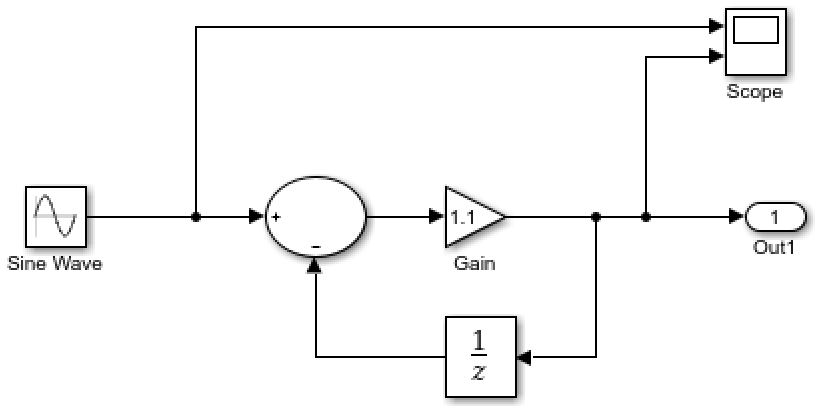
Choose a solver for your Simulink PLC Coder model.

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.

Configuring Simulink Models for Structured Text Code Generation

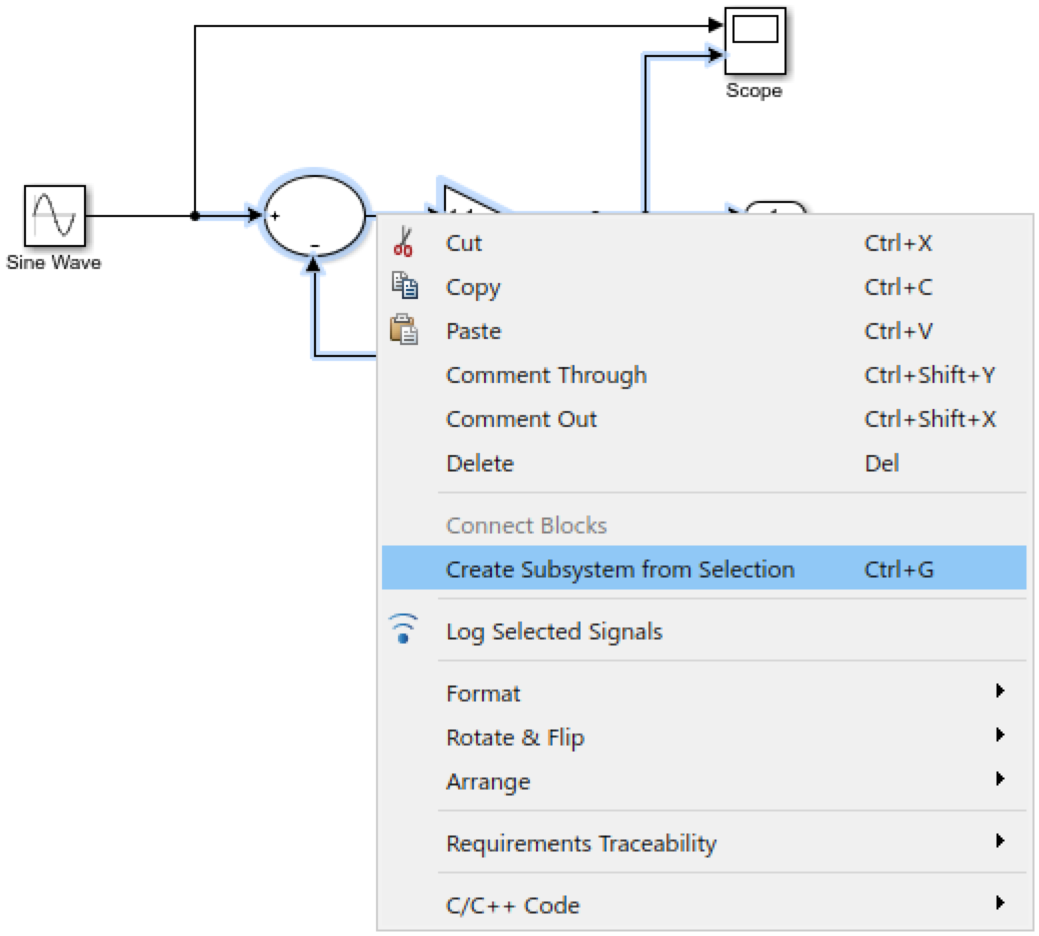
You must already have a model for which you want to generate and import code to a PLC IDE. Before you use this model, perform the following steps.

1 In the Command Window, open your model.

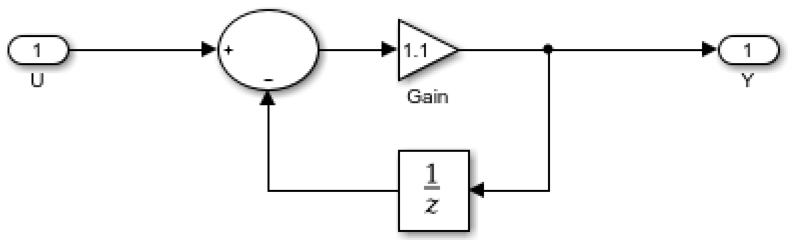


2. Configure the model to use the fixed-step discrete solver. Select **Simulation** > **Model Configuration Parameters** and in the Solver pane, set **Type** to `Fixed-step` and **Solver** to `discrete (no continuous states)`.

   If your model uses a continuous solver, has a subsystem, configure a fixed sample time for the subsystem for which you generate code.
3. Save this model as `plcdemo_simple_subsystem1`.
4. Create a subsystem containing the components for which you want to generate Structured Text code.



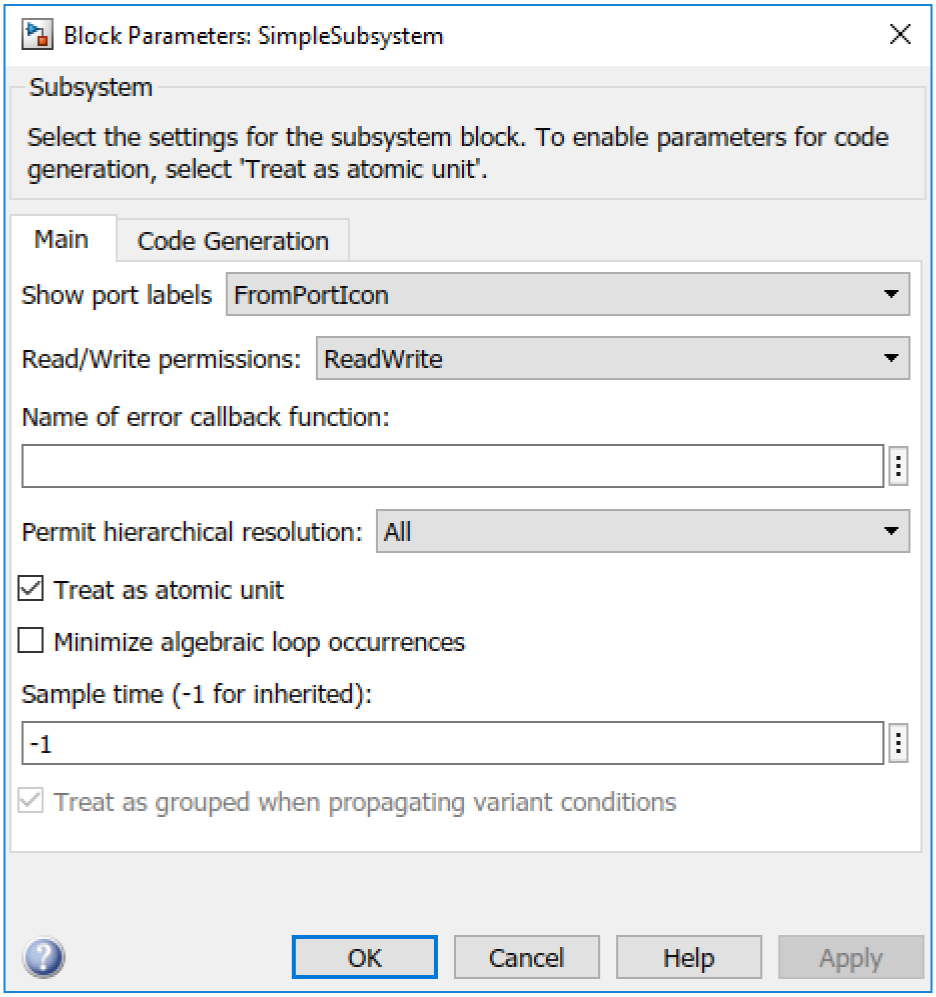
Optionally, rename `In1` and `Out1` to `U` and `Y` respectively. This operation results in a subsystem like the following figure:



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


Block Parameters: SimpleSubsystem

Subsystem

Select the settings for the subsystem block. To enable parameters for code generation, select 'Treat as atomic unit'.

Show port labels: FromPortIcon

Read/Write permissions: ReadWrite

Name of error callback function:

Permit hierarchical resolution: All

☑ Treat as atomic unit

☐ Minimize algebraic loop occurrences

Sample time (-1 for inherited): -1

☑ Treat as grouped when propagating variant conditions

8 Click **OK**.

9 Simulate your model.

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

You are now ready to:

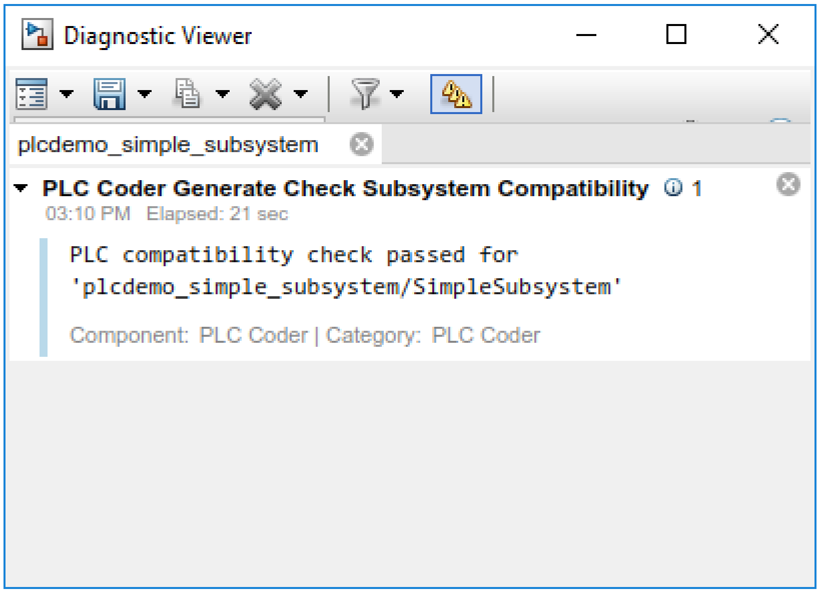
- Set up your subsystem to generate Structured Text code. See “Checking System Compatibility for Structured Text Code Generation” on page 1-14.
- Generate Structured Text code for your IDE. See “Generate and Examine Structured Text Code” on page 1-17.

Checking System Compatibility for Structured Text Code Generation

You must already have a model that you have configured to work with the Simulink PLC Coder software.

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

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

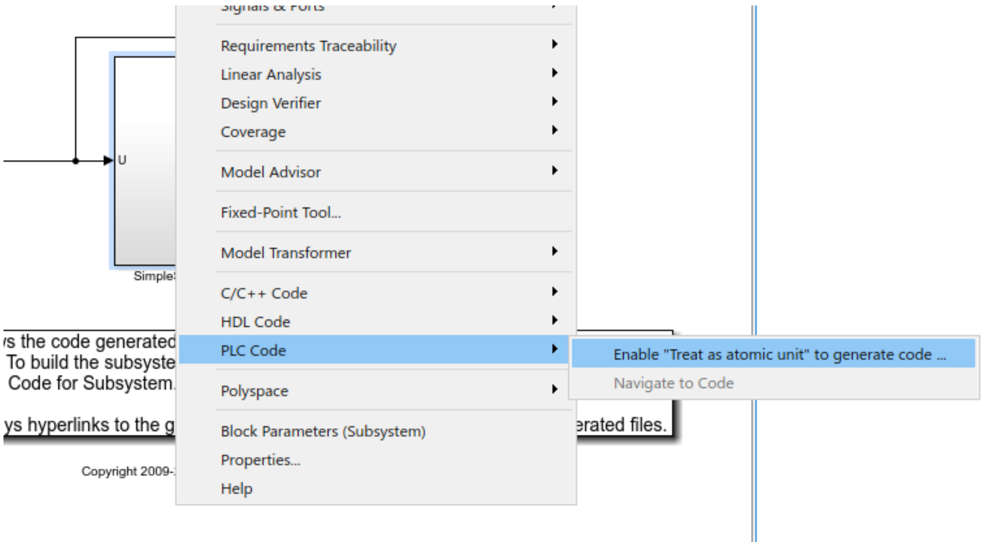


PLC Coder Generate Check Subsystem Compatibility 1
03:10 PM Elapsed: 21 sec

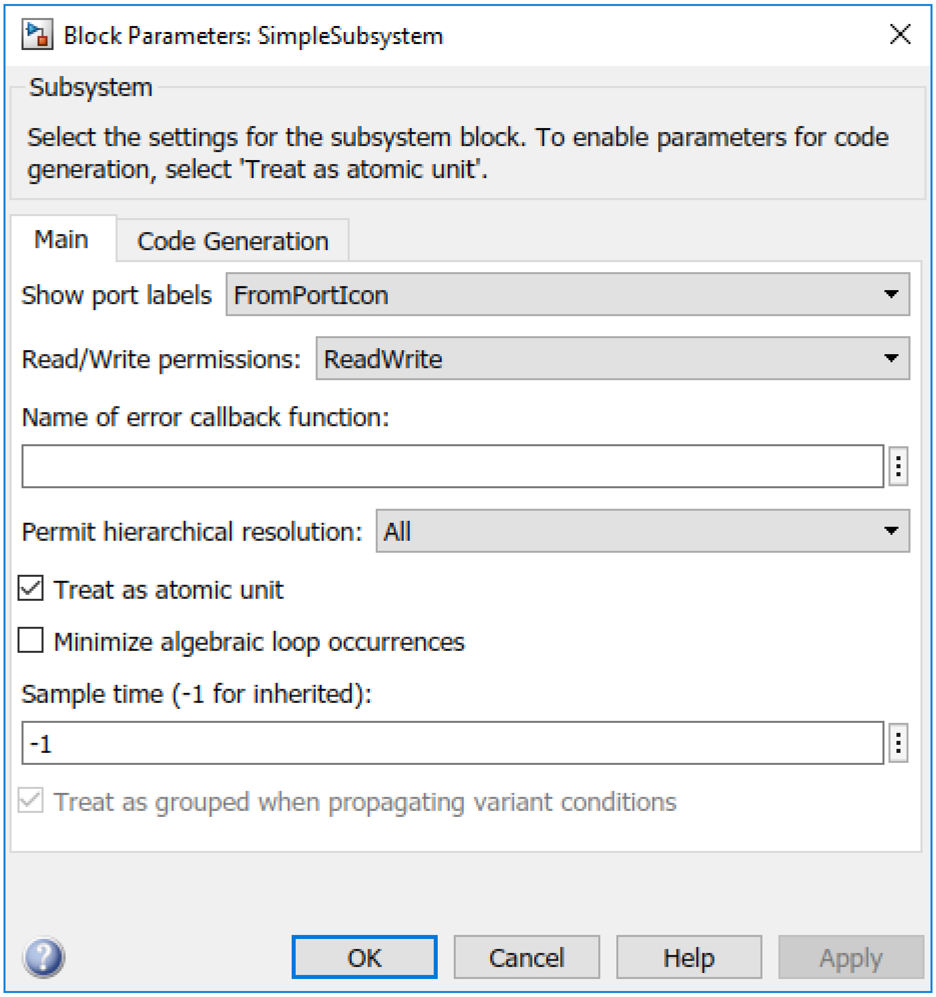
```
PLC compatibility check passed for
'plcdemo_simple_subsystem/SimpleSubsystem'
```

Component: PLC Coder | Category: PLC Coder

   If the subsystem is not atomic, right-click the Subsystem block and select **PLC Code**, which prompts **Enable “Treat as atomic unit” to generate code**.



This command opens the block parameter dialog box. Select **Treat as atomic unit**.



Block Parameters: SimpleSubsystem

Subsystem

Select the settings for the subsystem block. To enable parameters for code generation, select 'Treat as atomic unit'.

Show port labels: FromPortIcon

Read/Write permissions: ReadWrite

Name of error callback function:

Permit hierarchical resolution: All

☑ Treat as atomic unit

☐ Minimize algebraic loop occurrences

Sample time (-1 for inherited): -1

☑ Treat as grouped when propagating variant conditions

You are now ready to generate Structured Text code for your IDE. See “Generate and Examine Structured Text Code” on page 1-17.

Generate and Examine Structured Text Code

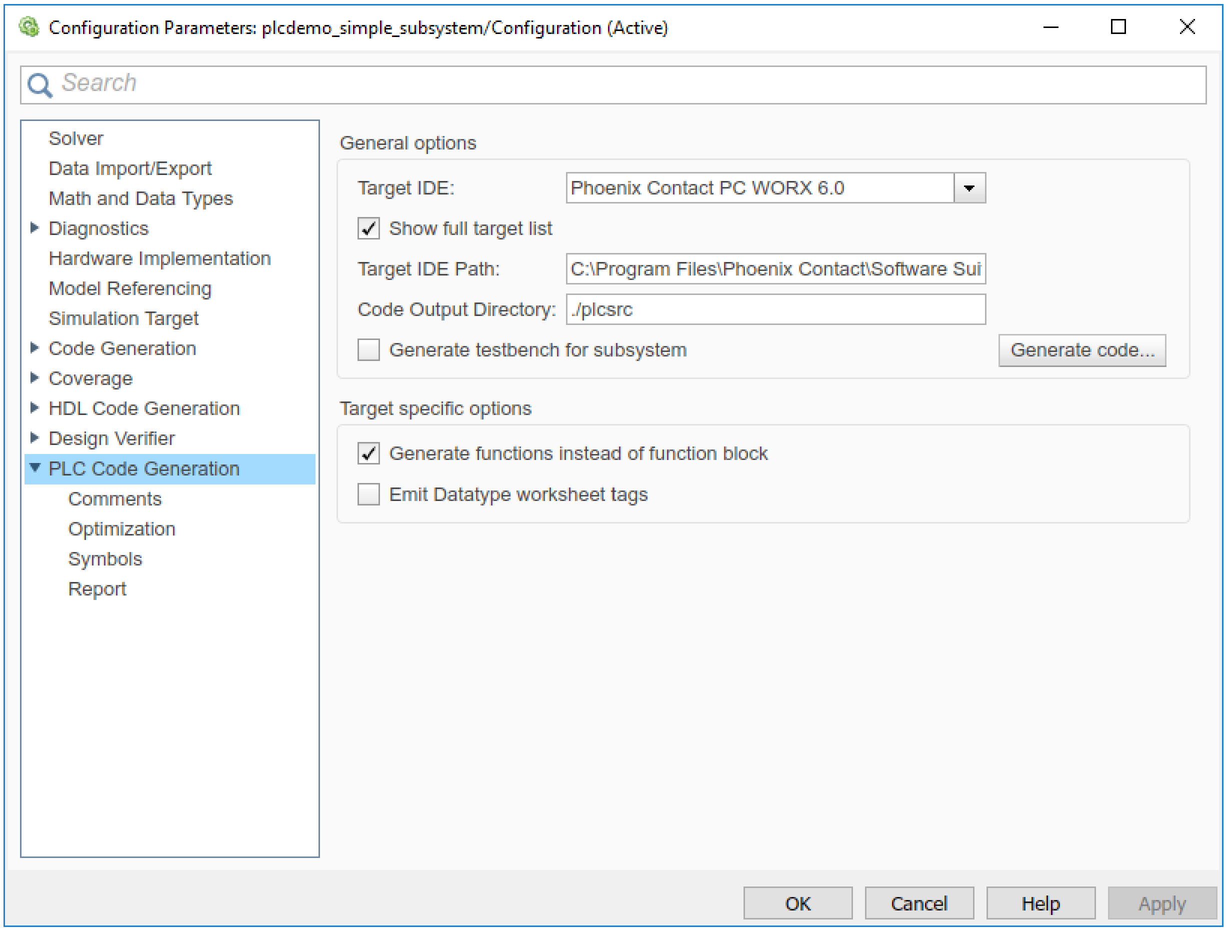
In this section...

Generate Structured Text from the Model Window

You must already have set up your environment and Simulink model to use the Simulink PLC Coder software to generate Structured Text code. If you have not yet done so, see "Prepare Model for Structured Text Generation" on page 1-9.

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

   The Configuration Parameters dialog box is displayed.


Configuration Parameters: plcdemo_simple_subsystem/Configuration (Active)

General options

Target IDE: Phoenix Contact PC WORX 6.0

[✓] Show full target list

Target IDE Path: C:\Program Files\Phoenix Contact\Software Sui

Code Output Directory: ./plcsrc

[ ] Generate testbench for subsystem — Generate code...

Target specific options

[✓] Generate functions instead of function block

[ ] Emit Datatype worksheet tags

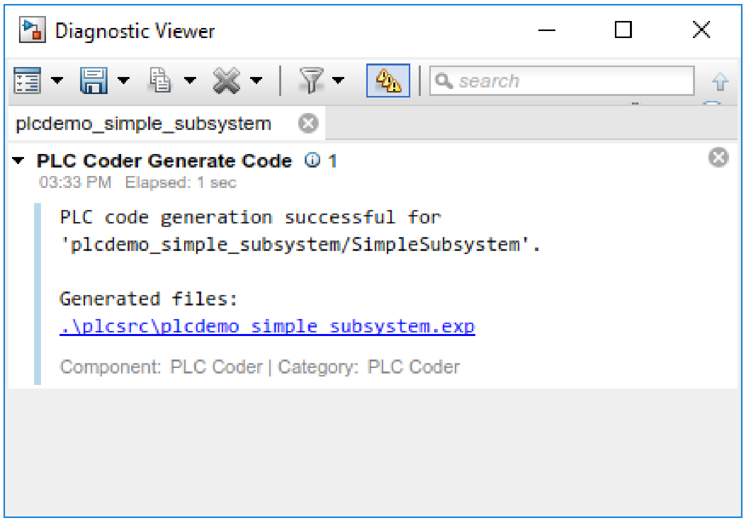
3. On the **PLC Code Generation** pane, select an option from the **Target IDE** list, for example, `3S CoDeSys 2.3`.

   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.
4. Click **Apply**.
5. 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.



```
PLC code generation successful for
'plcdemo_simple_subsystem/SimpleSubsystem'.

Generated files:
.\plcsrc\plcdemo_simple_subsystem.exp
```

Component: PLC Coder | Category: PLC Coder

This window has links that you can click to open the associated files. For more information, see “Files Generated with Simulink PLC Coder” on page 1-23.

Generate Structured Text with the MATLAB Interface

You can generate Structured Text code for a subsystem in the Command Window with 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 using the `plcopenconfigset` function:

   ```
   plcopenconfigset('plcdemo_simple_subsystem/SimpleSubsystem')
   ```
3. Select a target IDE.
4. Configure the subsystem as described in “Prepare Model for Structured Text Generation” on page 1-9.
5. Generate code for the subsystem:

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

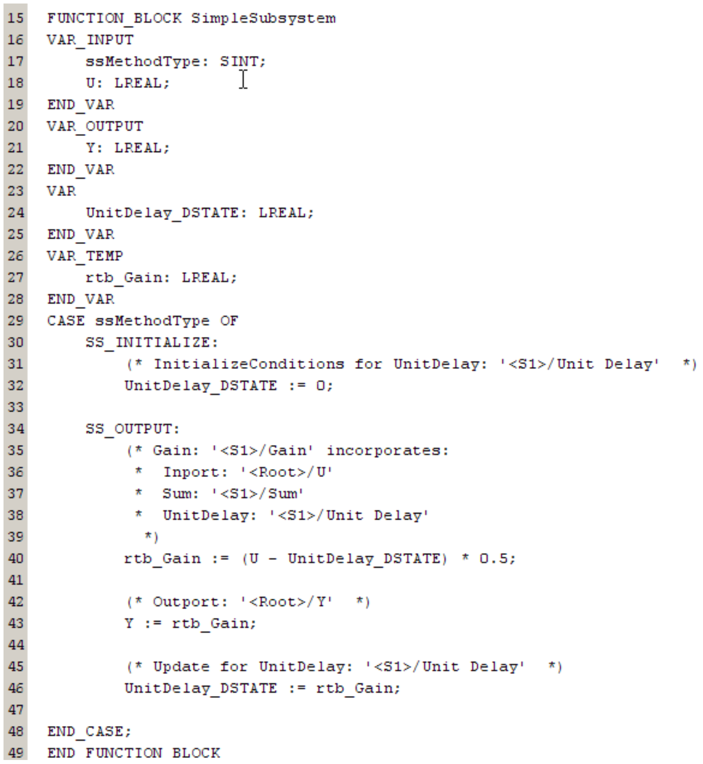
When using `plcgeneratecode` for code generation, all diagnostic messages are printed to the MATLAB command window.

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 “PLC Code Generation Basics”. In addition, note the following:

- Matrix data types: The 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 1-22.

The 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 other format and looks different.



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

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

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

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

END_CASE;
END_FUNCTION_BLOCK
```

If you are confident that the generated Structured Text is good, optionally change your workflow to automatically generate and import code to the target IDE. For more information, see "Import Structured Text Code Automatically" on page 1-27.

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 13-15.

Your block description appears as comments in the generated code.

Files Generated with 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 1-26.

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.

Platform	**Generated Files**
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.
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.

Platform	Generated Files
Rockwell Automation RSLogix 5000 IDE: Routine	*current_folder*`\plcsrc\model_name.L5X` — (If test bench is generated) Structured Text file for importing to the target IDE using routine constructs. This file is in XML format and contains the generated Structured Text code for your model. In *current_folder*`\plcsrc\model_name` (if test bench is not generated), the following files are generated: • *subsystem_block_name*`.L5X` — Structured Text file in XML format. Contains program tag and UDT type definitions and the routine code for the top-level subsystem block. • *routine_name*`.L5X` — Structured Text files in XML format. Contains routine code for other subsystem blocks.
Siemens SIMATIC STEP 7 IDE	*current_folder*`\plcsrc\model_name\model_name.scl` — Structured Text file for importing to the target IDE. *current_folder*`\plcsrc\model_name\model_name.asc` — (If test bench is generated) Text file. Structured Text file and symbol table for generated test bench code.
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 with Simulink PLC Coder” on page 1-23.

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

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

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

Function name options	Generated File Name
`Auto`	Default. Uses the model name, as listed in “Prepare Model for Structured Text Generation” on page 1-9, 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 That Qualify for Importing Code Automatically

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

- 3S-Smart Software Solutions CoDeSys Version 2.3
- PHOENIX CONTACT (previously KW) Software MULTIPROG Version 5.0 or 5.50
- Phoenix Contact PC WORX Version 6.0
- Rockwell Automation RSLogix 5000 Version 17, 18, or 19

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

Working with the default CoDeSys Version 2.3 IDE should require additional changes for only the PHOENIX CONTACT (previously KW) Software MULTIPROG 5.0/5.50 and Phoenix Contact PC WORX 6.0 IDE. For information about automatically importing Structured Text code to these IDEs, see "Import and Verify Structured Text to PHOENIX CONTACT (previously KW) Software MULTIPROG 5.0 and Phoenix Contact PC WORX 6.0 IDEs" on page 4-6.

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. Doing so might disrupt the process. When the process completes, you can resume normal operations.

You must have already installed your target PLC IDE in a default location, and it 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 13-7.

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

   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 run the Structured Text code, see “Import and Verify Structured Text Code” on page 4-5.

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 following versions of target IDEs for automatic import and verification:

- 3S-Smart Software Solutions CoDeSys Version 2.3
- 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 testbench code for automatic import and verification.
- Siemens SIMATIC STEP 7 Version 5.4 (English and German)

Note Some antivirus softwares falsely identify the executables that implement the automatic import feature as malware. This can be safely ignored. For more information, see “Issues with Anti-Virus Software” on page 1-5.

Unsupported Target IDEs

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

- 3S-Smart Software Solutions CoDeSys Version 3.3
- 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)

Possible Automatic Import Issues

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

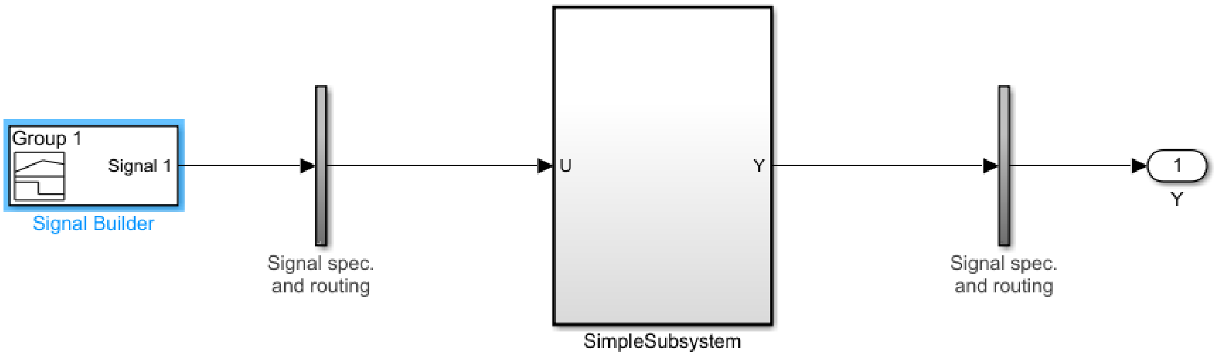
- Check that the coder supports the target IDE version and language setting combination.
- Check that you have specified the target IDE path in the subsystem Configuration Parameters dialog box.

- 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. Try to close other unrelated applications on the system and try the automatic import again.

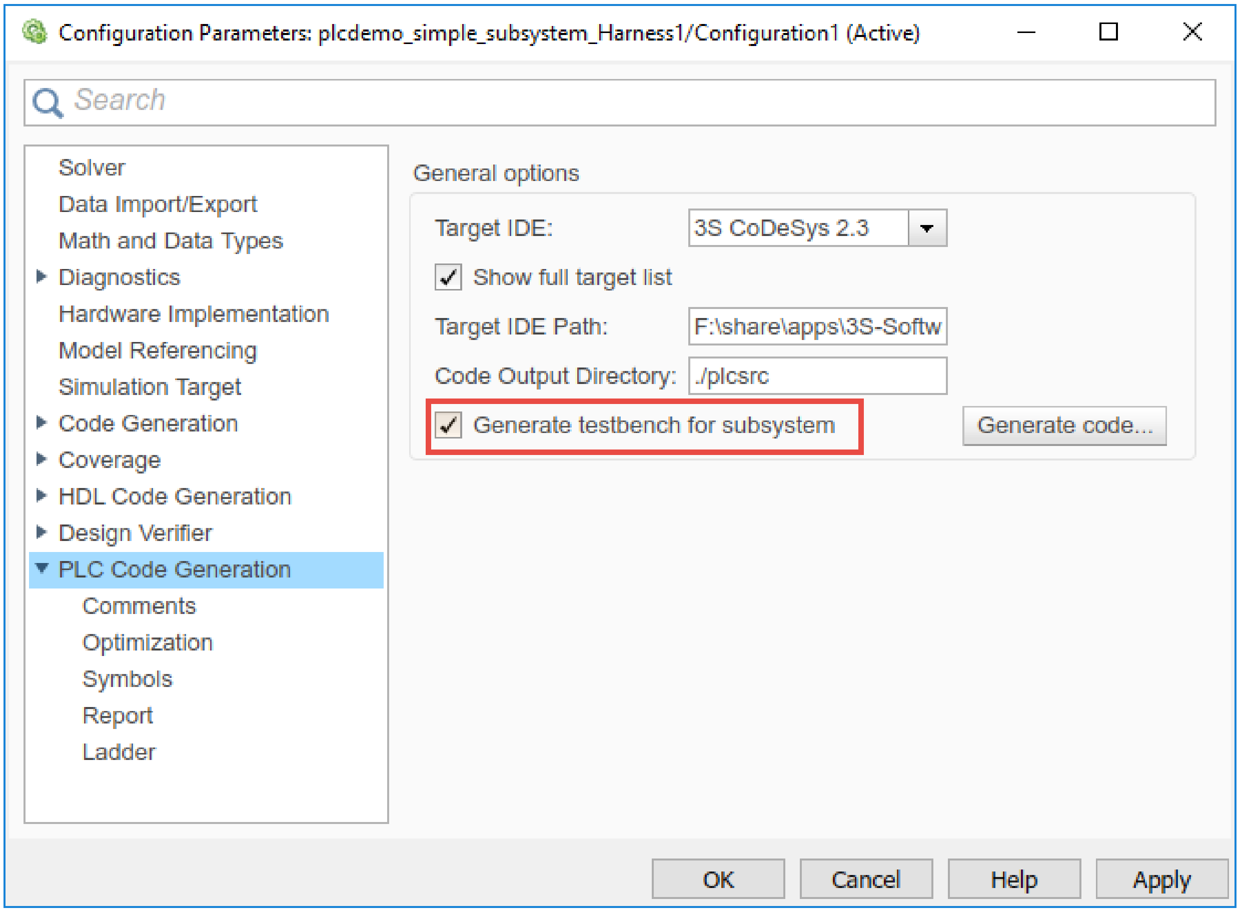
Using Simulink Test with Simulink PLC Coder

You can use Simulink Test™ with Simulink PLC Coder to author, manage, and execute simulation-based tests of the generated code.

1. If you do not have the `plcdemo_simple_subsystem` model open, open it now.
2. Create a signal build test harness for the subsystem as shown. To create a test harness for a subsystem, select the subsystem and select **Analysis** > **Test Harness** > **Create for <subsystem name>**. Set test harness properties using the `Create Test Harness` dialog box.



3. Right-click the Subsystem block and select **PLC Code** > **Options**. The Configuration Parameters dialog box is displayed.


Configuration Parameters: plcdemo_simple_subsystem_Harness1/Configuration1 (Active)

General options

Target IDE: 3S CoDeSys 2.3

✓ Show full target list

Target IDE Path: F:\share\apps\3S-Softw

Code Output Directory: ./plcsrc

✓ Generate testbench for subsystem

4 On the **PLC Code Generation** pane, select a target and enable the **Generate testbench for subsystem**option.

5 Click **Apply**.

6 Right-click and select **Generate code** for the subsystem from the `Test Harness Window`. 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 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 both the subsystem under test and the testing blocks (for example, say test sequence block) and generate code for this subsystem.
- Simulink PLC Coder does not yet support `verify` keyword in the test sequence block
- Simulink PLC Coder does support `duration` keyword in the test sequence block but it requires the generate code to be run with the same sample rate as in the Simulink model

Simulation and Code Generation of Motion Instructions

The Simulink PLC Coder software supports a workflow for the behavioral simulation and structured text code generation for the Rockwell Automation RSLogix motion control instructions.

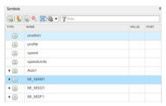
Workflow for Using Motion Instructions in Model

This workflow uses the “Simulating and Generating Structured Text Code for Rockwell Motion Instructions” example in the `plccoderdemos` folder. This example provides a template that you can use with motion instructions. It contains the following files:

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 new model you must load this MAT-file for simulation and code generation.
`Trajectory.m`	MATLAB class file for implementing trapezoidal velocity profile. This 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.

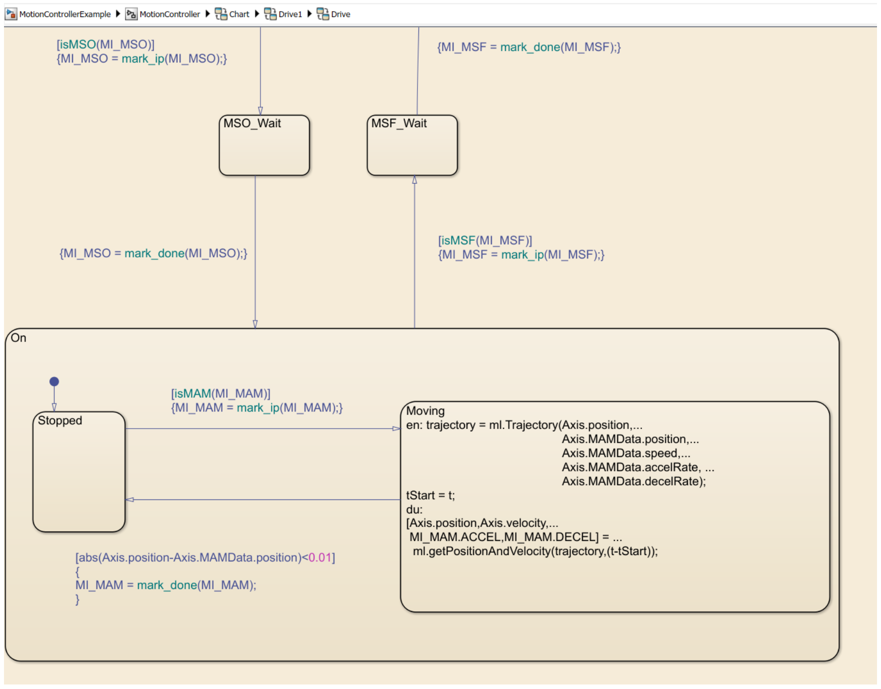
Before you start, copy the files in the example to the current working folder.

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)



TYPE	NAME	VALUE	PORT
	position		
	profile		
	speed		
	speedUnits		
	Axis1		
	MI_MAM1		
	MI_MSO1		
	MI_MSF1		

4. Copy the drive(axis) model from the `DriveLibrary.slx` file into the Stateflow chart. The drive model must be copied as an atomic subchart.



