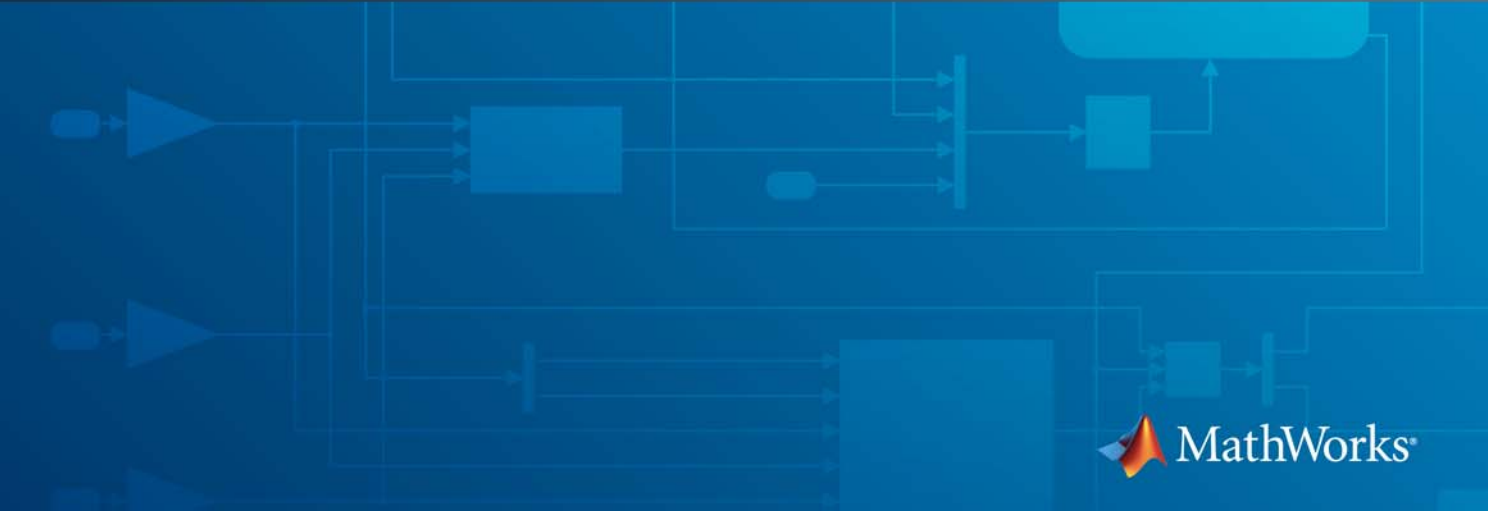
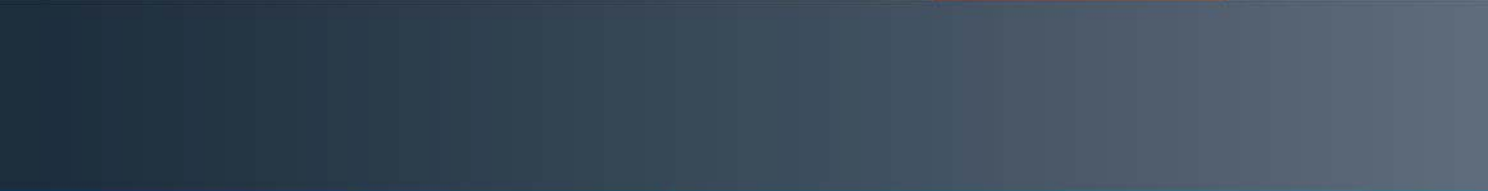
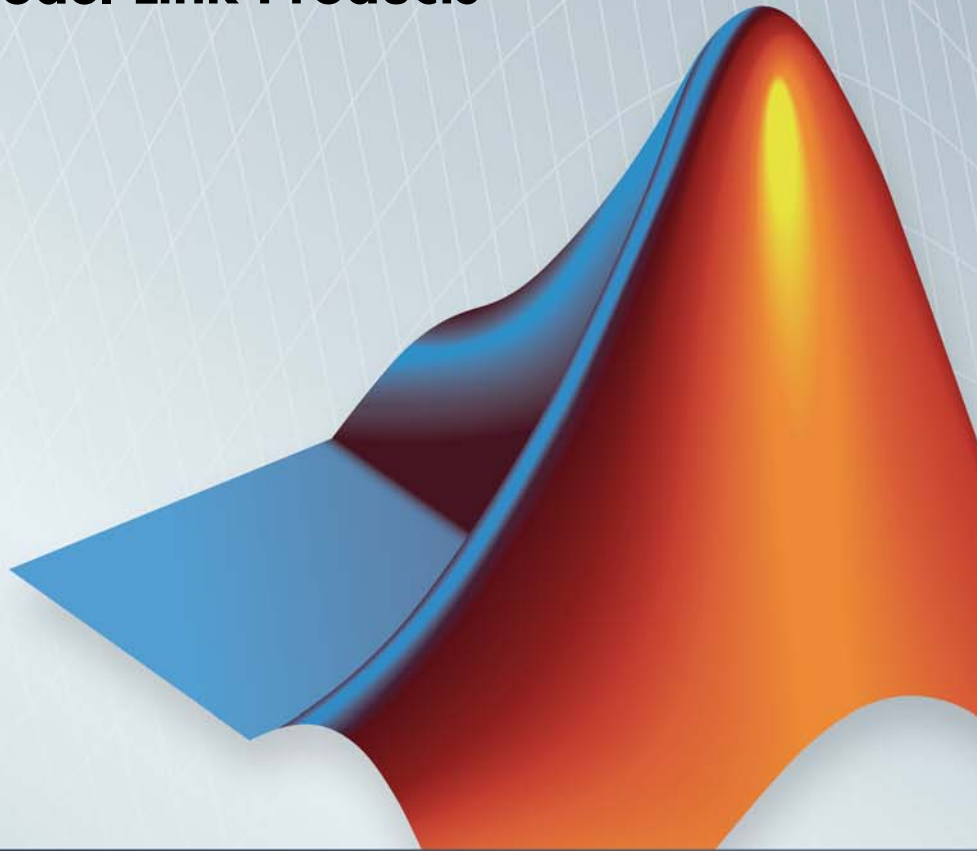


Polyspace® Model Link Products

User's Guide

R2013a



How to Contact MathWorks



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



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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

Polyspace® Model Link Products User's Guide

© COPYRIGHT 1999–2013 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

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

Patents

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

Revision History

| | | |
|----------------|-------------|--|
| March 2009 | Online Only | Revised for Version 5.3 (Release 2009a) |
| September 2009 | Online Only | Revised for Version 5.4 (Release 2009b) |
| March 2010 | Online Only | Revised for Version 5.5 (Release 2010a) |
| September 2010 | Online Only | Revised for Version 5.6 (Release 2010b) |
| April 2011 | Online Only | Revised for Version 5.7 (Release 2011a) |
| September 2011 | Online Only | Revised for Version 5.8 (Release 2011b) |
| March 2012 | Online Only | Revised for Version 5.9 (Release 2012a) |
| September 2012 | Online Only | Revised for Version 5.10 (Release 2012b) |
| March 2013 | Online Only | Revised for Version 5.11 (Release 2013a) |

Getting Started with Model Link Products

1

| | |
|--|------|
| Product Overview | 1-2 |
| Polyspace Model Link SL | 1-2 |
| Polyspace Model Link TL | 1-2 |
| | |
| Install Polyspace Model Link Products | 1-3 |
| | |
| Code Verification Approach | 1-4 |
| | |
| Verify Code from a Simple Model | 1-6 |
| Create Simulink Model and Generate Code | 1-6 |
| Run Polyspace Verification | 1-9 |
| View Results in Polyspace Verification Environment | 1-10 |
| Trace Error to Simulink Model | 1-12 |
| Specify Signal Ranges | 1-13 |
| Verify Updated Model | 1-15 |

Configure Model for Code Verification

2

| | |
|---|-----|
| Overview of Model Configuration for Code Generation and Verification | 2-2 |
| | |
| Configure Embedded IDE Link Model for Code Verification | 2-3 |
| | |
| Recommended Polyspace Settings for Code Verification | 2-5 |
| | |
| Check Simulink Model Settings | 2-7 |

| | |
|--|------|
| Specify Signal Ranges | 2-9 |
| Specify Signal Range through Source Block Parameters .. | 2-9 |
| Specify Signal Range through Base Workspace | 2-11 |
| | |
| Annotate Blocks with Known Checks or Coding-Rule Violations | 2-14 |
| | |
| Code Annotation for Justifying Polyspace Checks | 2-17 |