```
[isMSO(MI_MSO)]
{MI_MSO = mark_ip(MI_MSO);}
```

```
{MI_MSF = mark_done(MI_MSF);}
```

```
{MI_MSO = mark_done(MI_MSO);}
```

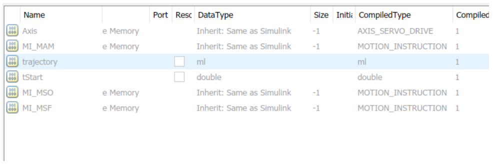
```
[isMSF(MI_MSF)]
{MI_MSF = mark_ip(MI_MSF);}
```

```
[isMAM(MI_MAM)]
{MI_MAM = mark_ip(MI_MAM);}
```

```
en: trajectory = ml.Trajectory(Axis.position,...
                              Axis.MAMData.position,...
                              Axis.MAMData.speed,...
                              Axis.MAMData.accelRate, ...
                              Axis.MAMData.decelRate);
tStart = t;
du:
[Axis.position,Axis.velocity,...
 MI_MAM.ACCEL,MI_MAM.DECEL] = ...
  ml.getPositionAndVelocity(trajectory,(t-tStart));
```

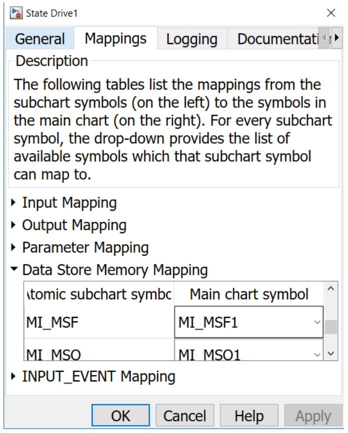
```
[abs(Axis.position-Axis.MAMData.position)<0.01]
{
MI_MAM = mark_done(MI_MAM);
}
```

The drive logic Stateflow chart models a real world drive with parameters such as trajectory and delay. Any drive subchart has the following data:



	Name		Port	Resc	DataType	Size	Initi	CompiledType	Compiled
	Axis	e Memory			Inherit: Same as Simulink	-1		AXIS_SERVO_DRIVE	1
	MI_MAM	e Memory			Inherit: Same as Simulink	-1		MOTION_INSTRUCTION	1
	trajectory			☐	ml			ml	1
	tStart			☐	double			double	1
	MI_MSO	e Memory			Inherit: Same as Simulink	-1		MOTION_INSTRUCTION	1
	MI_MSF	e Memory			Inherit: Same as Simulink	-1		MOTION_INSTRUCTION	1

5 Use the **Subchart Mappings** dialog 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). The “Simulating and Generating Structured Text Code for Rockwell Motion Instructions” example has the following mapping gor `Drive1`.



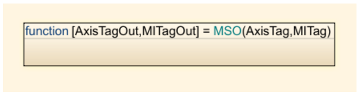
State Drive1

Description

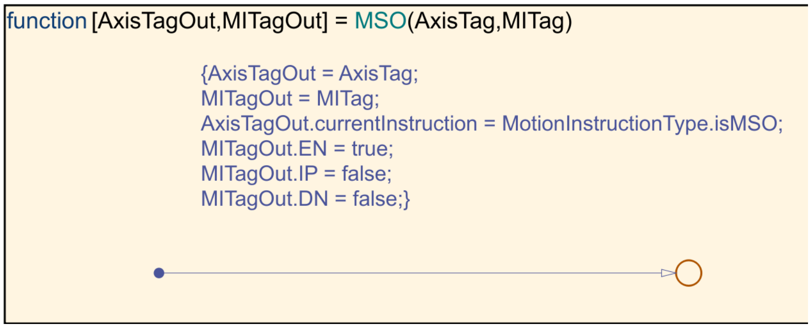
The following tables list the mappings from the subchart symbols (on the left) to the symbols in the main chart (on the right). For every subchart symbol, the drop-down provides the list of available symbols which that subchart symbol can map to.

ıtomic subchart symbc	Main chart symbol
MI_MSF	MI_MSF1
MI_MSO	MI_MSO1

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



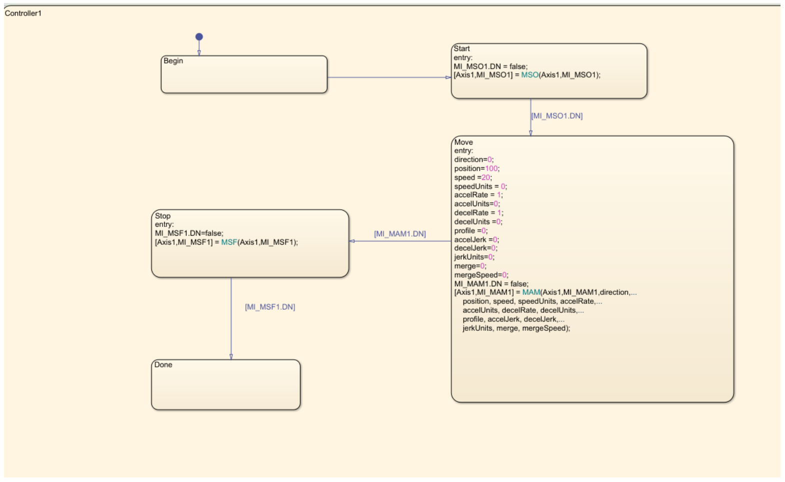
```
function [AxisTagOut,MITagOut] = MSO(AxisTag,MITag)
```



```
function [AxisTagOut,MITagOut] = MSO(AxisTag,MITag)

{AxisTagOut = AxisTag;
MITagOut = MITag;
AxisTagOut.currentInstruction = MotionInstructionType.isMSO;
MITagOut.EN = true;
MITagOut.IP = false;
MITagOut.DN = false;}
```

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. `Controller1` in the example has the following Stateflow chart.



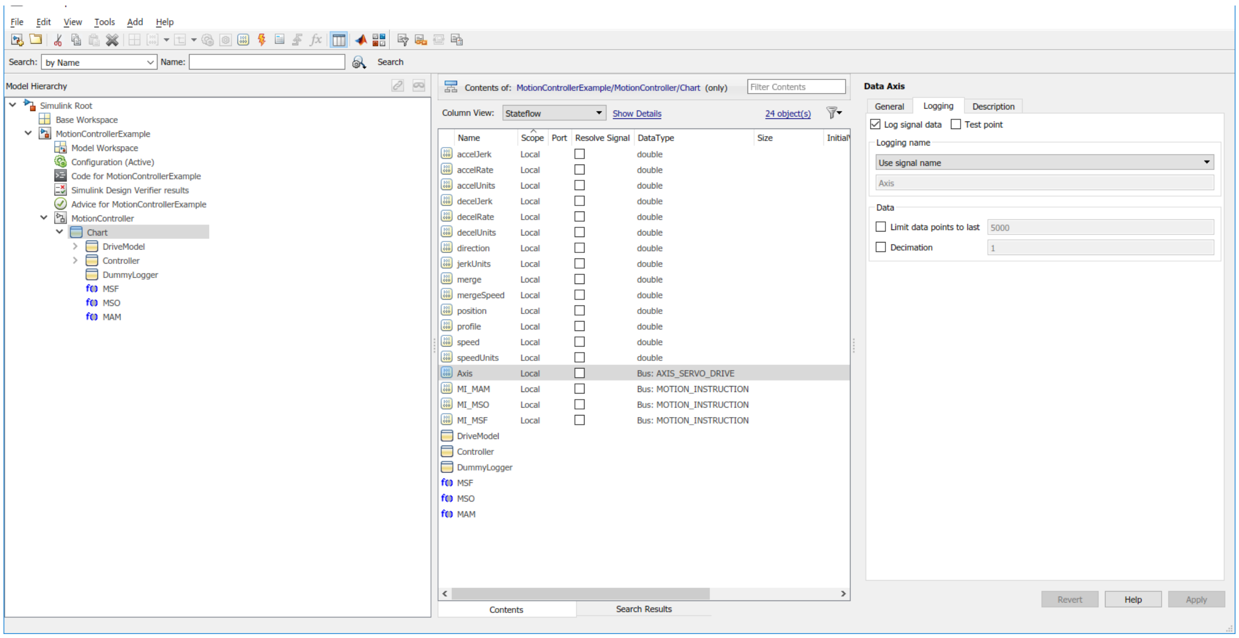
```
Start
entry:
MI_MSO1.DN = false;
[Axis1,MI_MSO1] = MSO(Axis1,MI_MSO1);
```

```
Move
entry:
direction=0;
position=100;
speed =20;
speedUnits = 0;
accelRate = 1;
accelUnits=0;
decelRate = 1;
decelUnits =0;
profile =0;
accelJerk =0;
decelJerk=0;
jerkUnits=0;
merge=0;
mergeSpeed=0;
MI_MAM1.DN = false;
[Axis1,MI_MAM1] = MAM(Axis1,MI_MAM1,direction,...
    position, speed, speedUnits, accelRate,...
    accelUnits, decelRate, decelUnits,...
    profile, accelJerk, decelJerk,...
    jerkUnits, merge, mergeSpeed);
```

```
Stop
entry:
MI_MSF1.DN=false;
[Axis1,MI_MSF1] = MSF(Axis1,MI_MSF1);
```

Simulation of the Motion API Model

You can run simulation on the model containing the motion instructions and see the state changes 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).



- Simulink Root
  - Base Workspace
  - MotionControllerExample
    - Model Workspace
    - Configuration (Active)
    - Code for MotionControllerExample
    - Simulink Design Verifier results
    - Advice for MotionControllerExample
    - MotionController
      - Chart
        - DriveModel
        - Controller
        - DummyLogger
        - MSF
        - MSO
        - MAM

Contents of: MotionControllerExample/MotionController/Chart (only)

Column View: Stateflow   Show Details   24 object(s)

Name	Scope	Port	Resolve Signal	DataType	Size	Initial
accelJerk	Local		☐	double		
accelRate	Local		☐	double		
accelUnits	Local		☐	double		
decelJerk	Local		☐	double		
decelRate	Local		☐	double		
decelUnits	Local		☐	double		
direction	Local		☐	double		
jerkUnits	Local		☐	double		
merge	Local		☐	double		
mergeSpeed	Local		☐	double		
position	Local		☐	double		
profile	Local		☐	double		
speed	Local		☐	double		
speedUnits	Local		☐	double		
Axis	Local		☐	Bus: AXIS_SERVO_DRIVE		
MI_MAM	Local		☐	Bus: MOTION_INSTRUCTION		
MI_MSO	Local		☐	Bus: MOTION_INSTRUCTION		
MI_MSF	Local		☐	Bus: MOTION_INSTRUCTION		
DriveModel						
Controller						
DummyLogger						
MSF						
MSO						
MAM						

Data Axis

General | Logging | Description

☑ Log signal data ☐ Test point

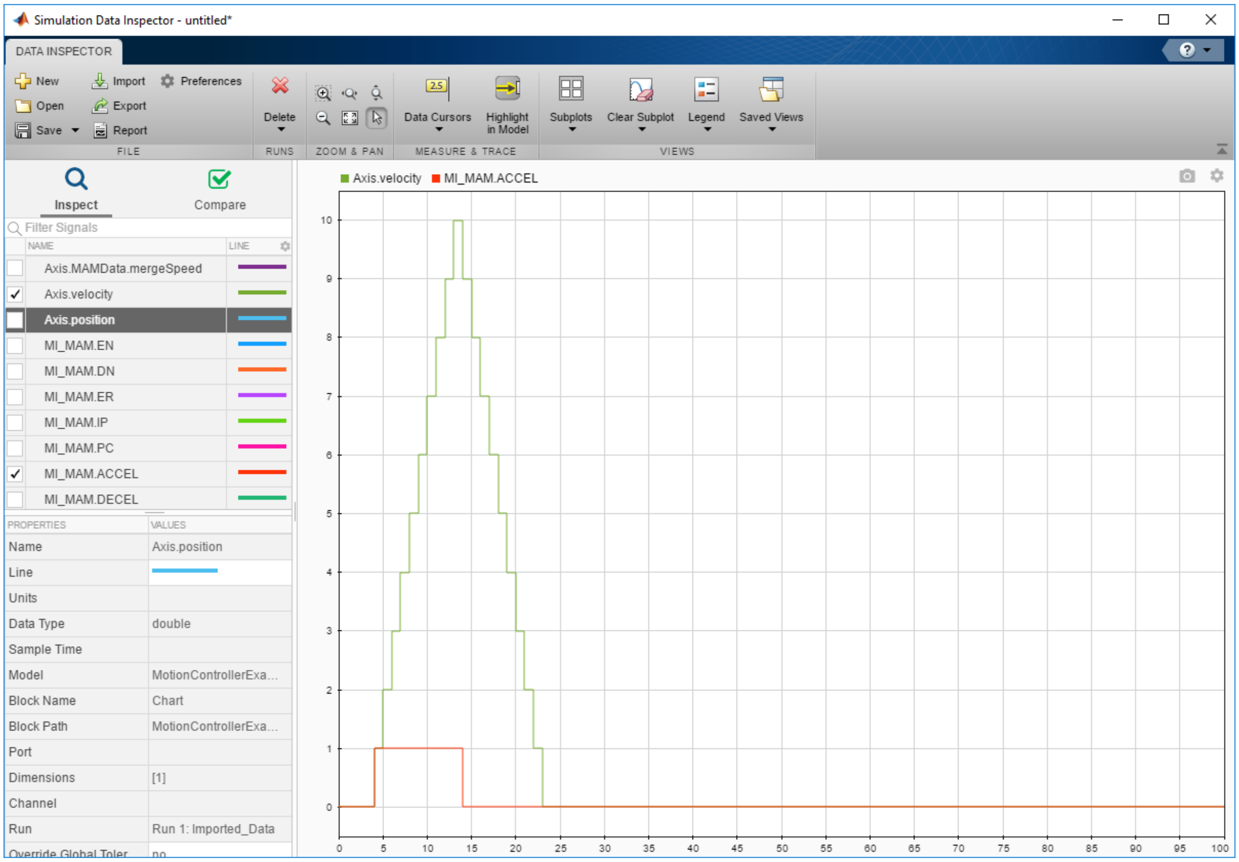
Logging name: Use signal name — Axis

Data

☐ Limit data points to last 5000

☐ Decimation 1

At the end of simulation, the logged signals are captured in the base workspace as a variable called `logsout`. This can be imported into Simulation Data Inspector.



NAME	LINE
☐ Axis.MAMData.mergeSpeed	
✓ Axis.velocity	
☐ Axis.position	
☐ MI_MAM.EN	
☐ MI_MAM.DN	
☐ MI_MAM.ER	
☐ MI_MAM.IP	
☐ MI_MAM.PC	
✓ MI_MAM.ACCEL	
☐ MI_MAM.DECEL	

PROPERTIES	VALUES
Name	Axis.position
Line	
Units	
Data Type	double
Sample Time	
Model	MotionControllerExa...
Block Name	Chart
Block Path	MotionControllerExa...
Port	
Dimensions	[1]
Channel	
Run	Run 1: Imported_Data
Override Global Toler	no

Structured Text Code Generation

Use the `plcgeneratemotionapicode` function to prepare the model for code generation and generate structured text code. The `plcgeneratemotionapicode` takes the full path name of subsystem containing the original chart as an input and creates a new model from which structured text code can be generated.

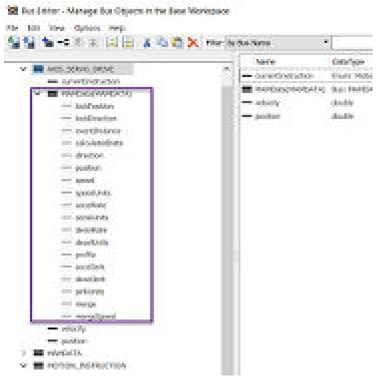
Adding Support for Other Motion Instructions

The `plcdemo_motion_api_rockwell` example has support for only the following motion instructions:

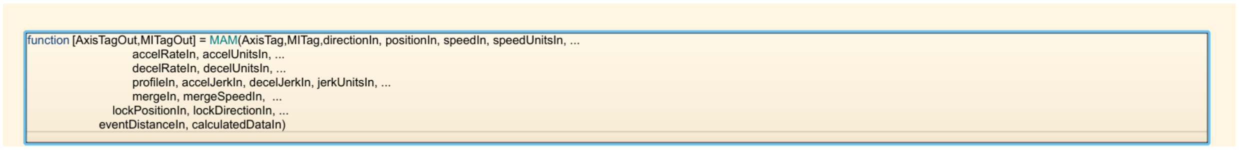
- `MAM`
- `MAS`
- `MSF`
- `MSO`

To use other Rockwell Automation RSLogix motion instructions in the model (For example, `Motion Axis Jog (MAJ)`), you must perform the following steps:

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

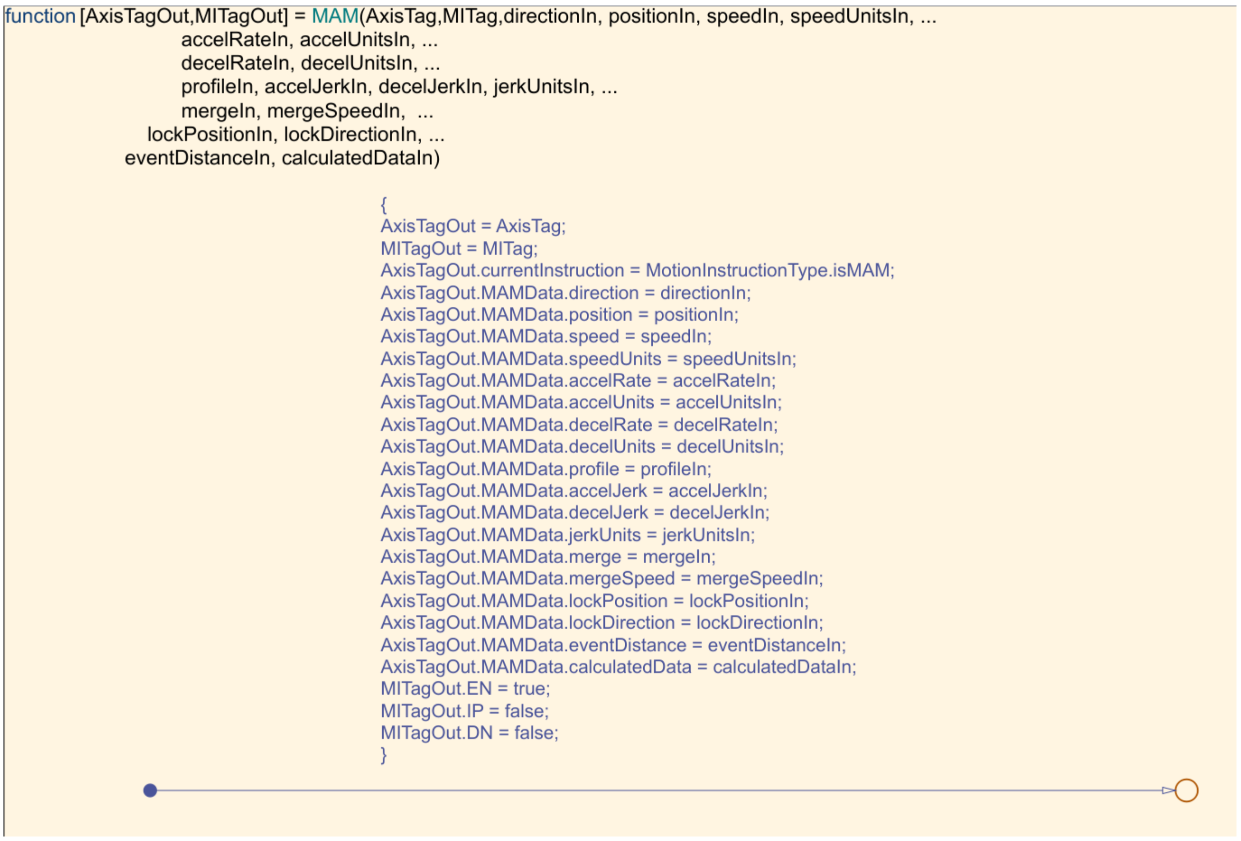


2. Update the `MotionTypesForSim.mat` file with the new definitions for `MAJDATA` and `AXIS_SERVO_DRIVE`.
3. In the Stateflow chart, create a graphical function representing `MAJ` (similar to `MAM`). Assign the appropriate inputs and outputs.



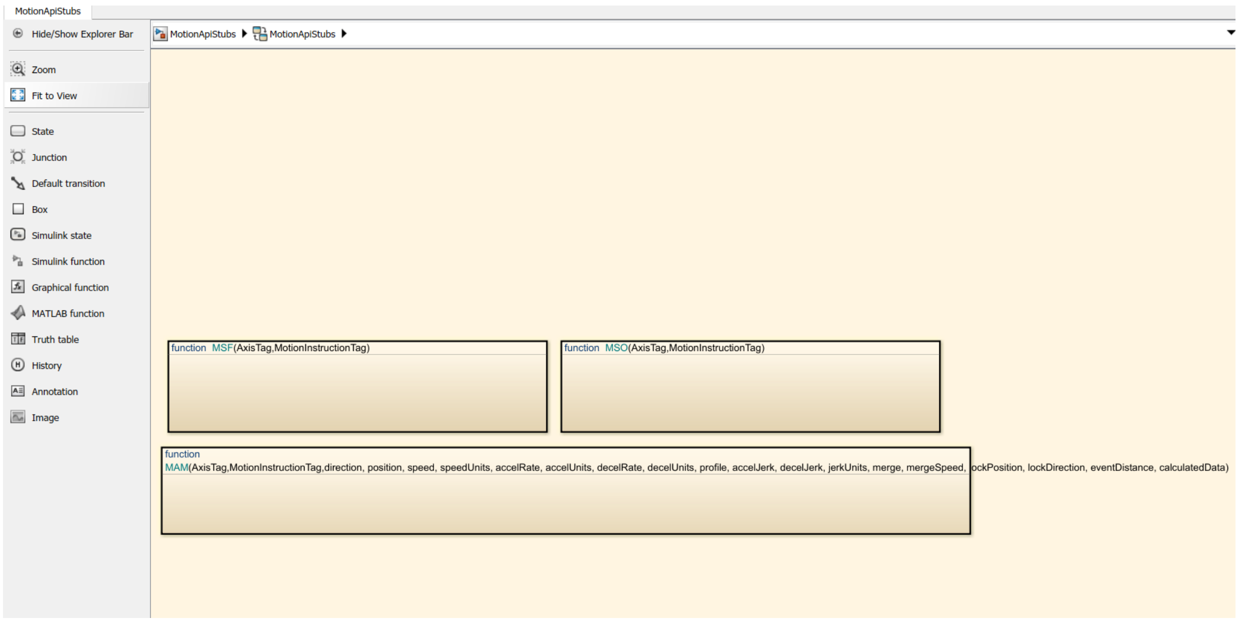
```
function [AxisTagOut,MITagOut] = MAM(AxisTag,MITag,directionIn, positionIn, speedIn, speedUnitsIn, ...
                        accelRateIn, accelUnitsIn, ...
                        decelRateIn, decelUnitsIn, ...
                        profileIn, accelJerkIn, decelJerkIn, jerkUnitsIn, ...
     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

4. Create 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`.

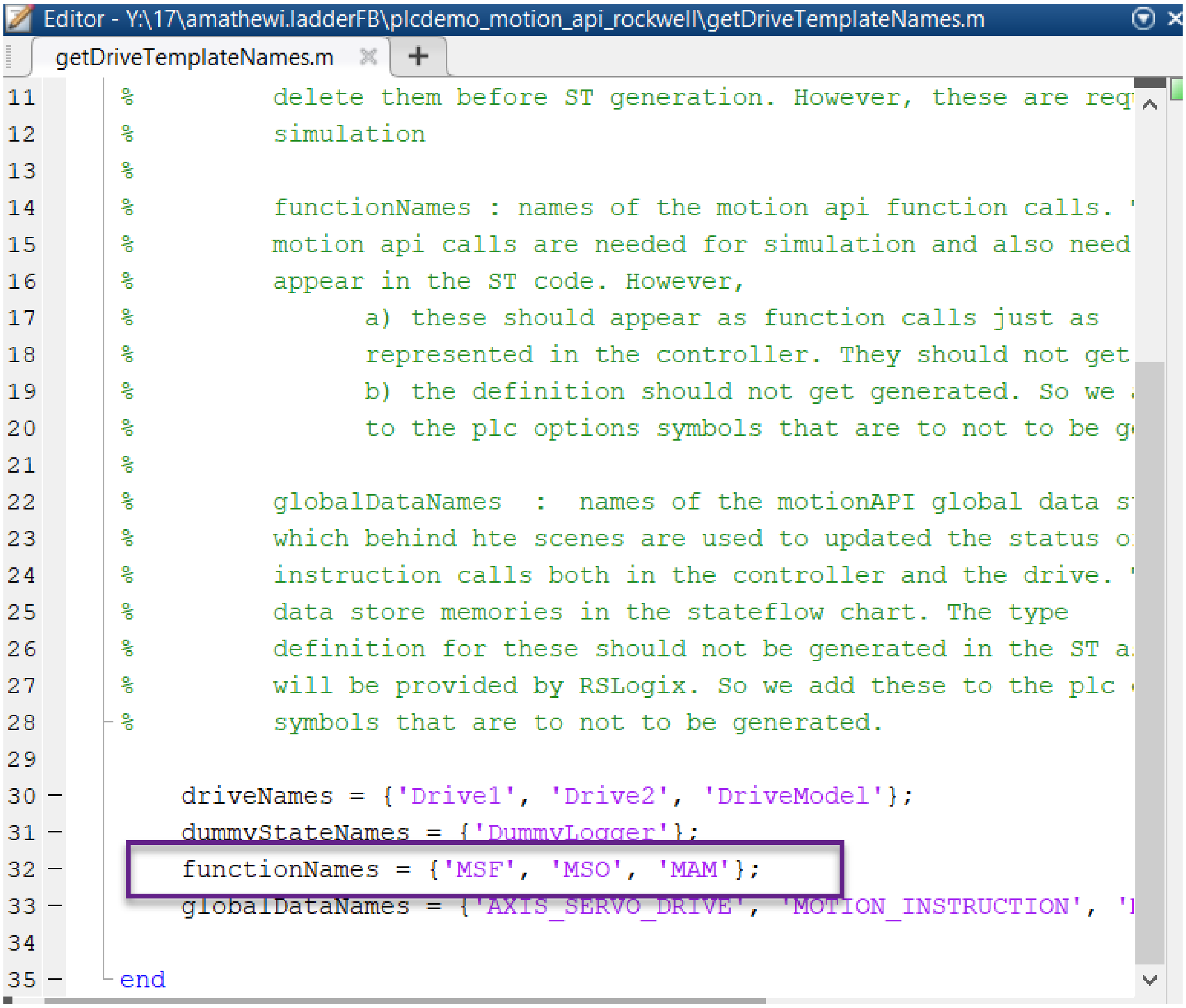


```
function  MSF(AxisTag,MotionInstructionTag)
```

```
function  MSO(AxisTag,MotionInstructionTag)
```

```
function
MAM(AxisTag,MotionInstructionTag,direction, position, speed, speedUnits, accelRate, accelUnits, decelRate, decelUnits, profile, accelJerk, decelJerk, jerkUnits, merge, mergeSpeed, ockPosition, lockDirection, eventDistance, calculatedData)
```

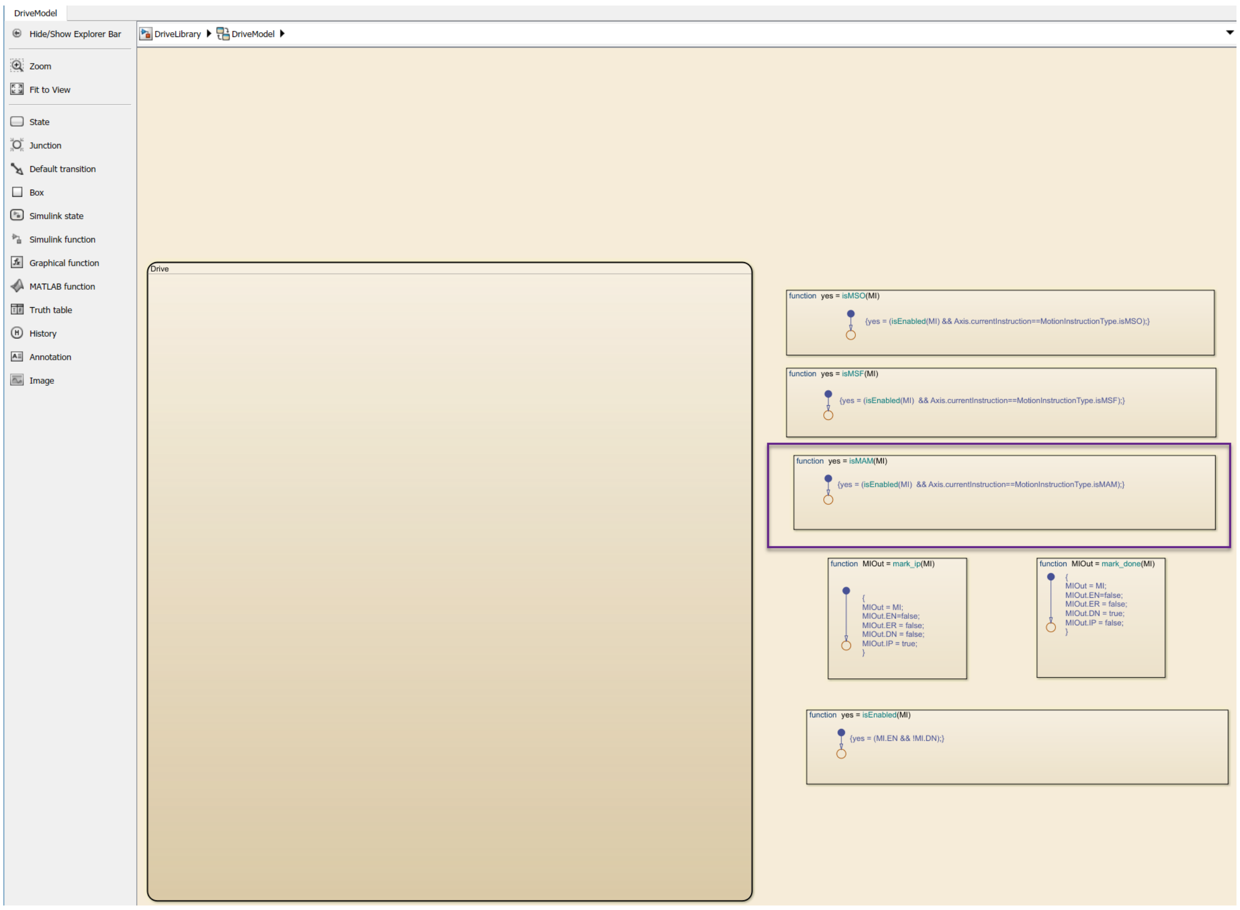
6 Update the `functionName` variable in the `getDriveTemplateNames.m` file to include `MAJ`.


Editor - Y:\17\amathewi.ladderFB\plcdemo_motion_api_rockwell\getDriveTemplateNames.m
getDriveTemplateNames.m

```
%         delete them before ST generation. However, these are req
%         simulation
%
%         functionNames : names of the motion api function calls. '
%         motion api calls are needed for simulation and also need
%         appear in the ST code. However,
%               a) these should appear as function calls just as
%               represented in the controller. They should not get
%               b) the definition should not get generated. So we .
%               to the plc options symbols that are to not to be g
%
%         globalDataNames  :  names of the motionAPI global data s
%         which behind hte scenes are used to updated the status o
%         instruction calls both in the controller and the drive. '
%         data store memories in the stateflow chart. The type
%         definition for these should not be generated in the ST a
%         will be provided by RSLogix. So we add these to the plc
%         symbols that are to not to be generated.

    driveNames = {'Drive1', 'Drive2', 'DriveModel'};
    dummyStateNames = {'DummyLogger'};
    functionNames = {'MSF', 'MSO', 'MAM'};
    globalDataNames = {'AXIS_SERVO_DRIVE', 'MOTION_INSTRUCTION', 'I

end
```

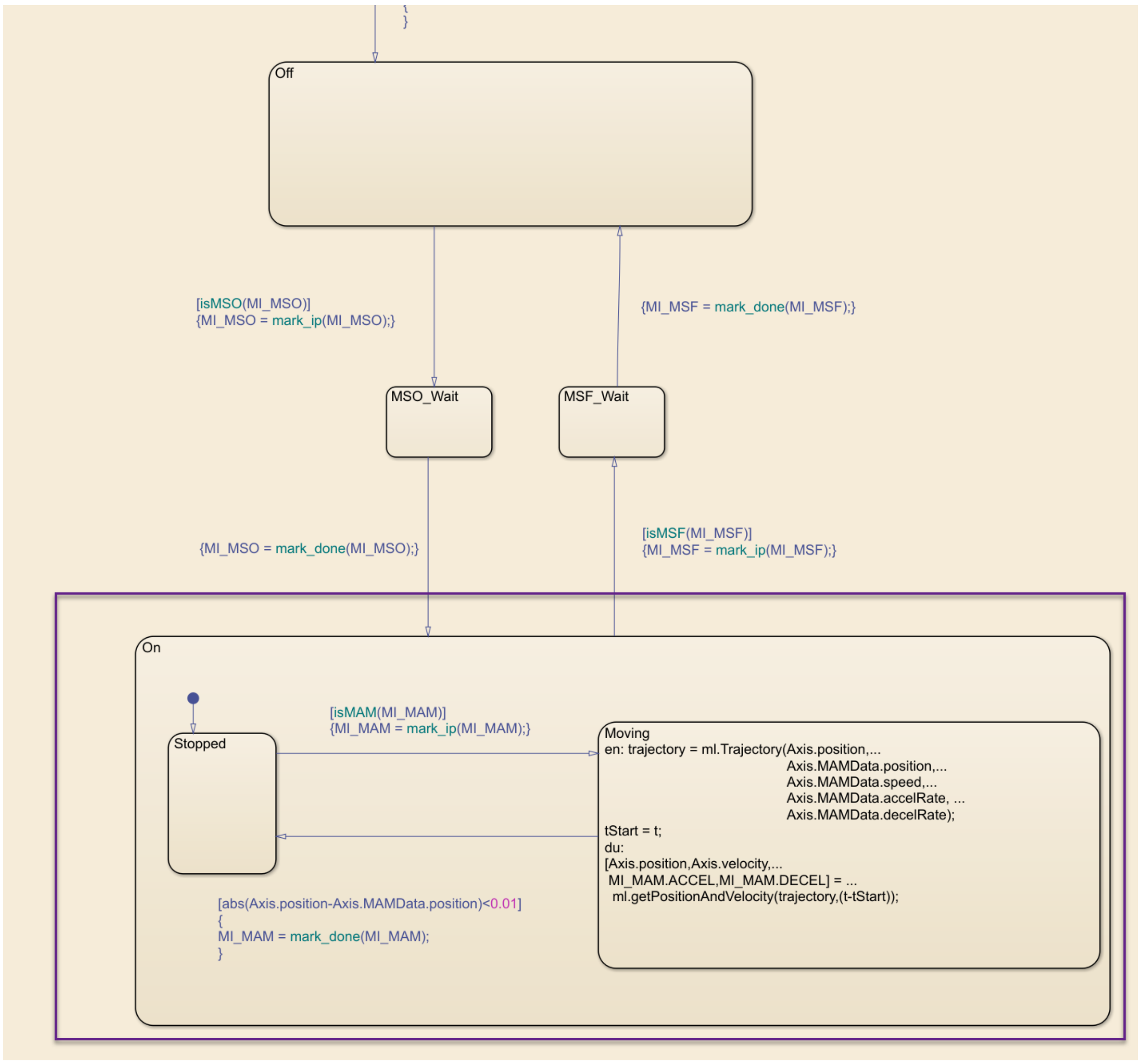
7 Update the `DriveLibrary.slx` file to respond to `MAJ` calls during simulation.

- Create `isMAJ` graphical function (similar to `isMAM`).



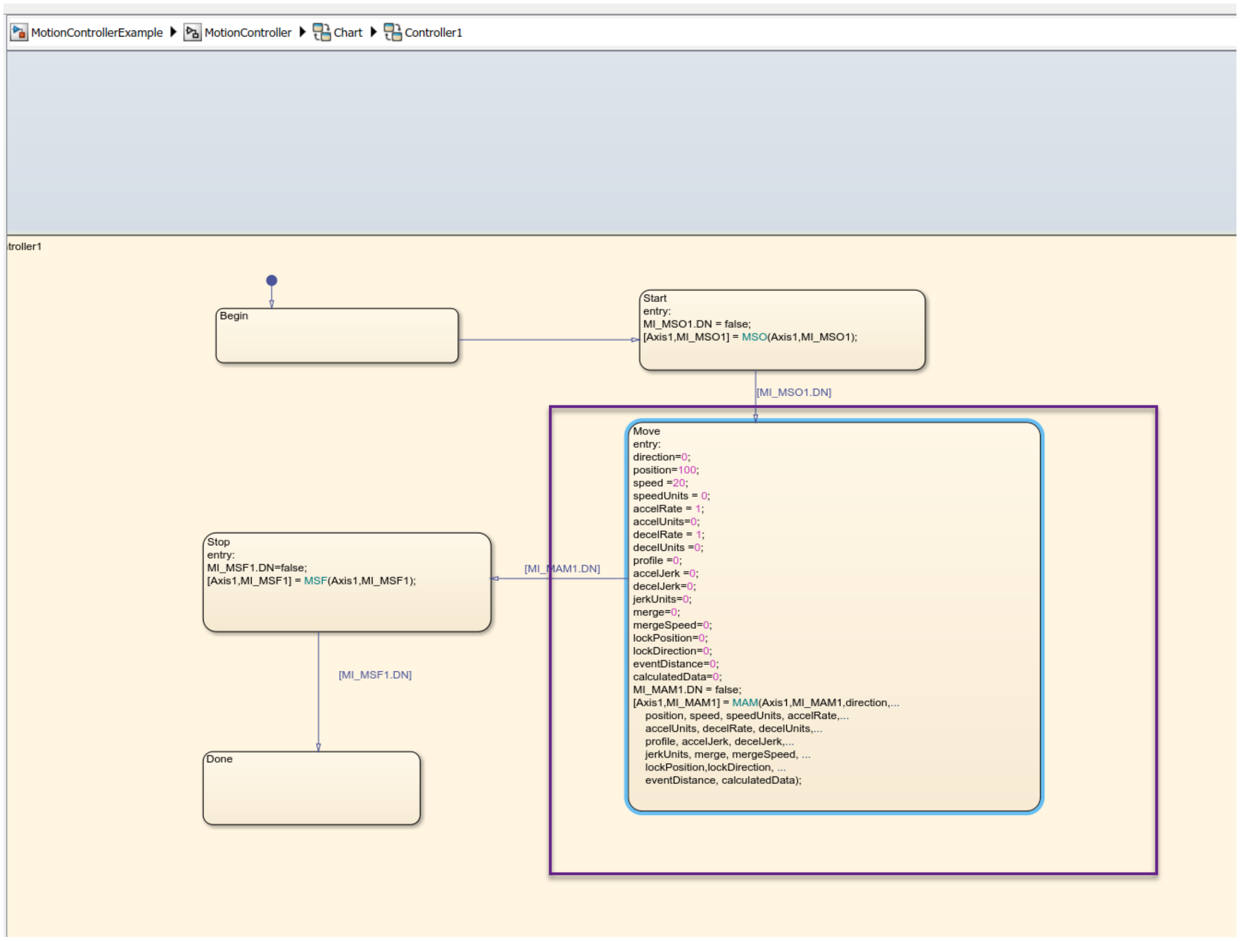
```
function  yes = isMSO(MI)
{yes = (isEnabled(MI) && Axis.currentInstruction==MotionInstructionType.isMSO);}

function  yes = isMSF(MI)
{yes = (isEnabled(MI)  && Axis.currentInstruction==MotionInstructionType.isMSF);}

function  yes = isMAM(MI)
{yes = (isEnabled(MI)  && Axis.currentInstruction==MotionInstructionType.isMAM);}

function  MIOut = mark_ip(MI)
{
MIOut = MI;
MIOut.EN=false;
MIOut.ER = false;
MIOut.DN = false;
MIOut.IP = true;
}

function  MIOut = mark_done(MI)
{
MIOut = MI;
MIOut.EN=false;
MIOut.ER = false;
MIOut.DN = true;
MIOut.IP = false;
}

function  yes = isEnabled(MI)
{yes = (MI.EN && !MI.DN);}
```

- Update the `Drive` subchart to respond to `MAJ` by implementing required transitions etc (similar to `MAM` as shown).


{
}
Off
[isMSO(MI_MSO)]
{MI_MSO = mark_ip(MI_MSO);}
{MI_MSF = mark_done(MI_MSF);}
MSO_Wait
MSF_Wait
{MI_MSO = mark_done(MI_MSO);}
[isMSF(MI_MSF)]
{MI_MSF = mark_ip(MI_MSF);}
On
[isMAM(MI_MAM)]
{MI_MAM = mark_ip(MI_MAM);}
Stopped
Moving
[abs(Axis.position-Axis.MAMData.position)<0.01]
{
MI_MAM = mark_done(MI_MAM);
}

```
Moving
en: trajectory = ml.Trajectory(Axis.position,...
                               Axis.MAMData.position,...
                               Axis.MAMData.speed,...
                               Axis.MAMData.accelRate, ...
                               Axis.MAMData.decelRate);
tStart = t;
du:
[Axis.position,Axis.velocity,...
 MI_MAM.ACCEL,MI_MAM.DECEL] = ...
  ml.getPositionAndVelocity(trajectory,(t-tStart));
```

8 Create or update the controller logic as required. Create a new state and add `MAJ` instruction to it (similar to the `MAM`)



```
Begin
```

```
Start
entry:
MI_MSO1.DN = false;
[Axis1,MI_MSO1] = MSO(Axis1,MI_MSO1);
```

```
Move
entry:
direction=0;
position=100;
speed =20;
speedUnits = 0;
accelRate = 1;
accelUnits=0;
decelRate = 1;
decelUnits =0;
profile =0;
accelJerk =0;
decelJerk=0;
jerkUnits=0;
merge=0;
mergeSpeed=0;
lockPosition=0;
lockDirection=0;
eventDistance=0;
calculatedData=0;
MI_MAM1.DN = false;
[Axis1,MI_MAM1] = MAM(Axis1,MI_MAM1,direction,...
    position, speed, speedUnits, accelRate,...
    accelUnits, decelRate, decelUnits,...
    profile, accelJerk, decelJerk,...
    jerkUnits, merge, mergeSpeed, ...
    lockPosition,lockDirection, ...
    eventDistance, calculatedData);
```

```
Stop
entry:
MI_MSF1.DN=false;
[Axis1,MI_MSF1] = MSF(Axis1,MI_MSF1);
```

```
Done
```

9 Perform simulation and generate code using the steps described earlier.

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 1-17.

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

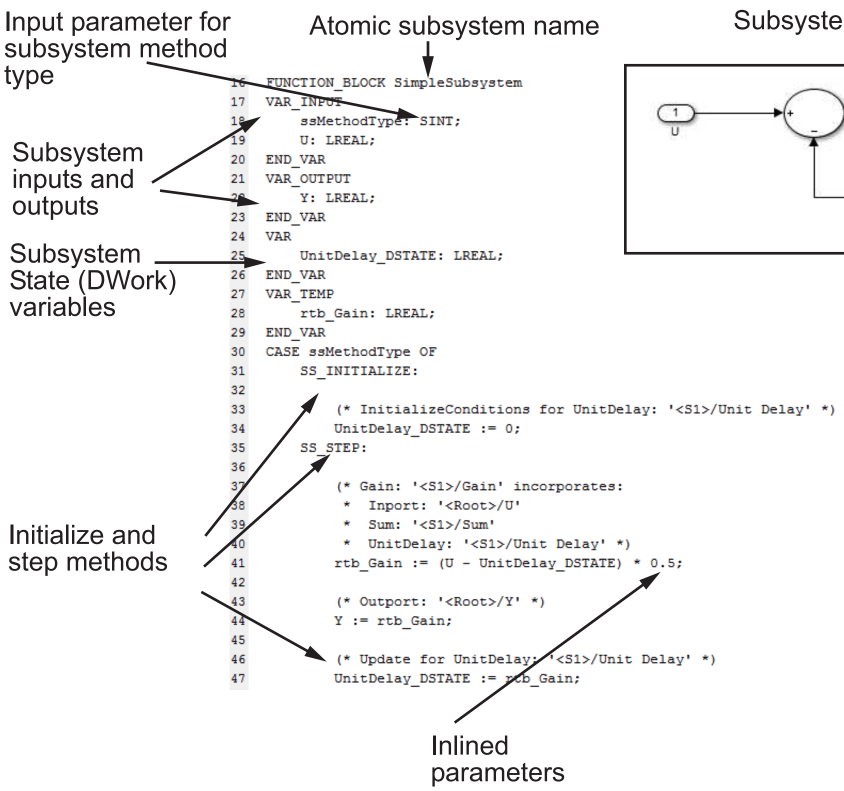
1 If you do not have the `plcdemo_simple_subsystem.exp` file open, open it in the MATLAB editor. In the folder that contains the file, type:

   ```
   edit plcdemo_simple_subsystem.exp
   ```

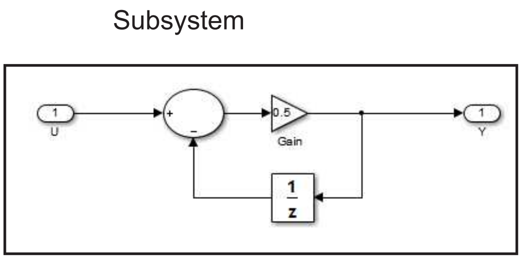
   A file like the following is displayed.

   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.


Input parameter for subsystem method type
Atomic subsystem name
Subsystem inputs and outputs
Subsystem State (DWork) variables
Initialize and step methods
Inlined parameters

```
FUNCTION_BLOCK SimpleSubsystem
VAR_INPUT
    ssMethodType: SINT;
    U: LREAL;
END_VAR
VAR_OUTPUT
    Y: LREAL;
END_VAR
VAR
    UnitDelay_DSTATE: LREAL;
END_VAR
VAR_TEMP
    rtb_Gain: LREAL;
END_VAR
CASE ssMethodType OF
    SS_INITIALIZE:

        (* InitializeConditions for UnitDelay: '<S1>/Unit Delay' *)
        UnitDelay_DSTATE := 0;
    SS_STEP:

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

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

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



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

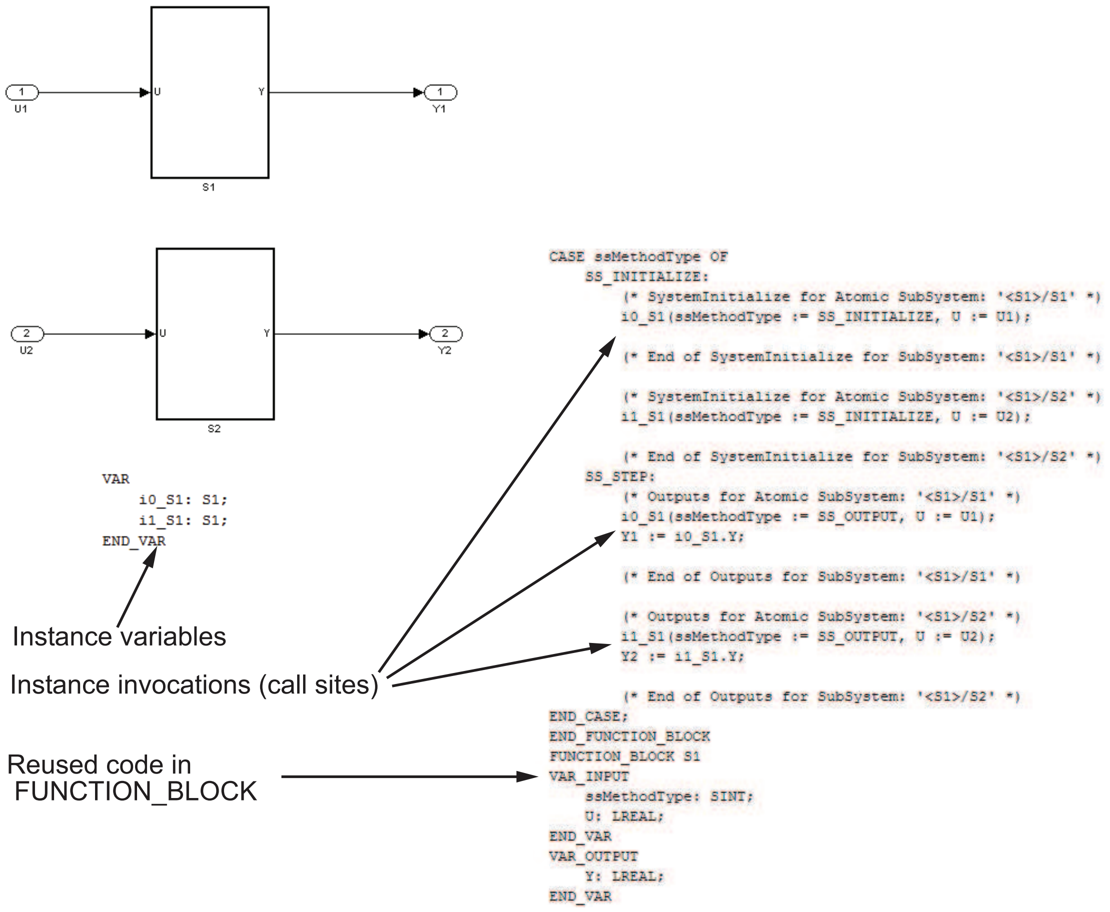
This topic assumes that you have generated Structured Text code from a Simulink model. If you have not yet done so, see “Generate Structured Text from the Model Window” on page 1-17.

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

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

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

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


1
U1
U
Y
S1
1
Y1
2
U2
U
Y
S2
2
Y2
Instance variables
Instance invocations (call sites)
Reused code in FUNCTION_BLOCK

```
VAR
    i0_S1: S1;
    i1_S1: S1;
END_VAR
```

```
CASE ssMethodType OF
    SS_INITIALIZE:
        (* SystemInitialize for Atomic SubSystem: '<S1>/S1' *)
        i0_S1(ssMethodType := SS_INITIALIZE, U := U1);

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

        (* SystemInitialize for Atomic SubSystem: '<S1>/S2' *)
        i1_S1(ssMethodType := SS_INITIALIZE, U := U2);

        (* End of SystemInitialize for SubSystem: '<S1>/S2' *)
    SS_STEP:
        (* Outputs for Atomic SubSystem: '<S1>/S1' *)
        i0_S1(ssMethodType := SS_OUTPUT, U := U1);
        Y1 := i0_S1.Y;

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

        (* Outputs for Atomic SubSystem: '<S1>/S2' *)
        i1_S1(ssMethodType := SS_OUTPUT, U := U2);
        Y2 := i1_S1.Y;

        (* End of Outputs for SubSystem: '<S1>/S2' *)
END_CASE;
END_FUNCTION_BLOCK
FUNCTION_BLOCK S1
VAR_INPUT
    ssMethodType: SINT;
    U: LREAL;
END_VAR
VAR_OUTPUT
    Y: LREAL;
END_VAR
```

4. 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.

5. 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

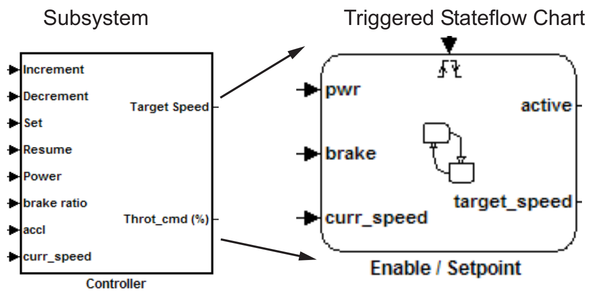
This topic assumes that you have generated Structured Text code from a Simulink model. If you have not yet done so, see “Generate Structured Text from the Model Window” on page 1-17.

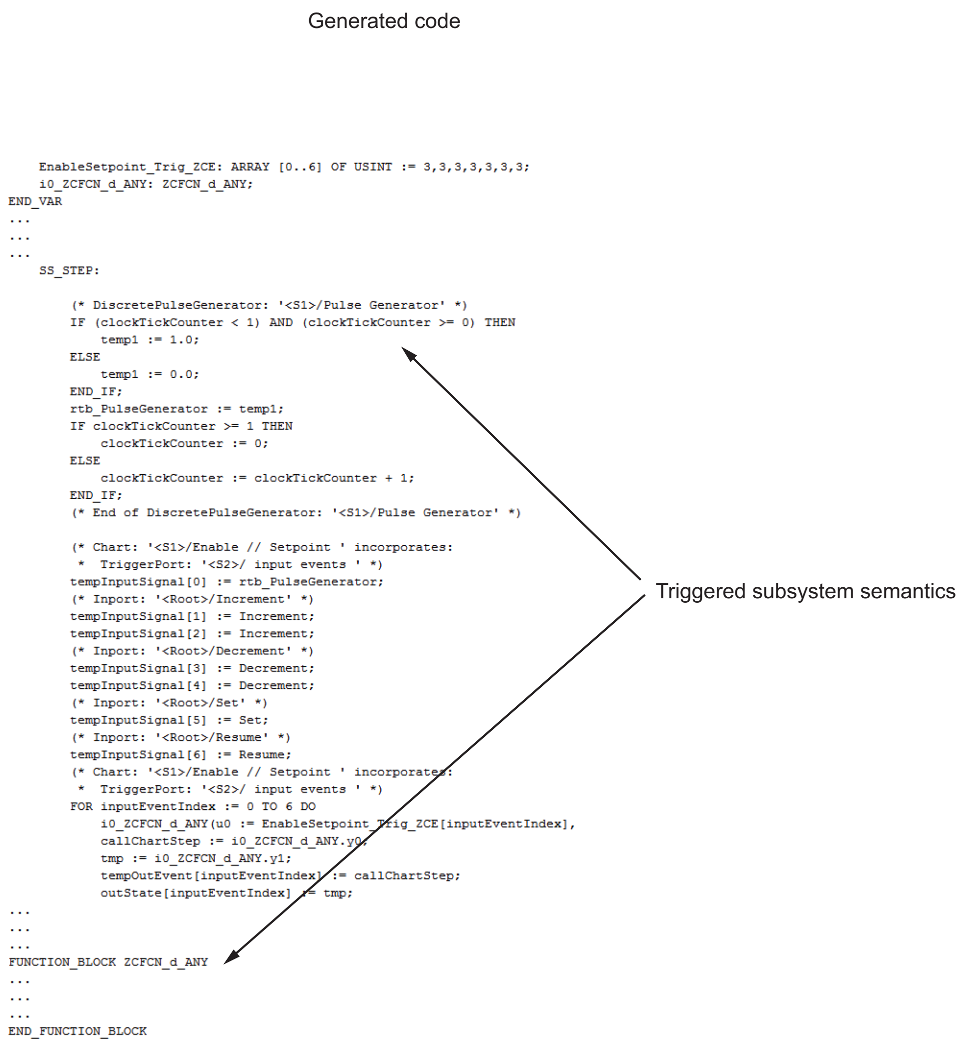
The example in this topic shows generated code for the CoDeSys Version 2.3 PLC IDE. Generated code for other IDE platforms looks different.

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

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

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





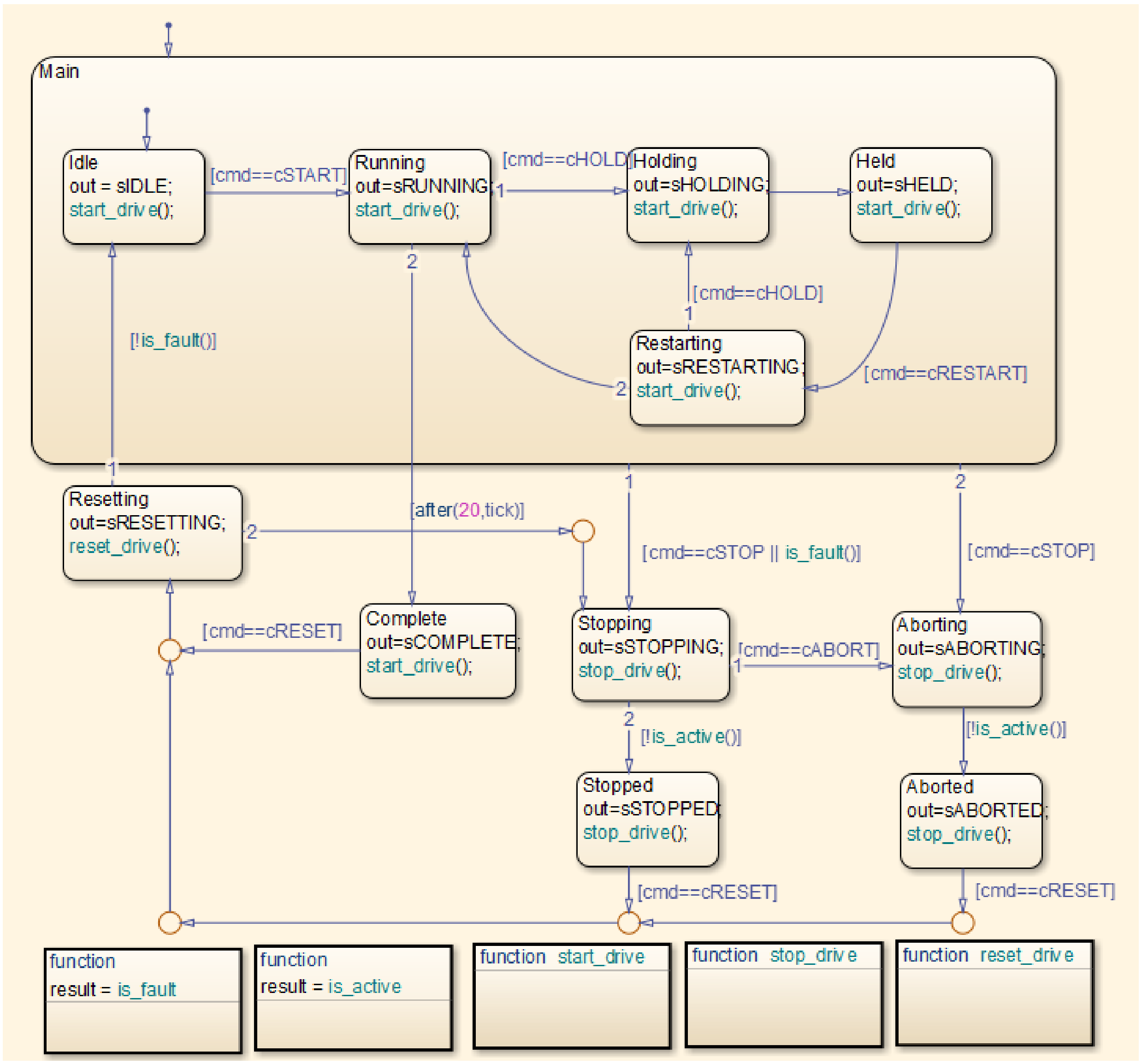
```
    EnableSetpoint_Trig_ZCE: ARRAY [0..6] OF USINT := 3,3,3,3,3,3,3;
    i0_ZCFCN_d_ANY: ZCFCN_d_ANY;
END_VAR
...
...
...
    SS_STEP:

        (* 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
```

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:



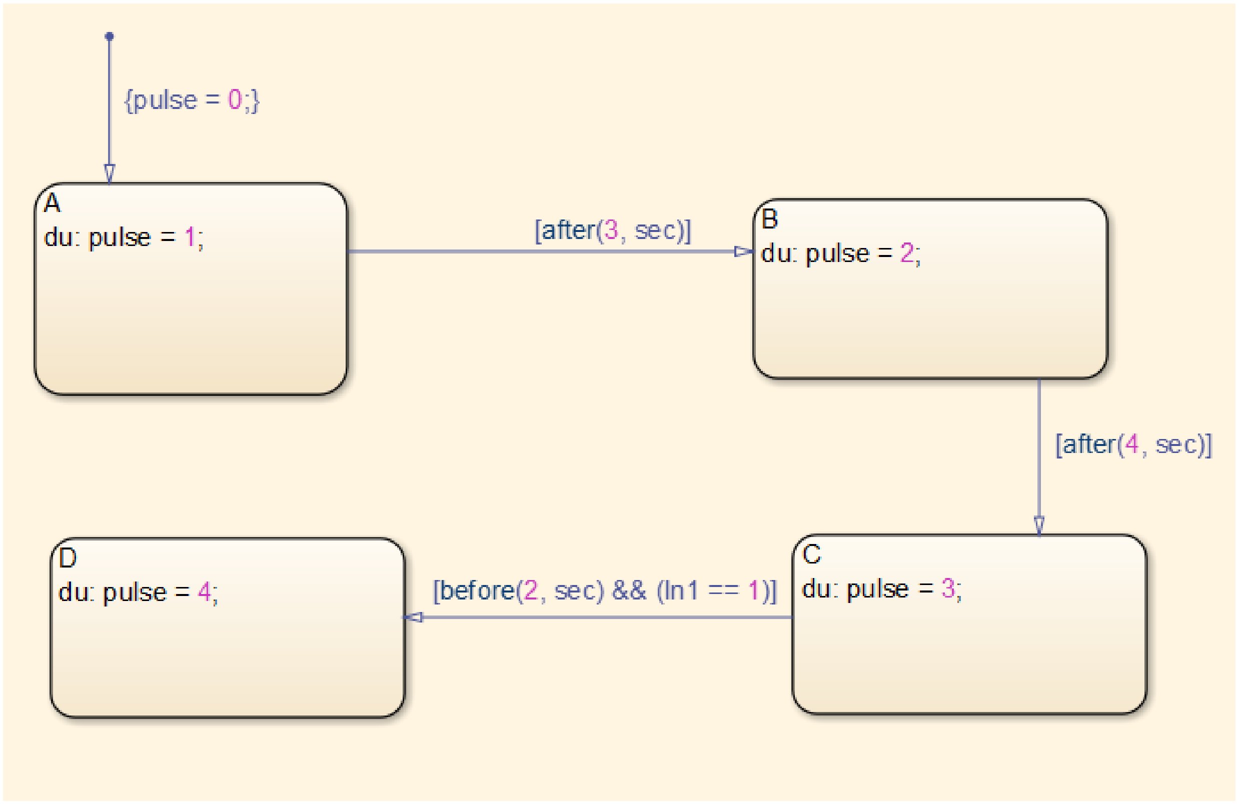
You can map the states and transitions in the chart to the generated code. For instance, the transition from the state `Aborting` to `Aborted` appears in the generated code as:

```
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 7-9.

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

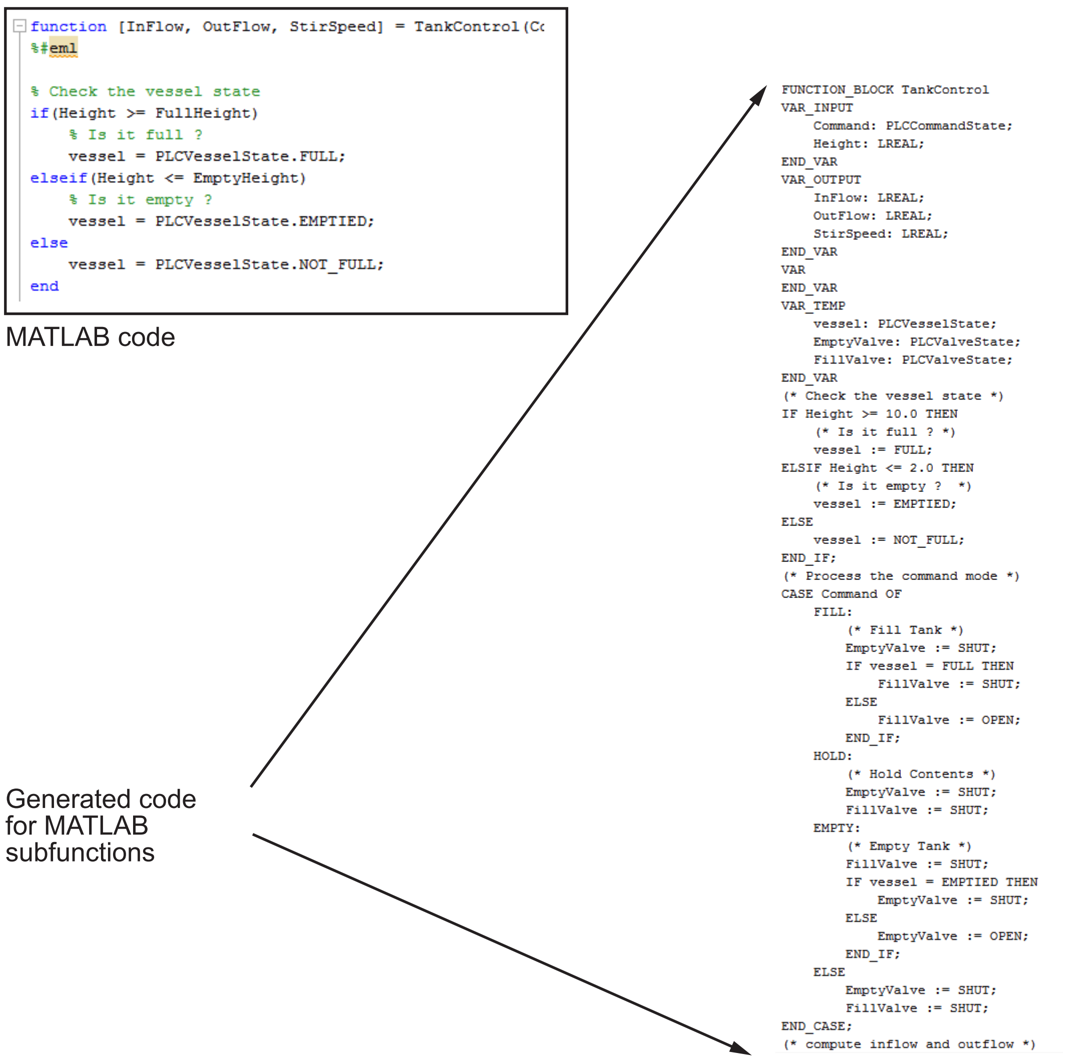
This topic assumes that you have generated Structured Text code from a Simulink model. If you have not yet done so, see “Generate Structured Text from the Model Window” on page 1-17.

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

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

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

   The following figure illustrates the mapping of the generated code to Structured Text components for a Simulink Subsystem block that contains a MATLAB Function block. The coder tries to perform inline optimization on the generated code for MATLAB 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`.
4. Examine the generated Structured Text code.



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

```
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 1-17.

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 Right-click the `SimpleSubsystem` block and select **PLC Code** > **Generate Code for Subsystem**.

   The Simulink PLC Coder software generates Structured Text code and places it in *`current_folder`*`/plcsrc/plcdemo_multirate.exp`.
3 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:
```

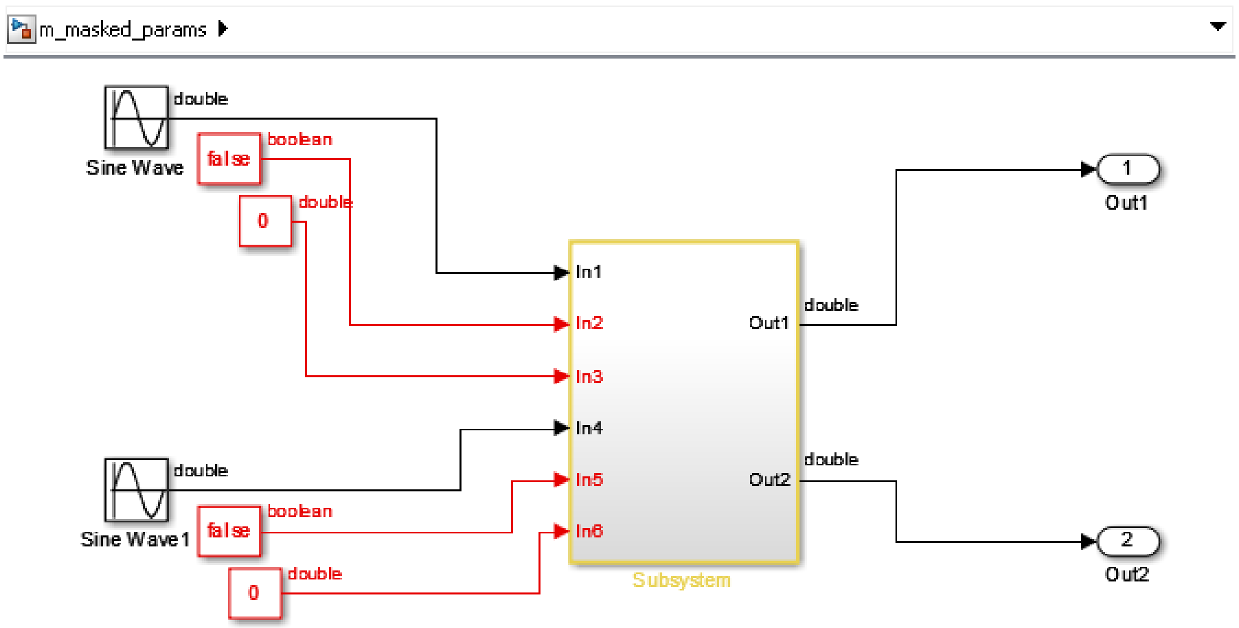
```
(* Gain: '<S1>/Gain' incorporates:
 *  Inport: '<Root>/U1'
 *  Sum: '<S1>/Sum'
 *  UnitDelay: '<S1>/Unit Delay' *)
rtb_Gain := (U1 - UnitDelay_DSTATE) * 0.5;

(* 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;
```

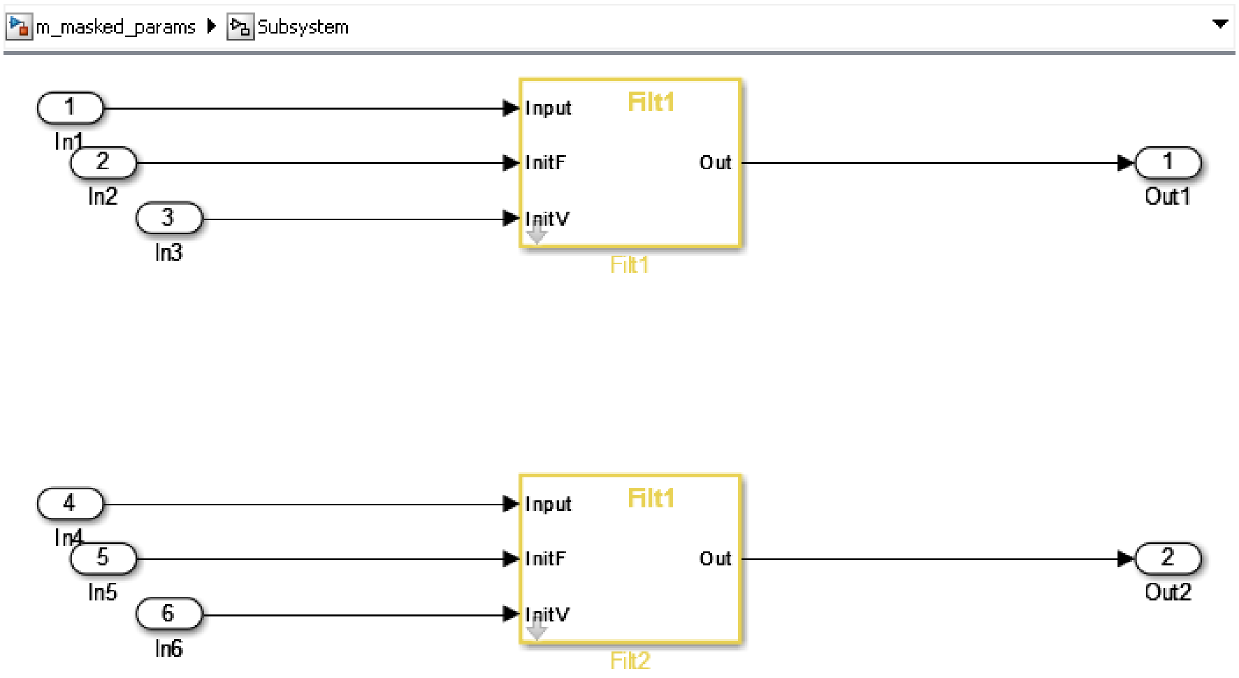
In general, for a subsystem with n different sample times, the generated code has `n-1` time step counter variables, corresponding to the `n-1` slower rates. Code generated from parts of the model running at the slower rates executes conditionally, based on the corresponding time step counter values.

Generated Code Structure for Subsystem Mask Parameters

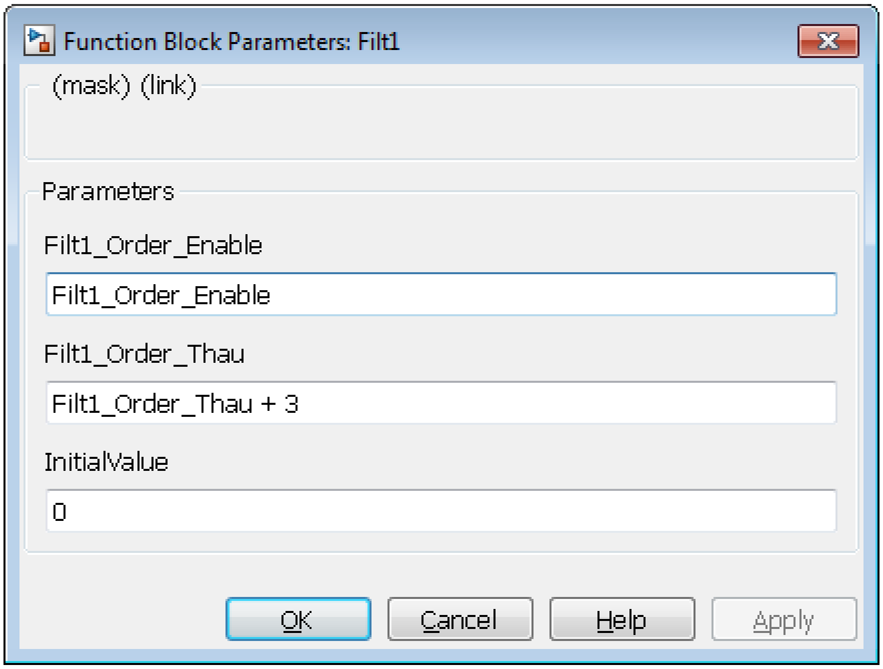
In the generated code for masked subsystems, the mask parameters map to function block inputs. The values you specify in the subsystem mask are assigned to these function block inputs in the generated code.

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.


Function Block Parameters: Filt1

(mask) (link)

Parameters

Filt1_Order_Enable

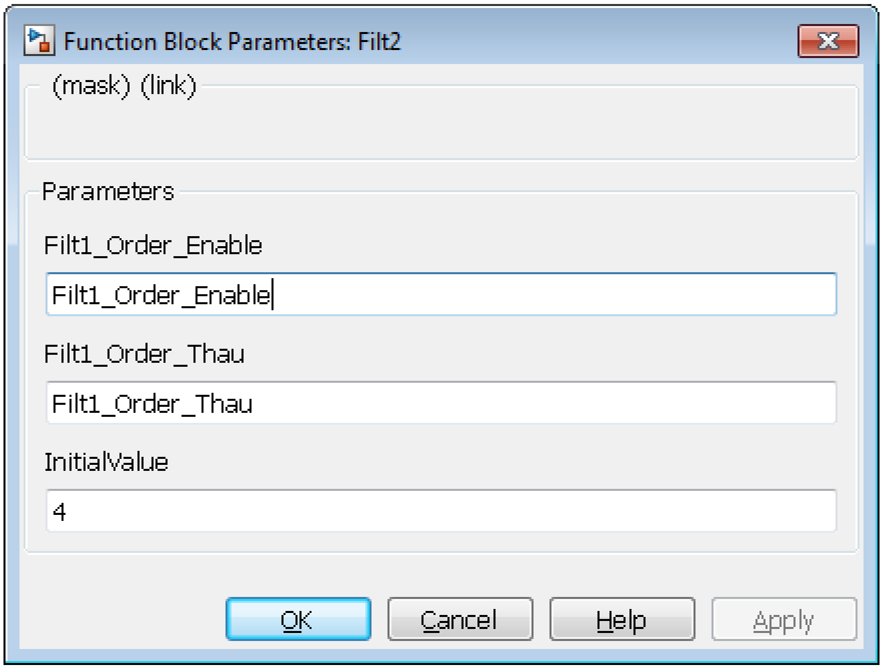
Filt1_Order_Enable

Filt1_Order_Thau

Filt1_Order_Thau + 3

InitialValue

0



Filt1_Order_Enable: Filt1_Order_Enable
Filt1_Order_Thau: Filt1_Order_Thau
InitialValue: 4

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 Non-Intrinsic Math Functions

When Simulink PLC Coder encounters a math function that is not intrinsic, it generates Structured Text by replacing the non-intrinsic 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. The issue can be addressed 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. So if *x>19*, return a value of 1.0000.

See Also

Generating Ladder Diagram

Ladder Diagram Generation for PLC Controllers

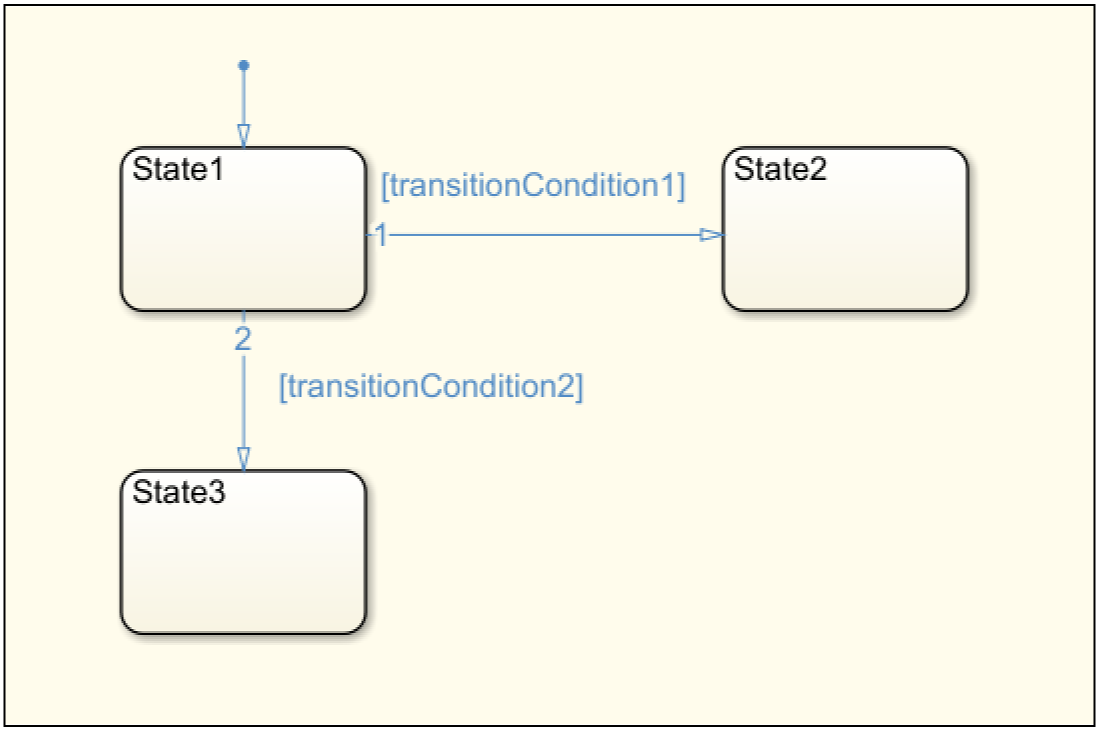
Ladder Diagram (LD) is a graphical programming language used to develop software for programmable logic controllers (PLCs). It is one of the languages that the IEC 61131 Standard specifies for use with PLCs.

A program in Ladder Diagram notation is a circuit diagram that emulates circuits of relay logic hardware. The underlying program uses Boolean expressions that translate readily to switches and relays. When you program complex applications directly in Ladder Diagram notation, it is challenging because you must write the programs with only Boolean variables and expressions.

Using Simulink PLC Coder, you can generate Ladder Diagram code for your applications from a Stateflow chart. The benefits are:

- You can design your application by using states and transitions in a Stateflow chart. Once you complete the design, you can generate Ladder Diagram code in XML or another format. You then import the generated code to an IDE such as CODESYS 3.5 or RSLogix AOI 5000 and view the Ladder Diagram.
- When you test your Stateflow chart by using a set of inputs, you can reuse these inputs to create a test bench for the Ladder Diagram code. You import the test bench to your PLC IDE and compare the results of simulation with the results of running the Ladder Diagram. If the results agree, the original Stateflow chart is equivalent to the generated Ladder Diagram code.

The figure shows a simple Stateflow chart with three states and two transitions. Based on the transition conditions, the chart transitions from one state to another.



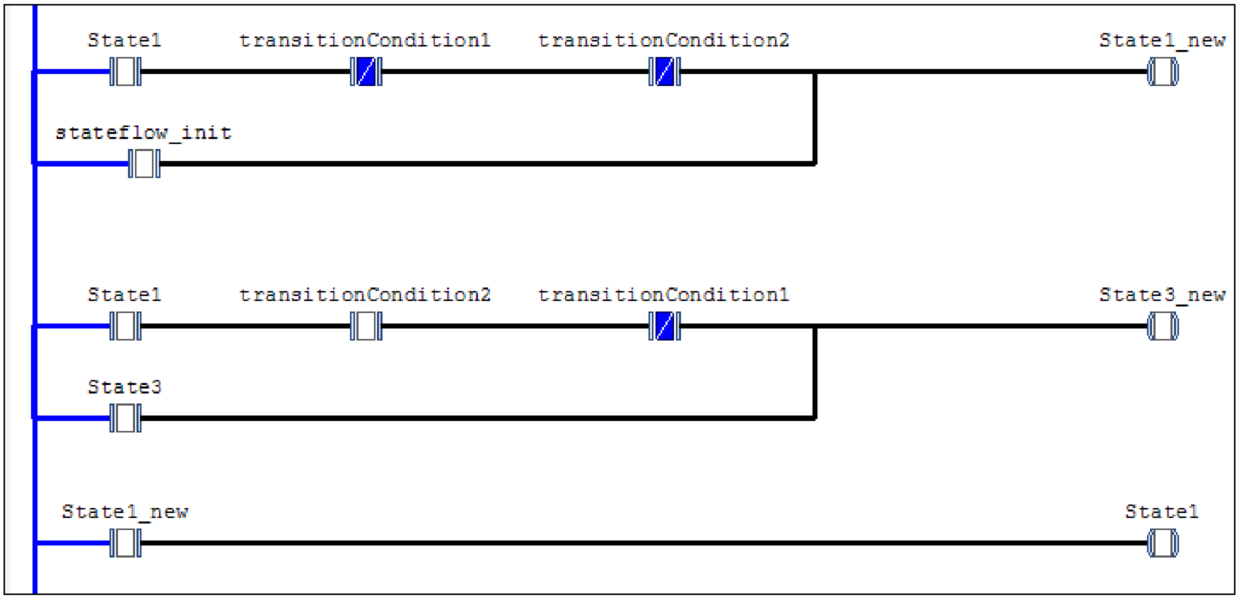
The state `State1` is active as long `transitionCondition1` and `transitionCondition2` are not true. This means, `State1` is active in one of these two cases:

- The chart has started execution through the default transition.
- The previous active state is also `State1` and the conditions `transitionCondition1` and `transitionCondition2` are false.

`State3` is active in one of these two cases:

- The previous active state is `State1`, `transitionCondition1` is false, and `transitionCondition2` is true.
- The previous active state is also `State3`. `State3` is a terminating state.

You can import the generated Ladder Diagram code to CODESYS 3.5 and view the diagram. A portion of the Ladder Diagram is shown.



In the preceding Ladder Diagram, each rung of the ladder ends in a coil. The coil corresponds to a state of the original chart. The contacts before the coil determine if the coil receives power. You can compare the Ladder Diagram visually with the Stateflow chart. For instance, the coil `State1_new` receives power in one of these two cases:

- The normally open contact `State1` is closed and the normally closed contacts `transitionCondition1` and `transitionCondition2` are open.
- The normally open contact `stateflow_init` is closed. This contact corresponds to the default transition.

Once the coil `State1_new` receives power, the contact `State1_new` further down in the ladder is then closed and the coil `State1` receives power.

The Ladder Diagram executes from top to bottom and from left to right.

Ladder Diagram Generation Workflow

1. Before generating Ladder Diagram code from your Stateflow chart, confirm that your chart is ready for code generation.

   See "Prepare Chart for Ladder Diagram Generation" on page 3-6.

2. Generate Ladder Diagram code from the Stateflow chart. The code is in a format suitable for import to an IDE.

   Generate a test bench along with the code. The test bench is in the Structured Text language. You can later import the code along with the test bench to your IDE. The test bench invokes the Ladder Diagram code and compares the output against the expected outputs from the original Stateflow chart.

   See “Generate Ladder Diagram Code from Stateflow Chart” on page 3-10.
3. Import the generated Ladder Diagram code to your CODESYS 3.5 IDE. Validate the diagram in the IDE by using the generated test bench.

   See “Import Ladder Diagram Code to CODESYS 3.5 IDE and Validate Diagram” on page 3-15.

Prepare Chart for Ladder Diagram Generation

This example shows how to prepare your Stateflow chart for Ladder Diagram code generation. Once your chart is ready, you can generate Ladder Diagram code from the chart.

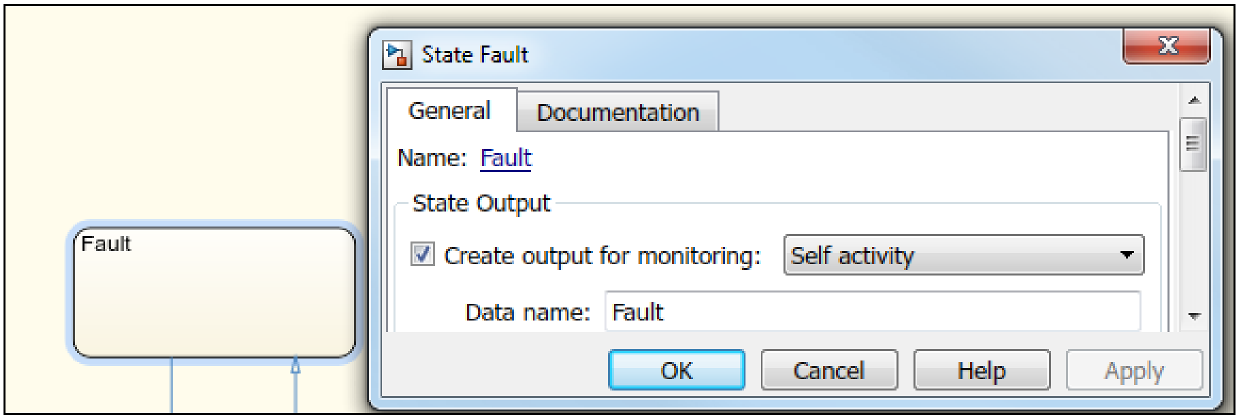
For the complete Ladder Diagram code generation workflow, see “Ladder Diagram Generation Workflow” on page 3-4.

Design PLC Application with Stateflow

Use Stateflow to design state machines that model PLC controllers. Your Stateflow chart must have these properties:

- The inputs and outputs to the chart must be Boolean. They correspond to the input and output terminals of your PLC.
- Each state in the chart must correspond to an output. The output is true if the state is active.

  To ensure that each state in the chart is mapped to an output, in the Properties dialog box of each state, select **Create output for monitoring**. Then, select `Self activity`.



Name: Fault

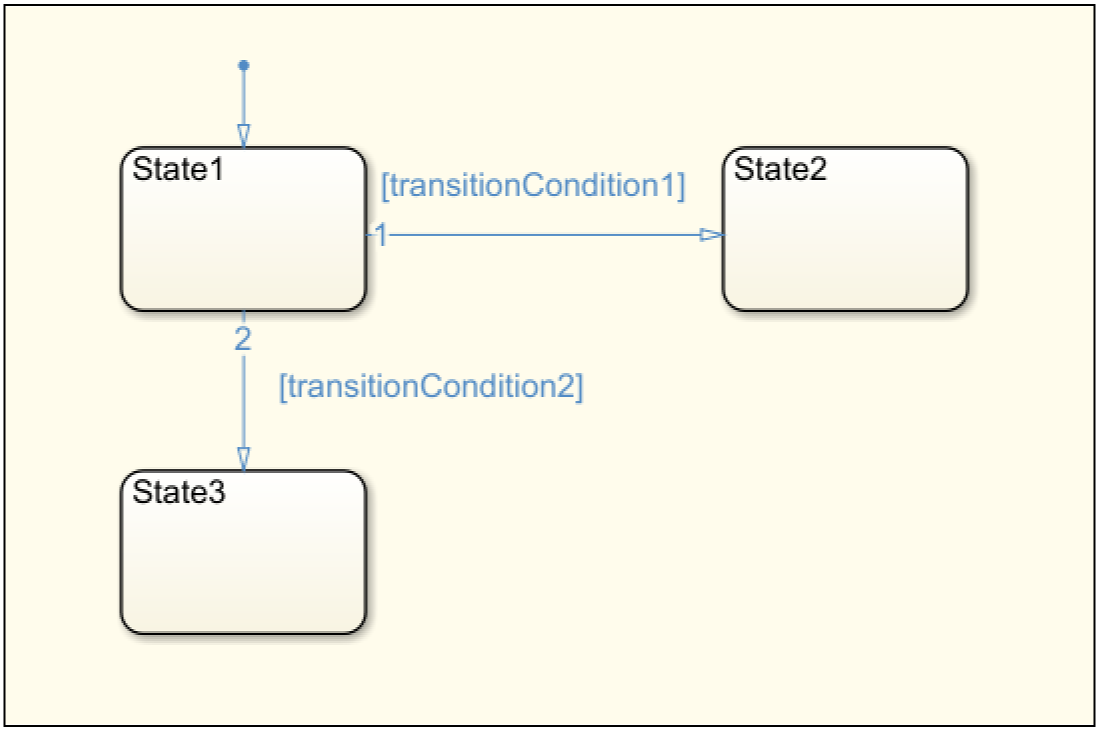
State Output

[✓] Create output for monitoring: Self activity

Data name: Fault

- The transition conditions must involve only Boolean operations such as ~, &, and | between the inputs.

For instance, in the following chart, `transitionCondition1`, and `transitionCondition2` are Boolean inputs to the model. `State1`, `State2`, and `State3` correspond to Boolean outputs from the model.



Some advanced Stateflow features on page 3-19 are not supported because of inherent restrictions in Ladder Diagram semantics. You can use the function `plccheckforladder` to check if the chart has the required properties. You can also use the function `plcprepareforladder` to change certain chart properties so that the chart is ready for Ladder Diagram code generation.

You can start generating Ladder Diagram code from the chart. See the example in “Generate Ladder Diagram Code from Stateflow Chart” on page 3-10.