Configure Code Verification Options

3

| | |
|---|------|
| Overview of Polyspace Configuration | 3-2 |
| | |
| Include Handwritten Code in Verification | 3-3 |
| | |
| Specify Client or Server Verification | 3-5 |
| | |
| Configure Data Range Settings | 3-6 |
| | |
| Configure Verification of Model Reference Code | 3-9 |
| | |
| Specify Location of Results | 3-10 |
| | |
| Check Coding Rules Compliance | 3-11 |
| | |
| Configure Polyspace Verification Options | 3-14 |
| | |
| Configure Polyspace Project Properties | 3-17 |
| | |
| Create a Polyspace Configuration File Template | 3-18 |
| | |
| Specify Header Files for Target Compiler | 3-21 |

| | |
|---|-------------|
| Open Polyspace Project Manager Automatically | 3-22 |
| Remove Polyspace Options From Simulink Model | 3-24 |
| Main Generation for Model Verification | 3-25 |
| Polyspace Model Link SL Considerations | 3-27 |
| Overview | 3-27 |
| Subsystems | 3-27 |
| Default Options | 3-27 |
| Data Range Specification | 3-28 |
| Recommended Polyspace Options for Verifying Generated Code | 3-29 |
| Hardware Mapping Between Simulink and Polyspace | 3-33 |
| Polyspace Model Link TL Considerations | 3-34 |
| Overview | 3-34 |
| Subsystems | 3-34 |
| Default Options | 3-34 |
| Data Range Specification | 3-35 |
| Lookup Tables | 3-36 |
| Code Generation Options | 3-36 |

Run Code Verification

4

| | |
|---|-------------|
| Specify Type of Analysis to Perform | 4-2 |
| Run Verification with Polyspace Model Link SL Software | 4-5 |
| Run Verification with Polyspace Model Link TL Software | 4-8 |
| Monitor Verification Progress | 4-10 |
| Client Verifications | 4-10 |
| Server Verifications | 4-10 |

| | |
|---|-------------|
| Code Generation and Verification with Configured Model | 4-12 |
| MATLAB Functions For Polyspace Batch Runs | 4-14 |
| Archive Files for Polyspace Verification | 4-15 |
| Template File in <i>MATLAB Installation folder\polyspace\</i> | 4-15 |
| Files Used in Model Folder | 4-15 |
| Auto-Generated Files in Model Folder | 4-16 |

Review Verification Results

| | | |
|----------|--|------------|
| 5 | | |
| | View Results in Polyspace Verification Environment .. | 5-2 |
| | Identify Errors in Simulink Models | 5-5 |

Functions

6 |

Getting Started with Model Link Products

- “Product Overview” on page 1-2
- “Install Polyspace Model Link Products” on page 1-3
- “Code Verification Approach” on page 1-4
- “Verify Code from a Simple Model” on page 1-6

Product Overview

| In this section... |
|---|
| “Polyspace® Model Link™ SL” on page 1-2 |
| “Polyspace® Model Link™ TL” on page 1-2 |

Polyspace Model Link SL

Polyspace® Model Link™ SL extends Polyspace Client™ for C/C++ and Polyspace Server™ for C/C++ with tools that let you trace Polyspace results from generated C code directly to your Simulink® model. As a result, you can identify parts of the model that do not generate code with run-time errors, and fix design problems that cause run-time errors. With Polyspace Model Link SL, you work in the Simulink environment to verify C code generated by Embedded Coder® software. You can verify a mix of generated and hand-written code before it is compiled.

Polyspace Model Link TL

Polyspace Model Link TL extends Polyspace Client for C/C++ and Polyspace Server for C/C++ with tools that let you verify C code generated by TargetLink® and trace Polyspace results from the generated C code to your model. As a result, you can identify parts of the model that do not generate code with run-time errors, and fix design problems that cause run-time errors. With Polyspace Model Link TL software, you work in the Simulink environment to verify C code generated by TargetLink. You can verify a mix of generated and hand-written code before it is compiled.

Install Polyspace Model Link Products

For MATLAB® versions R2011a and later:

- 1 Start MATLAB.
- 2 Change your working folder to:

```
Polyspace_Install\polyspace\toolbox\pslink\pslink
```

- 3 To install Polyspace Model Link products, in the Command Window, run:

```
pslinksetup('install')
```

To uninstall Polyspace Model Link products, in the Command Window, run:

```
pslinksetup('uninstall')
```

Alternatively:

- 1 Open a DOS command window.
- 2 Change your current folder to *matlabroot*\bin:

```
cd matlabroot\bin
```

matlabroot is the installation folder for your MATLAB software.

- 3 From *matlabroot*\bin, install Polyspace Model Link products:

```
matlab -r  
Polyspace_Install\polyspace\toolbox\pslink\pslink\pslinksetup('install')
```

Code Verification Approach

With Embedded Coder or dSPACE® TargetLink software, you can generate code from models in the Simulink Model-Based Design environment. Using Polyspace Model Link SL and Polyspace Model Link TL software, you can apply Polyspace verification to the generated code within the Simulink environment. The software detects run-time errors in the generated code and helps you to locate and fix model faults.

Note The documentation describes steps for the Polyspace Model Link SL product, but states differences between the Polyspace Model Link SL and Polyspace Model Link TL products where relevant.

Use the following approach:

- 1 Configure your Simulink model and generate code. See “Overview of Model Configuration for Code Generation and Verification” on page 2-2.
- 2 Configure Polyspace verification options. See “Overview of Polyspace Configuration” on page 3-2

Note After generating code, you can run a verification without manual configuration. By default, Polyspace automatically creates a project and extracts required information from your model. However, you can also customize your verification. See “Configure Polyspace Verification Options” on page 3-14.

- 3 Run Polyspace verification. See:
 - “Run Verification with Polyspace® Model Link™ SL Software” on page 4-5
 - “Run Verification with Polyspace® Model Link™ TL Software” on page 4-8
- 4 View results, analyze errors, locate and fix model faults. See “View Results in Polyspace Verification Environment” on page 5-2.

The software allows direct navigation from a run-time error in the generated code to the corresponding Simulink block or Stateflow® chart in the Simulink model. See “Identify Errors in Simulink Models” on page 5-5.

Verify Code from a Simple Model

In this section...

“Create Simulink Model and Generate Code” on page 1-6

“Run Polyspace Verification” on page 1-9

“View Results in Polyspace Verification Environment” on page 1-10

“Trace Error to Simulink Model” on page 1-12

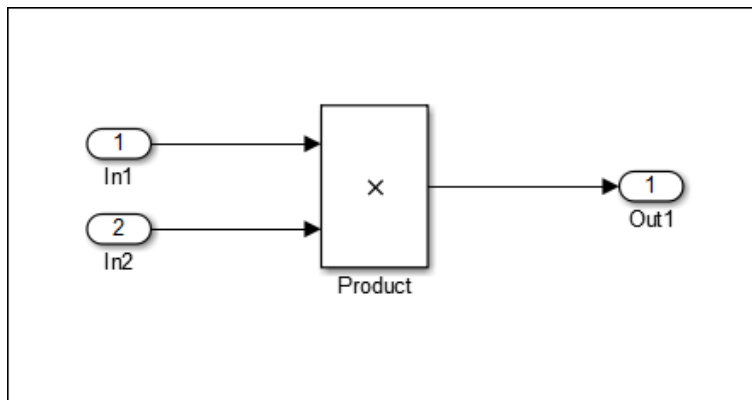
“Specify Signal Ranges” on page 1-13

“Verify Updated Model” on page 1-15

Create Simulink Model and Generate Code

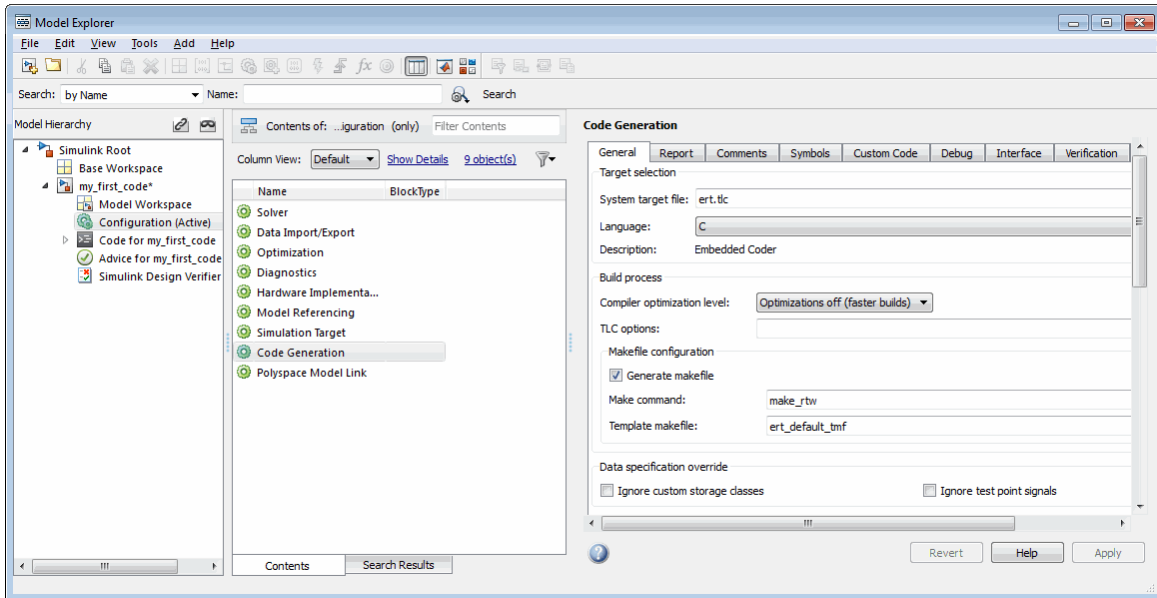
To create a simple Simulink model and generate code:

- 1 Open MATLAB. Then start Simulink software.
- 2 Construct the following model.



- 3 Select **File > Save**. Then name the model `my_first_code`.
- 4 Select **Tools > Model Explorer**. The Model Explorer opens.
- 5 From the **Model Hierarchy** tree, expand the node `my_first_code`.

- 6 Select **Configuration > Code Generation**, which displays Code Generation configuration parameters.



- 7 Select the **General** tab, and then set the **System target file** to `ert.tlc` (Embedded Coder).
- 8 Select the **Report** tab.
- 9 Select **Create code-generation report**, and then select **Code-to-model** navigation.
- 10 Select the **Templates** tab.
- 11 In the **Custom templates** section, clear the check box **Generate an example main program**.
- 12 Select the **Interface** tab.
- 13 In the **Code interface** section, select the **Suppress error status in real-time model data structure** check box.

- 14** Click **Apply** in the lower-right corner of the window.
- 15** Select **Configuration > Solver**, which displays Solver configuration parameters.
- 16** In the **Solver options** section, set the solver **Type** to **Fixed-step**. Then, set the **Solver** to **discrete (no continuous states)**.
- 17** Click **Apply**.
- 18** Select **Configuration > Optimization**, which displays Optimization configuration parameters. Then:
 - On the **General** tab, in the **Data initialization** section, select the **Remove root level I/O zero initialization** check box.
 - On the **General** tab, clear the **Use memset to initialize floats and doubles to 0.0** check box
 - On the **Signals and Parameters** tab, in the **Simulation and code generation** section, select the **Inline parameters** check box.
- 19** Click **Apply**.
- 20** To generate code, from the Simulink model window, select **Code > C/C++ Code > Build Model**.
- 21** Save your Simulink model.

Run Polyspace Verification

To start the Polyspace verification:

- 1 From the Simulink model window, select **Code > Polyspace > Polyspace for Embedded Coder > Verify Generated Code**.

The verification starts, and you see messages in the MATLAB Command Window.

```
### Starting Polyspace verification for Embedded Coder
### Creating results folder results_my_first_code for system my_first_code
### Parameters used for code verification:
System                : my_first_code
Results Folder       : C:\results_my_first_code
Additional Files     : 0
Remote               : 1
Verifier settings    : PrjConfig
DRS input mode       : DesignMinMax
DRS parameter mode   : None
DRS output mode      : None
Model Reference Depth : Current model only
Model by Model       : 0
### Creating results folder C:\results_my_first_code\my_first_code for system my_first_code
### Writing DRS table in C:\results_my_first_code\my_first_code\my_first_code_drs.txt
### Writing link data in C:\results_my_first_code\my_first_code\linksData.xml
### Writing model version in C:\results_my_first_code\my_first_code\code_generator_used
### Computing code verification options
```

- 2 Follow the progress of the verification in the MATLAB Command window.

Note Verification of this model takes about a minute. A 3,000 block model will take approximately one hour to verify, or about 15 minutes for each 2,000 lines of generated code.

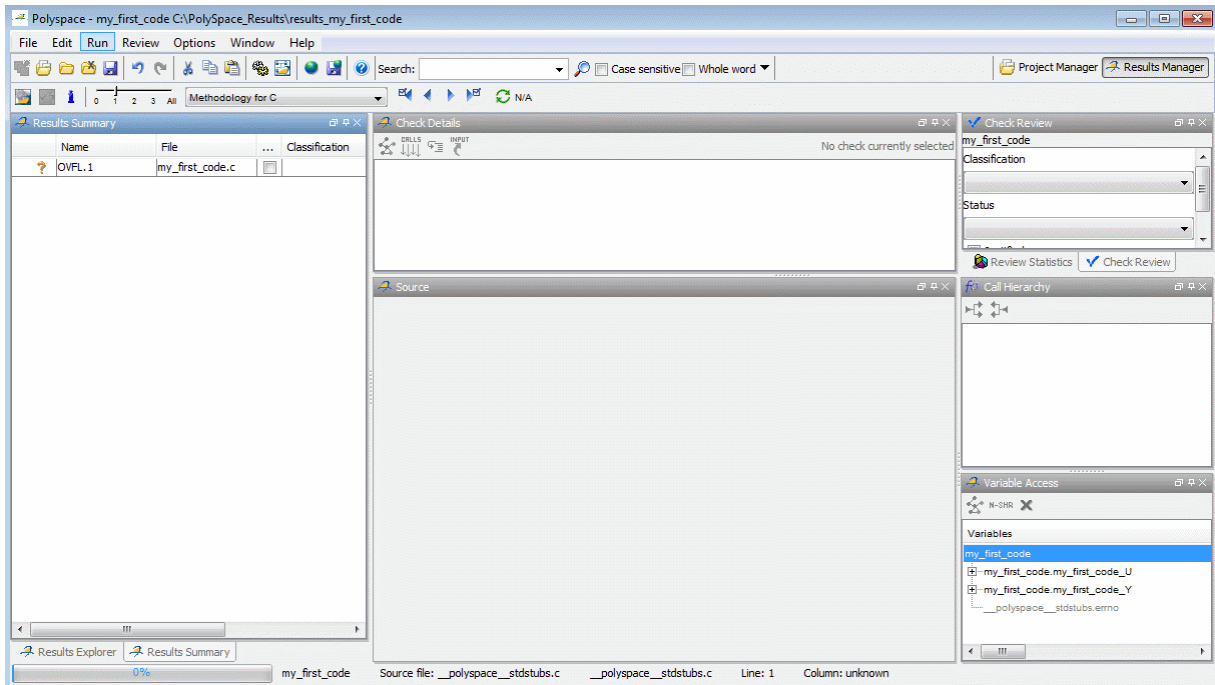
View Results in Polyspace Verification Environment

When the verification is complete, you can view the results using the Results Manager perspective of the Polyspace verification environment.

To view your results:

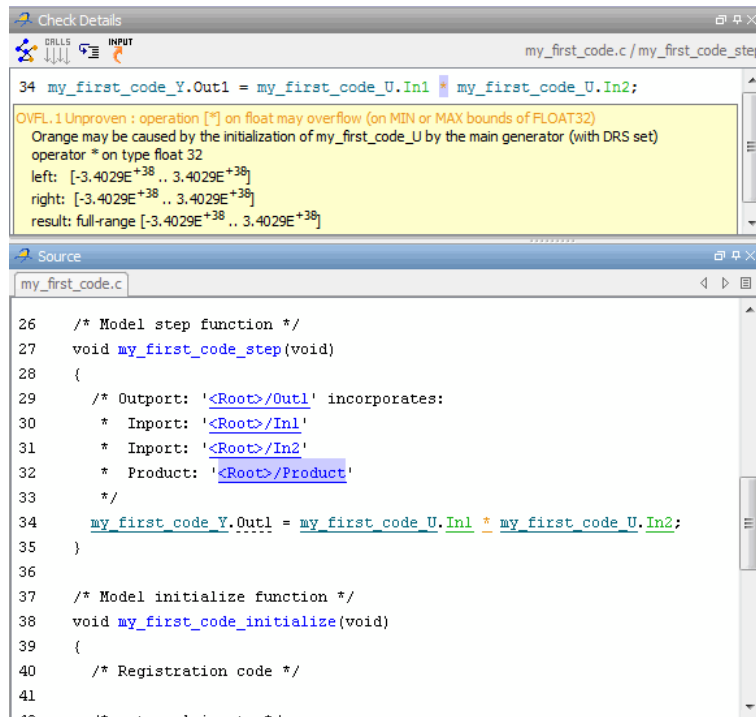
- 1 From the Simulink model window, select **Code > Polyspace > Open Results > For Generated Code**.

After a few seconds, the Results Manager perspective of the Polyspace verification environment opens.



- 2 On the **Results Summary** tab, select the orange check.

The **Check Details** pane shows information about the orange check, and the **Source** pane shows the source code containing the orange check.