Create Test Harness for Chart

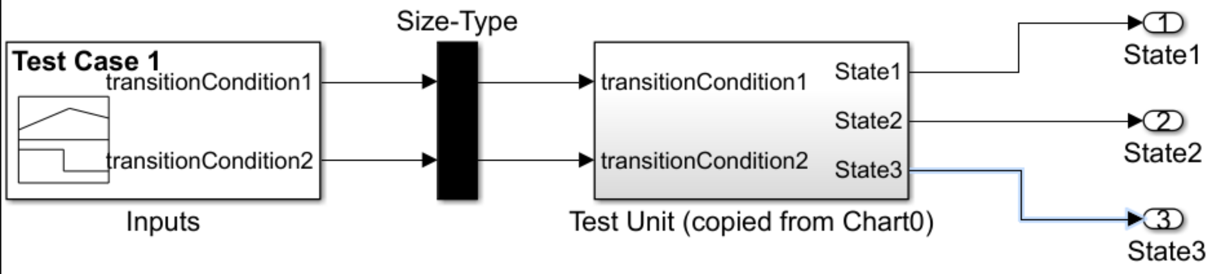
If you want to generate a test bench for validation of the Ladder Diagram code, create a test harness for the Stateflow chart. The test harness can consist of multiple test cases.

Using the test harness, Simulink PLC Coder can generate test benches for validation of the Ladder Diagram code.

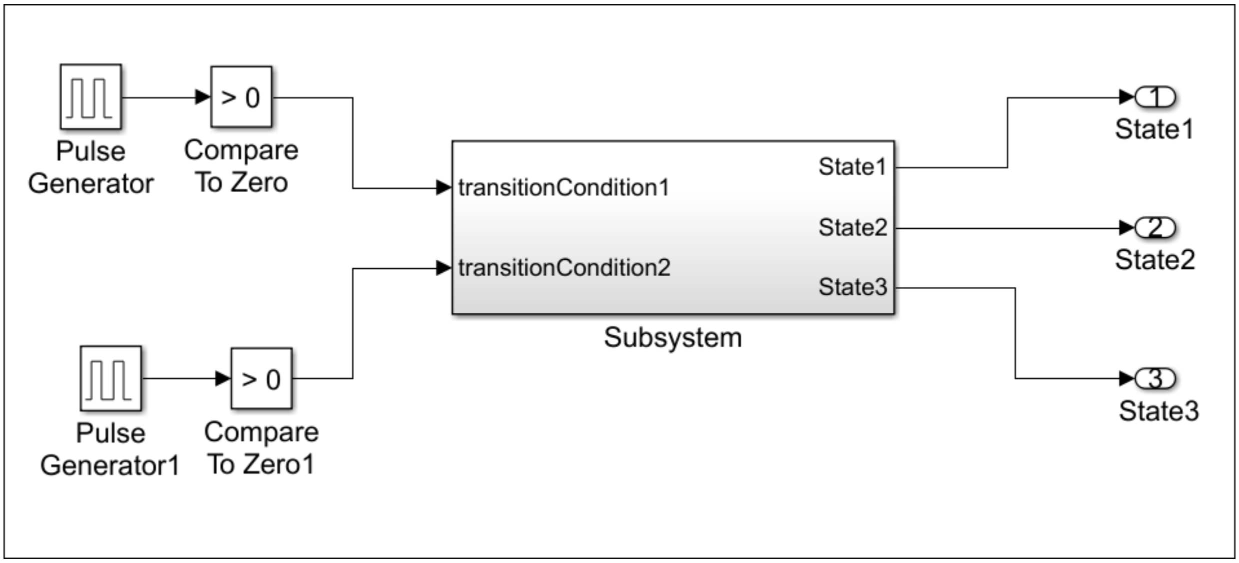
You can manually create a test harness by using the Signal Builder block or autogenerate a test harness by using Simulink Design Verifier™. To autogenerate the test harness:

1. Right-click the chart or a subsystem containing the chart. Select **Design Verifier** > **Generate Tests for Subsystem**.
2. After test creation, select **Create harness model**.

The harness model is created. The model consists of the original subsystem coupled with inputs from a Signal Builder block. The block consists of multiple test cases, so that the states and transitions in your model are covered at least once.



You can also create tests by using other blocks from the Simulink library. However, you must ensure that the inputs to the chart are Boolean.



You can now generate Ladder Diagram code from the chart and validate the diagram.

- To generate Ladder Diagram code only, use the original Stateflow chart.
- To generate Ladder Diagram code with test bench, use the Stateflow chart coupled with the Boolean inputs from the test cases. For instance, if you create a harness model with Simulink Design Verifier, use the harness model for the Ladder Diagram code and test bench generation instead of the original chart.

See "Generate Ladder Diagram Code from Stateflow Chart" on page 3-10.

Generate Ladder Diagram Code from Stateflow Chart

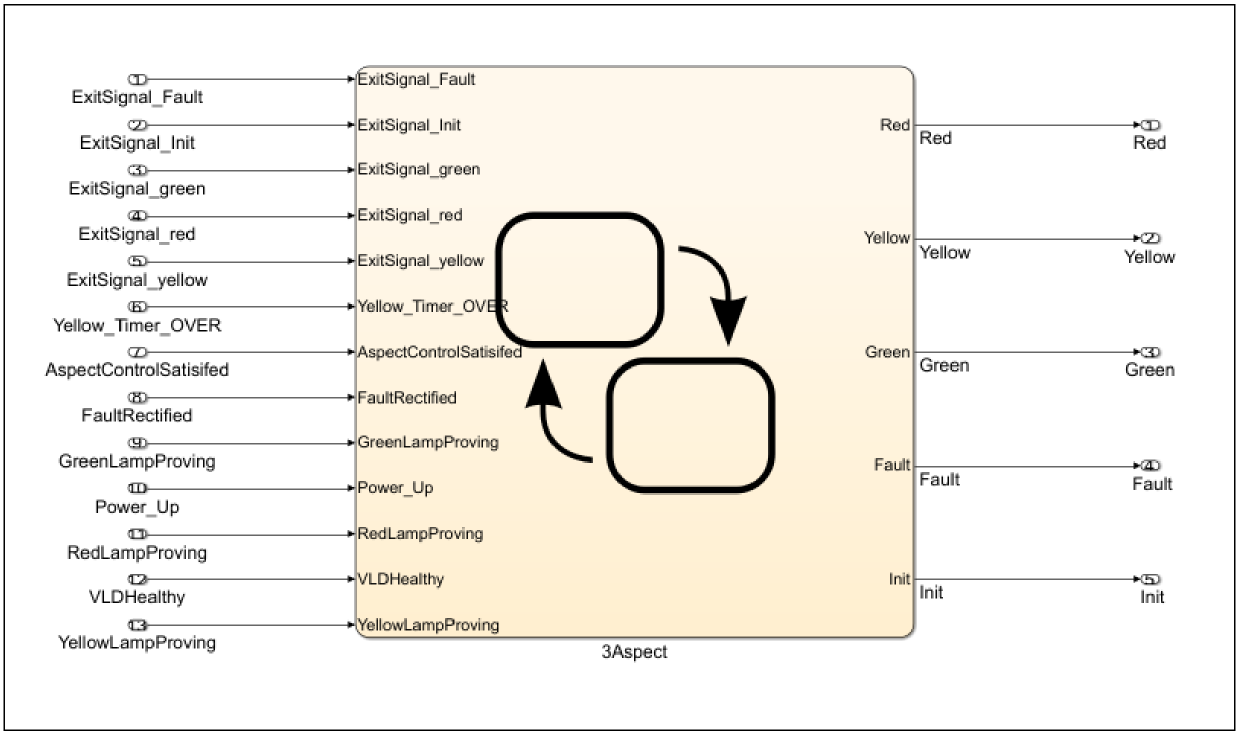
This example shows how to:

- Generate code from a Stateflow chart that you can view as Ladder Diagram in your IDE.
- Generate test bench for validation of the Ladder Diagram code in your IDE.

For the complete Ladder Diagram code generation workflow, see “Ladder Diagram Generation Workflow” on page 3-4.

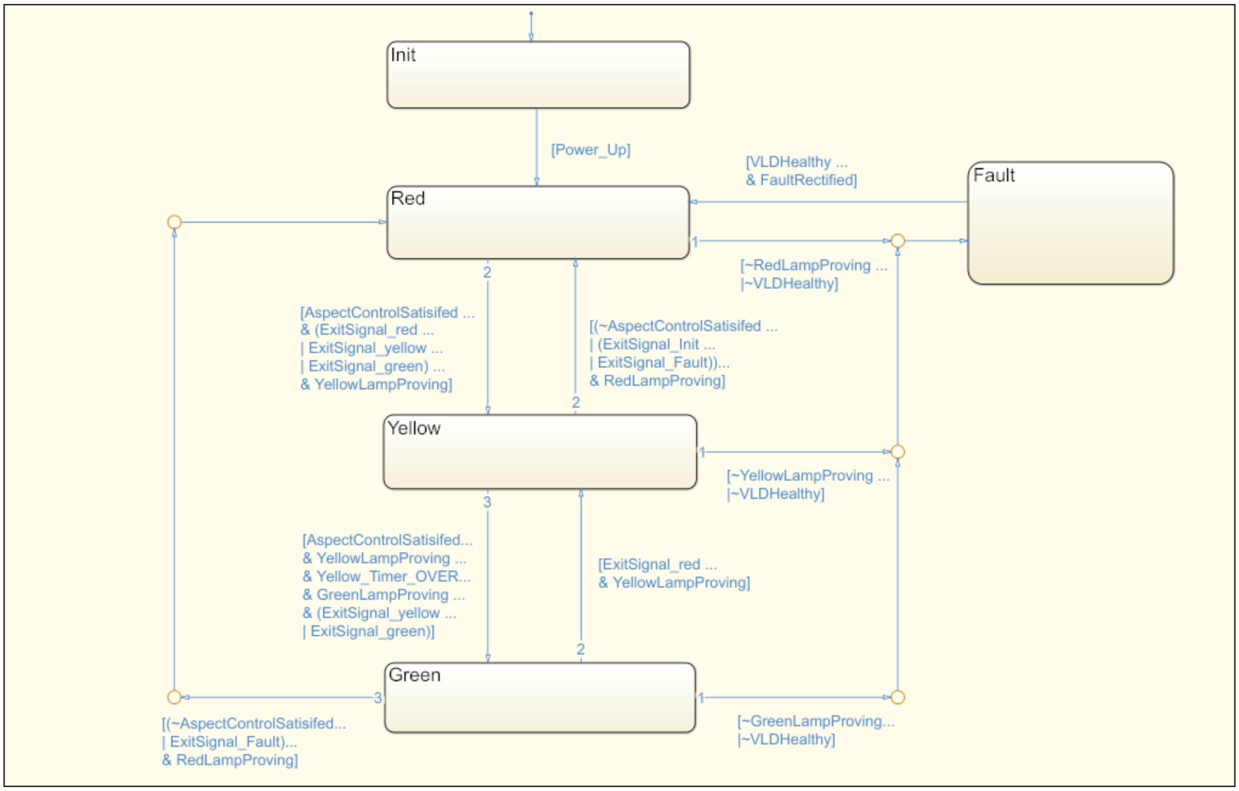
Stateflow Chart and Ladder Logic Diagram

The figure shows a Stateflow chart that implements three-aspect logic, a decision logic for many railway signaling applications.

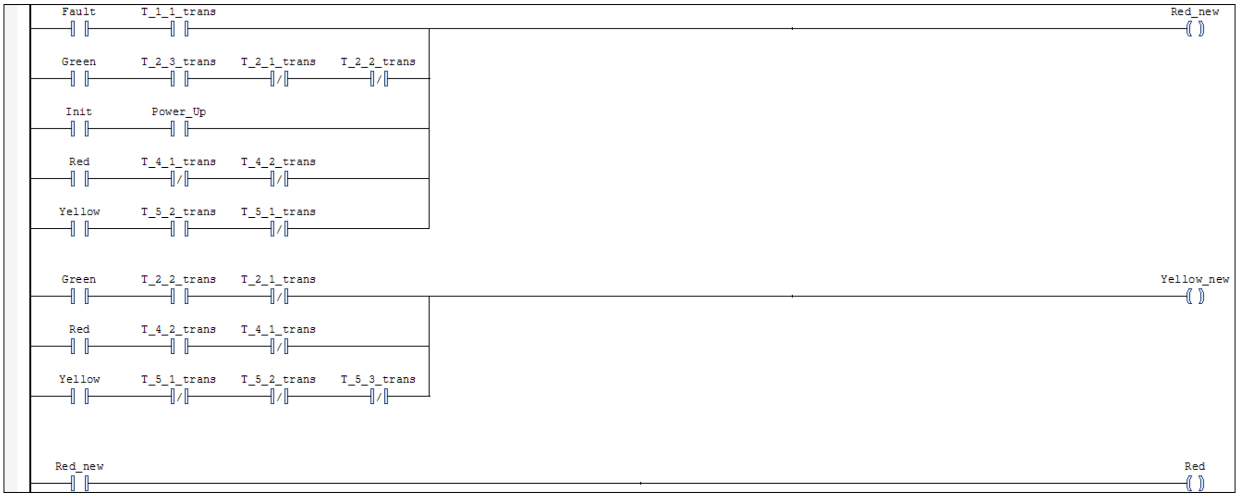


The chart consists of five states: `Init`, `Fault`, `Red`, `Yellow`, and `Green`. Based on the input to the chart, transitions to any of these states can take place. For instance, the state `Red` becomes active in the following scenarios:

- **Initialization and power up**: The previous state is `Init` and the condition `Power_Up` is true.
- **Fault rectification**: The previous state is `Fault` and the condition `VLDHealthy & FaultRectified` is true.
- **Transitions from other colors**: The previous state is `Green` or `Yellow`, the conditions that allow transition to `Red` are true, and the conditions that allow transition to another color or to the `Fault` state are false.
- **Staying red**: The previous state is `Red` and the conditions that allow transition to another state are false.



The figure shows a portion of the Ladder Diagram code generated from the chart when viewed in the CODESYS 3.5 IDE. The Ladder Diagram consists of contacts (normally open and normally closed) and coils (normal, set, and reset).



You can map elements of the original Stateflow chart to these coils and contacts. For instance, the coil `Red_new` corresponds to the update of the state `Red` in the Stateflow chart. For the coil to receive power, one of the following must be true:

- **Initialization and power up**: The normally open contacts `Init` and `Power_Up` must be closed.
- **Fault rectification**: The normally open contacts `Fault` and `T_1_1_trans` must be closed. The contact `T_1_1_trans` represents the transition condition `VLDHealthy & FaultRectified` in the chart.
- **Transitions from other colors**: The normally open contact `Green` must be closed and the following must be true:

  - The normally open contact `T_2_3_trans` must be closed. This contact corresponds to the chart condition that must be true for transition to the `Red` state.
  - The normally closed contacts `T_2_1_trans` and `T_2_2_trans` must stay closed. These contacts correspond to the chart condition that must be false for transition to the `Red` state. If the conditions are true, the contacts open and the coil no longer receives power.

- **Staying red**: The normally open contact `Red` must be closed, and the normally closed contacts `T_4_1_trans` and `T_4_2_trans` must stay closed. These contacts correspond to the chart conditions that must be false for the `Red` state to continue to be active. If the conditions are true, the contacts open and the coil no longer receives power.

Generate Ladder Diagram from Chart

To generate Ladder Diagram code from the model `plcdemo_ladder_three_aspect`:

1. Open the model.
2. Specify the target IDE for which to generate the Ladder Diagram code.

   Right-click the chart and select **PLC Code** > **Options**. Specify a supported IDE for the option “Target IDE” on page 13-3. See “IDEs Supported for Ladder Diagram Code Generation” on page 1-7.
3. Right-click the chart and select **PLC Code** > **Generate Ladder Logic for Chart**.

If code generation is successful, in the subfolder `plcsrc` of the current working folder, you see the file *`ModelName`*`.xml`. You import this file to your IDE and view the Ladder Diagram. For the CODESYS 3.5 IDE, see “Import Ladder Diagram Code to CODESYS 3.5 IDE and Validate Diagram” on page 3-15.

You can also use the function `plcgenerateladder` to generate Ladder Diagram code from a Stateflow chart.

Generate Ladder Diagram Along with Test Bench

You can generate a test bench to validate the generated Ladder Diagram code. You import the code together with the test bench in your IDE and validate the Ladder Diagram against the original Stateflow chart using the test bench. To generate test bench along with the Ladder Diagram code:

1. Right-click the chart and select **PLC Code** > **Options**. Select the option “Generate Testbench for Subsystem” on page 13-9.
2. Right-click the chart and select **PLC Code** > **Generate Ladder Logic for Chart**.

The test benches use the inputs to the original Stateflow chart. Therefore, you can create test harnesses for the original chart and reuse them for validation of the Ladder Diagram code.

You can also use the function `plcgenerateladder` to generate test benches.

After generating the Ladder Diagram code and the test benches, you can import them to your IDE. For the CODESYS 3.5 IDE, see “Import Ladder Diagram Code to CODESYS 3.5 IDE and Validate Diagram” on page 3-15.

Import Ladder Diagram Code to CODESYS 3.5 IDE and Validate Diagram

This example shows how to import generated Ladder Diagram code to an IDE and validate the generated code against the original Stateflow chart by using the generated test bench.

For this example, the CODESYS 3.5 IDE is used. You can also use one of the other supported IDE. See “IDEs Supported for Ladder Diagram Code Generation” on page 1-7.

For the complete Ladder Diagram code generation workflow, see “Ladder Diagram Generation Workflow” on page 3-4.

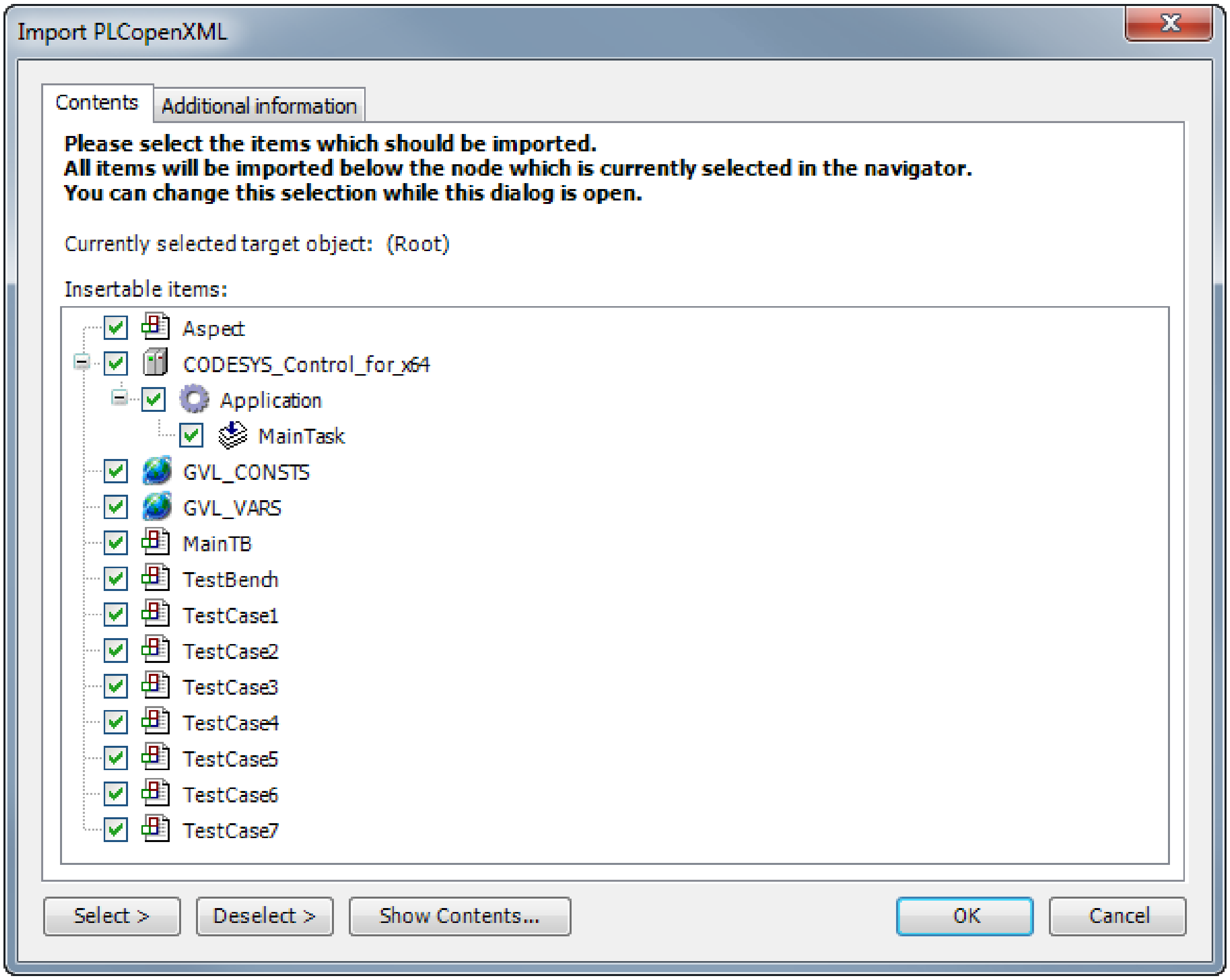
Import Ladder Diagram XML

After code generation, you see the Ladder Diagram code XML file `*ModelName*.xml` in a subfolder `plcsrc` of the current working folder. To import the generated XML and view the Ladder Diagram in the CODESYS 3.5 IDE:

1. Create an empty project.
2. Import the Ladder Diagram code to the project.

   Select **Project** > **Import PLCOpenXML** and navigate to the folder containing the XML file.

   A dialog box opens with a full list of the components imported from the XML. If you generate a test bench for validation, you also see the testbench.



Please select the items which should be imported.
All items will be imported below the node which is currently selected in the navigator.
You can change this selection while this dialog is open.

Currently selected target object: (Root)

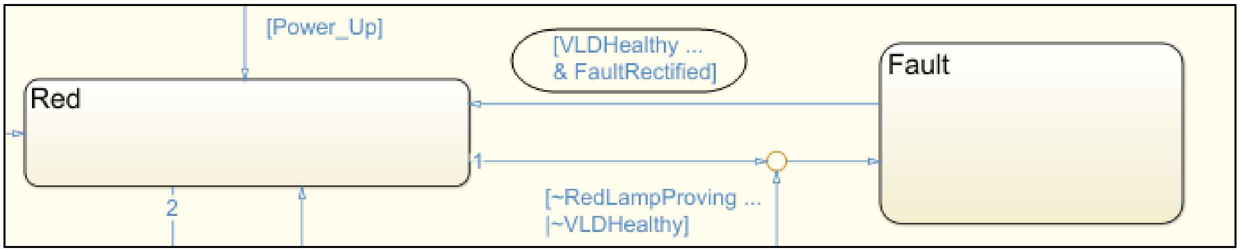
Insertable items:

- ✓ Aspect
- ✓ CODESYS_Control_for_x64
  - ✓ Application
    - ✓ MainTask
- ✓ GVL_CONSTS
- ✓ GVL_VARS
- ✓ MainTB
- ✓ TestBench
- ✓ TestCase1
- ✓ TestCase2
- ✓ TestCase3
- ✓ TestCase4
- ✓ TestCase5
- ✓ TestCase6
- ✓ TestCase7

3 On the **POUs** pane, you see the project. View the Ladder Diagram in the project.

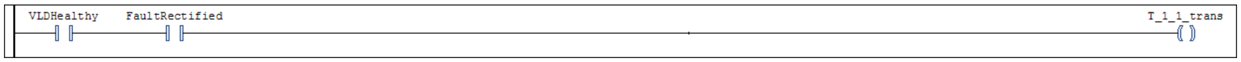
You can compare the Ladder Diagram with the original Stateflow chart.

For instance, if you generate Ladder Diagram code from the model `plcdemo_ladder_three_aspect`, in the Ladder Diagram, you can identify the transition from the `Fault` state to the `Red` state.

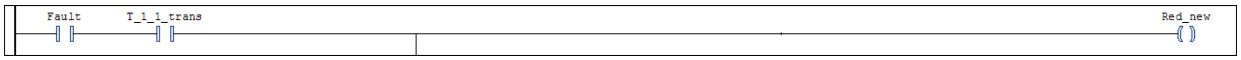


The transition appears in the Ladder Diagram in three steps:

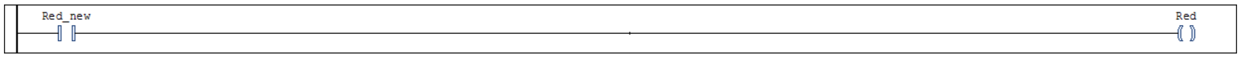
a The normally open contacts `VLDHealthy` and `FaultRectified` are closed. Coil `T_1_1_trans` receives power and is energized.



b The normally open contacts `Fault` and `T_1_1_trans` are closed. The coil `Red_new` receives power and is energized. Other conditions not shown in figure must also be satisfied.



c The normally open contact `Red_new` is closed. The coil `Red` receives power and is energized.



Besides coils and normally open contacts , the Ladder Diagram also uses:

- Normally closed contacts : They appear if the ~ operator is used in a transition condition.
- Set coils and reset coils : They are used in the Ladder Diagram for chart initialization. Reset coils are also used if you enforce additional safeguards against multiple states from being simultaneously active. See the argument `InsertGuardResets` in `plcgenerateladder`.

For more information about these symbols, refer to the IEC 61131-3 specifications.

4 Select **Online** > **Simulation**. Click the (Build) button and verify that there are no build errors.

   If the option is not active, you might have to change the version number in your XML. Search for the version number in the XML and depending on the patch that you have, replace it with the following version number:

   - CODESYS V3.5 SP6 Patch1: 3.5.4.30
   - CODESYS V3.5 SP6 Patch3: 3.5.6.30
   - CODESYS V3.5 SP8 Patch2: 3.5.8.20
   - CODESYS V3.5 SP8 Patch4: 3.5.8.40

Verify Ladder Diagram with Test Bench

In your project, you see the generated test bench. To simulate using the test bench and validate your generated code:

1 Click the (Login) button and log in to the emulator device.

2 Click the (Start) button and begin simulation.

3 Double-click a test bench in your project. You see the following variables updating to reflect the results of validation.

   - The variable `testCycleNum` increases from 0 to the number of cycles.
   - The variable `testVerify` remains `TRUE` as long as the test bench verification succeeds.

Restrictions on Stateflow Chart for Ladder Diagram Generation

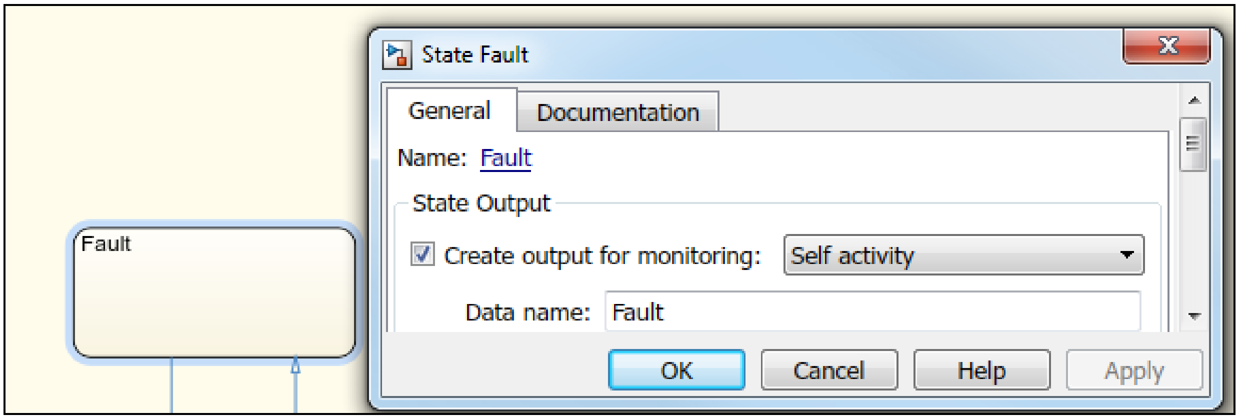
Ladder Diagram semantics must be represented with switches and relays. Therefore, if you intend to generate a Ladder Diagram from a Stateflow chart, you cannot use some advanced features in your chart. The Stateflow chart must have the following form:

- The inputs and outputs to the chart must be Boolean. These inputs and outputs correspond to the input and output terminals of your PLC.
- Each state of the chart must correspond to a chart output.
- The expressions controlling the transition between states must involve only Boolean operations between the inputs.

In addition, the chart must have the following properties. You can use the function `plccheckforladder` to check if the chart has the required properties. You can also use the function `plcprepareforladder` to change certain chart properties so that the chart is ready for Ladder Diagram code generation.

- The chart **Action Language** must be `C`.
- You must disable the following chart properties:
  - **Enable Super Step Semantics**
  - **Execute (enter) Chart At Initialization**
  - **Initialize Outputs Every Time Chart Wakes Up**
- The chart must have at least one input and output. The input and output data must be Boolean.
- Each output must correspond to a state in the chart. The output is true if the state is active.

  To ensure that each state in the chart is mapped to an output, in the Properties dialog box of each state, select **Create output for monitoring**. Then, select `Self activity`.



Name: Fault

State Output

☑ Create output for monitoring: Self activity

Data name: Fault

- The chart must not have data with scope other than input or output.
- The chart must not include:
  - Atomic subcharts
  - Multiple default transition
  - Simulink functions
  - Parallel states
  - State hierarchy
  - History junctions
  - Dangling transitions
  - Super transitions crossing subchart boundaries
  - Conditional default paths
  - Self transitions

See Also

Related Examples

- “Prepare Chart for Ladder Diagram Generation” on page 3-6
- “Generate Ladder Diagram Code from Stateflow Chart” on page 3-10
- “Import Ladder Diagram Code to CODESYS 3.5 IDE and Validate Diagram” on page 3-15

More About

- “Ladder Diagram Generation for PLC Controllers” on page 3-2

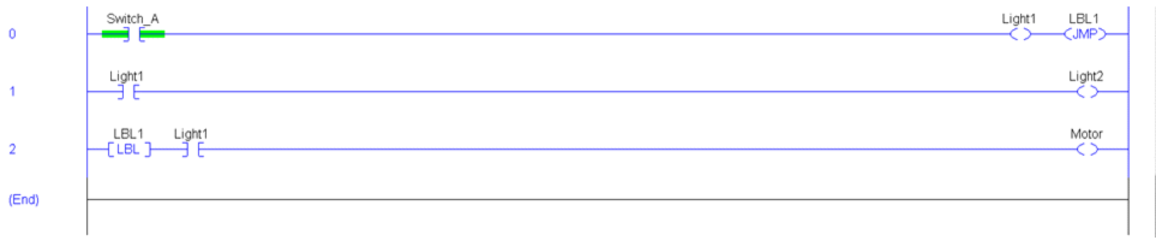
Import Ladder Diagram into Simulink

Ladder Diagram is a graphical programming language used to develop software for programmable logic controllers (PLCs). It is one of the languages that the IEC 61131 Standard specifies for use with PLCs. A program in Ladder Diagram notation is a circuit diagram that emulates circuits of relay logic hardware. The underlying program uses Boolean expressions that translate readily to switches and relays.

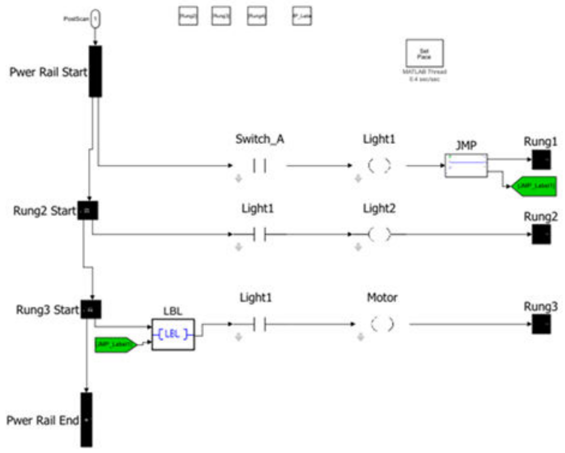
The ladder import feature of Simulink PLC Coder allows you to import Ladder Diagram created with Rockwell Automation IDEs such as RSLogix 5000 and Studio 5000 into the Simulink environment as a model. After importing, you can edit, simulate, and generate code for the Ladder Diagram models from within the Simulink environment.

- After importing the Ladder Diagram into Simulink, you can simulate the Ladder Diagram within Simulink.
- Generating C code from the imported Ladder Diagram allows you to integrate the code into your existing C language-based simulation environments.

The following is an example of a simple Ladder Diagram with the `JMP` element and its equivalent model in Simulink after ladder import.



The Simulink subsystem containing the ladder implementation.



Supported Features

Ladder import supports the following features:

- Import single RSLogix AOI or program.
- Boolean variables
- Support for basic ladder functions like XIC, XIO, and OTE.
- Data access to array elements, bus elements, bit, and constant variables.
- Multiple rungs
- Simple Jumps, returns, and other supported execution control elements.
- C code generation from the imported model.

Supported Ladder Elements

Simulink PLC Coder supports the following ladder elements:

XIC	XIO	OTE	OTL	OTU	JMP
LBL	ADD	AFI	AND	CLR	EQU
GEQ	GRT	LEQ	LES	MOV	MUL
NEQ	ONS	SUB	FRD	CTU	RTO

RES	FLL	UDT	Timer: TON	Timer: TOFF	Counter
Arrays	Tags	Alias Tags	Bit Access	COP	

See Also

More About

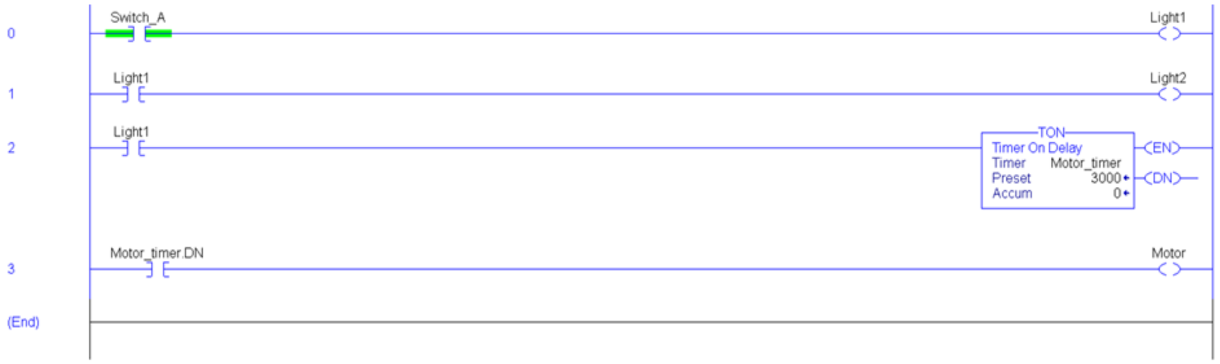
- “Import L5X Ladder Files into Simulink” on page 3-25

Import L5X Ladder Files into Simulink

This example shows how to import a Ladder Diagram from an L5X file created using Rockwell Automation IDEs such as RSLogix 5000 and Studio 5000 into the Simulink environment. The import operation is performed using the `plcimportladder` function.

Description of the Ladder Diagram

The figure shows a Ladder Diagram with a simple timer. The Ladder Diagram consists of four rungs with contacts (`Switch_A`, `Light1`, `Motor_timer.DN`), coils (`Light1`, `Light2`, `EN`, `DN`, `Motor`), and `TON` timer function.



The `simple_timer.L5X` file was created using the RSLogix 5000 IDE. A snippet of the L5X file is shown.

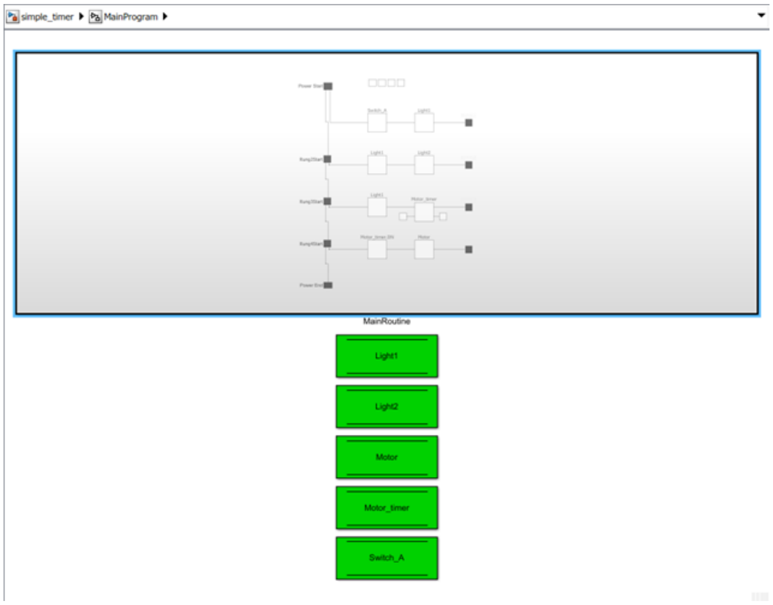
```
<RSLogix5000Content SchemaRevision="1.0" SoftwareRevision="24.01" TargetName="MainProgram"
TargetType="Program" ContainsContext="true" Owner="MathWorks User, MathWorks, Inc."
ExportDate="Wed Aug 30 14:39:26 2017" ExportOptions="References DecoratedData Context
Dependencies ForceProtectedEncoding AllProjDocTrans">
<Controller Use="Context" Name="timer_and_counter">
<DataTypes Use="Context">
</DataTypes>
<Programs Use="Context">
<Program Use="Target" Name="MainProgram" TestEdits="false" MainRoutineName="MainRoutine"
Disabled="false" UseAsFolder="false">
...
<Tags>
```

Import Ladder Diagram

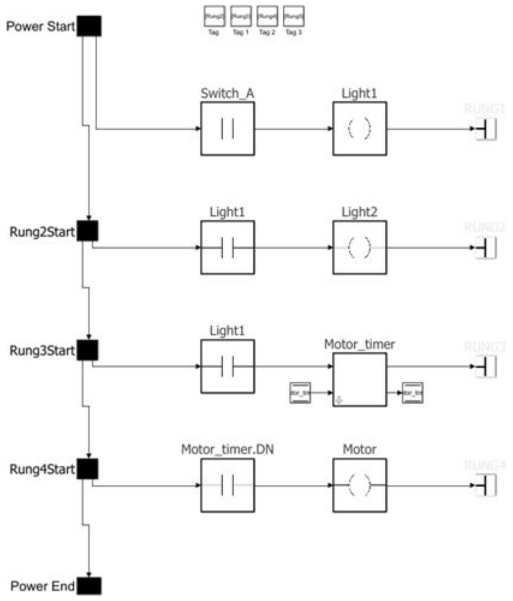
Use the `plcladderimport` function to import the ladder into Simulink. For this example, the program `Name` of the ladder is `MainProgram` and the `MainRoutineName` is `MainRoutine`.

```
>> plcimportladder('simple_timer.L5X','MainProgram','MainRoutine')
```

The Ladder Diagram is imported into the `plcout\simple_timer.mdl` Simulink model. The state information of the ladder elements is stored in the data store memory and updated by the model during simulation. The `plcout\simple_timer_initialValues.m` file gets called during the pre-load stage of the Simulink model. This file sets the timer initial values in `Motor_timer` data store memory.

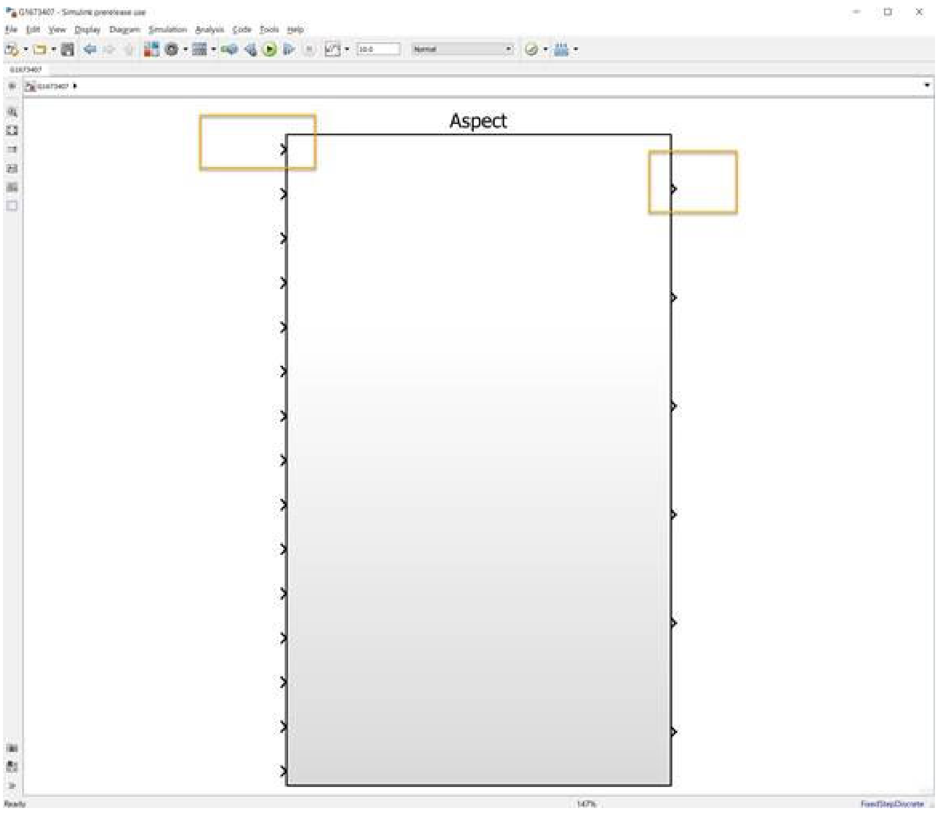


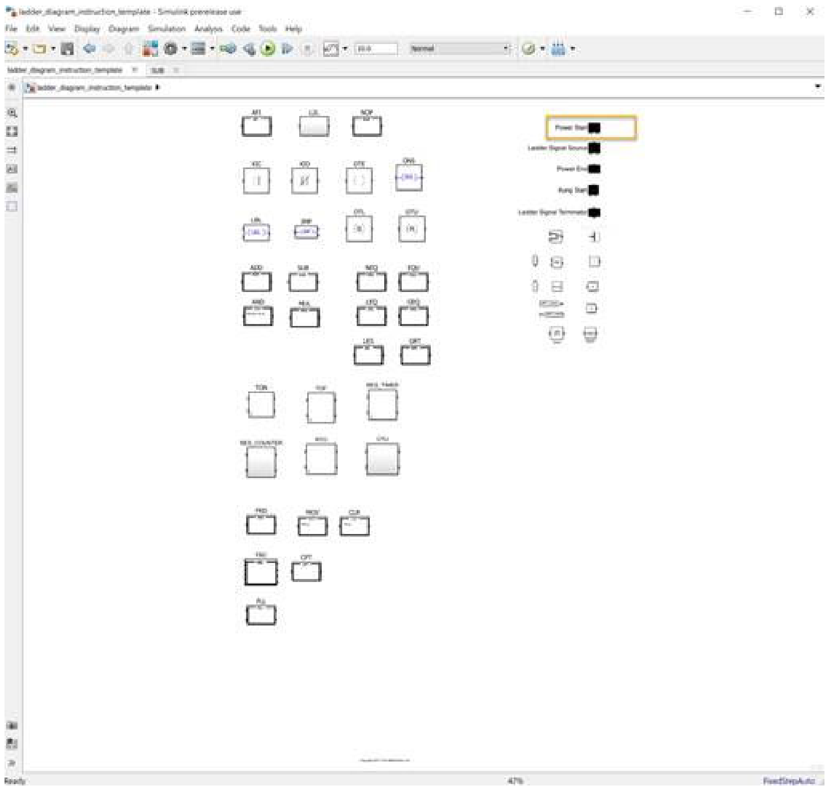
The `MainRoutine` subsystem contains the Simulink implementation of the `simple_timer.L5X` Ladder Diagram. The ladder rung executes from top to bottom and left to right.