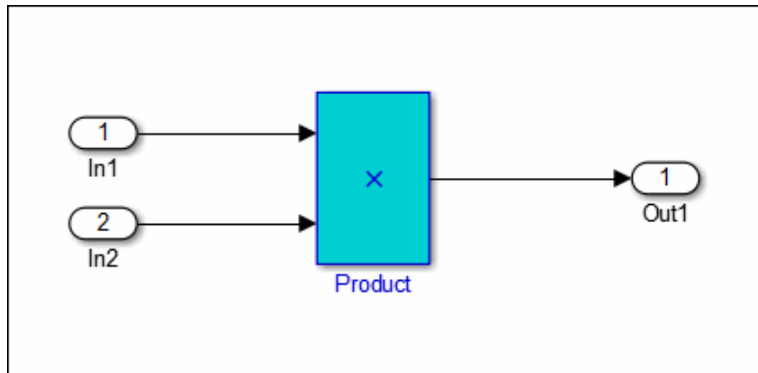
This orange check shows a potential overflow issue when multiplying the signals from the imports In1 and In2. Polyspace software assumes that the signal values are full range, and the multiplication of the two signals may result in an overflow.

Trace Error to Simulink Model

To fix this overflow issue, you must return to the Simulink model.

To trace the error to your model:

- 1 Click the blue underlined link ([<Root>/Product](#)) immediately before the check in the **Source** pane. The Simulink model opens, highlighting the block with the error.



- 2 Examine the model. The highlighted block multiplies two full-range signals, which could result in an overflow.

The verification has identified a potential bug. This could be a flaw in:

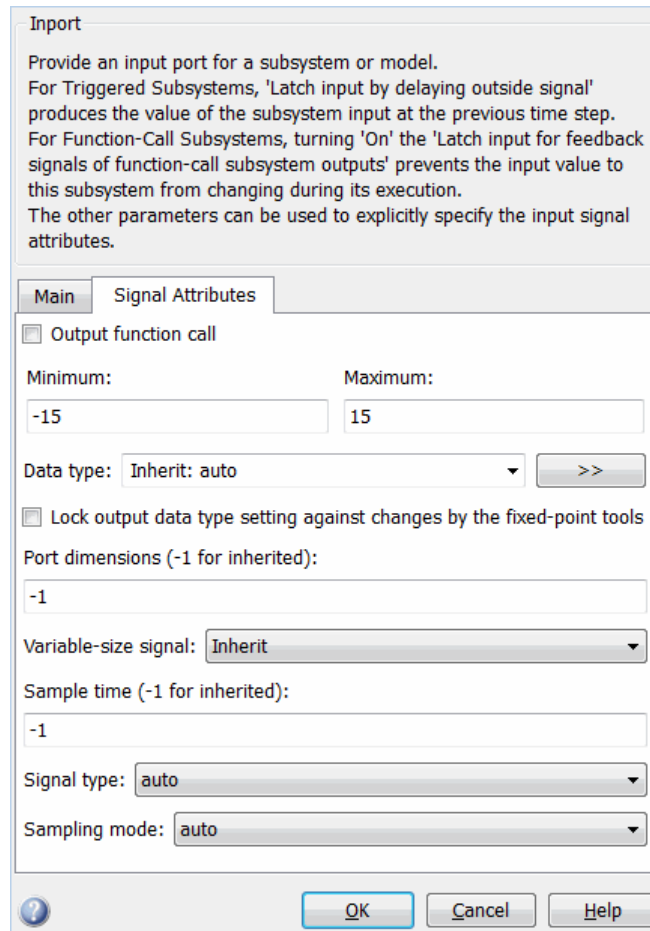
- **Design** — If the model should be robust for the full signal range, then the issue is a design flaw. In this case, you must change the model to accommodate the full signal range. For example, you could saturate the output of the previous block, or bound the signal with a Switch block.
- **Specifications** — If the model is supposed to work for specific input ranges, you can provide these ranges using block parameters or the base workspace. The next verification will read these ranges from the model, and the check will be green.

Specify Signal Ranges

If you constrain the signals in your Simulink model to specified ranges, Polyspace software automatically applies these constraints during verification of the generated code. The OVFL check will then be green in the verification results.

To specify signal ranges using source block parameters:

- 1** Double-click the In1 source block in your model. The Source Block Parameters dialog box opens.
- 2** Select the **Signal Attributes** tab.
- 3** Set the **Minimum** value for the signal to -15.
- 4** Set the **Maximum** value for the signal to 15.



- 5** Click **OK**.
- 6** Repeat steps 1–6 for the In2 block.
- 7** Save your model as `my_first_code_bounded`.

Verify Updated Model

After changing the model, you must regenerate code and run verification again.

To regenerate code and rerun the verification:

- 1 From the Simulink model, select **Code > C/C++ Code > Build Model**.

The software generates code for the updated model.

- 2 Select **Code > Polyspace > Polyspace for Embedded Coder > Verify Generated Code**.

The software verifies the generated code.

- 3 Select **Code > Polyspace > Open Results**.

Verification results open in the Polyspace verification environment.

- 4 In the Results Manager perspective, select the **Results Explorer** tab.

The screenshot displays two windows from the Polyspace verification environment. The top window, titled 'Check Details', shows a yellow error message: 'OVFL:1 Operation [*] on float does not overflow in FLOAT64 range'. It details the operation as 'operator * on type float 64', with left and right operands both in the range $[-1.5E+1 .. 1.5E+1]$ and a result in the range $[-2.2501E+2 .. 2.2501E+2]$. The bottom window, titled 'Source', shows the C code for 'my_first_code_bounded.c'. The code defines a function 'my_first_code_bounded_step' that takes a void pointer and performs a multiplication of two floating-point variables, 'my_first_code_bounded U.In1' and 'my_first_code_bounded U.In2', to produce 'my_first_code_bounded Y.Out1'.

```

34 my_first_code_bounded_Y.Out1 = my_first_code_bounded_U.In1
OVFL:1 Operation [*] on float does not overflow in FLOAT64 range
operator * on type float 64
left: [-1.5E+1 .. 1.5E+1]
right: [-1.5E+1 .. 1.5E+1]
result: [-2.2501E+2 .. 2.2501E+2]

Source
my_first_code_bounded.c
27 void my_first_code_bounded_step(void)
28 {
29 /* Output: '<Root>/Out1' incorporates:
30 * Import: '<Root>/In1'
31 * Import: '<Root>/In2'
32 * Product: '<Root>/Product'
33 */
34 my_first_code_bounded_Y.Out1 = my_first_code_bounded_U.In1
35 my_first_code_bounded_U.In2;
36 }

```

The OVFL check is now green. Polyspace verification shows that no run-time errors are present in the model.

Configure Model for Code Verification

- “Overview of Model Configuration for Code Generation and Verification” on page 2-2
- “Configure Embedded IDE Link Model for Code Verification” on page 2-3
- “Recommended Polyspace Settings for Code Verification” on page 2-5
- “Check Simulink Model Settings” on page 2-7
- “Specify Signal Ranges” on page 2-9
- “Annotate Blocks with Known Checks or Coding-Rule Violations” on page 2-14
- “Code Annotation for Justifying Polyspace Checks” on page 2-17

Overview of Model Configuration for Code Generation and Verification

To facilitate Polyspace code verification and the review of results:

- There are certain settings that you should apply to your model before generating code. See “Recommended Polyspace Settings for Code Verification” on page 2-5.
- Polyspace Model Link SL software allows you to check your model configuration before starting a verification. See “Check Simulink Model Settings” on page 2-7
- You can constrain signals in your model to lie within specified ranges. See “Specify Signal Ranges” on page 2-9.
- You can highlight blocks that you know contain checks or coding rule violations. See “Annotate Blocks with Known Checks or Coding-Rule Violations” on page 2-14.

Configure Embedded IDE Link Model for Code Verification

To configure a Simulink model for code generation and verification:

- 1 Open Model Explorer.
- 2 From the Model Hierarchy tree, expand the model node.
- 3 Select **Configuration > Code Generation**, which displays Code Generation configuration parameters.
- 4 Select the **General** tab, and then set the **System target file** to `ert.tlc` (Embedded Coder).
- 5 In the **Report** tab, select:
 - **Create code-generation report**
 - **Code-to-model** navigation.
- 6 In the **Templates** tab, clear **Generate an example main program**.
- 7 In the **Interface** tab, select **Suppress error status in real-time model data structure**.
- 8 Click **Apply**.
- 9 Select **Configuration > Solver**, which displays Solver configuration parameters.
- 10 In the **Solver options** section, set:
 - **Type** to Fixed-step.
 - **Solver** to discrete (no continuous states).
- 11 Click **Apply**.
- 12 Select **Configuration > Optimization**, which displays Optimization configuration parameters. Then:
 - On the **General** tab, in the **Data initialization** section, select the **Remove root level I/O zero initialization** check box.

- On the **General** tab, clear the **Use memset to initialize floats and doubles to 0.0** check box
- On the **Signals and Parameters** tab, in the **Simulation and code generation** section, select the **Inline parameters** check box.

13 Save your model.

Recommended Polyspace Settings for Code Verification

For Polyspace verification, you should configure your model with the following settings before generating code.