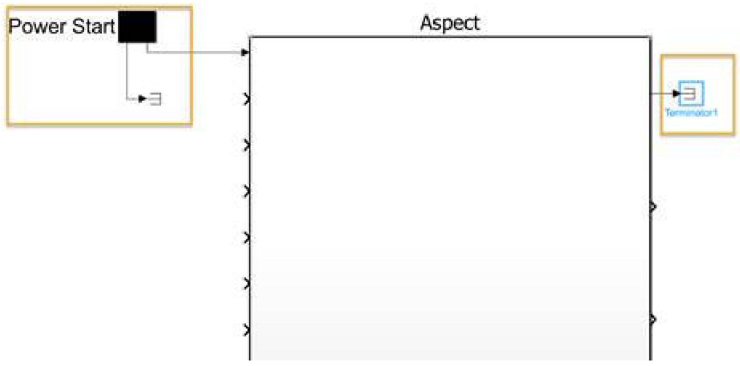


You can use the Signal Builder block to generate test inputs for `Switch_A` and verify the operation of the imported ladder. You can also generate C code for the top-level subsystem. If you want to edit the imported ladder, the Simulink blocks are in the template `ladder_diagram_instruction_template`.

To perform simulation on imported AOI, you must perform additional steps. This is because the AOI structure that is imported is not connected to any input, so you must add the `Power Start`, and `Terminator` blocks as shown.


Aspect




Power Start
Aspect
Terminator1

See Also

`plcgeneratecode` | `plcimportladder`

More About

- “Import Ladder Diagram into Simulink” on page 3-22

Generating Test Bench Code

How Test Bench Verification Works

The Simulink PLC Coder software simulates your model and automatically captures the input and output signals for the subsystem that contains your algorithm. This set of input and output signal data is the test bench data. The coder also automatically generates a test bench, or test harness, using the test bench data.

The test bench runs the generated code to verify that the output is functionally and numerically equivalent to the output from the execution of a Simulink model. The following table shows how the test bench compares the expected and actual data 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 the following logic:

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

To verify the generated code using the test bench, import the generated Structured Text and the test bench data into your target IDE. You can import test bench code:

- Manually.
- Automatically, including running the test bench.

For more information, see “Import and Verify Structured Text Code” on page 4-5.

Depending on the target IDE platform, the Simulink PLC Coder software generates code into one or more files. See “Files Generated with Simulink PLC Coder” on page 1-23 for list of the target IDE platforms and the possible generated files.

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.
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.

Has Internal State	ssMethodType Contains...
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

After you generate code and test benches for your subsystem, you can import them to your target IDE. Using the test bench data, you can verify that the results from your generated code match your simulation results.

If you want to import the generated code, see “Generate and Automatically Import Structured Text Code” on page 1-27.

Generate, Import, and Verify Structured Text

If you are working with the PHOENIX CONTACT (previously KW) Software MULTIPROG 5.0/5.50 or Phoenix Contact PC WORX 6.0 IDE, see “Import and Verify Structured Text to PHOENIX CONTACT (previously KW) Software MULTIPROG 5.0 and Phoenix Contact PC WORX 6.0 IDEs” on page 4-6.

Otherwise, to generate, import, and verify Structured Text code:

1. Specify that test bench code must be generated for the subsystem.

   a. Right-click your subsystem and select **PLC Code** > **Options**.
   b. Select “Generate Testbench for Subsystem” on page 13-9.

   If you do not specify that test bench code must be generated, when you automatically verify the generated code, you see the error `Testbench not selected`.
2. You can generate the code and testbench, and manually import them to your target IDE. For information on how to import generated code, see the user manual for your target IDE.

   Alternatively, after code generation, import and verify the generated code automatically. Right-click the subsystem and select **PLC Code** > **Generate, Import, and Verify Code for Subsystem**. The software:

   a. Generates the code and test bench.
   b. Starts the target IDE.
   c. Creates a project.
   d. Imports the generated code and test bench to the new project in the target IDE.
   e. Runs the generated code on the target IDE to verify it.

For information on:

- IDEs not supported for automatic import and verification, see “Troubleshoot Automatic Import Issues” on page 1-28.
- Possible reasons for long testbench code generation time, see “Troubleshooting: Long Test Bench Code Generation Time” on page 4-7.

Import and Verify Structured Text to PHOENIX CONTACT (previously KW) Software MULTIPROG 5.0 and Phoenix Contact PC WORX 6.0 IDEs

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

1. Start the PHOENIX CONTACT (previously KW) Software MULTIPROG 5.0/5.50 or Phoenix Contact PC WORX 6.0 IDE.
2. Select **File** > **Delete Template**. Delete any template named `Empty`, and click **OK** when done.
3. Select **File** > **New Project**, select `Project Wizard`, then click **OK**.

   a. In the **Project Name** field, type `Empty`,

   b. In the **Project Path** field, type or select a path to which you have write privileges.

   c. Click **Next**.

   d. In the remaining wizard pages, click **Next** to leave the default selections. At the end of the wizard, click **Finish**.

   The IDE is updated with the new `Empty` project tree.
4. In the project, delete everything under the following nodes:

   - Logical POUs
   - Physical Hardware
5. Verify that the project tree has only top-level nodes for `Libraries`, `Data Types`, `Logical POUs`, and `Physical Hardware`. There must not be any subtree nodes.
6. In the IDE, select **File** > **Save As Template**.

7. In **Template Name**, type `Empty`.
8. Click **OK**.
9. Close the IDE interface.

Open your model, right-click the Subsystem block, and select one of the following:

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

If you automatically generate, import, and verify code, 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.

Troubleshooting: Long Test Bench Code Generation Time

If code generation with test bench takes too long, one possible reason is that the test bench data size exceeds the limit that Simulink PLC Coder can handle. The test bench data size is directly related to the number of times the input signal is sampled during simulation. For large simulation time or more frequent sampling, the test bench data can be large.

To reduce test bench generation time, do one of the following:

- 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 with duration [0, `t`], divide the input into multiple parts with durations [0, t_1], [t_1, t_2], [t_2, t_3], etc., 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

- “Verify Generated Code with Multiple Test Benches” on page 4-9

Verify Generated Code with Multiple Test Benches

You can generate code with 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.

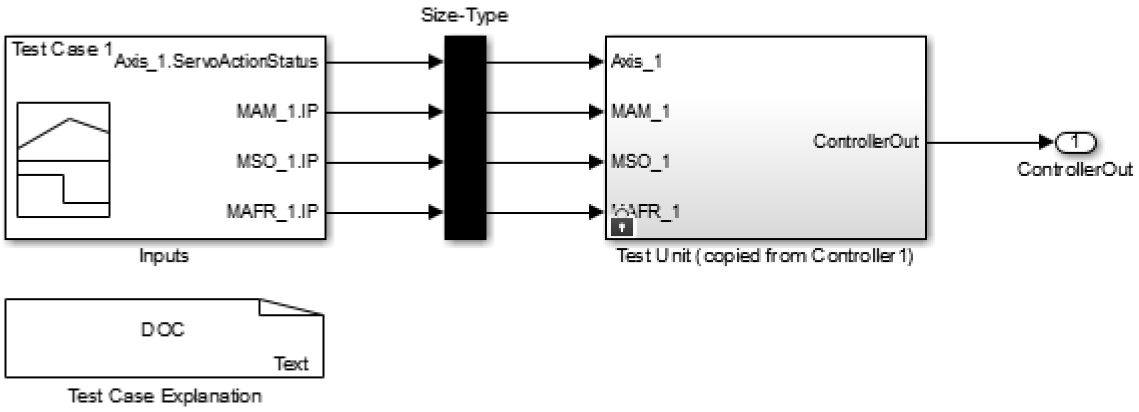
To generate multiple test benches for your subsystem:

1. Provide multiple signal groups as inputs by using a Signal Builder block with multiple signal groups (Simulink).

   Instead of manually entering a Signal Builder block and creating multiple signal groups, you can use Simulink Design Verifier to create a test harness model from the subsystem. In the test harness model, a Signal Builder block with one or more signal groups provides input to the subsystem. You can use this Signal Builder block to provide inputs to your subsystem. However, if your model is complex, Simulink Design Verifier can create large number of signal groups. See “Troubleshooting: Test Data Exceeds Target Data Size” on page 4-11.

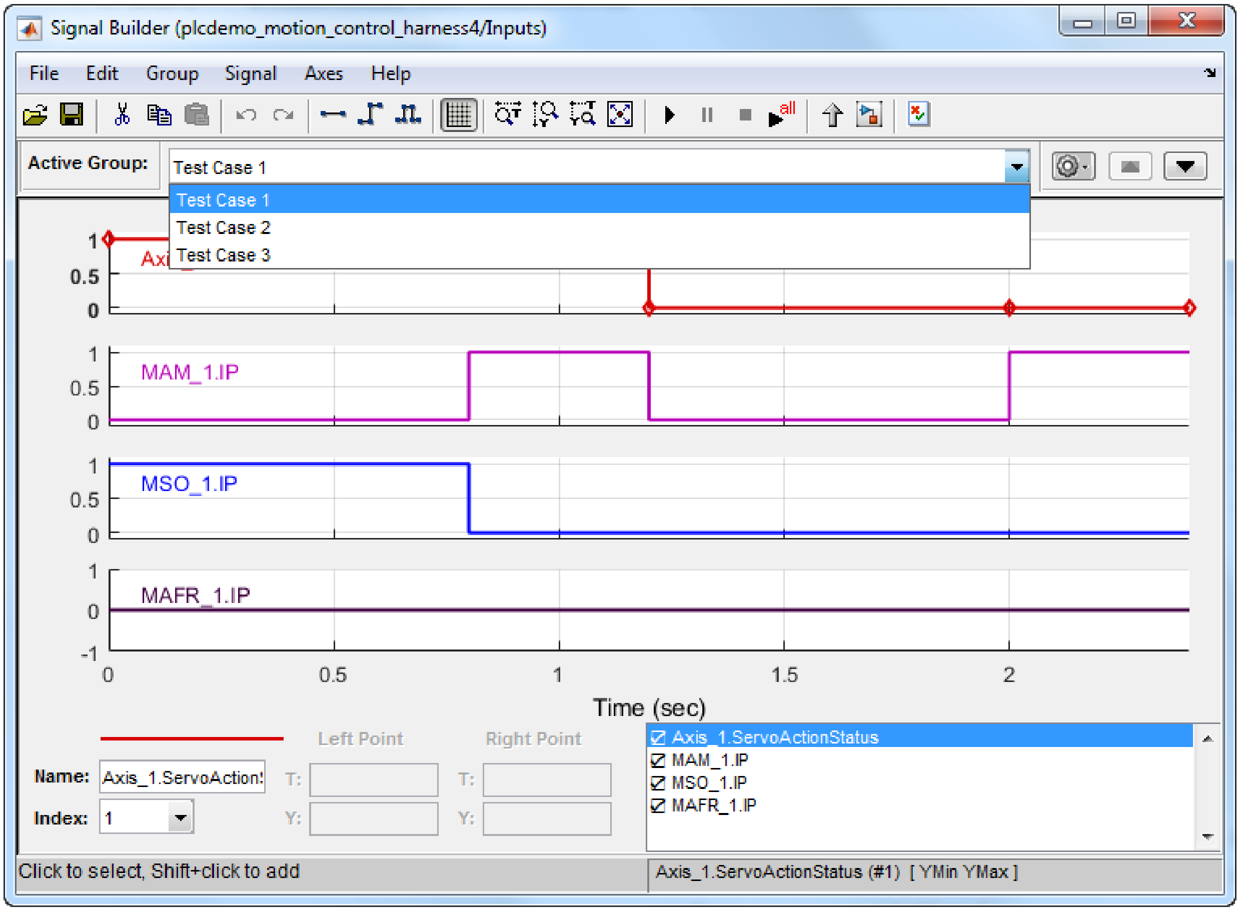
   To create your Signal Builder block with Simulink Design Verifier:

   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.


Signal Builder (plcdemo_motion_control_harness4/Inputs)
File Edit Group Signal Axes Help
Active Group: Test Case 1
Test Case 1
Test Case 2
Test Case 3
MAM_1.IP
MSO_1.IP
MAFR_1.IP
Time (sec)
Left Point
Right Point
Name: Axis_1.ServoAction
Index: 1
T:
Y:
Axis_1.ServoActionStatus
MAM_1.IP
MSO_1.IP
MAFR_1.IP
Click to select, Shift+click to add
Axis_1.ServoActionStatus (#1) [YMin YMax]

d Copy the Signal Builder block from the test harness model and use this block to provide inputs to your original subsystem.

2 Specify that test benches must be generated for the subsystem.

a Right-click your subsystem and select **PLC Code > Options**.

    b Select “Generate Testbench for Subsystem” on page 13-9.

3. Right-click the subsystem and select **PLC Code > Generate, Import and Verify Code for Subsystem**.

   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.

See Also

Related Examples

- “Import and Verify Structured Text Code” on page 4-5

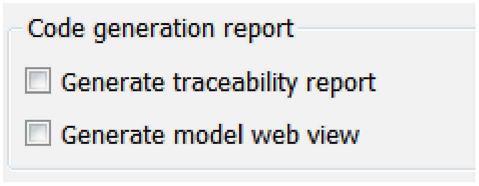
Code Generation Reports

Information in Code Generation Reports

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

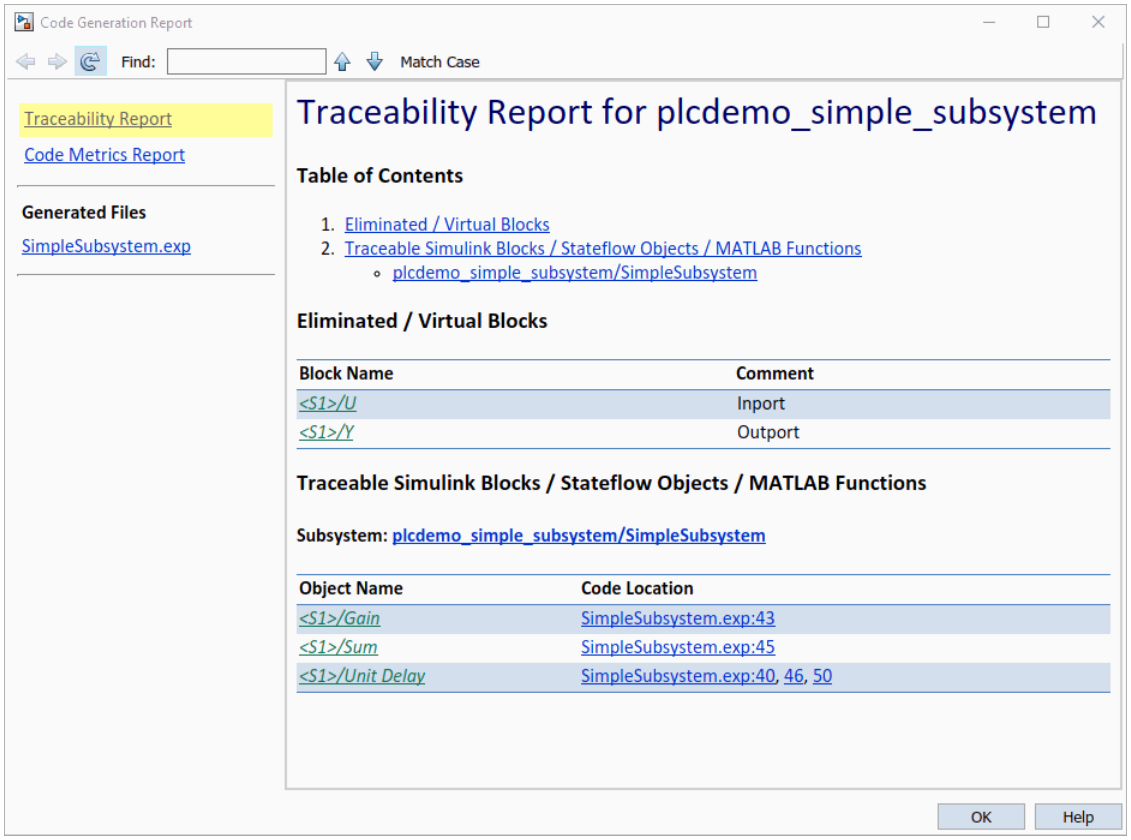
GUI Option	Command-Line Property	Description
Generate traceability report	`PLC_GenerateReport`	Specify whether to create code generation report.
Generate model web view	`PLC_GenerateWebview`	Include the model web view in the code generation report to navigate between the code and model within the same 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 11-4. A typical traceability report looks something like this figure:



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

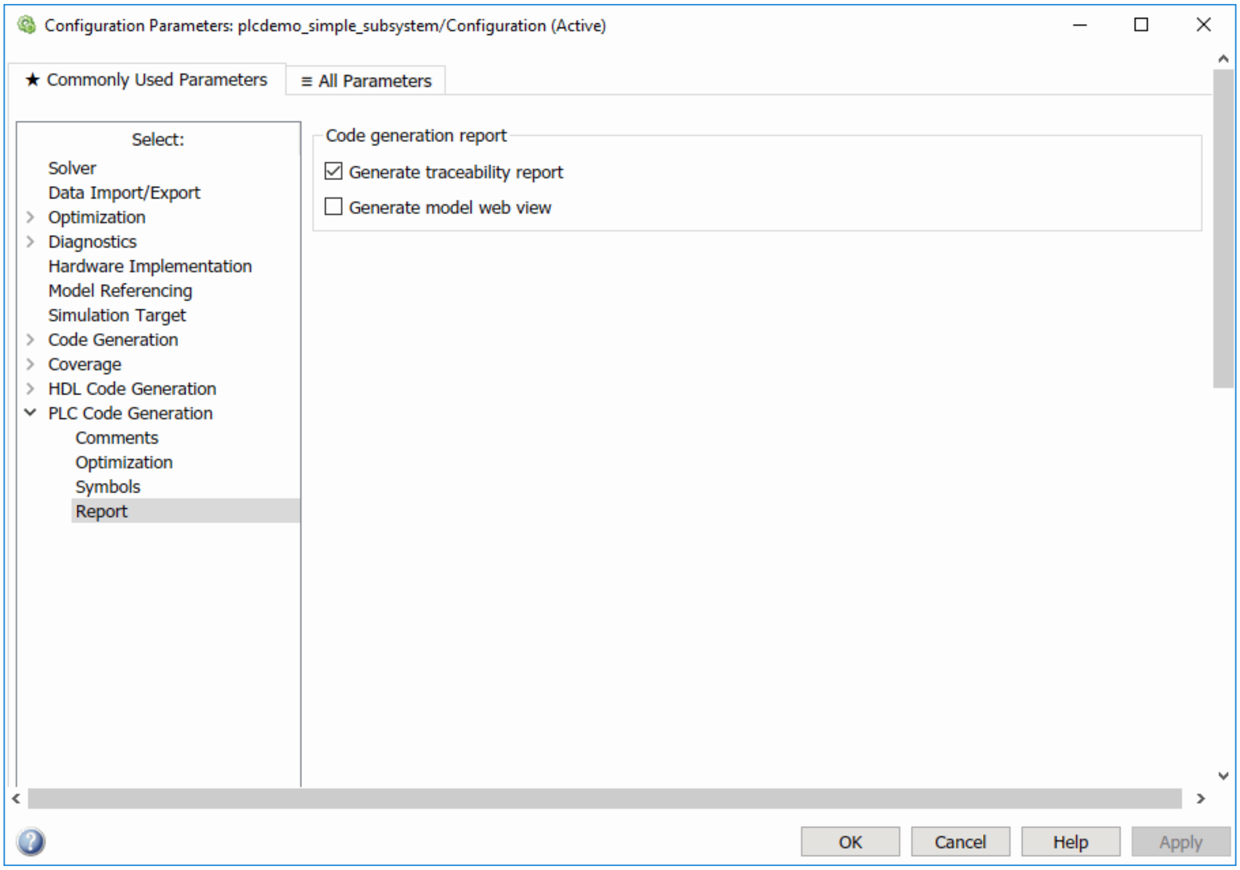
Create and Use Code Generation Reports

In this section...

Generate a Traceability Report from Configuration Parameters

To generate a Simulink PLC Coder code generation report from the Configuration Parameters dialog box:

1. Verify that the model is open.
2. Open the Configuration Parameters dialog box and navigate to the **PLC Code Generation** pane.
3. To enable report generation, select **Report > Generate traceability report**.
4. Click **Apply**.

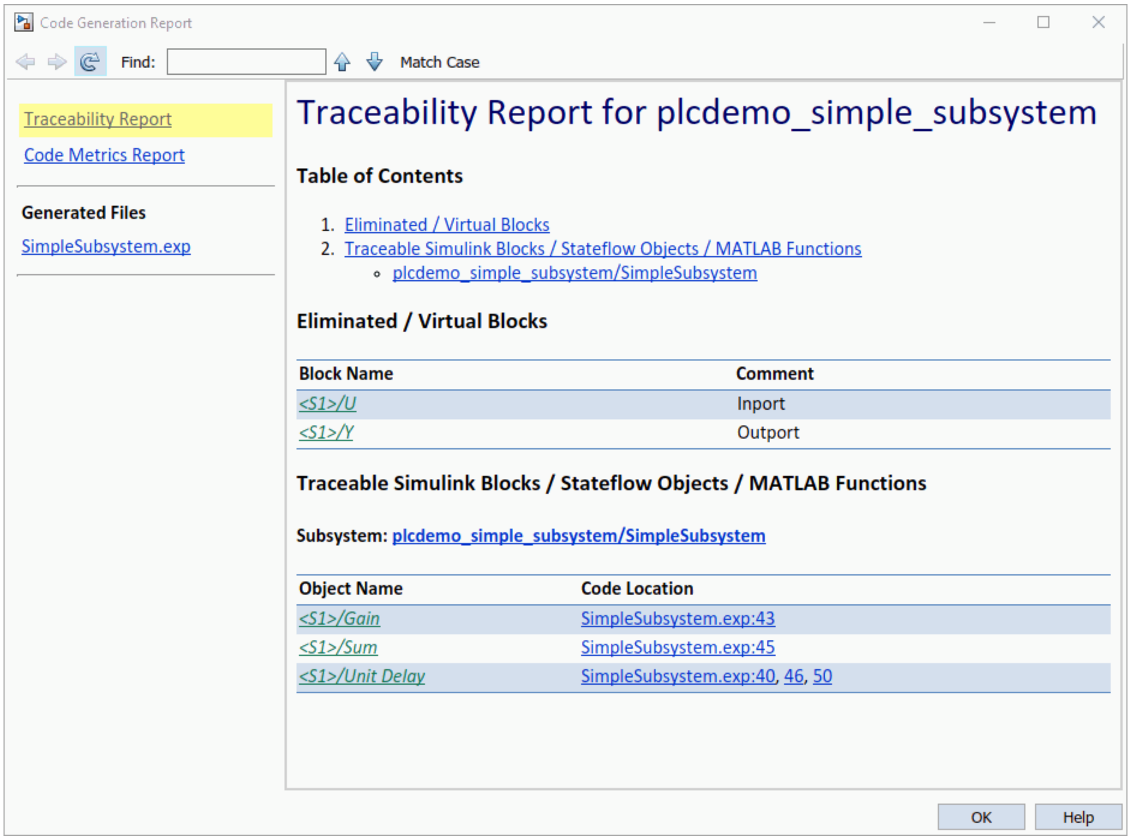


Code generation report

- [x] Generate traceability report
- [ ] Generate model web view

5 Click **PLC Code Generation > Generate code** to initiate code and report generation. The coder generates HTML report files as part of the code generation process.

The HTML report appears:



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

For more information, see:

- “Trace from Code to Model” on page 5-7
- “Trace from Model to Code” on page 5-8

Keep the Report Current

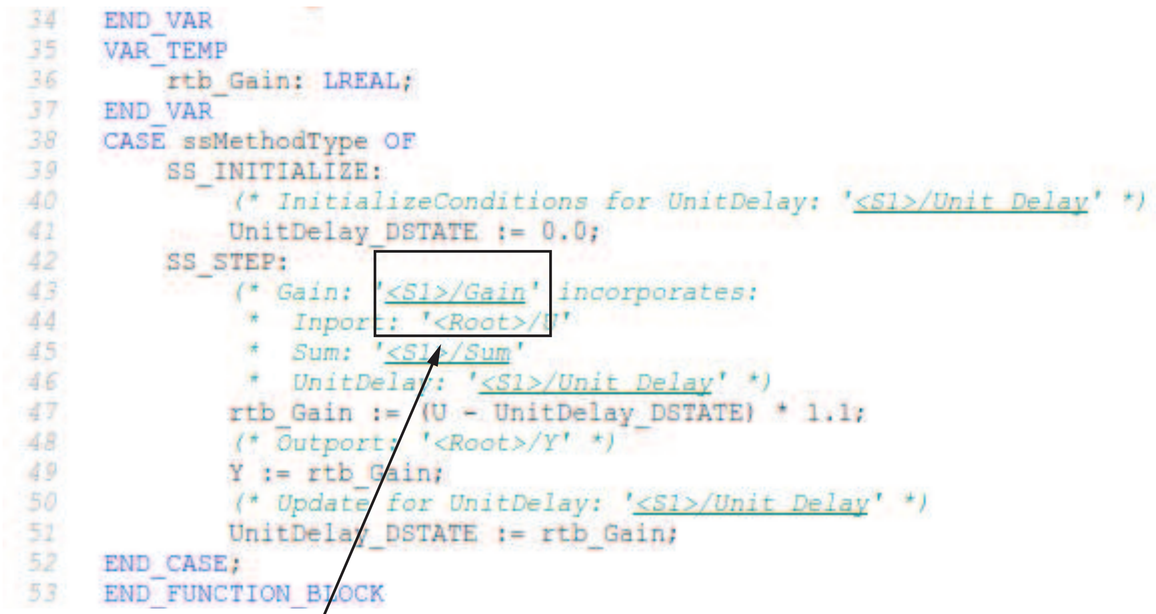
If you generate a code generation report for a model, and then change the model, the report becomes invalid. To keep your code generation report current, after modifying the source model, regenerate code and the report. If you close and then reopen a model, regenerate the report.

Trace from Code to Model

You must have already generated code with a traceability report. If not, see “Generate a Traceability Report from Configuration Parameters” on page 5-4 or “Generate a Traceability Report from the Command Line” on page 5-14.

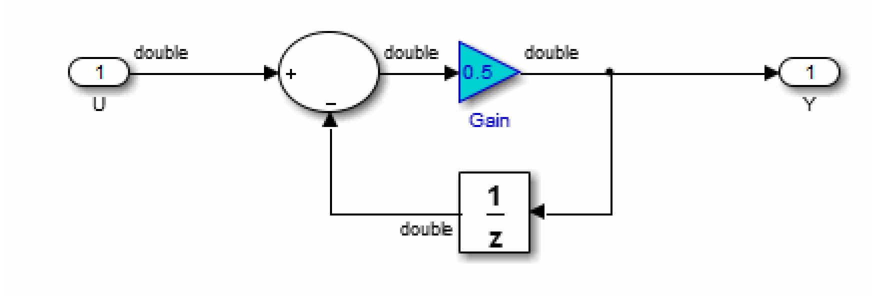
To trace generated code to your model:

1. In the generated code HTML report display, look for `<S1>/Gain`. Code Generation Report has syntax highlighting for easy readability. PLC-specific keywords are highlighted in blue, comments in green, and the rest of the code in black.



```
END_VAR
VAR_TEMP
    rtb_Gain: LREAL;
END_VAR
CASE ssMethodType OF
    SS_INITIALIZE:
        (* InitializeConditions for UnitDelay: '<S1>/Unit Delay' *)
        UnitDelay_DSTATE := 0.0;
    SS_STEP:
        (* Gain: '<S1>/Gain' incorporates:
         *  Inport: '<Root>/U'
         *  Sum: '<S1>/Sum'
         *  UnitDelay: '<S1>/Unit Delay' *)
        rtb_Gain := (U - UnitDelay_DSTATE) * 1.1;
        (* Outport: '<Root>/Y' *)
        Y := rtb_Gain;
        (* Update for UnitDelay: '<S1>/Unit Delay' *)
        UnitDelay_DSTATE := rtb_Gain;
END_CASE;
END_FUNCTION_BLOCK
```

2. In the HTML report window, click a link to highlight the corresponding source block. For example, in the HTML report shown in the previous figure, you click the hyperlink for the Gain block (highlighted) to view that block in the model. Clicking the hyperlink locates and displays the corresponding block in the model editor window. You can use the same method to trace other block from the HTML report.



Trace from Model to Code

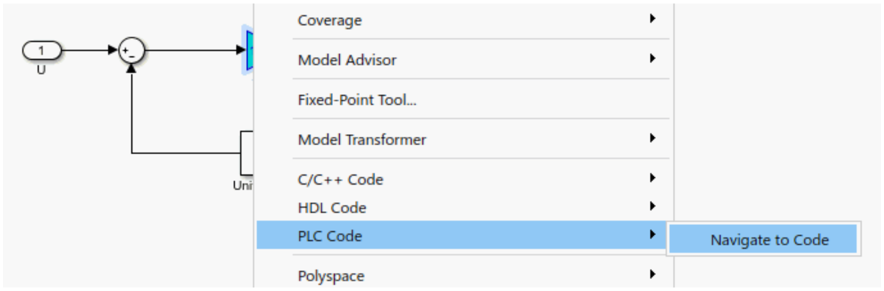
You can select a component at any level of the model with model-to-code traceability. You can also view the code references to that component in the HTML code generation report. You can select the following objects for tracing:

- Subsystem
- Simulink block
- MATLAB Function block
- Truth Table block
- State Transition Table block
- Stateflow chart, or the following elements of a Stateflow chart:
  - State
  - Transition
  - Graphical function
  - MATLAB function
  - Truth table function

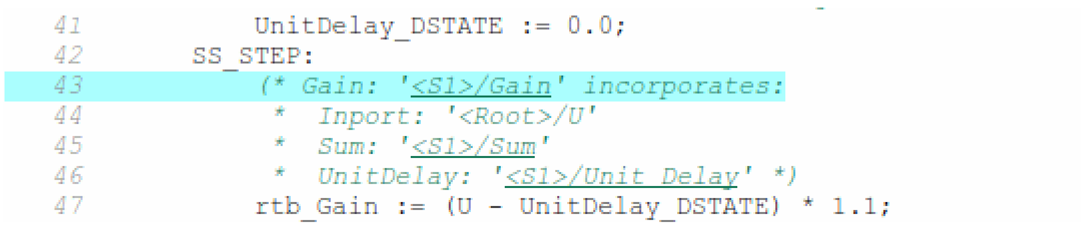
You must have already generated code with a traceability report to trace a model component to the generated code. If not, see “Generate a Traceability Report from Configuration Parameters” on page 5-4 or “Generate a Traceability Report from the Command Line” on page 5-14.

To trace a model component to the generated code:

1. In the model window, right-click the component and select **PLC Code > Navigate to Code**.



2 Selecting **Navigate to Code** activates the HTML code generation report. The following figure shows the result of tracing the Gain block within the subsystem.



```
            UnitDelay_DSTATE := 0.0;
        SS_STEP:
            (* Gain: '<S1>/Gain' incorporates:
             *  Inport: '<Root>/U'
             *  Sum: '<S1>/Sum'
             *  UnitDelay: '<S1>/Unit Delay' *)
            rtb_Gain := (U - UnitDelay_DSTATE) * 1.1;
```

In the report, the highlighted tag S1/Gain indicates the beginning of the generated code for the block. You can use the same method to trace from other Simulink, Stateflow, and MATLAB objects to the generated traceability report.

For a programmatic way to trace a block in the model to generated code, see `rtwtrace`.

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 need 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 the following web browsers:

- Mozilla® Firefox® Version 1.5 or later, which has native support for SVG. To download the Firefox browser, go to `www.mozilla.com/`.
- Apple Safari Web browser
- The Microsoft® Internet Explorer® web browser with the Adobe® SVG Viewer plugin. To download the Adobe SVG Viewer plugin, go to `www.adobe.com/svg/`.

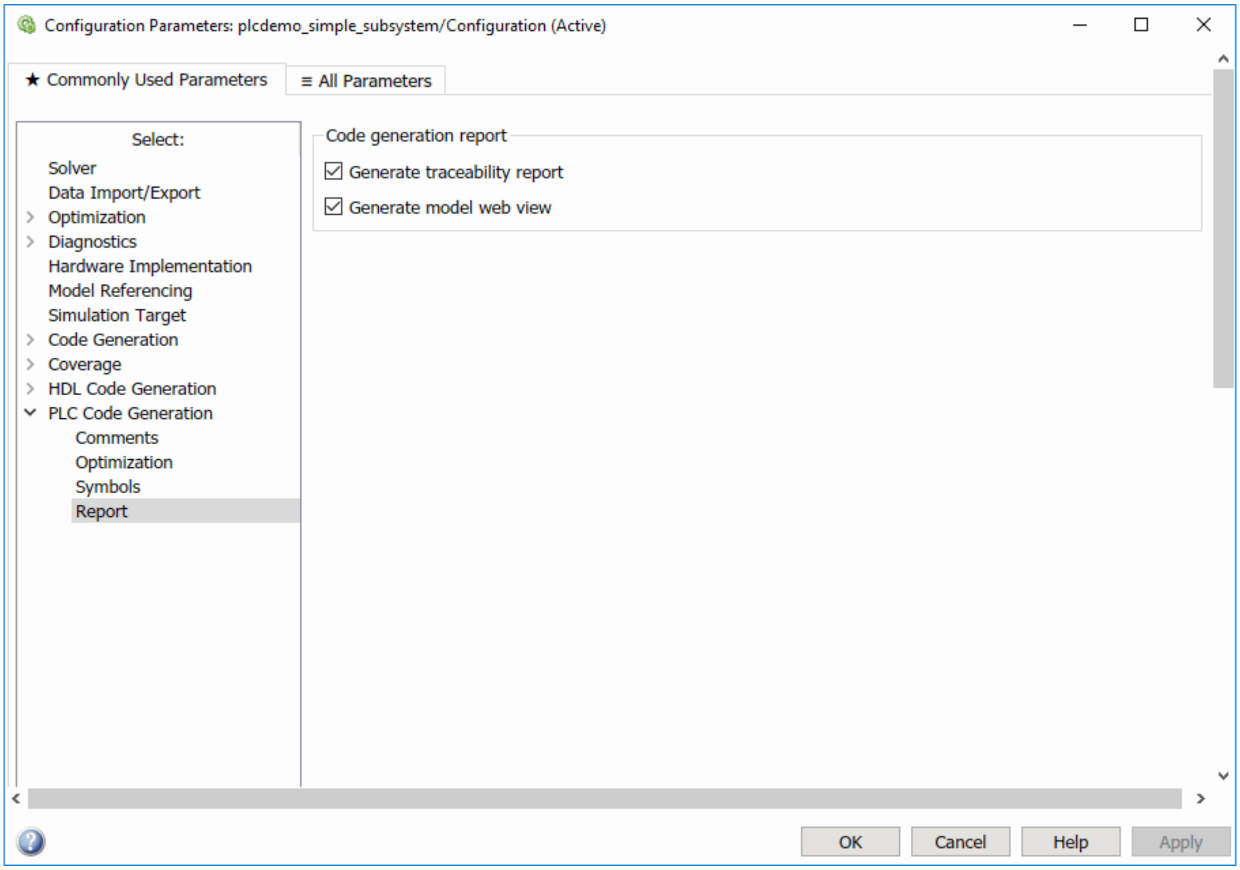
Note Web views do not currently support Microsoft Internet Explorer 9.

Generate HTML Code Generation Report with Model Web View

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

1. Open the `plcdemo_simple_subsystem` model.
2. Open the Configuration Parameters dialog box and navigate to the Code Generation pane.
3. To enable report generation, select **Report** > **Generate traceability report**.
4. To enable model web view, select **Report** > **Generate model web view**.
5. Click **Apply**.

   The dialog box looks something like this figure:

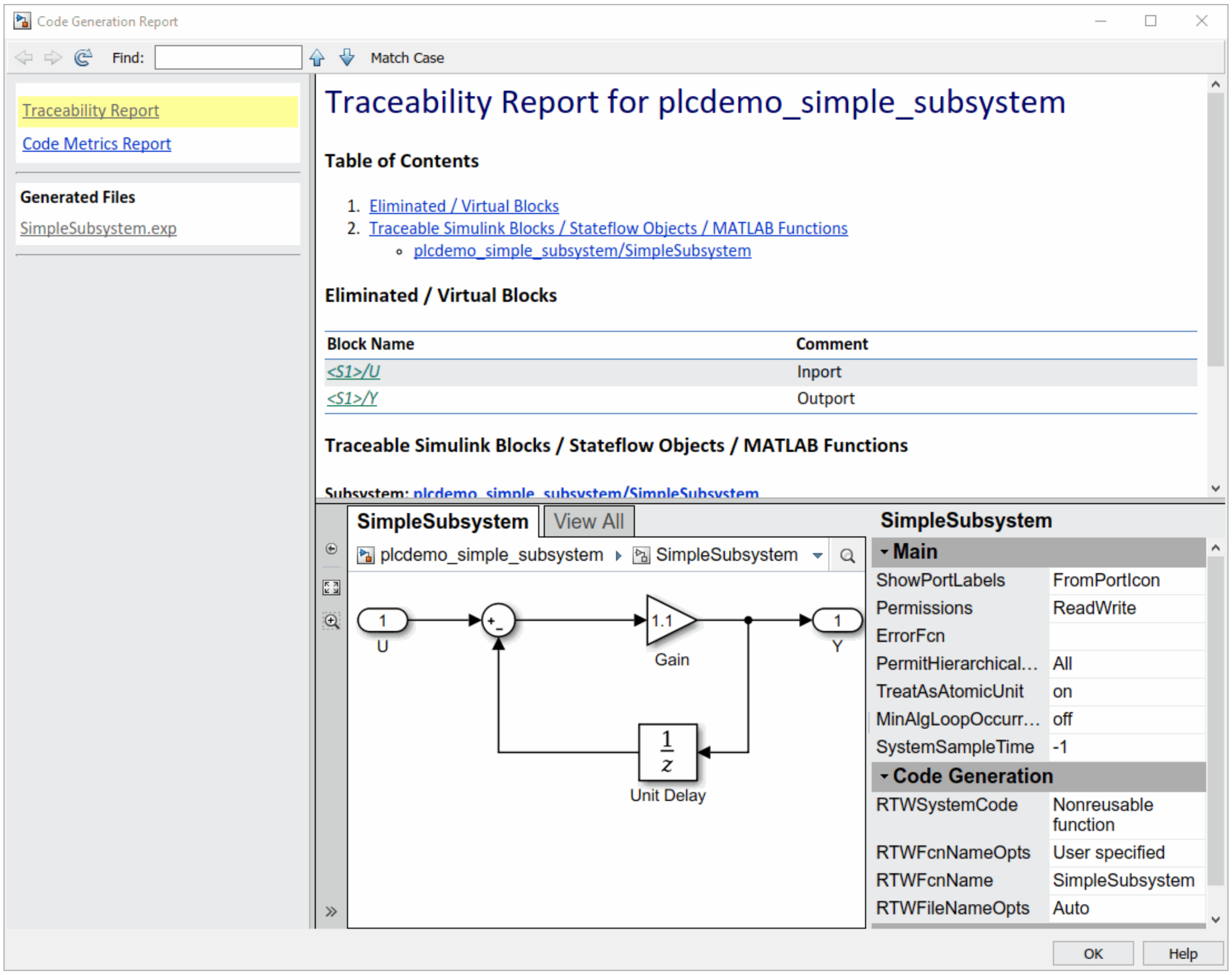

Configuration Parameters: plcdemo_simple_subsystem/Configuration (Active)

Code generation report

☑ Generate traceability report

☑ Generate model web view

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



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

SimpleSubsystem

Main	
ShowPortLabels	FromPortIcon
Permissions	ReadWrite
ErrorFcn	
PermitHierarchical…	All
TreatAsAtomicUnit	on
MinAlgLoopOccurr…	off
SystemSampleTime	-1
Code Generation	
RTWSystemCode	Nonreusable function
RTWFcnNameOpts	User specified
RTWFcnName	SimpleSubsystem
RTWFileNameOpts	Auto

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

8 Click a link in the code. The model web view displays and highlights the corresponding block in the model.

9 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 from Code to Model" on page 5-7
- "Trace from Model to Code" on page 5-8

Model Web View Limitations

When you are using the model web view, the HTML code generation report includes the following limitations:

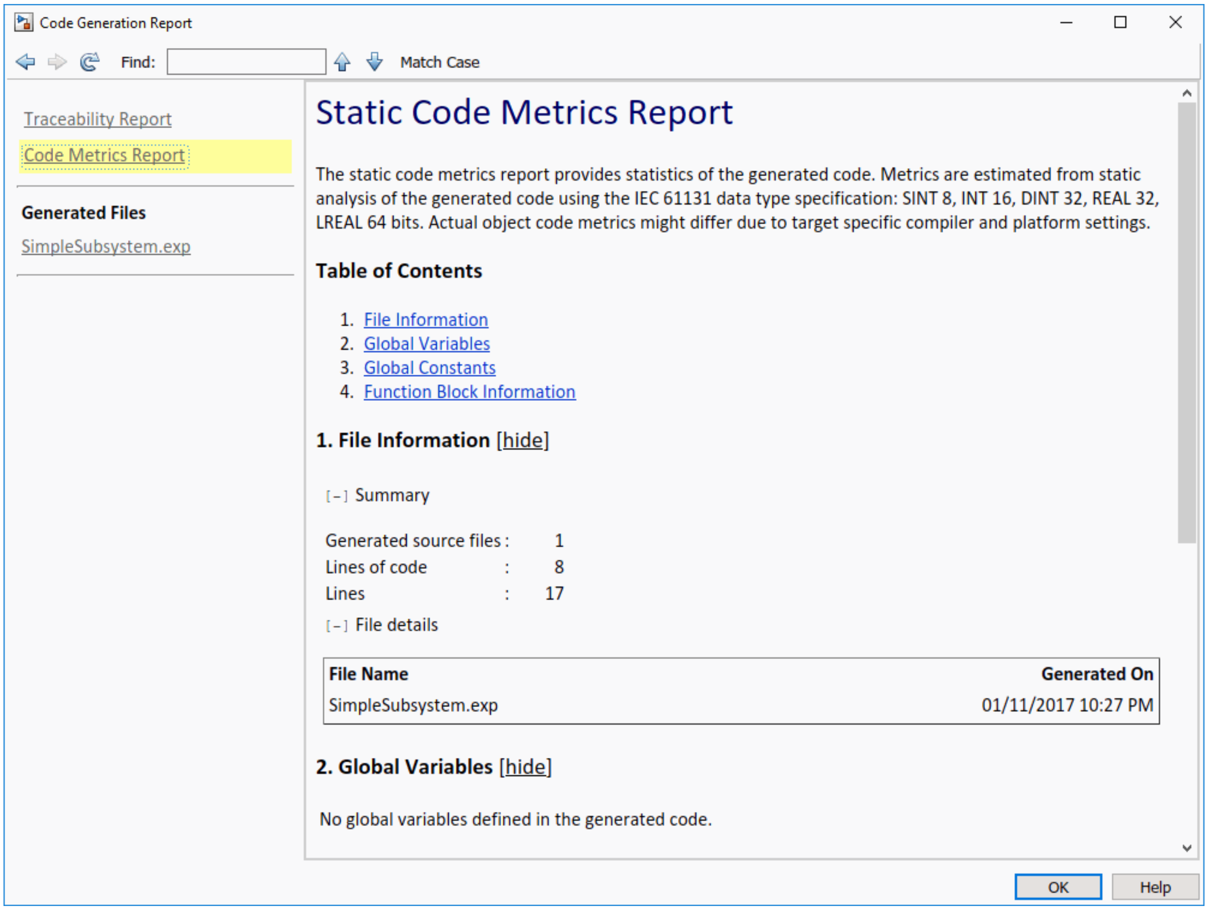
- Code is not generated for virtual blocks. In the model web view, if you click a virtual block, the code generation report clears highlighting in the source code files.
- Stateflow truth tables, events, and links to library charts are not supported in the model web view.
- Searching in the code generation report does not find or highlight text in the model web view.
- 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, see "Open Code Generation Report" (Simulink Coder).

Generate a Static Code Metrics Report

The PLC Coder Static Code Metrics report provides statistics of the generated code. 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. For more information, see "Working with the Static Code Metrics Report" on page 5-17.

The procedure is the same as generating the Traceability Report.

1. Open the Configuration Parameters dialog box and navigate to the **PLC Code Generation** pane.
2. To enable report generation, select **Report > Generate traceability report**.
3. Click **Apply**.
4. Click **PLC Code Generation > Generate code** to initiate code and report generation. The coder generates HTML report files as part of the code generation process. The Code Metrics Report is shown on the left navigation pane.



Static Code Metrics Report

The static code metrics report provides statistics of the generated code. Metrics are estimated from static analysis of the generated code using the IEC 61131 data type specification: SINT 8, INT 16, DINT 32, REAL 32, LREAL 64 bits. Actual object code metrics might differ due to target specific compiler and platform settings.

Table of Contents

1. File Information
2. Global Variables
3. Global Constants
4. Function Block Information

1. File Information [hide]

[-] Summary

```
Generated source files :   1
Lines of code          :   8
Lines                  :  17
```

[-] File details

File Name	Generated On
SimpleSubsystem.exp	01/11/2017 10:27 PM

2. Global Variables [hide]

No global variables defined in the generated code.

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.

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 Simulink Requirements™ 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" (Simulink Requirements).
2. For the subsystem for which you want to generate code, specify the following configuration parameters.

Option	Purpose
Include comments on page 13-14	Model information must appear in code comments.
Generate traceability report on page 13-38	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.

Working with the Static Code Metrics Report

In this section...

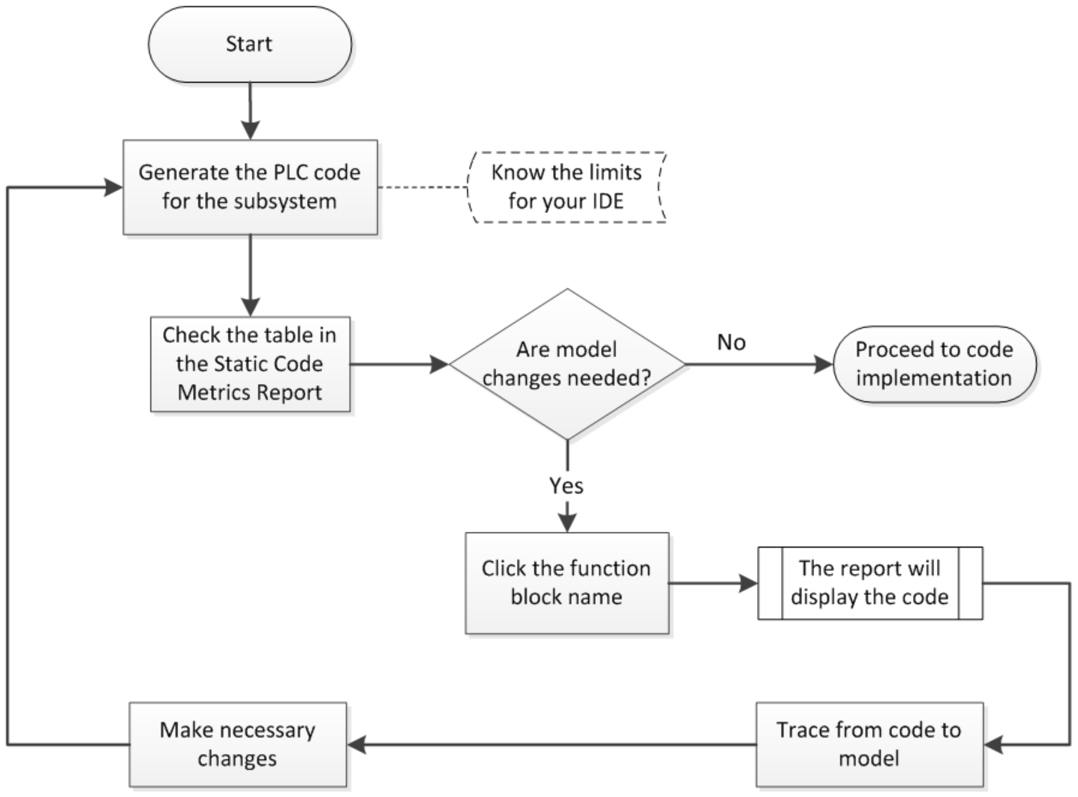
You can use the information in the Static Code Metrics Report to assess the generated code and make model changes before code implementation in your target IDE.

Before starting, you must 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 a Static Code Metrics Report” on page 5-13.

Workflow for Static Code Metrics Report

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


Start
Generate the PLC code for the subsystem
Know the limits for your IDE
Check the table in the Static Code Metrics Report
Are model changes needed?
No
Proceed to code implementation
Yes
Click the function block name
The report will display the code
Trace from code to model
Make necessary changes

Report Contents

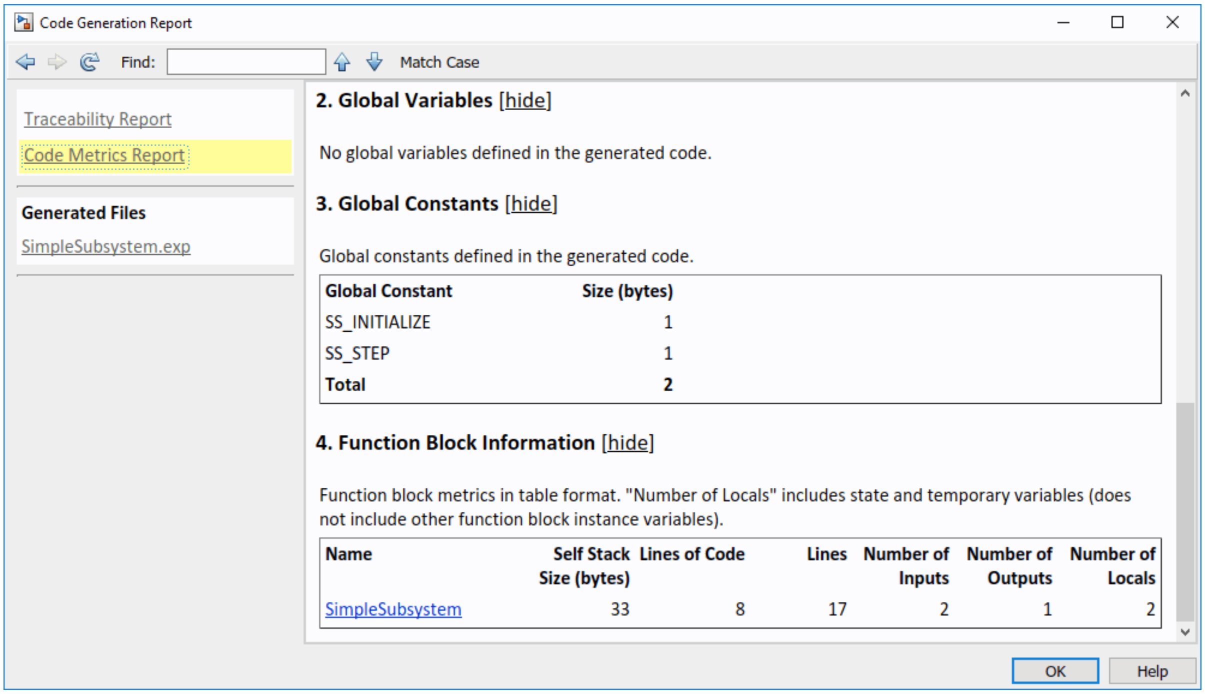
The Static Code Metrics Report is divided into the following sections:

- **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. From here, you can trace from the code to the model. For more information, see "Trace from Code to Model" on page 5-7.



2. Global Variables [hide]

No global variables defined in the generated code.

3. Global Constants [hide]

Global constants defined in the generated code.

Global Constant	Size (bytes)
SS_INITIALIZE	1
SS_STEP	1
Total	**2**

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
SimpleSubsystem	33	8	17	2	1	2

Working with Tunable Parameters in the Simulink PLC Coder Environment

Block Parameters in Generated Code

Block parameters appear in the generated code as variables. You can choose how the variables appear in the generated code. For instance, you can control the following variable characteristics:

- Whether the variables are inlined in generated code
- Whether the variables are local to a function block, global, or not defined

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 6-5.

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			
	SimulinkGlobal	**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`.

Target IDE	**Parameter Storage Class**			
	`SimulinkGlobal`	**`ExportedGlobal`**	**`ImportedExtern`**	**`Imported-ExternPointer`**
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`.
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`.
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`.

Target IDE	Parameter Storage Class			
	`SimulinkGlobal`	`ExportedGlobal`	`ImportedExtern`	`Imported-ExternPointer`
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	Local function block variable	Local function block variable.	Ignored. If you set the parameter to this value, the software treats it the same as `ImportedExtern`.
Generic	Local function block variable	Global variable	Variable 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 instance, you can control the following variable characteristics:

- Whether the variables are inlined in generated code
- Whether the variables are local to a function block, global, or not defined

For more information, see “Block Parameters in Generated Code” on page 6-2.

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 11-3.

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.

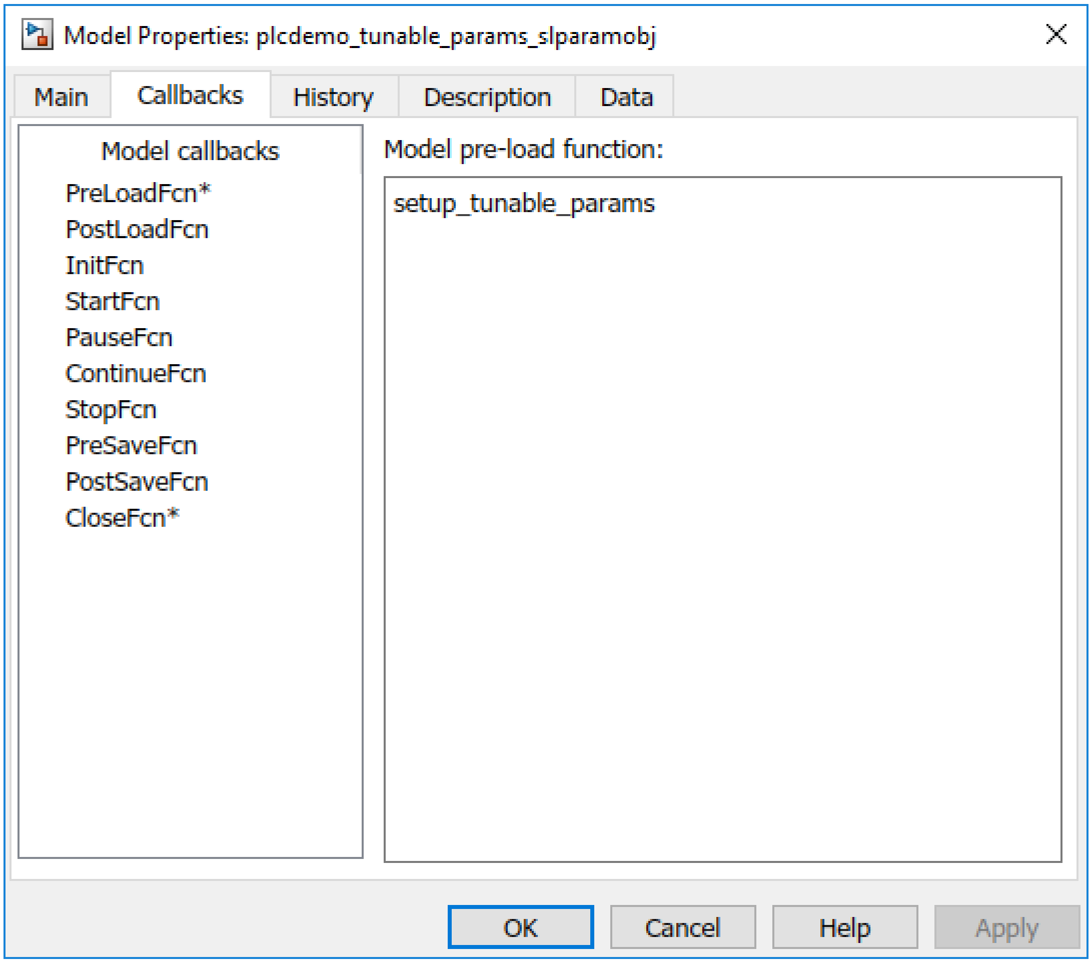
   The following script `setup_tunable_params.m` creates the constants `K1`, `K2`, and `K3` as `Simulink.Parameter` objects, assigns values, and sets the storage classes for these constants. For more information on the storage classes, see "Block Parameters in Generated Code" on page 6-2.

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

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

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

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

2 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, select **File > Model Properties > Model Properties**.

   b In the Model Properties dialog box, on the **Callbacks** tab, select `PreLoadFcn`. Enter `setup_tunable_params` for **Model pre-load function**.



Model pre-load function:

```
setup_tunable_params
```

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	`ExportedGlobal` and `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

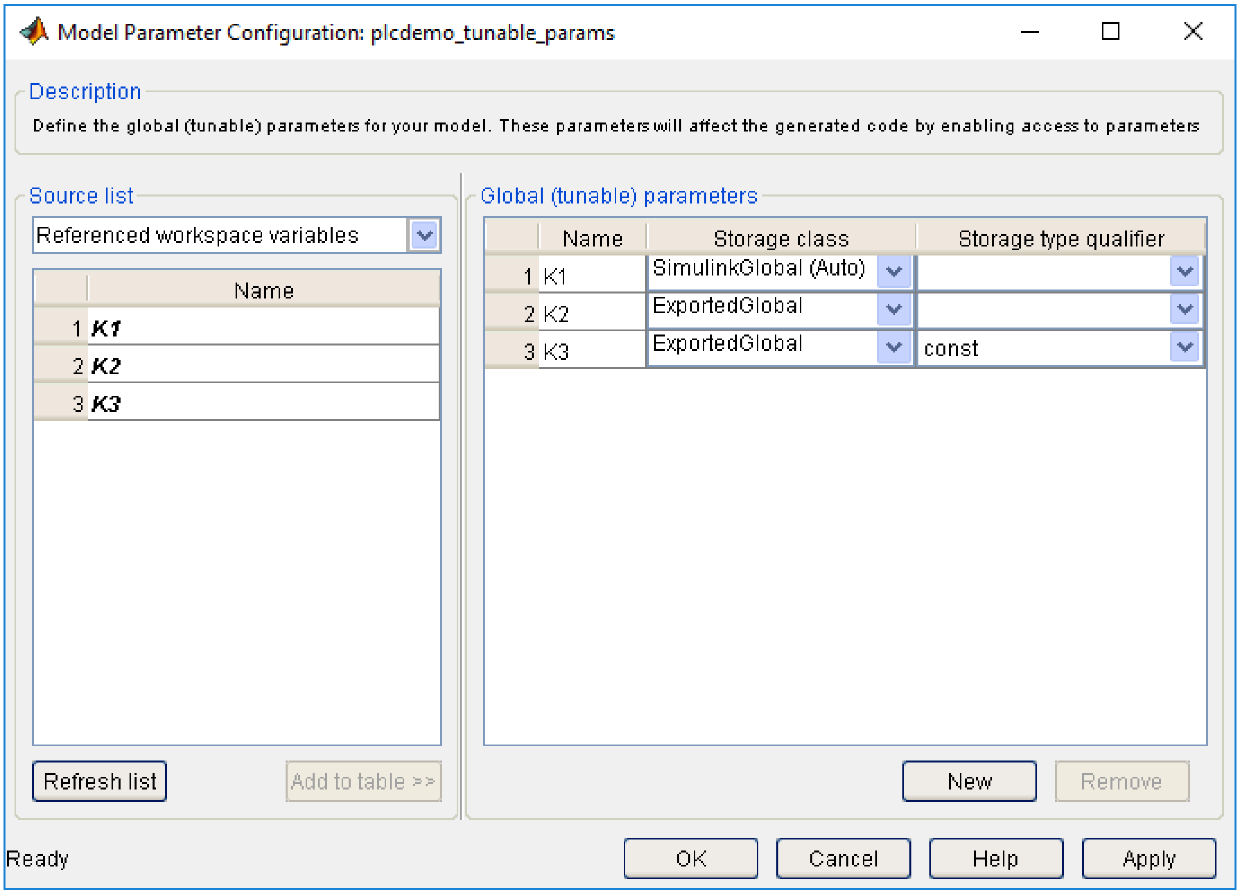
This example shows how to make parameters tunable using the Model Configuration Parameters dialog box.

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. Select **File** > **Model Properties** > **Model Properties**.
   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. Select **Simulation** > **Model Configuration Parameters**.
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 6-2.


Model Parameter Configuration: plcdemo_tunable_params

Description

Define the global (tunable) parameters for your model. These parameters will affect the generated code by enabling access to parameters

Source list

Referenced workspace variables

	Name
1	***K1***
2	***K2***
3	***K3***

Global (tunable) parameters

	Name	Storage class	Storage type qualifier
1	K1	SimulinkGlobal (Auto)	
2	K2	ExportedGlobal	
3	K3	ExportedGlobal	const

6 Generate code and inspect it.

Variable	Storage Class	Generated Code (3S CoDeSys 2.3)

`K1`	`SimulinkGlobal`	`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`	`ExportedGlobal` 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 model 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, select **View > Model Data Editor**. On the **Data Stores** tab, set the **Change View** drop-down to `Code`. Enable **Resolve** for the Data Store Memory block. For more information, see “Configure Data Properties by Using the Model Data Editor” (Simulink) .
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

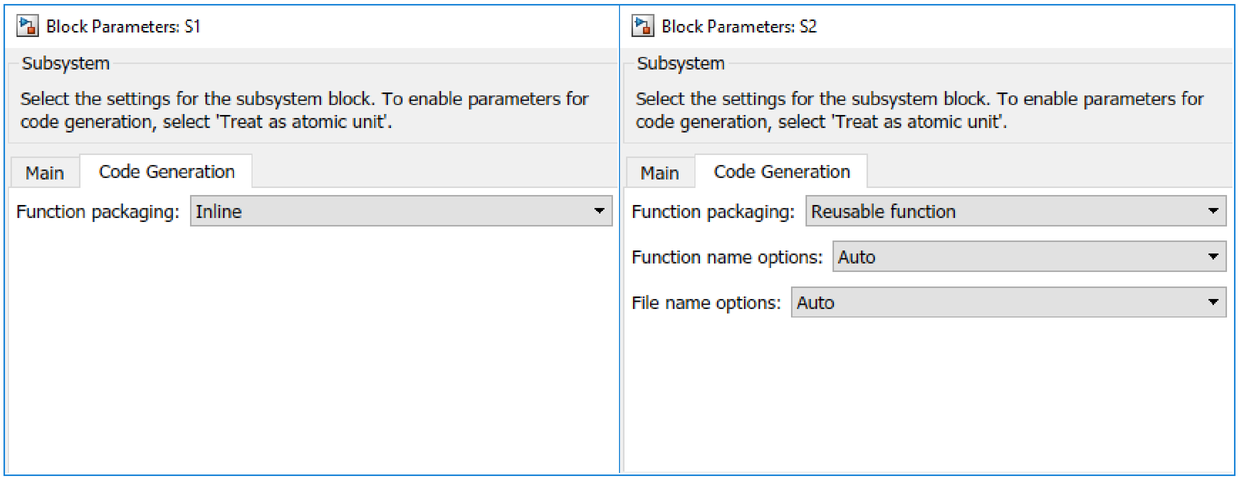
Use the **Function packaging** parameter to specify the code format to generate for an atomic (nonvirtual) subsystem. The Simulink PLC Coder software interprets this parameter depending on the setting that you choose:

Setting	Coder Interpretation
`Auto`	Uses the optimal format based on the type and number of subsystem instances in the model.

Setting	Coder Interpretation
`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, “Permanent Limitations” on page 11-6.
`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.


Block Parameters: S1

Subsystem

Select the settings for the subsystem block. To enable parameters for code generation, select 'Treat as atomic unit'.

Main | Code Generation

Function packaging: Inline

Block Parameters: S2

Subsystem

Select the settings for the subsystem block. To enable parameters for code generation, select 'Treat as atomic unit'.

Main | Code Generation

Function packaging: Reusable function

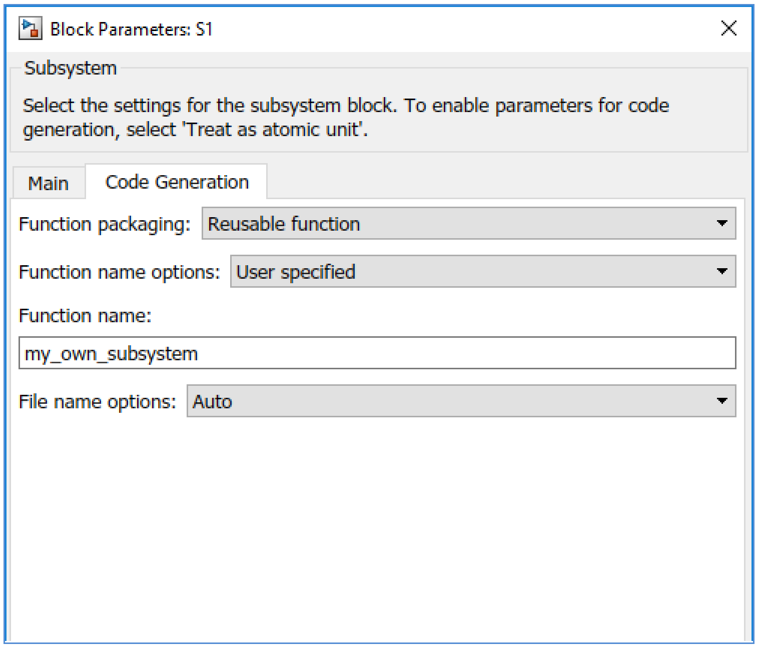
Function name options: Auto

File name options: Auto

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`.


Block Parameters: S1

Subsystem

Select the settings for the subsystem block. To enable parameters for code generation, select 'Treat as atomic unit'.

Function packaging: Reusable function

Function name options: User specified

Function name: my_own_subsystem

File name options: Auto

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

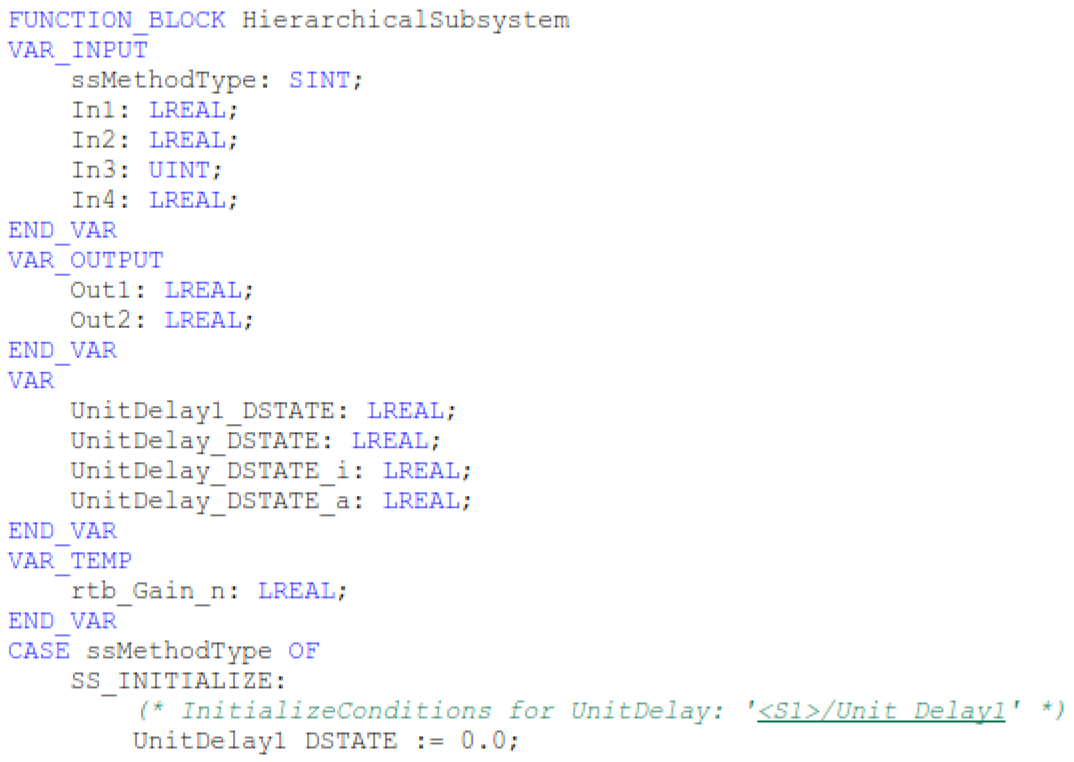
The code for `plcdemo_simple_subsystem` is an example of generating code with one function block. The atomic subsystem for which you generate code does not contain other subsystems.

```
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`

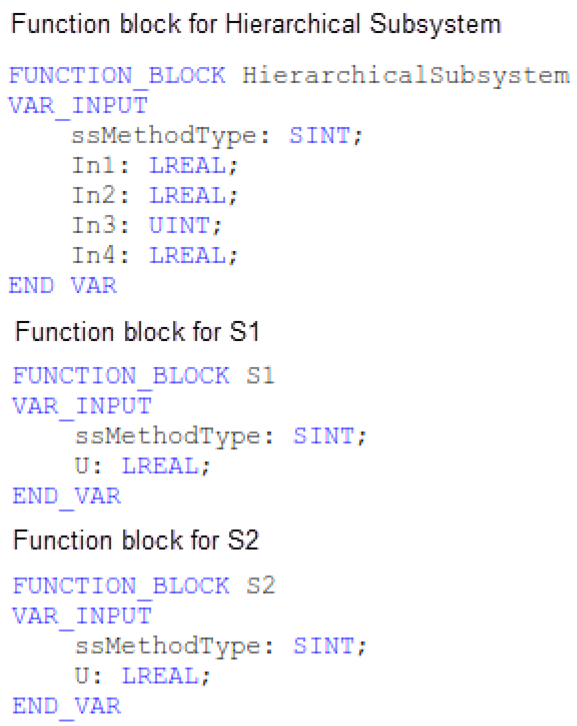
HierarchicalSubsystem component. There are no additional function blocks for the S1 and S2 subsystems.



```
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 MATLAB Function Properties in a Chart” (Stateflow).

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

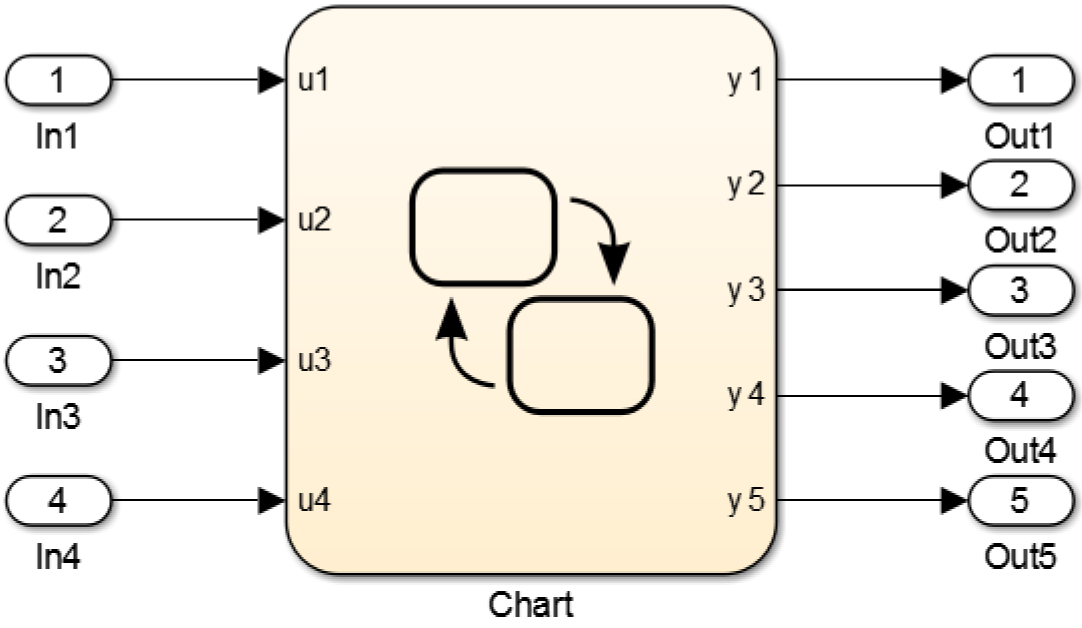
- If a MATLAB function accesses data that is local to the chart, it is inlined in generated code even if you specify that the function must not be inlined.

  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.

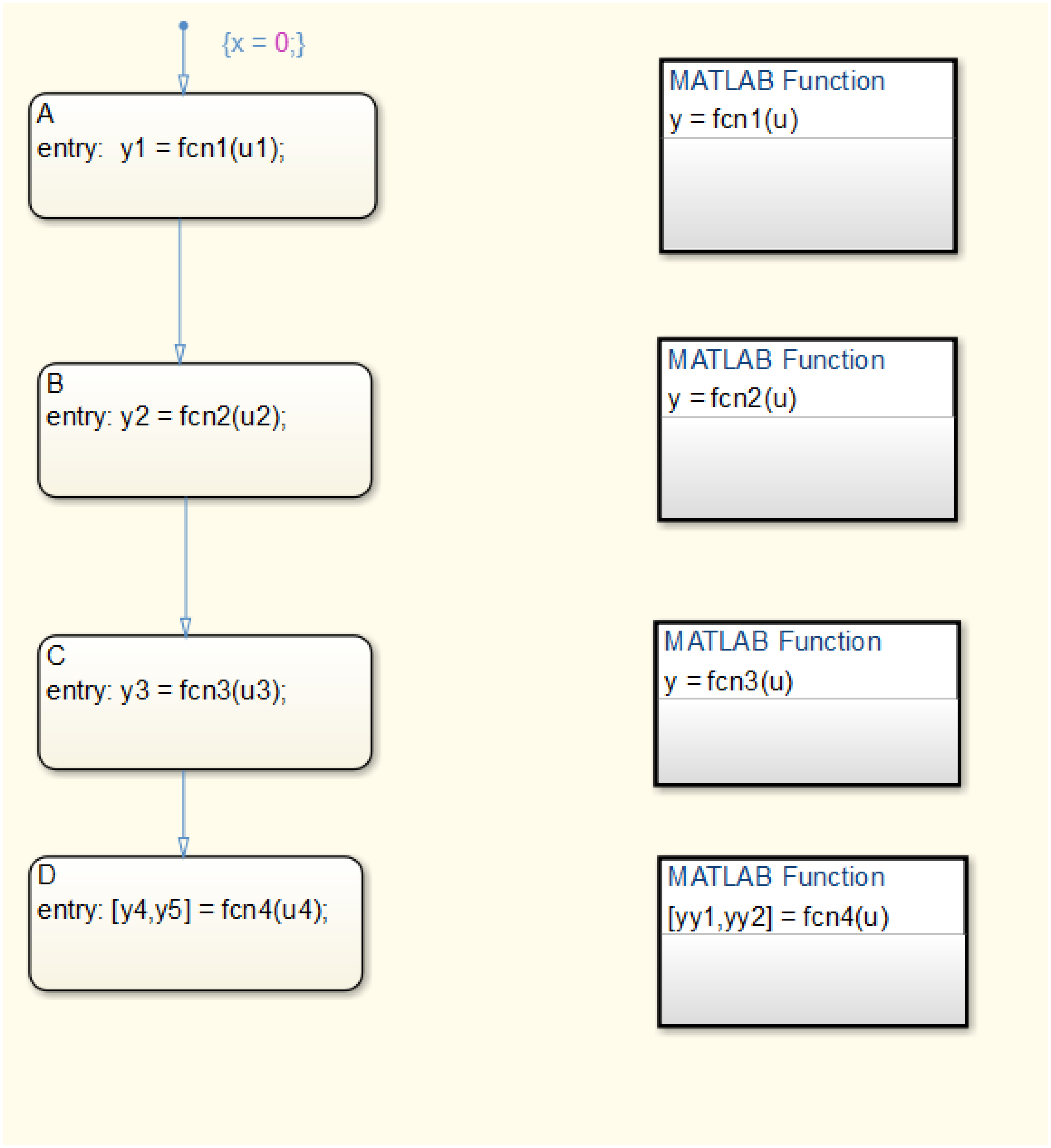
  Explanation: A Structured Text function cannot have multiple outputs, therefore the MATLAB function cannot be converted to a Structured Text function.

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.


{x = 0;}
A
entry: y1 = fcn1(u1);
B
entry: y2 = fcn2(u2);
C
entry: y3 = fcn3(u3);
D
entry: [y4,y5] = fcn4(u4);
MATLAB Function
y = fcn1(u)
MATLAB Function
y = fcn2(u)
MATLAB Function
y = fcn3(u)
MATLAB Function
[yy1,yy2] = fcn4(u)

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`

Function	Inlining Specification	Generated Code

`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 Symbols

Integrate Externally Defined Symbols

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

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

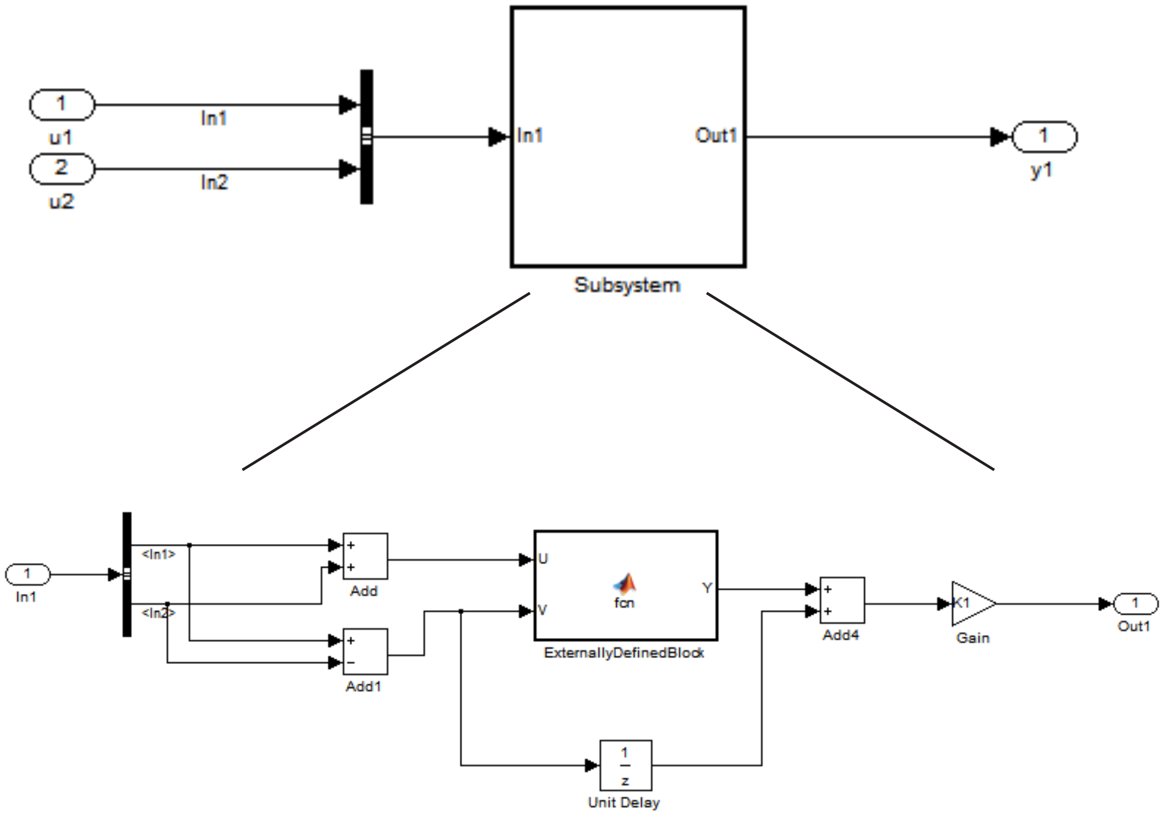
For a description of how to integrate a custom function block, see “Integrate Custom Function Block in Generated Code” on page 8-3. For a description of the **Externally Defined Symbols** parameter, see “Externally Defined Symbols” on page 13-34.

Integrate Custom Function Block in Generated Code

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

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

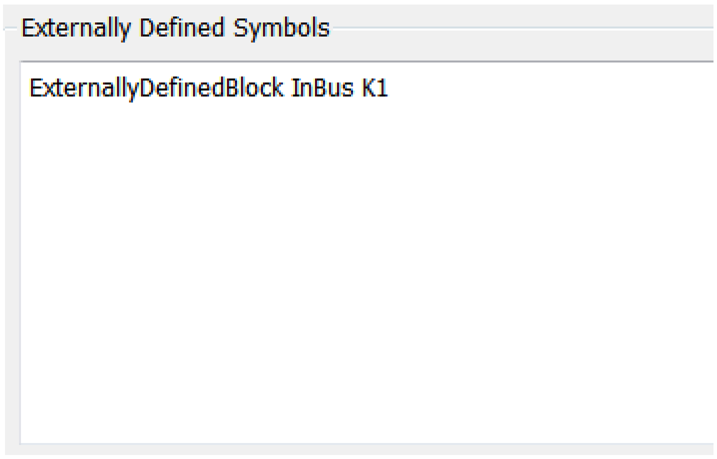
   ```
   function Y = fcn(U,V)
   % Stub behavior for simulation. This block
   % is replaced during code generation
   Y = U + V;
   ```
4. Change the MATLAB Function block name to ExternallyDefinedBlock.
5. Create a subsystem from this MATLAB Function block.
6. Complete the model to look like `plcdemo_external_symbols`.



7 Open the Configuration Parameters dialog box for the model.

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

9 The `plcdemo_external_symbols` model also suppresses `K1` and `InBus`. Add these symbol names to the **Externally Defined Symbols** field, separated by spaces 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 1-17.
- The location of generated code, see “Files Generated with Simulink PLC Coder” on page 1-23.

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.

   Unless you assigned a custom name, the file is called `*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.

Unless you assigned a custom name, the file is called `*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.

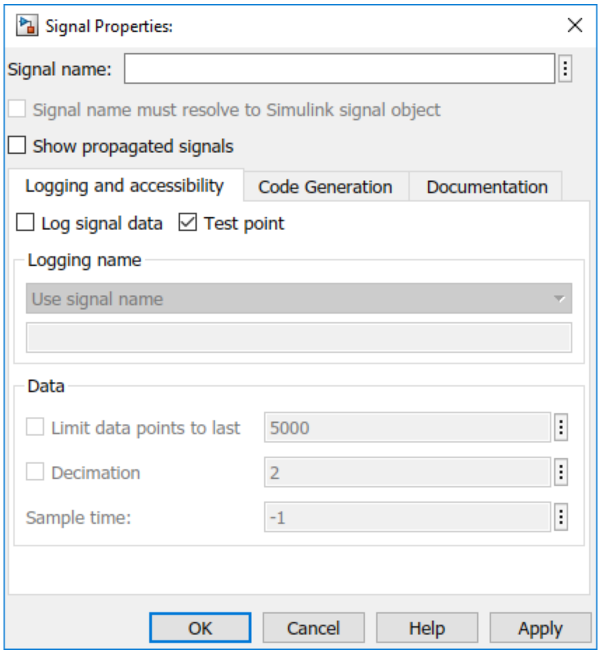
Use Internal Signals for Debugging in RSLogix 5000 IDE

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?” (Simulink).

Rockwell Automation RSLogix Considerations

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 AOIs for Add-On instruction format, not FUNCTION_BLOCK.

Double-Precision Data Types

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

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 numerical 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, the 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 Data Type Limitations” on page 11-4.

Enumerated Data Types

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

Considerations for Siemens IDEs

Following are considerations for this target IDE platform.

Double-Precision Floating-Point Data Types

The Siemens SIMATIC STEP 7 target IDE does not support double-precision floating-point data types. At code generation, the Simulink PLC Coder converts this data type to single-precision real data types in the generated code. Design your model so that the possible precision loss of numerical results of the generated code does not change the expected semantics of the model.

For Siemens PLC devices that support double-precision floating point types, use `Siemens TIA Portal: Double Precision` as **Target IDE** for generating code. 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 and Unsigned Integer Types

The SCL language for Siemens IDEs does not support int8 and unsigned integer data types. At code generation, the Simulink PLC Coder converts int8 and unsigned integer data types to int16 or int32 in 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 numerical results using int8 or unsigned integer data types can differ from the numerical results produced by the generated Structured Text.

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

Unsigned Fixed-Point Data Types

In the generated code, Simulink PLC Coder converts fixed-point data types to target IDE integer data types. Because the Siemens target IDEs do 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 Data Type Limitations” on page 11-4.

Enumerated Data Types

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

INOUT Variables

The Siemens SIMATIC STEP 7 and the TIA Portal single-precision targets do not support `INOUT` variables. If your Simulink model contains MATLAB Function blocks with *y = f (y)* style in-place variables, coder generates code using normal input and output variables. However, if the code generation option for the MATLAB Function block is set to use **Reusable function**, this conversion is not possible. To fix this issue, rewrite the MATLAB Function block without using in-place variables or change the block code generation option to either **Auto** or **Inline**.