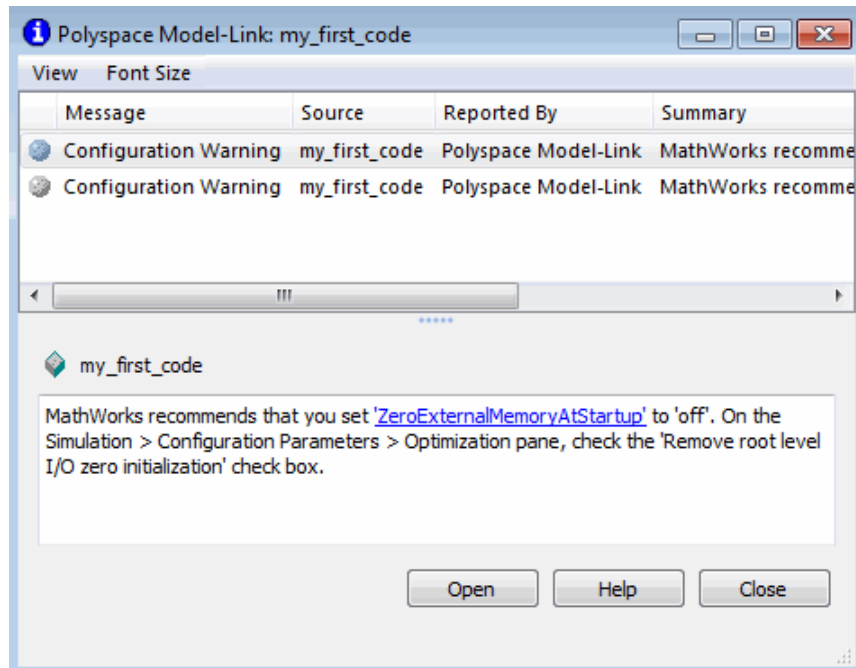
| Parameter | Recommended value for Polyspace verification | How you specify value in Configuration Parameters dialog box | If you do not use recommended value, Polyspace Model Link SL generates ... |
|-----------------------|--|--|--|
| InitFltsAndDblsToZero | 'on' | Select check box Optimization > Use memset to initialize floats and doubles to 0.0 | Warning |
| InlineParams | 'on' | Select check box Optimization > Signals and Parameters > Inline parameters | Warning |
| MatFileLogging | 'off' | Clear check box Code Generation > Interface > MAT-file logging | Warning |
| Solver | 'FixedStepDiscrete' | Select discrete (no continuous states) from Solver > Solver drop-down list | Warning |

| Parameter | Recommended value for Polyspace verification | How you specify value in Configuration Parameters dialog box | If you do not use recommended value, Polyspace Model Link SL generates ... |
|------------------------------|--|--|--|
| SystemTargetFile | 'ert.tlc' | Specify ert.tlc (for Embedded Coder) in Code Generation > System target file | Error |
| ZeroExternalMemory AtStartup | 'off' when Configuration Parameters > Polyspace Model Link > Data Range Management > Output is Global assert | Clear check box Optimization > Remove root level I/O zero initialization | Warning |

Check Simulink Model Settings

With Polyspace Model Link SL software, you can check your model settings before starting a verification:

- 1 From the Simulink model window, select **Code > Polyspace > Options**. The Configuration Parameters dialog box opens, displaying the Polyspace Model Link pane.
- 2 Click **Check configuration**. If your model settings are not optimal for Polyspace verification, the software displays warning messages with recommendations.



For more information on model settings, see “Recommended Polyspace Settings for Code Verification” on page 2-5.

Note If you alter your model settings, build the model again to generate fresh code. If the generated code version does not match your model version, the software produces warnings when you run a verification.

Specify Signal Ranges

If you constrain signals in your Simulink model to lie within specified ranges, Polyspace software automatically applies these constraints during verification of the generated code. This can reduce the number of orange checks in your verification results.

You can specify a range for a model signal by:

- Applying constraints through source block parameters. See “Specify Signal Range through Source Block Parameters” on page 2-9.
- Constraining signals through the base workspace. See “Specify Signal Range through Base Workspace” on page 2-11.

Note You can also manually define data ranges using the DRS feature in the Polyspace verification environment. If you manually define a DRS file, the software automatically appends any signal range information from your model to the DRS file. However, manually defined DRS information overrides information generated from the model for all variables.

Specify Signal Range through Source Block Parameters

You can specify a signal range by applying constraints to source block parameters.

Specifying a range through source block parameters is often easier than creating signal objects in the base workspace, but must be repeated for each source block. For information on using the base workspace, see “Specify Signal Range through Base Workspace” on page 2-11.

To specify a signal range using source block parameters:

- 1** Double-click the source block in your model. The Source Block Parameters dialog box opens.
- 2** Select the **Signal Attributes** tab.

- 3 Specify the **Minimum** value for the signal, for example, -15.
- 4 Specify the **Maximum** value for the signal, for example, 15.

Inport

Provide an input port for a subsystem or model.
For Triggered Subsystems, 'Latch input by delaying outside signal' produces the value of the subsystem input at the previous time step.
For Function-Call Subsystems, turning 'On' the 'Latch input for feedback signals of function-call subsystem outputs' prevents the input value to this subsystem from changing during its execution.
The other parameters can be used to explicitly specify the input signal attributes.

Main | **Signal Attributes**

Output function call

Minimum: Maximum:

Data type:

Lock output data type setting against changes by the fixed-point tools

Port dimensions (-1 for inherited):

Variable-size signal:

Sample time (-1 for inherited):

Signal type:

Sampling mode:

- 5 Click **OK**.

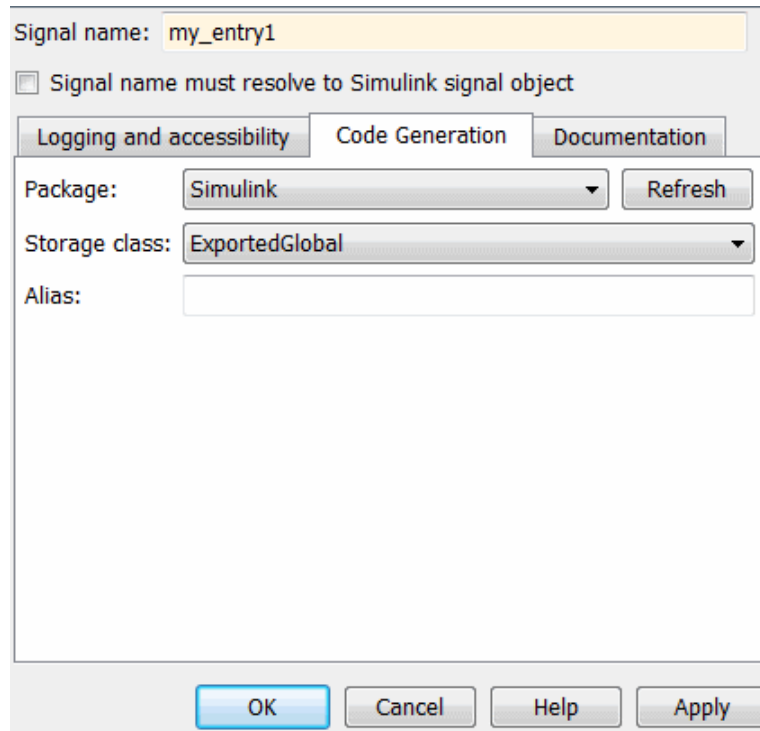
Specify Signal Range through Base Workspace

You can specify a signal range by creating signal objects in the MATLAB workspace. This information is used to initialize each global variable to the range of valid values, as defined by the min-max information in the workspace.

Note You can also specify a signal range by applying constraints to individual source block parameters. This method can be easier than creating signal objects in the base workspace, but must be repeated for each source block. For more information, see “Specify Signal Range through Source Block Parameters” on page 2-9.

To specify an input signal range through the base workspace:

- 1** Configure the signal to use, for example, the `ExportedGlobal` storage class:
 - a** Right-click the signal. From the context menu, select **Properties**. The Signal Properties dialog box opens.
 - b** In the **Signal name** field, enter a name, for example, `my_entry1`.
 - c** Select the **Code Generation** tab.
 - d** From the **Package** drop-down menu, select `Simulink`.
 - e** In the **Storage class** drop-down menu, select `ExportedGlobal`.



- f** Click **OK**, which applies your changes and closes the dialog box.

Note For information about supported storage classes, see “Data Range Specification” on page 3-28.

- 2** Using Model Explorer, specify the signal range:
 - a** Select **Tools > Model Explorer** to open Model Explorer.
 - b** From the **Model Hierarchy** tree, select **Base Workspace**.
 - c** Click the **Add Simulink Signal** button to create a signal. Rename this signal, for example, `my_entry1`.
 - d** Set the **Minimum** value for the signal, for example, to `-15`.
 - e** Set the **Maximum** value for the signal, for example, to `15`.

- f** From the **Storage class** drop-down list, select `ExportedGlobal`.
- g** Click **Apply**.