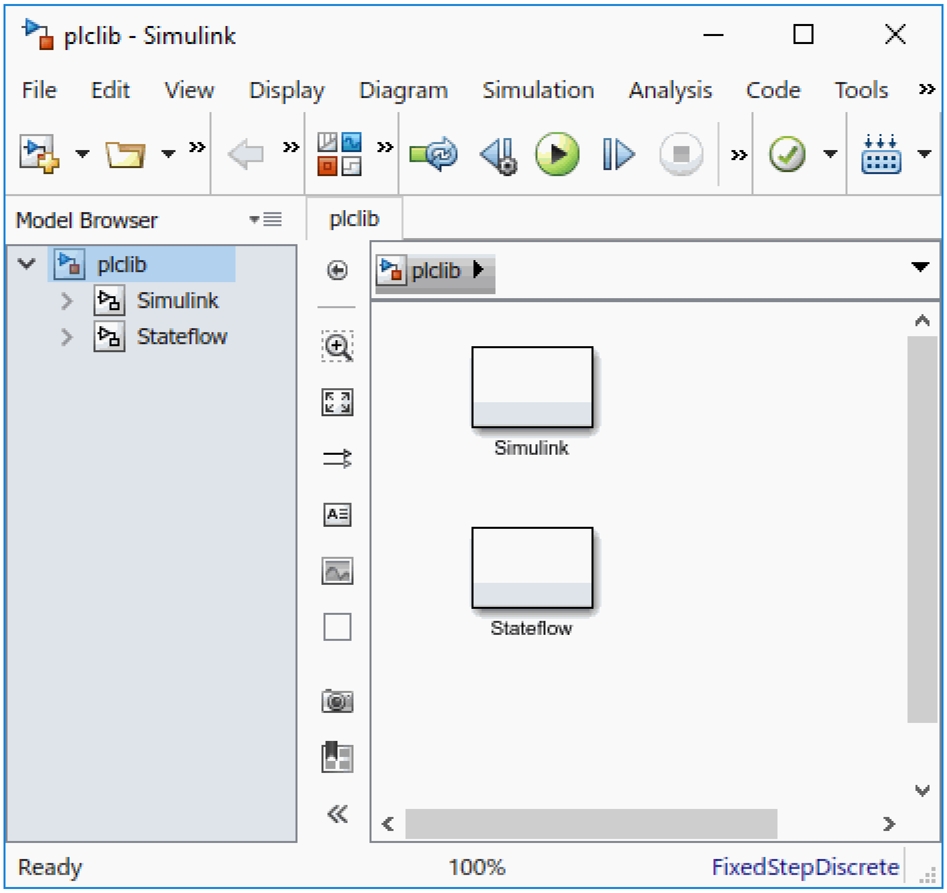
Supported Simulink and Stateflow Blocks

Supported Blocks

For Simulink semantics not supported by Simulink PLC Coder, see “Coder Limitations” on page 11-2.

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

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

XY Graph

Sources

Constant

Counter Free-Running

Counter Limited

Enumerated Constant

Ground

Pulse Generator

Repeating Sequence Interpolated

Repeating Sequence Stair

User-Defined Functions

MATLAB Function (MATLAB Function Block)

Fcn

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 “Coder Limitations” on page 11-2 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” (Simulink).

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.

11

Limitations

Coder Limitations

In this section...

Current Limitations

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

- Complex data types
- Model reference
- Stateflow machine-parented data and events
- Stateflow messages
- Limited support for math functions
- Merge block
- Step block
- Clock block
- Signal and state storage classes
- Shared state variables between subsystems
- Virtual buses at the input ports of the top-level Atomic Subsystem block
- For Each Subsystem block
- Variable-size signals and parameters
- Objects defined in the Simulink data dictionary, including model parameters, signals, and state objects.
- MATLAB System block or system objects

- Width block

  Use a MATLAB Function block instead. In the MATLAB function on the block, use the `length` function to compute input vector width.
- Cell arrays in MATLAB Function blocks
- In MATLAB Function blocks, only standard MATLAB functions are supported. Functions from toolboxes are not supported. For the list of standard functions supported for code generation, see “Functions and Objects Supported for C/C++ Code Generation — Category List” (Simulink).
- The use of `Simulink.CoderInfo` `Alias` name property with `Simulink.Parameter` and `Simulink.Signal` objects.

rand Function Support Limitations

Simulink PLC Coder generates Structured Text code for MATLAB Function blocks 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.

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';
```

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.

Fixed-Point Data Type Limitations

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

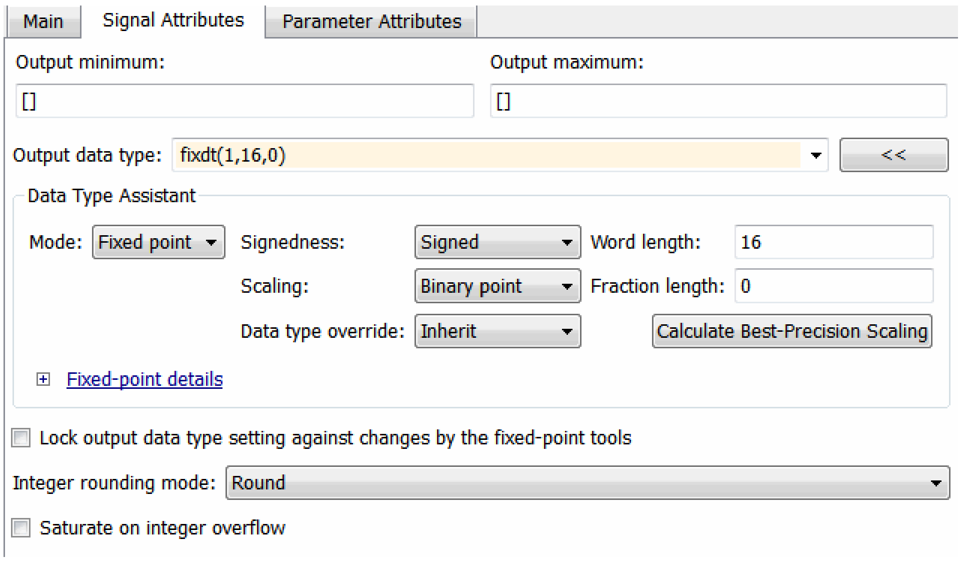
Note If you do not configure the blocks and models as directed, the generated Structured Text might:

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

Block Parameters

Properly configure block parameters:

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



Output minimum: []
Output maximum: []
Output data type: fixdt(1,16,0)
Data Type Assistant
Mode: Fixed point
Signedness: Signed
Word length: 16
Scaling: Binary point
Fraction length: 0
Data type override: Inherit
☐ Lock output data type setting against changes by the fixed-point tools
Integer rounding mode: Round
☐ Saturate on integer overflow

9 Click **OK**.

Be sure to edit the model configuration parameters (see “Model Configuration Parameters” on page 11-5).

Model Configuration Parameters

Properly configure model configuration parameters:

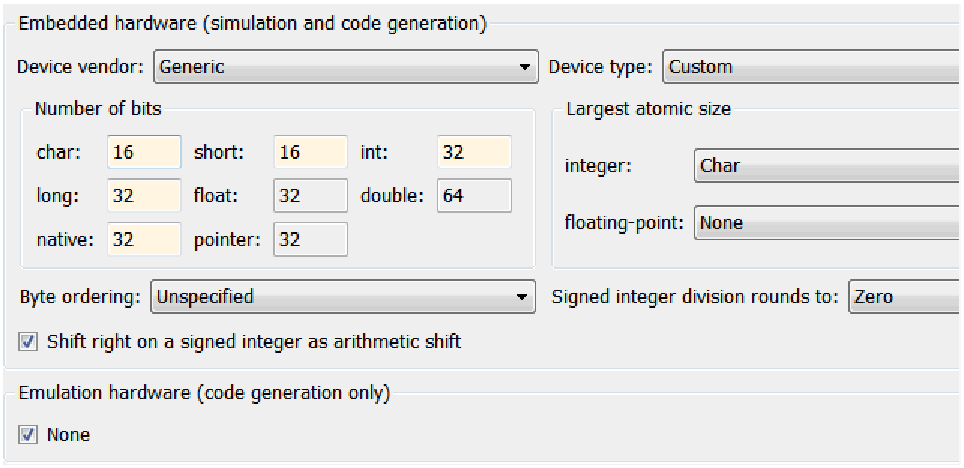
1 In model Configuration Parameters dialog box, click the `Hardware Implementation` node.

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

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

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

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



Embedded hardware (simulation and code generation)

Device vendor: Generic | Device type: Custom

Number of bits

char:	16	short:	16	int:	32
long:	32	float:	32	double:	64
native:	32	pointer:	32		

Largest atomic size

integer: Char

floating-point: None

Byte ordering: Unspecified

Signed integer division rounds to: Zero

[✓] Shift right on a signed integer as arithmetic shift

Emulation hardware (code generation only)

[✓] None

Multirate Model Limitations

To generate Structured Text from a multirate model, you must configure the model as follows:

- Change any continuous time input signals in the top-level subsystem to use discrete fixed sample times.
- For the solver, select single-tasking execution.

The B&R Automation Studio IDE is not supported for multirate model code generation.

When you deploy code generated from a multirate model, you must run the code at the fundamental sample rate.

Permanent Limitations

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

- The Simulink PLC Coder software supports code generation only for atomic subsystems.
- The Simulink PLC Coder software supports automatic, inline, or reusable function packaging for code generation. Nonreusable function packaging is not supported.

- No blocks that require continuous time semantics. This restriction includes continuous integrators, zero-crossing blocks, physical modeling blocks, and so on.
- No pointer data types.
- No recursion (including recursive events).
- Nonfinite data, for example `NaN` or `Inf`, is not supported.

12

Functions — Alphabetical List

plccoderdemos

Product examples

Syntax

`plccoderdemos`

Description

`plccoderdemos` displays the Simulink PLC Coder examples.

Examples

Display Simulink PLC Coder Examples

Enter the following at the command prompt: `plccoderdemos`

See Also

`plcopenconfigset`

Introduced in R2010a

plccoderpref

Manage user preferences

Syntax

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

Description

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

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

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

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

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

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

`plccoderpref('plctargetlist')` displays the target IDEs that appear in the reduced **Target IDE** list in the Simulink Configuration Parameters dialog box. For more information, see “Target IDE” on page 13-3 and “Show Full Target List” on page 13-6.

`plccoderpref('plctargetlist',`*`targetlist`*`)` sets the target IDEs that appear in the reduced **Target IDE** list to the values that you specify in *`targetlist`*.

Input Arguments

`plctargetide`

String directive that specifies the default target IDE.

Value	Description
`codesys23`	3S-Smart Software Solutions CoDeSys Version 2.3 (default) target IDE
`codesys33`	3S-Smart Software Solutions CoDeSys Version 3.3 target IDE
`codesys35`	3S-Smart Software Solutions CoDeSys Version 3.5 target IDE
`brautomation30`	B&R Automation Studio 3.0 target IDE
`brautomation40`	B&R Automation Studio 4.0 target IDE
`generic`	Generic target IDE
`indraworks`	Rexroth IndraWorks version 13V12 IDE
`multiprog50`	PHOENIX CONTACT (previously KW) Software MULTIPROG 5.0 or 5.50 target IDE
`omron`	OMRON Sysmac Studio
`plcopen`	PLCopen XML target IDE
`pcworx60`	Phoenix Contact PC WORX 6.0
`rslogix5000`	Rockwell Automation RSLogix 5000 Series target IDE for AOI format
`rslogix5000_routine`	Rockwell Automation RSLogix 5000 Series target IDE for routine format

Value	Description
step7	Siemens SIMATIC STEP 7 Version 5 target IDE
studio5000	Rockwell Studio 5000 Logix Designer target IDE for AOI format
studio5000_routine	Rockwell Studio 5000 Logix Designer target IDE for routine format
twincat211	Beckhoff TwinCAT 2.11 target IDE
twincat3	Beckhoff TwinCAT 3 target IDE
tiaportal	Siemens TIA Portal
tiaportal_double	Siemens TIA Portal with support for double precision (LREAL type)

Default: `codesys23`

plctargetidepaths

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

```
codesys23: 'C:\Program Files\3S Software'
codesys33: 'C:\Program Files\3S CoDeSys'
codesys35: 'C:\Program Files\3S CoDeSys'
studio5000: 'C:\Program Files\Rockwell Software'
studio5000_routine: 'C:\Program Files\Rockwell Software'
rslogix5000: 'C:\Program Files\Rockwell Software'
rslogix5000_routine: 'C:\Program Files\Rockwell Software'
brautomation30: 'C:\Program Files\BrAutomation'
brautomation40: 'C:\Program Files\BrAutomation'
multiprog50: 'C:\Program Files\KW-Software\MULTIPROG 5.0'
pcworx60: 'C:\Program Files\Phoenix Contact\Software Suite 150'
step7: 'C:\Program Files\Siemens'
tiaportal: 'C:\Program Files\Siemens\Automation'
tiaportal_double: 'C:\Program Files\Siemens\Automation'
plcopen: ''
twincat211: 'C:\TwinCAT'
twincat3: 'C:\TwinCAT'
generic: ''
indraworks: ''
omron: ''
```

default

String that sets your preferences to the factory default.

plctargetlist

Cell array of strings. Each string specifies a target IDE. You can specify any target IDE that is available for the `plctargetide` argument.

Use the string `default` to reset the reduced **Target IDE** list.

Examples

Return Current Default Target IDE

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

```
ans =
'rslogix5000'
```

Set `rslogix5000` as New Default Target IDE

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

```
ans =
'rslogix5000'
```

See Installation Paths of Supported Target IDEs

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

```
plccoderpref('plctargetidepaths')
```

```
ans = struct with fields:
              codesys23: 'E:/hub/hub_share/share/apps/3S-Software/CoDeSys/v2.3'
              codesys33: 'C:\Program Files\3S CoDeSys'
              codesys35: 'C:\Program Files\3S CoDeSys'
             studio5000: ''
     studio5000_routine: ''
            rslogix5000: ''
```

```
    rslogix5000_routine: ''
         brautomation30: 'C:\Program Files\BrAutomation'
         brautomation40: 'C:\Program Files\BrAutomation'
            multiprog50: 'C:\Program Files\KW-Software\MULTIPROG 5.0'
               pcworx60: 'C:\Program Files\Phoenix Contact\Software Suite 150'
                  step7: 'C:\Program Files\Siemens'
              tiaportal: 'C:\Program Files\Siemens\Automation'
       tiaportal_double: 'C:\Program Files\Siemens\Automation'
                plcopen: ''
             twincat211: 'C:\TwinCAT'
               twincat3: 'C:\TwinCAT'
                generic: ''
             indraworks: ''
                  omron: ''
```

Customize Reduced Target IDE List

If you disable **Show full target list**, the drop down for **Target IDE** shows only a subset of the supported IDEs. Customize this reduced list to contain only the IDEs CoDeSys 2.3 and Rockwell Automation RSLogix 5000 Series for AOI format.

```
targetlist = {'codesys23','rslogix5000'};
plccoderpref('plctargetlist',targetlist)
```

```
ans = 1x2 cell array
    {'codesys23'}    {'rslogix5000'}
```

Reset Reduced Target IDE List

Reset the reduced **Target IDE** list to the default subset.

```
plccoderpref('plctargetlist','default')
```

```
ans = 1x5 cell array
    {'codesys23'}    {'studio5000'}    {'step7'}    {'omron'}    {'plcopen'}
```

Append Another IDE to Default Reduced Target IDE List

Append the IDE CoDeSys 3.5 to the default reduced **Target IDE** list.

```
plccoderpref('plctargetlist', [plccoderpref('plctargetlist', 'default') 'codesys35'])
```

```
ans = 1x6 cell array
  Columns 1 through 5

    {'codesys23'}    {'studio5000'}    {'step7'}    {'omron'}    {'plcopen'}

  Column 6

    {'codesys35'}
```

Append Another IDE to Current Reduced Target IDE List

Append the IDE CoDeSys 3.5 to the current reduced **Target IDE** list.

```
plccoderpref('plctargetlist', [plccoderpref('plctargetlist') 'codesys35'])
```

```
ans = 1x6 cell array
  Columns 1 through 5

    {'codesys23'}    {'studio5000'}    {'step7'}    {'omron'}    {'plcopen'}

  Column 6

    {'codesys35'}
```

Tips

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

Introduced in R2010a

plcgeneratecode

Generate Structured Text for subsystem

Syntax

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

Description

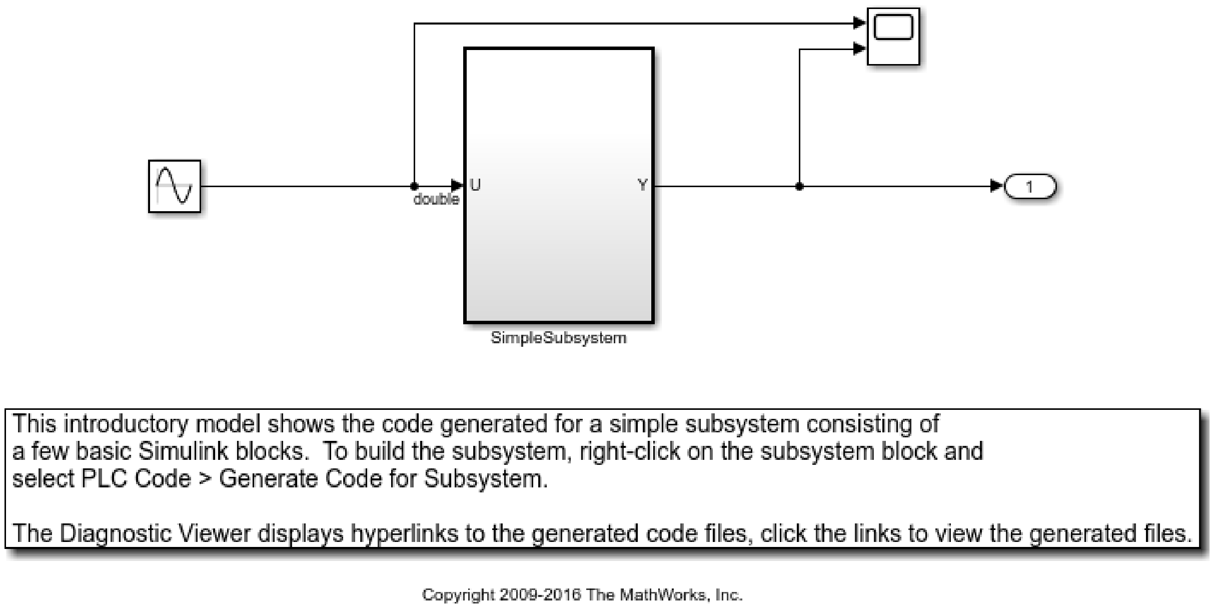
`generatedfiles = plcgeneratecode(`*`subsystem`*`)` generates Structured Text for the specified atomic subsystem in a model. *`subsystem`* is the fully qualified path name of the atomic subsystem. *`generatedfiles`* is a cell array of the generated file names. You must first load or start the model.

Examples

Generate Structured Text Code for Subsystem

Open or load the model containing the subsystem.

```
plcdemo_simple_subsystem
```



This introductory model shows the code generated for a simple subsystem consisting of a few basic Simulink blocks. 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.

Copyright 2009-2016 The MathWorks, Inc.

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

```
generatedFiles = plcgeneratecode('plcdemo_simple_subsystem/SimpleSubsystem');
```

```
PLC code generation successful for 'plcdemo_simple_subsystem/SimpleSubsystem'.

Generated files:
<a href="matlab: edit('.\plcsrc\plcdemo_simple_subsystem.exp')">.\plcsrc\plcdemo_simple
```

See Also

`plcopenconfigset`

Introduced in R2010a

plcopenconfigset

Open Configuration Parameters dialog box for subsystem

Syntax

```
plcopenconfigset(subsystem)
```

Description

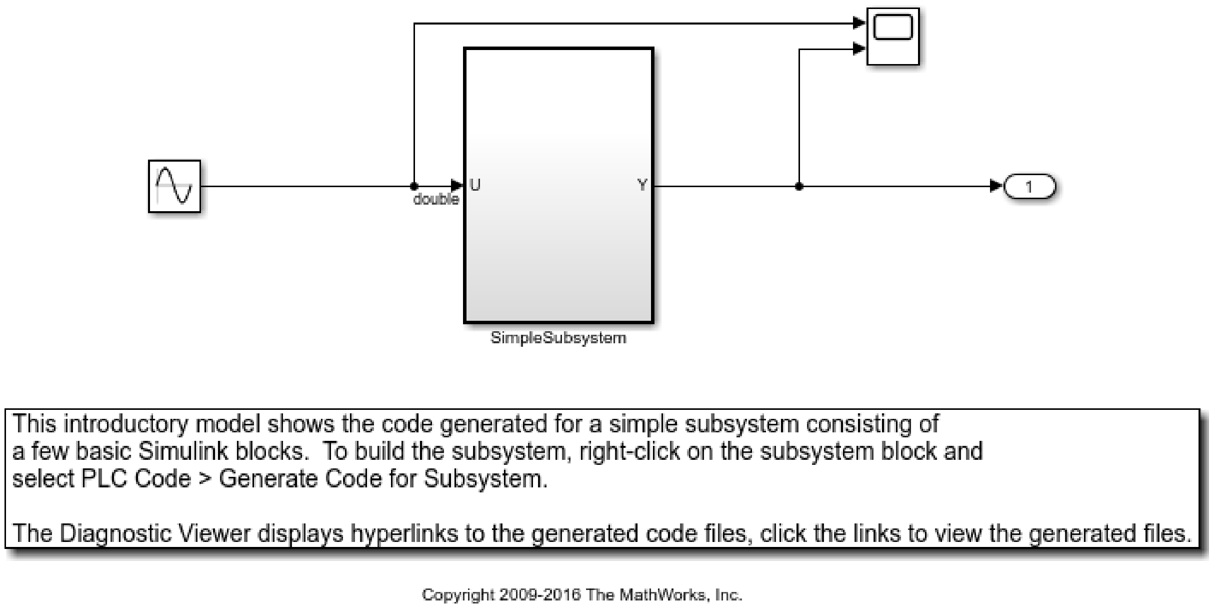
`plcopenconfigset(`*`subsystem`*`)` opens the Configuration Parameters dialog box for the specified atomic subsystem in the model. *subsystem* is the fully qualified path name of the atomic subsystem.

Examples

Open Configuration Parameters for Subsystem

Open the model containing the subsystem.

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



This introductory model shows the code generated for a simple subsystem consisting of a few basic Simulink blocks. 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.

Copyright 2009-2016 The MathWorks, Inc.

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

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

See Also

`plcgeneratecode`

Introduced in R2010a

plccheckforladder

Check whether Stateflow chart is ready for Ladder Diagram code generation

Syntax

```
plccheckforladder(chartPath)
```

Description

`plccheckforladder(chartPath)` checks whether a Stateflow chart is ready for Ladder Diagram code generation. If the chart has properties that do not allow Ladder Diagram code generation on page 3-19, the function shows errors in the Diagnostic Viewer window.

Examples

Preparation of Stateflow Chart for Ladder Diagram Code Generation

Open the model `plcdemo_ladder_three_aspect`.

```
open_system('plcdemo_ladder_three_aspect')
```

The model contains a subsystem `Subsys`, which contains a Stateflow chart, `3Aspect`. Save the model elsewhere with the name `plcdemo_ladder_three_aspect_copy`.

Enable super step semantics for the chart. In the chart properties, select **Enable Super Step Semantics**.

Check whether the Stateflow chart is ready for Ladder Diagram code generation.

```
plccheckforladder('plcdemo_ladder_three_aspect_copy/Subsys/3Aspect')
```

You see the following error message in the Diagnostic Viewer window:

```
Chart must not have superstep semantics enabled in Objects: 'Subsys/3Aspect'
```

Prepare the chart for Ladder Diagram code generation.

```
plcprepareforladder('plcdemo_ladder_three_aspect_copy/Subsys/3Aspect')
```

Check again whether the chart is ready for Ladder Diagram code generation.

```
plccheckforladder('plcdemo_ladder_three_aspect_copy/Subsys/3Aspect')
```

There are no more error messages. The function `plcprepareforladder` has disabled super step semantics for the chart.

Input Arguments

`chartPath` — Full path name of the Stateflow chart
character vector

Full path name of the Stateflow chart relative to the top level Simulink model, specified as a character vector. To obtain the full path, select the Stateflow chart in your model and use the `gcb` function.

Example: `gcb`, `'ThreeAspectAutoSignal/Subsystem/AutoSignalChart'`

See Also

`plcgenerateladder` | `plcprepareforladder`

Topics

“Prepare Chart for Ladder Diagram Generation” on page 3-6
“Generate Ladder Diagram Code from Stateflow Chart” on page 3-10
“Import Ladder Diagram Code to CODESYS 3.5 IDE and Validate Diagram” on page 3-15
“Ladder Diagram Generation for PLC Controllers” on page 3-2
“Supported IDE Platforms” on page 1-6

Introduced in R2016b

plcprepareforladder

Change some Stateflow chart properties to enable Ladder Diagram code generation

Syntax

```
plcprepareforladder(chartPath)
```

Description

`plcprepareforladder(chartPath)` changes certain properties of a Stateflow chart so that the chart is ready for Ladder Diagram code generation. The following properties are changed:

- The data types of inputs and outputs are changed to Boolean.
- The action language of the chart is changed to C.
- Super step semantics and chart initialization at execution are disabled.

Examples

Preparation of Stateflow Chart for Ladder Diagram Code Generation

Open the model `plcdemo_ladder_three_aspect`.

```
open_system('plcdemo_ladder_three_aspect')
```

The model contains a subsystem `Subsys`, which contains a Stateflow chart, `3Aspect`. Save the model elsewhere with the name `plcdemo_ladder_three_aspect_copy`.

Enable super step semantics for the chart. In the chart properties, select **Enable Super Step Semantics**.

Check whether the Stateflow chart is ready for Ladder Diagram code generation.

```
plccheckforladder('plcdemo_ladder_three_aspect_copy/Subsys/3Aspect')
```

You see the following error message in the Diagnostic Viewer window:

```
Chart must not have superstep semantics enabled in Objects: 'Subsys/3Aspect'
```

Prepare the chart for Ladder Diagram code generation.

```
plcprepareforladder('plcdemo_ladder_three_aspect_copy/Subsys/3Aspect')
```

Check again whether the chart is ready for Ladder Diagram code generation.

```
plccheckforladder('plcdemo_ladder_three_aspect_copy/Subsys/3Aspect')
```

There are no more error messages. The function `plcprepareforladder` has disabled super step semantics for the chart.

Tips

- Before you use this function, make a backup copy of your model because the function changes chart properties.
- The function does not change all properties that would allow for Ladder Diagram code generation. You must explicitly change certain properties. For the full list of chart properties that are not allowed, see “Restrictions on Stateflow Chart for Ladder Diagram Generation” on page 3-19.

Input Arguments

`chartPath` — Full path name of the Stateflow chart
character vector

Full path name of the Stateflow chart relative to the top level Simulink model, specified as a character vector. To obtain the full path, select the Stateflow chart in your model and use the `gcb` function.

Example: `gcb`, `'ThreeAspectAutoSignal/Subsystem/AutoSignalChart'`

See Also

`plccheckforladder` | `plcgenerateladder`

Topics

“Prepare Chart for Ladder Diagram Generation” on page 3-6
“Generate Ladder Diagram Code from Stateflow Chart” on page 3-10
“Import Ladder Diagram Code to CODESYS 3.5 IDE and Validate Diagram” on page 3-15
“Ladder Diagram Generation for PLC Controllers” on page 3-2
“Supported IDE Platforms” on page 1-6

Introduced in R2016b

plcgenerateladder

Generate Ladder Diagram code from Stateflow chart

Syntax

```
plcgenerateladder(chartPath)
plcgenerateladder(chartPath,Name,Value)
```

Description

`plcgenerateladder(chartPath)` generates code from a Stateflow chart that you can import to an IDE such as CODESYS 3.5 and view as a ladder diagram.

`plcgenerateladder(chartPath,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments. For instance, you can create a validation model or test bench to compare the generated Ladder Diagram code against the original Stateflow chart.

Examples

Generate Ladder Diagram Code from Stateflow Chart

Load the model, `plcdemo_ladder_three_aspect`. The model contains a subsystem `Subsys`, which contains a Stateflow chart, `3Aspect`.

```
load_system('plcdemo_ladder_three_aspect')
```

Specify the target IDE as CODESYS 3.5 or as a Rockwell Automation AOI.

```
set_param('plcdemo_ladder_three_aspect','PLC_TargetIDE','codesys35')
```

Generate Ladder Diagram code from the Stateflow chart.

```
plcgenerateladder('plcdemo_ladder_three_aspect/Subsys/3Aspect')
```

```
Conformance check results written to file : plcsrc\plcdemo_ladder_three_aspect_Conforma
Textual ladder logic equations written to file : plcsrc\plcdemo_ladder_three_aspect_Equ
Generating xml representation of the ladder equations
PLC code generation successful for 'plcdemo_ladder_three_aspect/Subsys/3Aspect'.

Generated files:
plcsrc\plcdemo_ladder_three_aspect.xml

ans = 1x3 cell array
    {'plcsrc\plcdemo_...'}    {'plcsrc\plcdemo_...'}    {'plcsrc\plcdemo_...'}
```

If code generation is successful, three files are generated in the subfolder plcsrc of your current folder:

- `ModelName_Equations.txt` : Text file containing the Ladder Diagram code.
- `ModelName.xml` : File containing the Ladder Diagram code in a format suitable for import. You use this file to import the code to your IDE and view the ladder diagram. The format is XML for CODESYS 3.5. For other target IDEs, the file uses an appropriate format suitable for importing.
- `ModelName_ConformanceChecks.txt` : Text file showing the result of conformance checks on the Stateflow chart. The conformance checks determine if the Stateflow chart is ready for generation of Ladder Diagram code. If code generation fails, this file lists the conformance checks that were not satisfied.

Generate Ladder Diagram Code with Testbench from Stateflow Chart

Load the model `plcdemo_ladder_three_aspect`, which contains a Stateflow chart.

```
load_system('plcdemo_ladder_three_aspect')
```

Generate Ladder Diagram code from the Stateflow chart, `3Aspect`, along with a test bench.

```
plcgenerateladder('plcdemo_ladder_three_aspect/Subsys/3Aspect', ...
     'GenerateTestBench','on')
```

```
Conformance check results written to file : plcsrc\plcdemo_ladder_three_aspect_Conforma
Textual ladder logic equations written to file : plcsrc\plcdemo_ladder_three_aspect_Equ
Generating xml representation of the ladder equations
PLC code generation successful for 'plcdemo_ladder_three_aspect/Subsys/3Aspect'.
```

```
Generated files:
plcsrc\plcdemo_ladder_three_aspect.xml

ans = 1x3 cell array
    {'plcsrc\plcdemo_...'}    {'plcsrc\plcdemo_...'}    {'plcsrc\plcdemo_...'}
```

You can import the Ladder Diagram code and the test bench together to a target IDE such as CODESYS 3.5. In the IDE, you can validate the ladder diagram against the test bench.

Input Arguments

`chartPath` — Full path name of the Stateflow chart

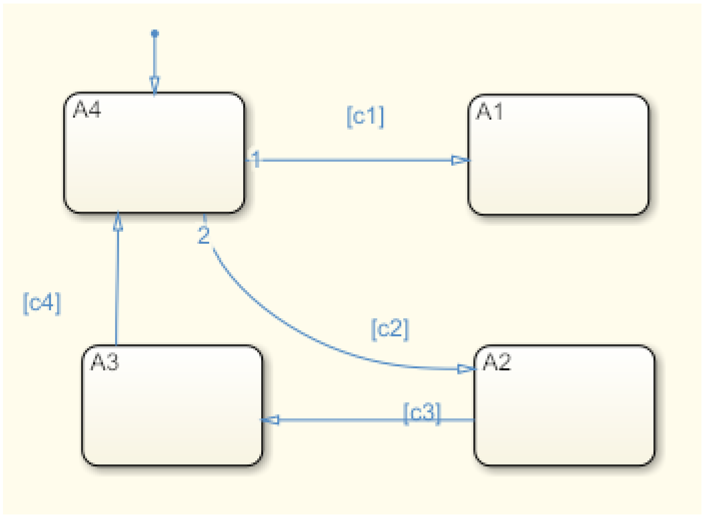
character vector

Full path name of the Stateflow chart relative to the top level Simulink model, specified as a character vector. To obtain the full path, select the Stateflow chart in your model and use the `gcb` function.

The Stateflow chart must have these properties:

- The inputs and outputs to the chart must be Boolean. These inputs and outputs correspond to the input and output terminals of your PLC.
- Each state of the chart must correspond to a chart output.
- The expressions controlling the transition between states must involve only Boolean operations between the inputs.

For instance, in the following chart, `c1`, `c2`, `c3`, and `c4` are Boolean inputs to the model. `A1`, `A2`, `A3`, and `A4` are Boolean outputs from the model.



Some advanced Stateflow features on page 3-19 are not supported because of inherent restrictions in ladder logic semantics. See the full list of unsupported features.

Example: `gcb`, `'ThreeAspectAutoSignal/Subsystem/AutoSignalChart'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'GenerateTestBench','on','PLC_OutputDir','laddereqn'` generates test bench code in addition to the ladder diagram and places the generated files in the subfolder `laddereqn` of the current working folder.

`GenerateTestBench` — Generate test bench for validation

`'off'` (default) | `'on'`

Specify whether a test bench must be generated.

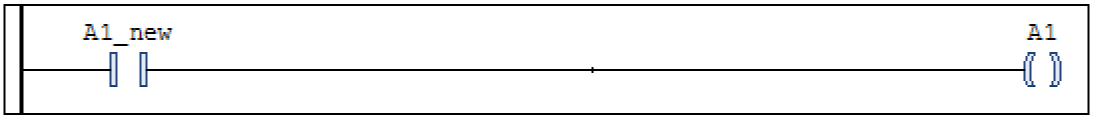
You can import the Ladder Diagram code and the test bench together to a target IDE such as CODESYS 3.5. In the IDE, you can validate the ladder diagram against the test bench.

`InsertGuardResets` — Add reset coils to safeguard against multiple active states
`'off'` (default) | `'on'`

In the ladder diagram, when the output coil corresponding to the active state is turned on, reset coils can be used to force deactivation of other states. The reset coils act as a safeguard against multiple states being simultaneously active. Specify whether the reset coils must be generated.

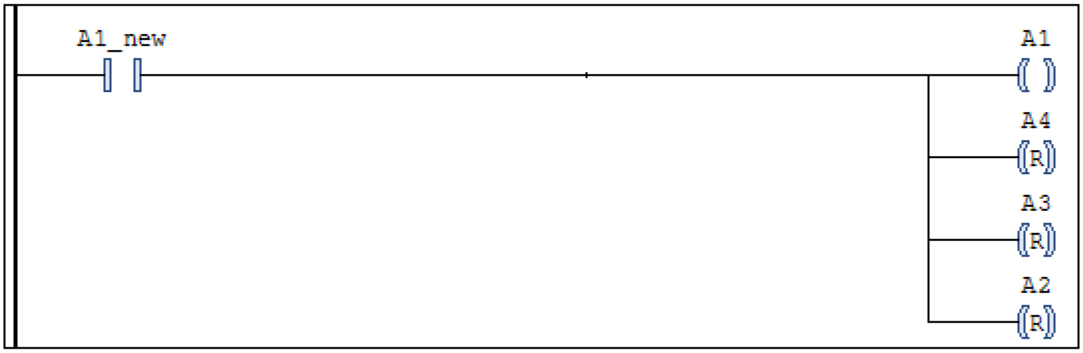
- If you do not enable this option, each output is a coil that represents a state in the chart.

  The following figure shows an output of the diagram when imported into the CODESYS 3.5 IDE. The output coil represents a state `A1` in the chart. When the state is active, the coil receives power.



- If you enable this option, each output is a coil that represents a state of the chart. The output is also coupled with reset coils that represent the other states. When a particular state is active, the reset coils force deactivation of the other states.

  The following figure shows an output in the ladder diagram when viewed in the CODESYS 3.5 IDE. The output coil represents a state `A1`. To avoid multiple states from being simultaneously active, the signal that turns the coil on also turns on the reset coils associated with the other states `A2`, `A3`, and `A4`.



`GenerateValidationModel` — Generate model with Ladder Diagram code for validation
`'off'` (default) | `'on'`

Specify whether a validation model must be generated. You can use the validation model to compare the generated Ladder Diagram code against the original Stateflow chart.

The validation model has two Subsystem blocks:

- The first block has the original Stateflow Chart.
- The second block has the Ladder Diagram code in a MATLAB Function block.

When you simulate this validation model, for all inputs, the software verifies the output of the second block against the first block. If the output of the second Subsystem block does not match the first, the simulation fails.

`PLC_OutputDir` — Path relative to current folder where generated files are placed
`'plcsrc'` (default) | character vector

Path relative to the current folder, specified as a character vector. The generated code files are placed in this subfolder. If you do not specify a value, the subfolder `plcsrc` is used.

The output folder must not have the same name as the current folder. For instance, if you do not specify an output folder, `plcsrc` is used. If the current folder is also `plcsrc`, an error occurs.

Example: `'out\plccode'`

See Also
`plccheckforladder` | `plcprepareforladder`

Topics
“Prepare Chart for Ladder Diagram Generation” on page 3-6
“Generate Ladder Diagram Code from Stateflow Chart” on page 3-10
“Import Ladder Diagram Code to CODESYS 3.5 IDE and Validate Diagram” on page 3-15
“Ladder Diagram Generation for PLC Controllers” on page 3-2
“Supported IDE Platforms” on page 1-6

Introduced in R2016b

plcimportladder

Import ladder diagram into a Simulink subsystem

Syntax

```
genmdlname = plcimportladder(filepath,program_name,routine_name)
```

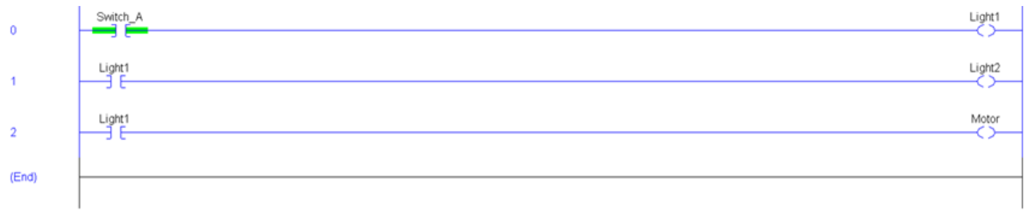
Description

`genmdlname = plcimportladder(filepath,program_name,routine_name)` generates a Simulink representation of the ladder diagram in the L5X file created using Rockwell Automation IDEs such as RSLogix 5000 and Studio 5000.

Examples

Import Simple Ladder Diagram into Simulink

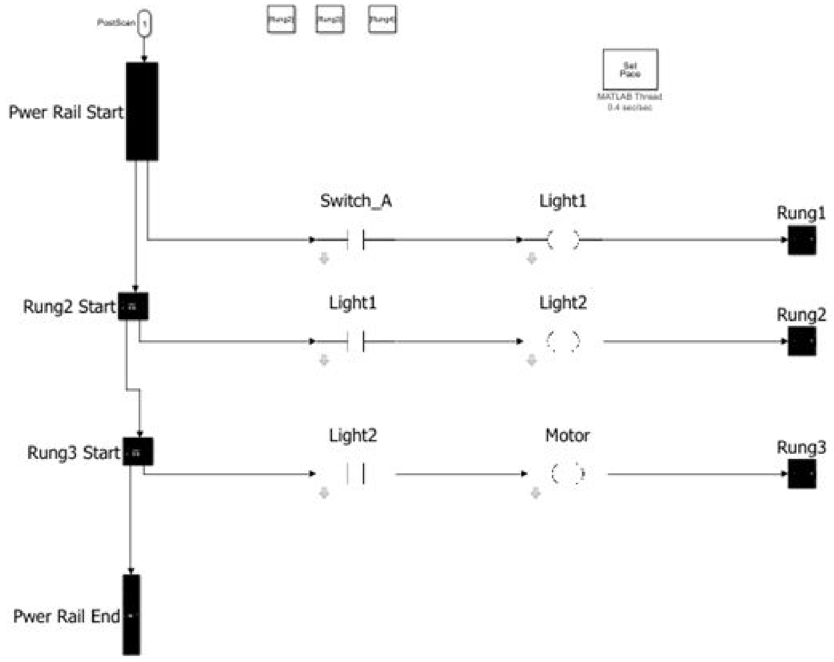
The following example demonstrates how to import a simple ladder diagram from an L5X file (`simple_ladder.L5X`) into the Simulink environment. The ladder L5X file was created using RSLogix 5000 IDE and contains contacts and coils representing motor and lights. The following is a snapshot of the ladder structure.



Use the `plcladderimport` function to import the ladder into Simulink. For this example, the program `Name` of the ladder is `MainProgram` and the `MainRoutineName` is `MainRoutine`.

```
plcimportladder('simple_ladder.L5X','MainProgram','MainRoutine')
```

The model imported into Simulink has blocks that implement the functionality of the contacts and coils.



Input Arguments

`filepath` — Full file path

character vector

Specifies the relative or absolute path to the ladder L5X file, exported from the Rockwell Automation IDEs.

`program_name` — Program or AOI name

character vector

Specifies the name of the `AddOnInstruction` tag (AOI tag) or the `ProgramName` tag containing the ladder logic.

.

`routine_name` — Routine Name
character vector

Specifies the name of the ladder logic routine structure.

Output Arguments

`genmdlname` — Simulink model name
character array

Specifies the name of the generated Simulink model.

See Also

`plcgeneratecode` | `plcgenerateladder` | `plcopenconfigset`

Topics

"Import Ladder Diagram into Simulink" on page 3-22
"Import L5X Ladder Files into Simulink" on page 3-25

Introduced in R2018a

plcdispextmodedata

Display the external mode logging data

Syntax

```
plcdispextmodedata(filename)
```

Description

`plcdispextmodedata(filename)` displays logging data information contained in the `filename` MAT-file on the MATLAB command window. The OPC Toolbox™ is required to run the external mode visualization.

Examples

Display Logging Data Information

The following example reads the logging data information stored in `plc_log_data.mat` and displays it on the command window.

```
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
```

Input Arguments

`filename` — Name of the MAT-file
character vector

Name of the MAT-file containing the logging information.

See Also

`plcgeneratecode` | `plcrunextmode`

Topics

“External Mode Logging” on page 14-2
“Generate Structured Text Code with Logging Instrumentation” on page 14-3
“Use the Simulation Data Inspector to Visualize and Monitor the Logging Data” on page 14-7

Introduced in R2018a

plcrunextmode

Run external mode visualization

Syntax

```
plcrunextmode(opc_host,target_ide,mdl_name,log_file)
plcrunextmode( ___ ,idx_list)
plcrunextmode( ___ ,name_list)
```

Description

`plcrunextmode(opc_host,target_ide,mdl_name,log_file)` runs external mode visualization using the settings specified in the arguments. All the logged signals are displayed in the Simulation Data Inspector.

`plcrunextmode( ___ ,idx_list)` runs external mode visualization and displays only the logged signals identified by the indices in the `idx_list` .

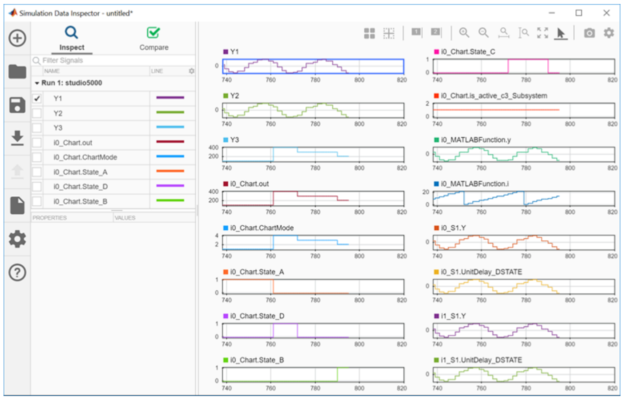
`plcrunextmode( ___ ,name_list)` runs external mode visualization and displays only the logged signals identified by the names in the `name_list`.

Examples

Visualize Logging Data

The following example uses `plcrunextmode` to connect to an OPC server and stream log data in to Simulink Data Inspector.