Annotate Blocks with Known Checks or Coding-Rule Violations

You can annotate individual blocks in your Simulink model to inform Polyspace software of known run-time checks or coding-rule violations. This allows you to highlight and categorize checks identified in previous verifications, so you can focus on new checks when reviewing results.

The Polyspace Results Manager perspective displays the information that you provide with block annotations, and marks the checks as Justified.

To annotate a block:

- 1 In the Simulink model window, right-click the block you want to annotate.
- 2 From the context menu, select **Polyspace Annotations > Edit**. The Polyspace Annotation dialog box opens.

The screenshot shows the Polyspace Annotation dialog box. It is divided into two main sections: 'Description' and 'Annotation'. The 'Description' section contains the following text: "You can annotate blocks in your Simulink model to inform Polyspace software of known run-time checks or coding-rule violations. This allows you to highlight previously identified checks in your verification results, so you can focus on new checks." The 'Annotation' section contains several controls: a dropdown menu for 'Annotation type' with 'Check' selected; a checked checkbox for 'Only 1 check'; a dropdown menu for 'Select RTE check kind'; a dropdown menu for 'Status'; a dropdown menu for 'Classification'; and a text area for 'Comment'. At the bottom of the dialog are four buttons: 'OK', 'Cancel', 'Help', and 'Apply'.

- 3 From the **Annotation type** drop-down list, select one of the following:
 - Check — To indicate a run-time error
 - MISRA-C — To indicate a MISRA C[®] coding rule violation

- MISRA-C++ — To indicate a MISRA® C++ coding rule violation
- JSF — To indicate a JSF® C++ coding rule violation

4 If you want to highlight only one check, select **Only 1 check** and the relevant run-time check (or coding rule) from the **Select RTE check kind** (or **Select MISRA rule**, **Select MISRA C++ rule**, or **Select JSF rule**) drop-down list.

If you want to highlight a list of checks, clear **Only 1 check**. In the **Enter a list of checks** (or **Enter a list of rule numbers**) field, specify the run-time checks or MISRA rules that you want to highlight.

5 Select a **Status** to describe how you intend to address the issue:

- Fix
- Improve
- Investigate
- Justify with annotations
- No Action Planned
- Other
- Restart with different options
- Undecided

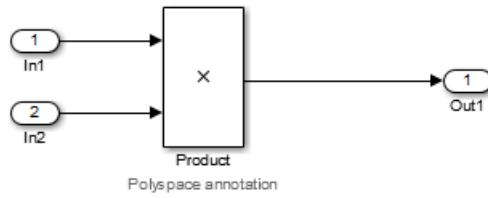
6 Select a **Classification** to describe the severity of the issue:

- High
- Medium
- Low
- Not a defect

7 In the **Comment** field, enter additional information about the check.

8 Click **OK**. The software adds the Polyspace annotation is to the block.

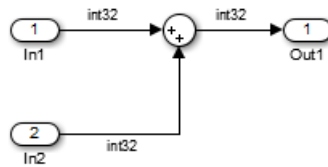
2 Configure Model for Code Verification



Code Annotation for Justifying Polyspace Checks

With the Polyspace Model Link SL product you can apply Polyspace verification to Embedded Coder generated code. The software detects run-time errors in the generated code and helps you to locate and fix model faults.

Polyspace might highlight overflows for certain operations that are legitimate because of the way Embedded Coder implements these operations. Consider the following model and the corresponding generated code.



```

32 /* Sum: '<Root>/Sum' incorporates:
33  * Inport: '<Root>/In1'
34  * Inport: '<Root>/In2'
35  */
36 qY_0 = sat_add_U.In1 + sat_add_U.In2;
37 if ((sat_add_U.In1 < 0) && ((sat_add_U.In2 < 0) && (qY_0 >= 0))) {
38   qY_0 = MIN_int32_T;
39 } else {
40   if ((sat_add_U.In1 > 0) && ((sat_add_U.In2 > 0) && (qY_0 <= 0))) {
41     qY_0 = MAX_int32_T;
42   }
43 }
  
```

Embedded Coder software recognizes that the largest built-in data type is 32-bit. It is not possible to saturate the results of the additions and subtractions using `MIN_INT32` and `MAX_INT32`, and a bigger single-word integer data type. Instead the software detects the results overflow and the direction of the overflow, and saturates the result.

If you do not provide justification for the addition operator on line 36, a Polyspace verification generates an orange check that indicates a potential overflow. The verification does not take into account the saturation function of lines 37 to 43. In addition, the trace-back functionality of Polyspace Model Link SL does not identify the reason for the orange check.

To justify overflows from operators that are legitimate, on the **Configuration Parameters > Code Generation > Comments** pane:

- Under **Overall control**, select the **Include comments** check box.
- Under **Auto generate comments**, select the **Operator annotations check box**.

When you generate code, the Embedded Coder software annotates the code with comments for Polyspace. For example:

```
32 /* Sum: '<Root>/Sum' incorporates:
33  * Inport: '<Root>/In1'
34  * Inport: '<Root>/In2'
35  */
36 qY_0 = sat_add_U.In1 +/*MW:0vOk*/ sat_add_U.In2;
```

When you run a verification using Polyspace Model Link SL, the Polyspace software uses the annotations to justify the operator-related orange checks and assigns the `Not a defect` classification to the checks.

Configure Code Verification Options

- “Overview of Polyspace Configuration” on page 3-2
- “Include Handwritten Code in Verification” on page 3-3
- “Specify Client or Server Verification” on page 3-5
- “Configure Data Range Settings” on page 3-6
- “Configure Verification of Model Reference Code” on page 3-9
- “Specify Location of Results” on page 3-10
- “Check Coding Rules Compliance” on page 3-11
- “Configure Polyspace Verification Options” on page 3-14
- “Configure Polyspace Project Properties” on page 3-17
- “Create a Polyspace Configuration File Template” on page 3-18
- “Specify Header Files for Target Compiler” on page 3-21
- “Open Polyspace Project Manager Automatically” on page 3-22
- “Remove Polyspace Options From Simulink Model” on page 3-24
- “Main Generation for Model Verification” on page 3-25
- “Polyspace® Model Link™ SL Considerations” on page 3-27
- “Polyspace® Model Link™ TL Considerations” on page 3-34

Overview of Polyspace Configuration

You do not have to manually create a Polyspace project or specify Polyspace options before running a verification for your generated code. By default, when you start a verification, Polyspace automatically creates a project and extracts the required information from your model. However, you can modify or specify additional options for your verification:

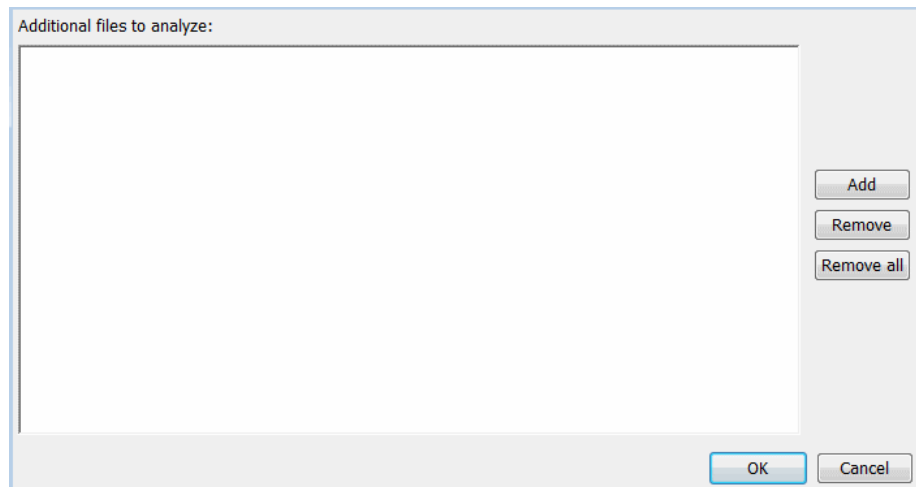
- You may incorporate separately created code within the code generated from your Simulink model. See “Include Handwritten Code in Verification” on page 3-3.
- By default, the Polyspace verification is contextual and treats tunable parameters as constants. You can specify a verification that considers robustness, including tunable parameters that lie within a range of values. See “Configure Data Range Settings” on page 3-6.
- You may customize the options for your verification. For example, to specify the target environment or adjust precision settings. See “Configure Polyspace Verification Options” on page 3-14 and “Recommended Polyspace Options for Verifying Generated Code” on page 3-29.
- You may create specific configurations for batch runs. See “Create a Polyspace Configuration File Template” on page 3-18.
- If you want to verify code generated for a 16-bit target processor, you must specify header files for your 16-bit compiler. See “Specify Header Files for Target Compiler” on page 3-21.

Include Handwritten Code in Verification

Files such as S-function wrappers are, by default, not part of the Polyspace verification. You can add these files manually. Steps for Polyspace Model Link SL are shown here.

To add a file to a verification:

- 1 From the Simulink model window, select **Code > Polyspace > Options**. The Configuration Parameters dialog box opens, displaying the **Polyspace Model Link** pane.
- 2 Select the **Enable additional file list** check box. Then click **Select files**. The Files Selector dialog box opens.



- 3 Click **Add**. The Select files to add dialog box opens.
- 4 Use the Select files to add dialog box to:
 - Navigate to the relevant folder
 - Add the required files.

The software displays the selected files as a list under **Additional files to analyze**.

Note To remove a file from the list, select the file and click **Remove**. To remove all files from the list, click **Remove all**.

5 Click **OK**.

Specify Client or Server Verification

By default, the software runs the code verification on your Polyspace server.
To specify a client verification:

- 1** From the Simulink model window, select **Code > Polyspace > Options**.
The Configuration Parameters dialog box opens, displaying the **Polyspace Model Link** pane.
- 2** Clear the **Send to Polyspace server** check box.
- 3** Click **Apply**.

Configure Data Range Settings

There are two approaches to code verification, which can produce results that are slightly different:

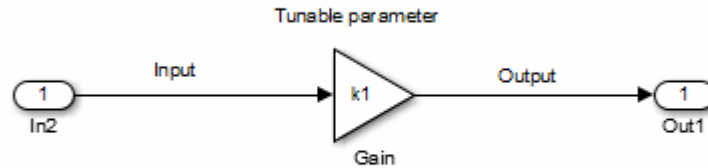
- **Contextual Verification** — Prove code works under predefined working conditions. This limits the scope of the verification to specific variable ranges, and verifies the code within these ranges.
- **Robustness Verification** — Prove code works under all conditions, including “abnormal” conditions for which the code was not designed. This can be thought of as “worst case” verification.

For more information, see:

- “Choose Robustness or Contextual Verification”.
- Data Range Specification — Model Link SL
- Data Range Specification — Model Link TL

Note The software supports data range management only with Simulink Version 7.4 (R2009b) or later.

Both Polyspace Model Link SL and Polyspace Model Link TL allow you to run either contextual or robustness verification by the way you specify data ranges for model inputs, outputs, and tunable parameters within the model.



To specify data range settings for your model:

- 1** From the Simulink model window, select **Code > Polyspace > Options**. The Configuration Parameters dialog box opens, displaying the **Polyspace Model Link** pane.
- 2** In the Data Range Management section, specify how you want the verification to treat:
 - a Input** — Select one of the following:
 - Use specified minimum and maximum values (Default) — Apply data ranges defined in blocks or base workspace to increase the precision of the verification. See “Specify Signal Ranges” on page 2-9.
 - Unbounded inputs — Assume all inputs are full-range values (min...max)
 - b Tunable parameters** — Select one of the following:
 - Use calibration data (Default) — Use value of constant parameter specified in code.
 - Use specified minimum and maximum values — Use a parameter range defined in the block or base workspace. See “Specify Signal Ranges” on page 2-9. If no range is defined, use full range (min...max).
 - c Output** — Select one of the following:
 - No verification (Default) — No assertion ranges on outputs.

- Verify outputs are within minimum and maximum values — Use assertion ranges on outputs.

Note This mode is incompatible with the Automatic Orange Tester.

In general, you should use the following combinations:

- To maximize verification precision, select Use specified minimum and maximum values for **Input** and **Tunable parameters**.
- To verify the extreme cases of program execution, select Unbounded inputs for **Input** and Use calibration data for **Tunable parameters**.

Configure Verification of Model Reference Code

From the **Polyspace Model Link** pane, you can specify the verification of generated code with respect to model reference hierarchy levels:

- **Model reference verification depth** — From the drop-down list, select one of the following:
 - **Current model only** — Default. The software verifies code from the top level only. The software creates stubs to represent code from lower hierarchy levels.
 - **1** — The software verifies code from the top level and the next level. For subsequent hierarchy levels, the software creates stubs.
 - **2** — The software verifies code from the top level and the next two hierarchy levels. For subsequent hierarchy levels, the software creates stubs.
 - **3** — The software verifies code from the top level and the next three hierarchy levels. For subsequent hierarchy levels, the software creates stubs.
 - **All** — The software verifies code from the top level and all lower hierarchy levels.
- **Model by model verification** — Select this check box if you want the software to verify code from each model separately.

Note The same configuration settings apply to all referenced models within a top model. It does not matter whether you open the **Polyspace Model Link** pane from the top model window (**Code > Polyspace > Options**) or through the right-click context menu of a particular Model block within the top model. However, you can run verifications for code generated from specific Model blocks. See “Run Verification with Polyspace® Model Link™ SL Software” on page 4-5.

Specify Location of Results

With Polyspace Model Link SL, you can specify a location for the results of your verification:

- 1** From the Simulink model window, select **Code > Polyspace > Options**. The Configuration Parameters dialog box opens with the Polyspace Model Link pane displayed.
- 2** In the **Output folder** field, specify the full path for your results folder. By default, the software stores results in `C:\PolySpace_Results\results_model_name`.
- 3** If you want to avoid overwriting results from previous verifications, select the **Make output folder name unique by adding a suffix** check box. Instead of overwriting an existing folder, the software specifies a new location for the results folder by appending a unique number to the folder name.

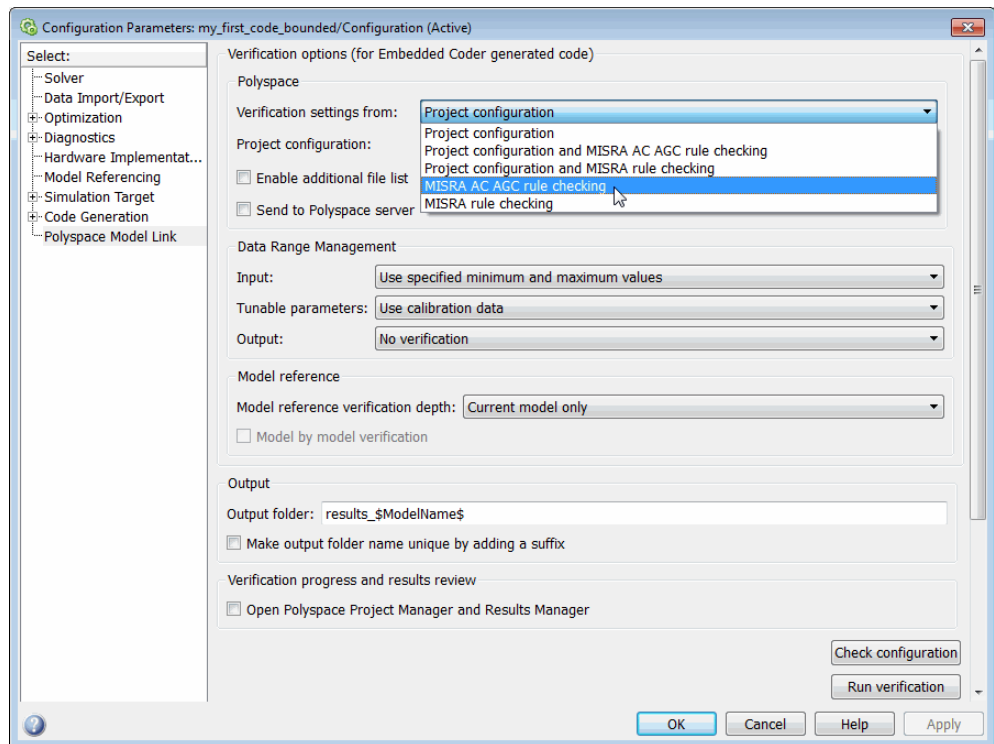
Check Coding Rules Compliance

Polyspace Model Link SL software allows you to check compliance with MISRA C and MISRA AC AGC coding rules directly from your Simulink model.

You can choose to run coding rules checking either with or without full code verification.

To configure coding rules checking:

- 1 From the Simulink model window, select **Code > Polyspace > Options**. The **Polyspace Model Link** pane opens.



- 2** In the **Verification settings** from drop-down menu, select the type of analysis you want to perform.

Depending on the type of code generated, different verification settings are available. The following tables describe the different Verification settings.