```
plcrunextmode ('localhost', 'studio5000', 'ext_demo1', 'plc_log_data.mat');
```



Input Arguments

opc_host — Host address
character vector

Host address of the OPC server.

Example: `'localhost'`

target_ide — Target IDE string
character vector

Specifies the name of the PLC target IDE.

Example: `'studio5000'`

mdl_name — Simulink model name
character vector

Specifies the Simulink model for which the code was generated with logging instrumentation.

Example: `'extmode_demo'`

log_file — Logging data MAT-file
character vector

Full file path of the logging data MAT-file.

Example: `'C:\plc_log_data.mat'`

`idx_list` — Index list of logged data
integer vector

The index vector specifying the indices of the logged data signals to display. This argument is optional.

Example: `[1 2 3]`

`name_list` — Name list of the logged data
vector

The name vector specifying the names of the logged data signals to display. This argument is optional.

Example: `{'Y1', 'Y2', 'i0_S1.Y'}`

See Also

`plcdispextmodedata` | `plcgeneratecode`

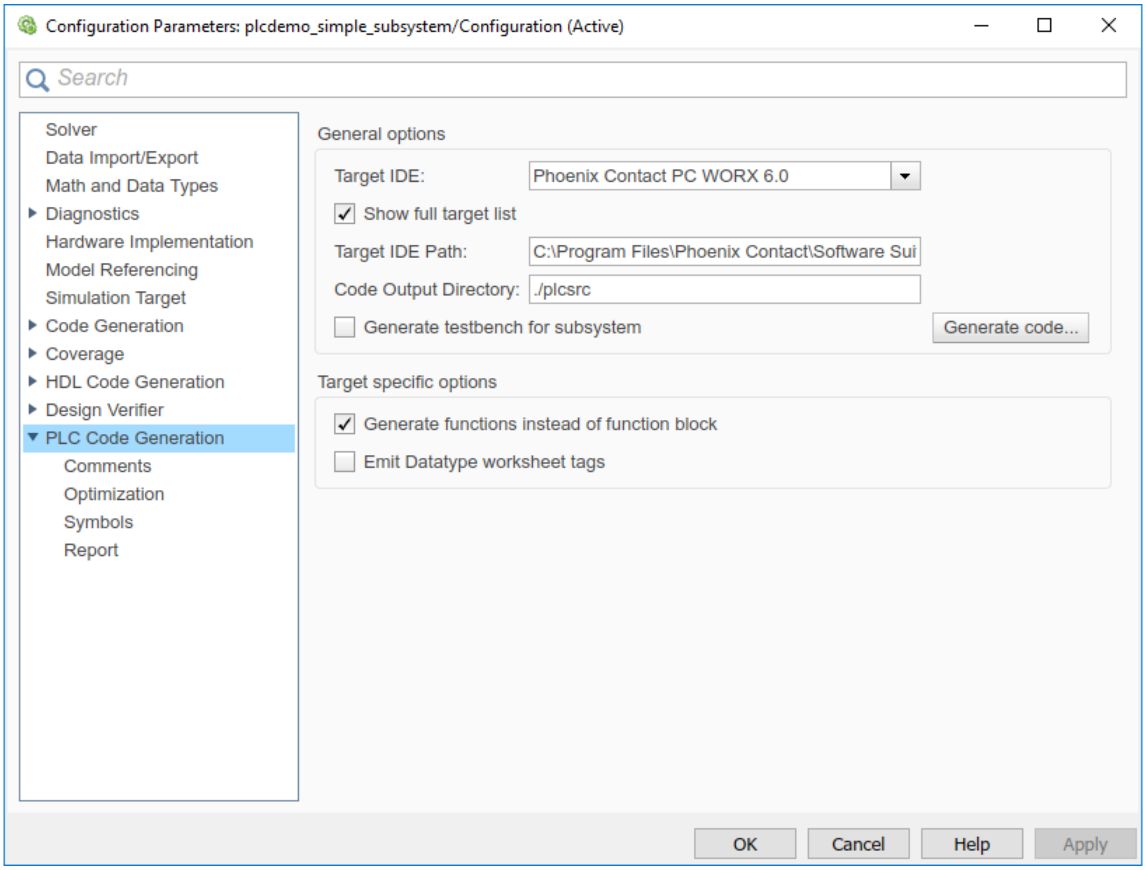
Topics

"External Mode Logging" on page 14-2
"Generate Structured Text Code with Logging Instrumentation" on page 14-3
"Use the Simulation Data Inspector to Visualize and Monitor the Logging Data" on page 14-7

Introduced in R2018a

Configuration Parameters for Simulink PLC Coder Models

PLC Coder: General


Configuration Parameters: plcdemo_simple_subsystem/Configuration (Active)

General options

Target IDE: Phoenix Contact PC WORX 6.0

☑ Show full target list

Target IDE Path: C:\Program Files\Phoenix Contact\Software Sui

Code Output Directory: ./plcsrc

☐ Generate testbench for subsystem

Target specific options

☑ Generate functions instead of function block

☐ Emit Datatype worksheet tags

In this section...

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 1-9

"Generate Structured Text from the Model Window" on page 1-17

Target IDE

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

The default **Target IDE** list shows the full set of supported targets. See “Show Full Target List” on page 13-6.

To see a reduced subset of targets, disable 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” on page 1-6.

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.

`Siemens TIA Portal: Double Precision`

Generates Structured Text code for Siemens TIA Portal. 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.

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 1-17

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" on page 1-6.

☐ 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 1-27

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 1-17

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 1-17

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 1-17

Emit Datatype 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 1-17

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.

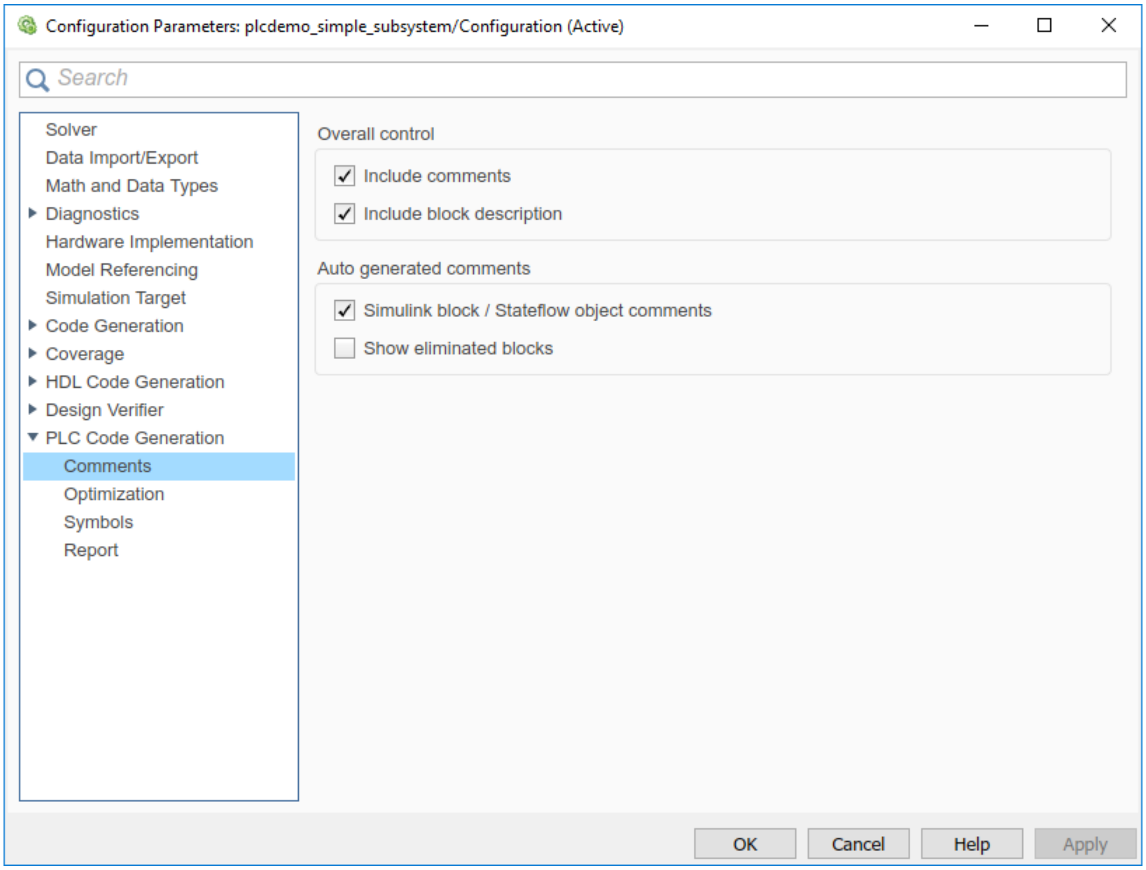
Command-Line Information

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

See Also

- "Generate Structured Text from the Model Window" on page 1-17
- "Generated Code Structure for Simple Simulink Subsystems" on page 2-2

PLC Coder: Comments


Configuration Parameters: plcdemo_simple_subsystem/Configuration (Active)

Overall control

- [✓] Include comments
- [✓] Include block description

Auto generated comments

- [✓] Simulink block / Stateflow object comments
- [ ] Show eliminated blocks

In this section...
"Comments Overview" on page 13-14
"Include Comments" on page 13-14
"Include Block Description" on page 13-15
"Simulink Block / Stateflow Object Comments" on page 13-15
"Show Eliminated Blocks" on page 13-16

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 1-17

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 Simulink Requirements 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 1-17

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'`

See Also

- “Propagate Block Descriptions to Code Comments” on page 1-22
- “Generate Structured Text from the Model Window” on page 1-17

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'`

See Also

"Generate Structured Text from the Model Window" on page 1-17

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

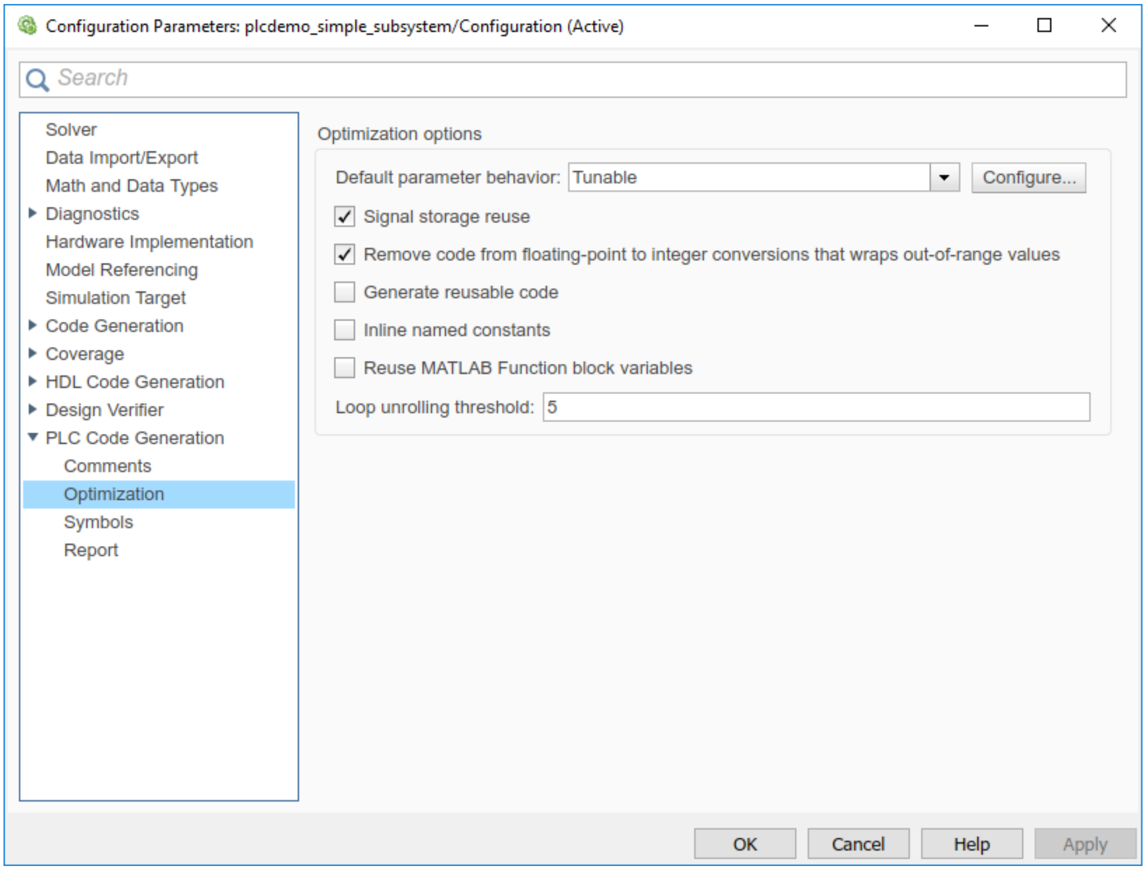
Suppresses statements.

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

See Also

“Generate Structured Text from the Model Window” on page 1-17

PLC Coder: Optimization



Optimization options

Default parameter behavior: Tunable

[✓] Signal storage reuse

[✓] Remove code from floating-point to integer conversions that wraps out-of-range values

[ ] Generate reusable code

[ ] Inline named constants

[ ] Reuse MATLAB Function block variables

Loop unrolling threshold: 5

In this section...

In this section...

Optimization Overview

Select the code generation optimization settings.

See Also

"Generate Structured Text from the Model Window" on page 1-17

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 1-17

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 1-17

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 1-17

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 1-17
- “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 1-17
- “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 1-17
- "Generated Code Structure for MATLAB Function Block" on page 2-14

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.

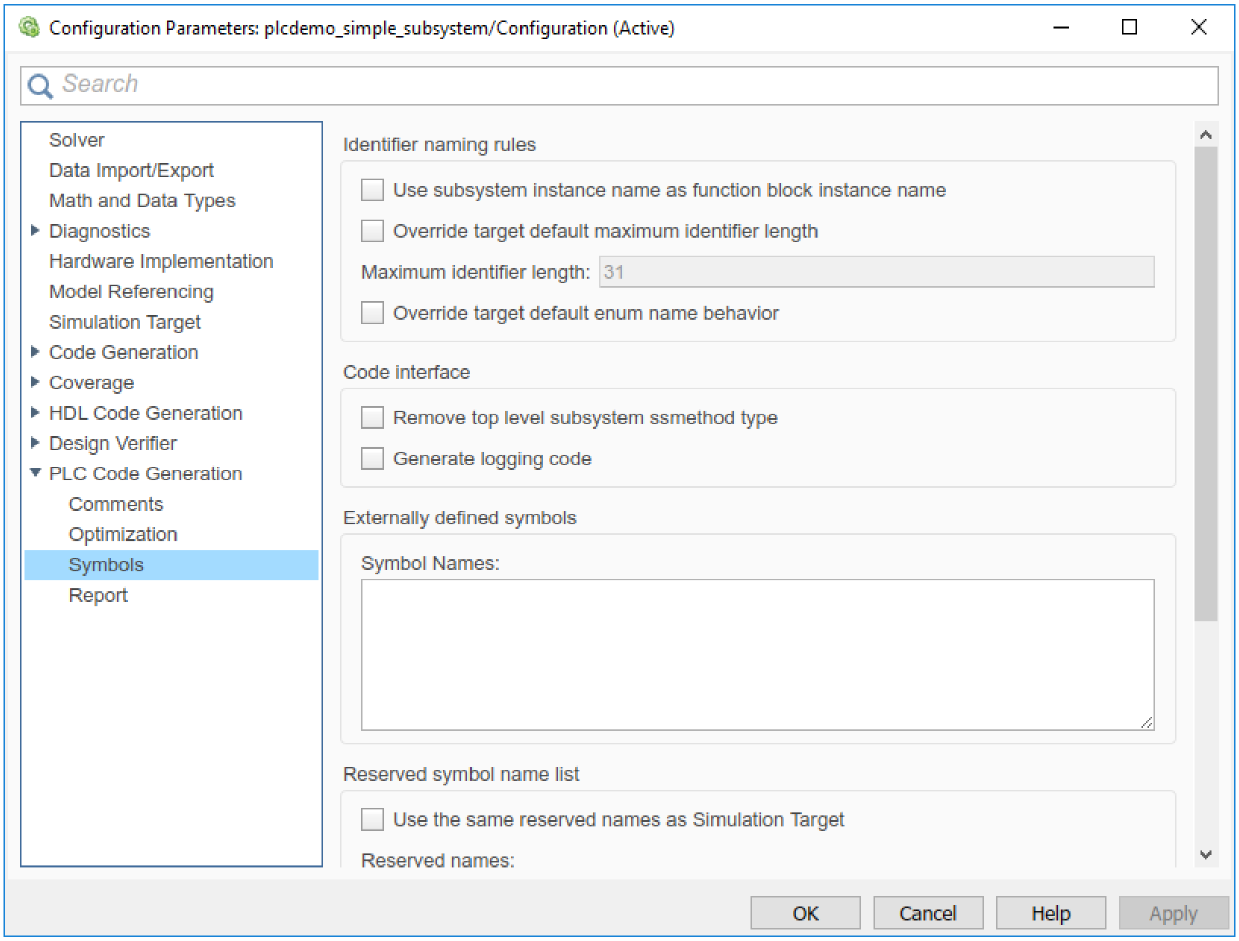
Command-Line Information

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

See Also

"Generate Structured Text from the Model Window" on page 1-17

PLC Coder: Symbols


Configuration Parameters: plcdemo_simple_subsystem/Configuration (Active)

Search

Identifier naming rules

- [ ] Use subsystem instance name as function block instance name
- [ ] Override target default maximum identifier length

Maximum identifier length: 31

- [ ] Override target default enum name behavior

Code interface

- [ ] Remove top level subsystem ssmethod type
- [ ] Generate logging code

Externally defined symbols

Symbol Names:

Reserved symbol name list

- [ ] Use the same reserved names as Simulation Target

Reserved names:

In this section...

In this section...

Symbols Overview

Select the automatically generated identifier naming rules.

See Also

"Generate Structured Text from the Model Window" on page 1-17

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 > Symbols** 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 1-17

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 > Symbols** 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 1-17

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 > Symbols** 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 1-17

Override Target Default enum Name Behavior

Use this option to enable enum names to be used as the symbols names instead of enum values. The PLC target IDE must support enum type.

This option is available on the **PLC Code Generation > Symbols** 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 1-17

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 > Symbols** 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 1-17

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** > **Symbols** 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 1-17

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 > Symbols** pane in the Configuration Parameters dialog box.

Settings

Default: off

☑ On

Uses the same reserved names as those specified in the 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 > Symbols 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 1-17

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 > Symbols** 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 1-17

Externally Defined Symbols

Specify the names of identifiers for which you want to suppress definitions. This option is available on the **PLC Code Generation > Symbols** 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 1-17
- "Integrate Externally Defined Symbols" on page 8-2
- `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 > Symbols** 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.

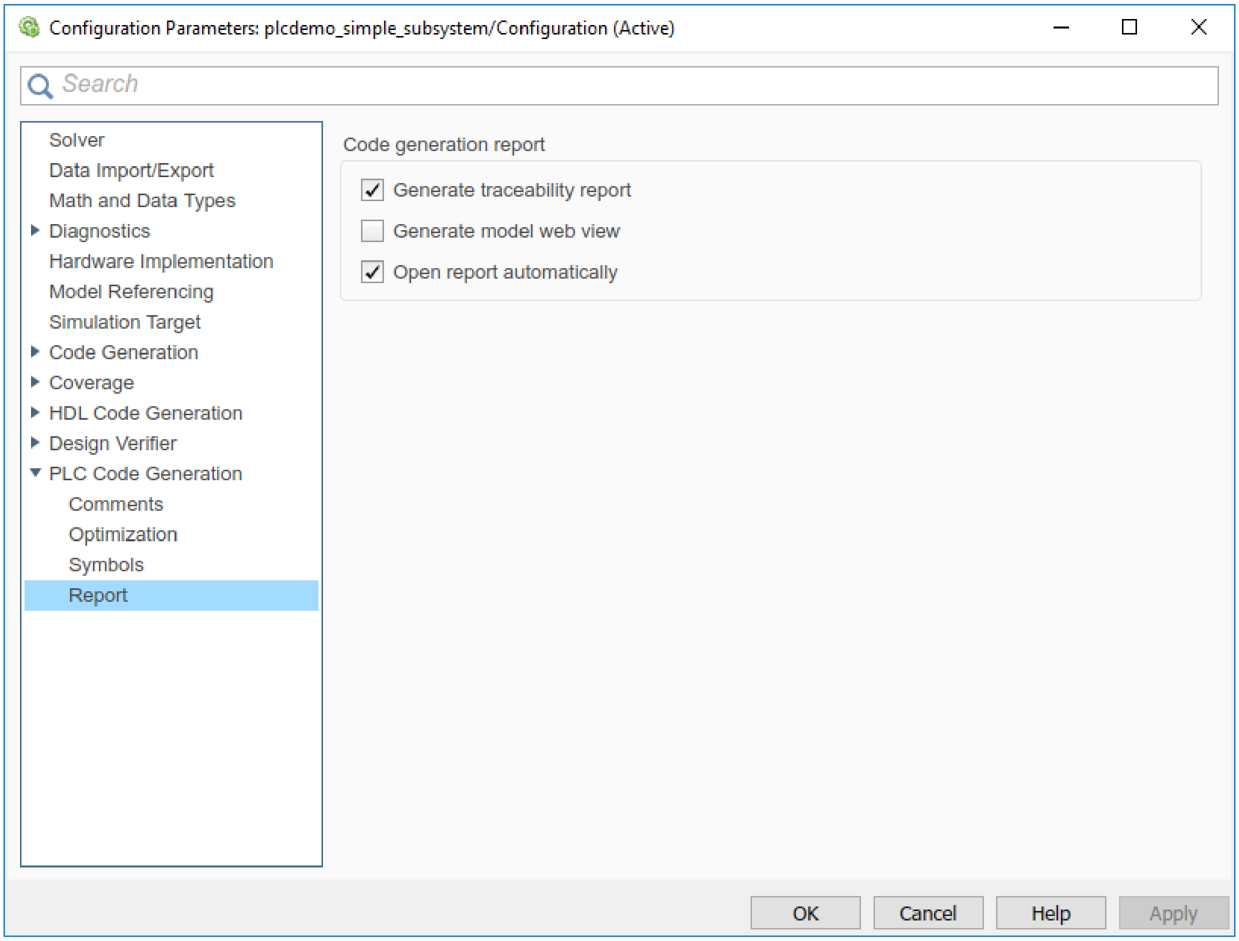
Tips

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'`

PLC Coder: Report



Code generation report

- [✓] Generate traceability report
- [ ] Generate model web view
- [✓] Open report automatically

In this section...

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 1-17

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: off

☑ 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: `'off'`

See Also

"Generate Structured Text from the Model Window" on page 1-17

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 1-17

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 1-17

External Mode

External Mode Logging

With external mode logging, you can generate code from Simulink models with logging instrumentation to collect run-time data on PLC targets. You can enable this feature by using **Generate logging code** option in the configuration parameters or by using the `PLC_GenerateLoggingCode` command-line property. The PLC target IDEs must have support for `inout` variables. You can generate logging code for one of the following target PLC IDEs:

- 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

For Rockwell Automation targets, you can set up an Open Platform Communications (OPC) server and use the Simulation Data Inspector in Simulink to visualize and monitor the logging data. The OPC Toolbox is required to run the external mode visualization.

See Also

More About

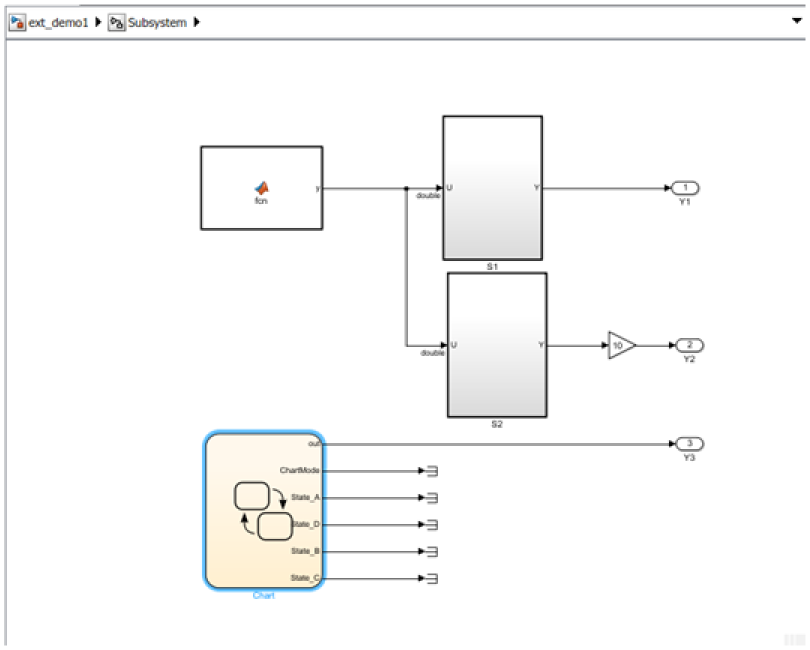
- "Generate Structured Text Code with Logging Instrumentation" on page 14-3
- "Use the Simulation Data Inspector to Visualize and Monitor the Logging Data" on page 14-7

Generate Structured Text Code with Logging Instrumentation

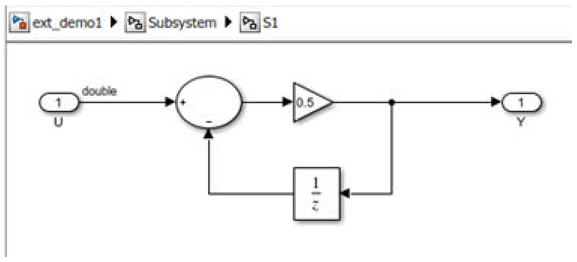
This topic assumes that you have generated Structured Text code from a Simulink model. If you have not yet done so, see “Generate Structured Text from the Model Window” on page 1-17.

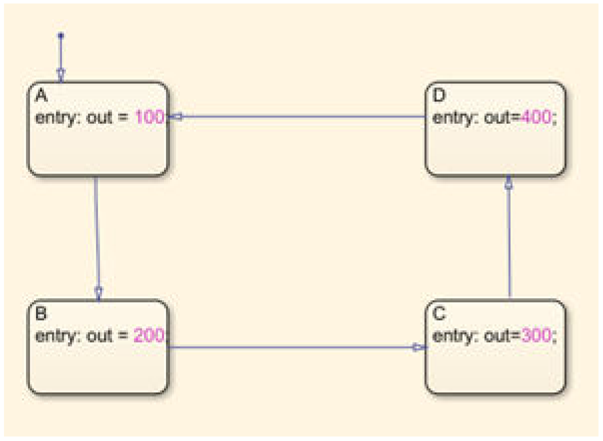
The example in this topic shows generated code for the Rockwell Automation Studio 5000 IDE. Generated code for other IDE platforms looks different.

1 Create a Simulink model `ext_demo1.slx` containing a top-level subsystem with two child subsystems `S1`, `S2`, a MATLAB Function block and a Stateflowchart.



2 The `S1`, `S2` blocks are identical and contain simple feedback loop.The Stateflow chart contains a simple state machine.

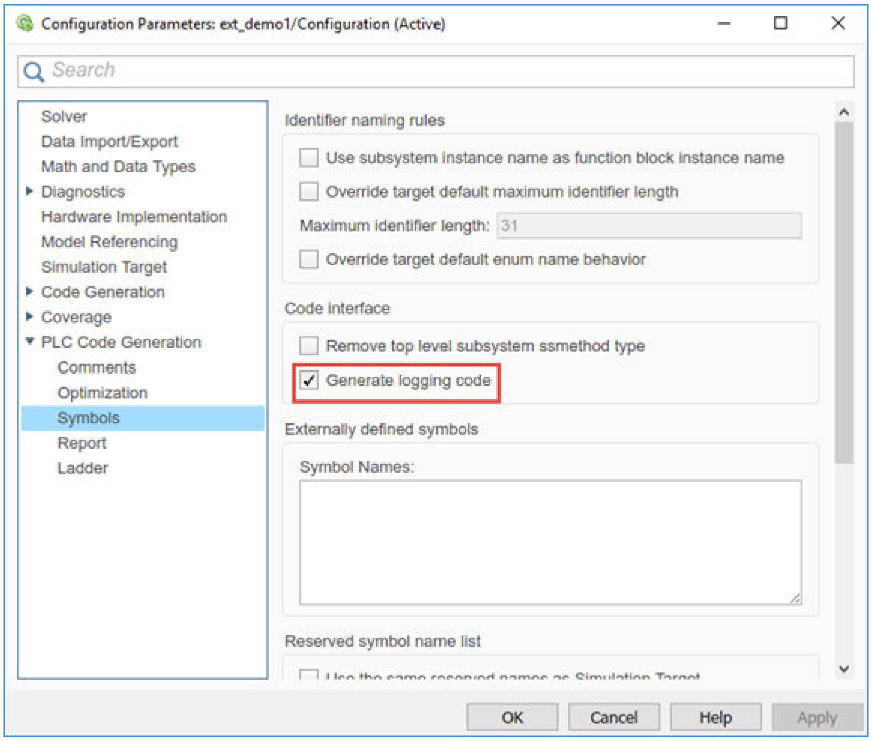




3 The MATLAB function block implements the following code:

```
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);
```

4 Select the top-level subsystem and open the configuration parameters window. On the **PLC Code Generation** pane, select the **Target IDE** as `Rockwell Studio 5000: AOI`. On the **Symbols** pane, select **Generate logging code**.



Configuration Parameters: ext_demo1/Configuration (Active)

Identifier naming rules

- [ ] Use subsystem instance name as function block instance name
- [ ] Override target default maximum identifier length

Maximum identifier length: 31

- [ ] Override target default enum name behavior

Code interface

- [ ] Remove top level subsystem ssmethod type
- [x] Generate logging code

Externally defined symbols

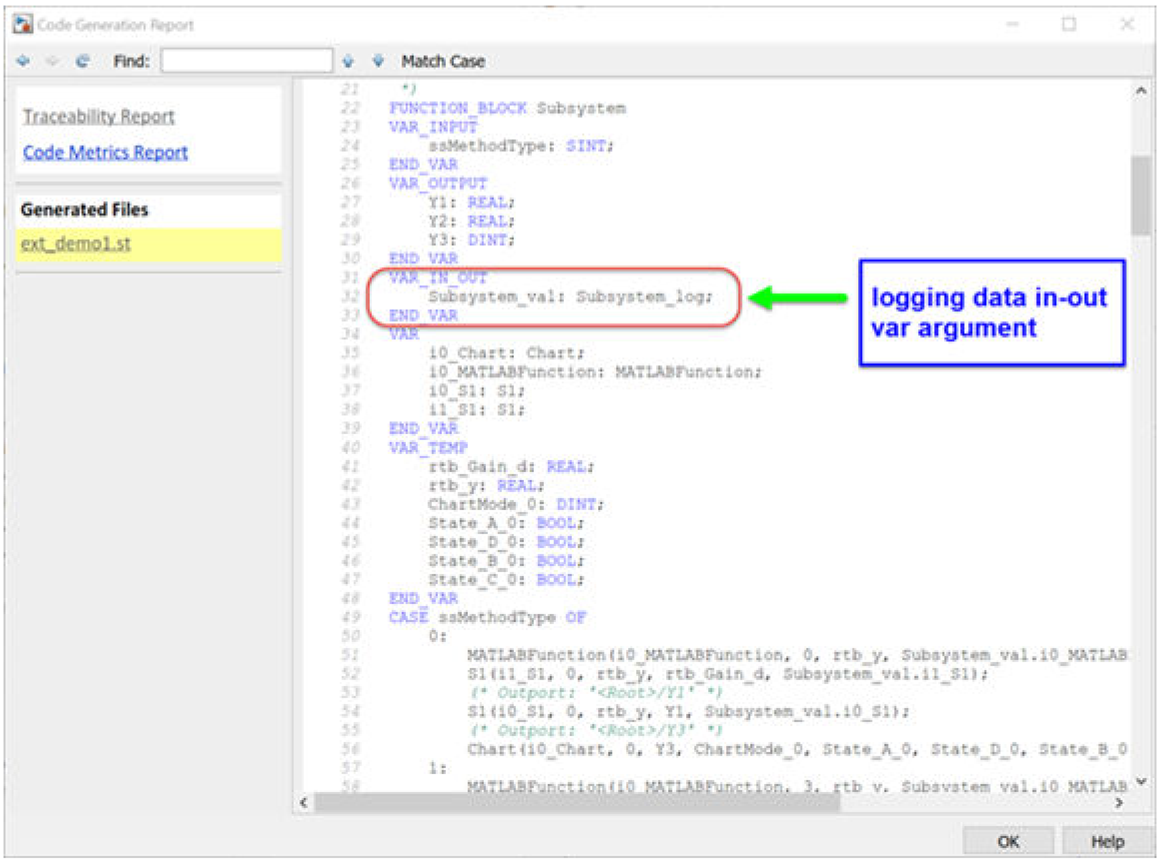
Symbol Names:

Reserved symbol name list

- [ ] Use the same reserved names as Simulation Target

5 In the model, select the top subsystem block, right-click, and choose **PLC Code>Generate Code for Subsystem**.

This operation generates L5X AOI code for the top subsystem block and the children S1, S2, MATLAB function, and Stateflow chart blocks. In the code folder, it also generates `plc_log_data.mat` which has the logging data information.



```
  *)
FUNCTION_BLOCK Subsystem
VAR_INPUT
    ssMethodType: SINT;
END_VAR
VAR_OUTPUT
    Y1: REAL;
    Y2: REAL;
    Y3: DINT;
END_VAR
VAR_IN_OUT
    Subsystem_val: Subsystem_log;
END_VAR
VAR
    i0_Chart: Chart;
    i0_MATLABFunction: MATLABFunction;
    i0_S1: S1;
    i1_S1: S1;
END_VAR
VAR_TEMP
    rtb_Gain_d: REAL;
    rtb_y: REAL;
    ChartMode_0: DINT;
    State_A_0: BOOL;
    State_D_0: BOOL;
    State_B_0: BOOL;
    State_C_0: BOOL;
END_VAR
CASE ssMethodType OF
    0:
        MATLABFunction(i0_MATLABFunction, 0, rtb_y, Subsystem_val.i0_MATLAB
        S1(i1_S1, 0, rtb_y, rtb_Gain_d, Subsystem_val.i1_S1);
        (* Outport: '<Root>/Y1' *)
        S1(i0_S1, 0, rtb_y, Y1, Subsystem_val.i0_S1);
        (* Outport: '<Root>/Y3' *)
        Chart(i0_Chart, 0, Y3, ChartMode_0, State_A_0, State_D_0, State_B_0
    1:
        MATLABFunction(i0_MATLABFunction, 3, rtb_y, Subsystem_val.i0_MATLAB
```

6 After generating the code, you can download and run the logging code from the PLC IDE.

See Also

More About

- “External Mode Logging” on page 14-2
- “Use the Simulation Data Inspector to Visualize and Monitor the Logging Data” on page 14-7

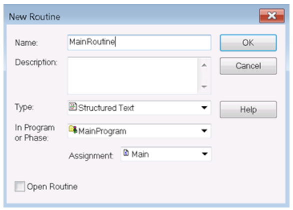
Use the Simulation Data Inspector to Visualize and Monitor the Logging Data

This workflow is supported for Rockwell Automation targets. This workflow shows you how to set up an Open Platform Communications (OPC) server and use the Simulation Data Inspector in Simulink to visualize and monitor the logging data.

Set Up and Download Code to the Studio 5000 IDE

The following procedure shows you how to create a Studio 5000 project to import the generated logging code. You can use a similar procedure to import the generated code into an existing project.

1. Start the Studio 5000 IDE and create project with the name `ext_demo1`.
2. Import the generated `ext_demo.L5X` to the Add-On Instructions tree node of the project.
3. In the `MainProgram` node, delete the ladder `MainRoutine` and create an ST `MainRoutine` node.



New Routine

Name: MainRoutine

Description:

Type: Structured Text

In Program or Phase: MainProgram

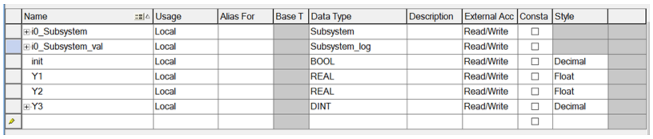
Assignment: Main

Open Routine

4. In ST `MainRoutine`, define the following tags:

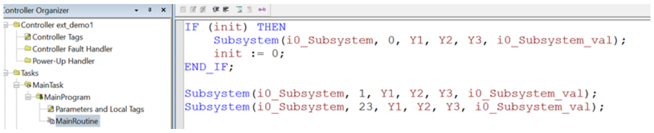
Tag Name	Tag Type
`i0_Subsystem`	`Subsystem`
`i0_Subsystem_val`	`Subsystem_log`
`Init`	`BOOL`
`Y1`	`REAL`
`Y2`	`REAL`
`Y3`	`DINT`

5 The tag definition looks like the following in Studio 5000 IDE, `i0_Subsystem tag` is the instance of the top subsystem AOI, the `i0_Subsystem_val tag` is the log data with structure type `Subsystem_log`. Set the initial value of `init` tag to 1.



Name	Usage	Alias For	Base T	Data Type	Description	External Acc	Consta	Style
⊞ i0_Subsystem	Local			Subsystem		Read/Write	☐	
⊞ i0_Subsystem_val	Local			Subsystem_log		Read/Write	☐	
init	Local			BOOL		Read/Write	☐	Decimal
Y1	Local			REAL		Read/Write	☐	Float
Y2	Local			REAL		Read/Write	☐	Float
⊞ Y3	Local			DINT		Read/Write	☐	Decimal
							☐	

6 Double-click `MainRoutine` tree node and type in the following code. 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.



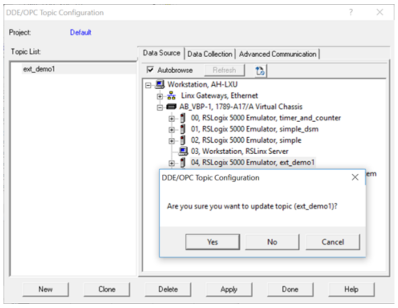
```
IF (init) THEN
    Subsystem(i0_Subsystem, 0, Y1, Y2, Y3, i0_Subsystem_val);
    init := 0;
END_IF;

Subsystem(i0_Subsystem, 1, Y1, Y2, Y3, i0_Subsystem_val);
Subsystem(i0_Subsystem, 23, Y1, Y2, Y3, i0_Subsystem_val);
```

7 Compile the project in Studio 5000 IDE, connect, and download to the PLC target.

Configure RSLinx OPC Server

1 Start RSLinx Classic Gateway, select the menu item `DDE/OPC->Topic Configuration`.

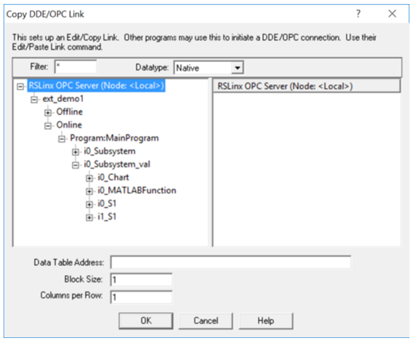
2 In the resulting pop-up dialog box, create a topic `ext_demo1` by using the `New` button. Select the target PLC from the PLC list.


DDE/OPC Topic Configuration
Project: Default
Topic List:
ext_demo1
Data Source
Data Collection
Advanced Communication
Autobrowse
Refresh
Workstation, AH-LXU
Linx Gateways, Ethernet
AB_VBP-1, 1789-A17/A Virtual Chassis
00, RSLogix 5000 Emulator, timer_and_counter
01, RSLogix 5000 Emulator, simple_dsm
02, RSLogix 5000 Emulator, simple
03, Workstation, RSLinx Server
04, RSLogix 5000 Emulator, ext_demo1
DDE/OPC Topic Configuration
Yes
No
Cancel
New
Clone
Delete
Apply
Done
Help

Are you sure you want to update topic (ext_demo1)?

3 Click `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.



Copy DDE/OPC Link

This sets up an Edit/Copy Link. Other programs may use this to initiate a DDE/OPC connection. Use their Edit/Paste Link command.

Filter: *
Datatype: Native

```
RSLinx OPC Server (Node: <Local>)
  ext_demo1
    Offline
    Online
      Program:MainProgram
        i0_Subsystem
        i0_Subsystem_val
          i0_Chart
          i0_MATLABFunction
          i0_S1
          i1_S1
```

RSLinx OPC Server (Node: <Local>)

Data Table Address:
Block Size: 1
Columns per Row: 1

Use PLC External Mode Commands to Stream and Display Live Log Data

After the RSLinx OPC Server is configured, you can use the PLC external mode commands to connect to the server, stream, and display live logging data on the Simulink Data Inspector. The log data information is in the `plc_log_data.mat` file which can be found in `plcsrc` folder. You can use the `plcdispextmodedata` function to display the contents of the MAT-file. In the MATLAB command, type:

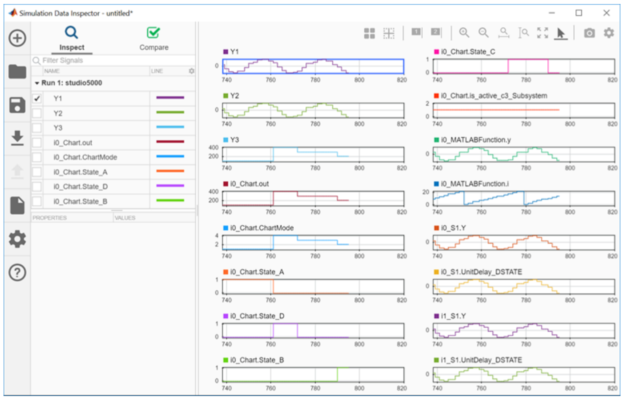
```
>>cd plcsrc
>>plcdispextmodedata plc_log_data.mat
```

```
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 are named using the dot notation to represent the function block instances that own the data. The index and name of the log data can be used with the `plcrunextmode` command to specify a subset of log data for streaming and visualization.

Use the `plcrunextmode` function to connect to the OPC server and stream log data. For example, executing `plcrunextmode ('localhost', 'studio5000', 'ext_demo1', 'plc_log_data.mat');` command streams live log data for the example model in to Simulink Data Inspector.



The `plcrunextmode` command continues to run and stream log data. To exit streaming, type Ctrl-C in MATLAB to stop.

See Also

`plcdispextmodedata` | `plcrunextmode`

More About

- "External Mode Logging" on page 14-2
- "Generate Structured Text Code with Logging Instrumentation" on page 14-3