C Code Verification Settings

| Verification Setting | Description |
|--|--|
| Project configuration | Run code verification using the options specified in the Project configuration. |
| Project configuration and MISRA AC AGC rule checking | Check compliance with the MISRA AC-AGC rule set, and run code verification using the options specified in the Project configuration. |
| Project configuration and MISRA rule checking | Check compliance with all MISRA C coding rules, and run code verification using the options specified in the Project configuration. |
| MISRA AC AGC rule checking | Check compliance with the MISRA AC-AGC rule set. Verification stops after rules checking. |
| MISRA rule checking | Check compliance with all MISRA C coding rules. Verification stops after rules checking. |

C++ Code Verification Settings

| Verification Setting | Description |
|---|---|
| Project configuration | Run code verification using the options specified in the Project configuration. |
| Project configuration and MISRA C++ rule checking | Check compliance with the MISRA C++ coding rules, and run code verification using the options specified in the Project configuration. |
| Project configuration and JSF C++ rule checking | Check compliance with all JSF C++ coding rules, and run code verification using the options specified in the Project configuration. |
| MISRA C++ rule checking | Check compliance with the MISRA C++ coding rules. Verification stops after rules checking. |
| JSF C++ rule checking | Check compliance with all JSF C++ coding rules. Verification stops after rules checking. |

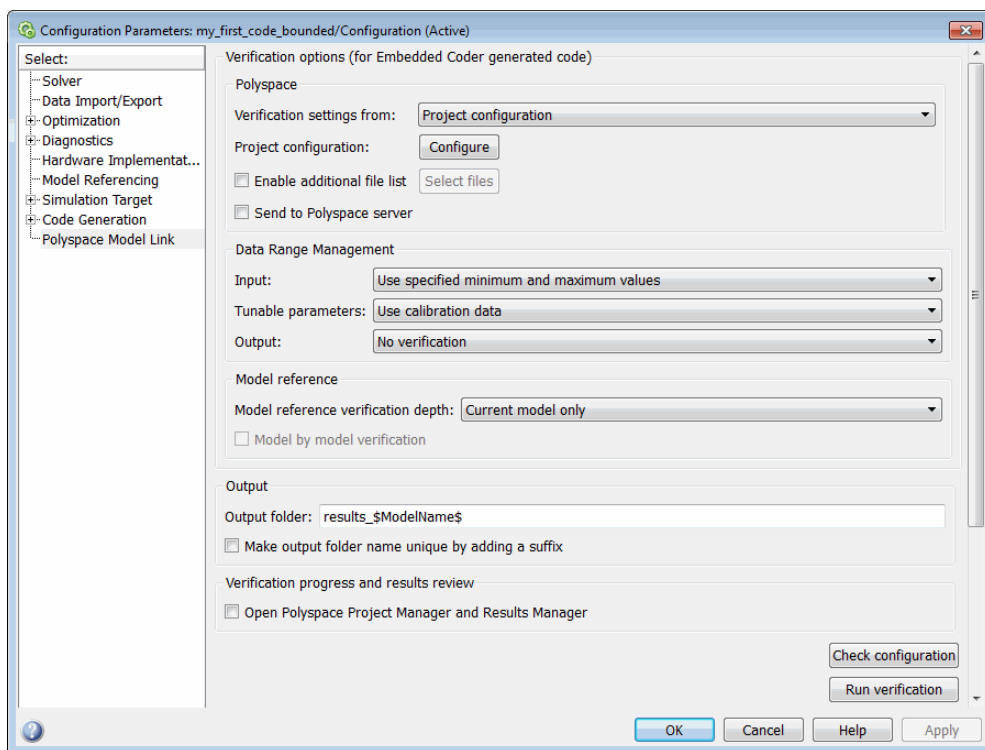
3 Click **Apply** to save your settings.

Configure Polyspace Verification Options

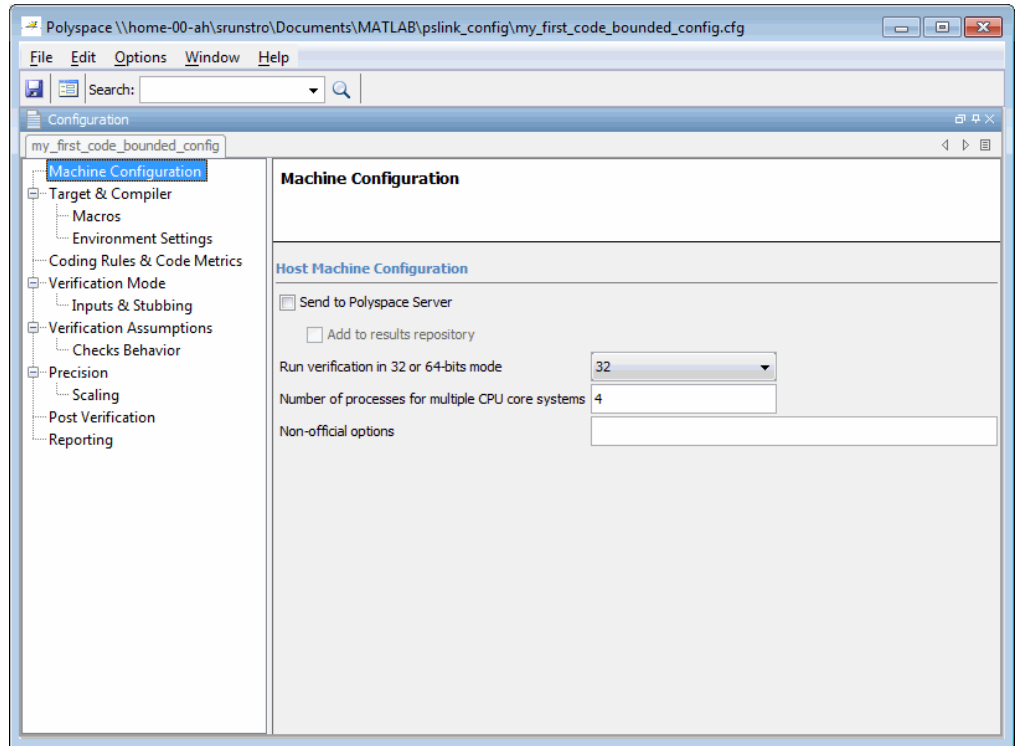
Polyspace Model Link software supports a simplified version of the Polyspace Project Manager, which allows you to customize the options and project properties for your verification. For example, you can specify the target processor type, target operating system, and compilation flags.

To open the **Configuration** pane of the Project Manager:

- 1 From the Simulink model window, select **Code > Polyspace > Options**. The **Polyspace Model Link** pane opens.



- 2 Click **Configure**. The Project Manager opens, displaying the **Machine Configuration** pane.



The first time you open the configuration, the software sets the following options:

- **Target operating system** (-OS-target) – Set to no-predefined-OS
- **Use result folder** (-results-dir) – Set to results_*modelName*

The software also configures other options automatically, but the settings depend on the code generator used. See “Polyspace® Model Link™ SL Considerations” on page 3-27 and “Polyspace® Model Link™ TL Considerations” on page 3-34.


3 Set other options required by your application.

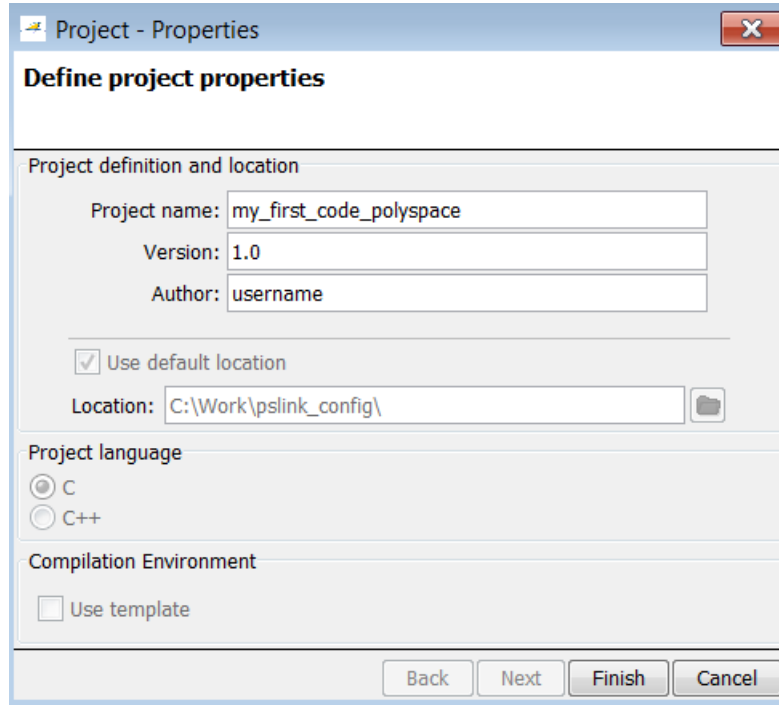
For recommended options for verifying generated code, see “Recommended Polyspace Options for Verifying Generated Code” on page 3-29.

For descriptions of all options, see “Analysis Options for C Code” or “Analysis Options for C++ Code”.

Configure Polyspace Project Properties

You can specify project properties, for example, your project name, through the Polyspace Project - Properties dialog box. To open this dialog box,

- 1 From the Simulink model window, select **Code > Polyspace > Options**. The **Polyspace Model Link** pane opens.
- 2 Click **Configure**. The Project Manager opens.
- 3 On the Project Manager toolbar, click the **Project properties** icon .



Project - Properties

Define project properties

Project definition and location

Project name: my_first_code_polyspace

Version: 1.0

Author: username

Use default location

Location: C:\Work\pslink_config\

Project language

C

C++

Compilation Environment

Use template

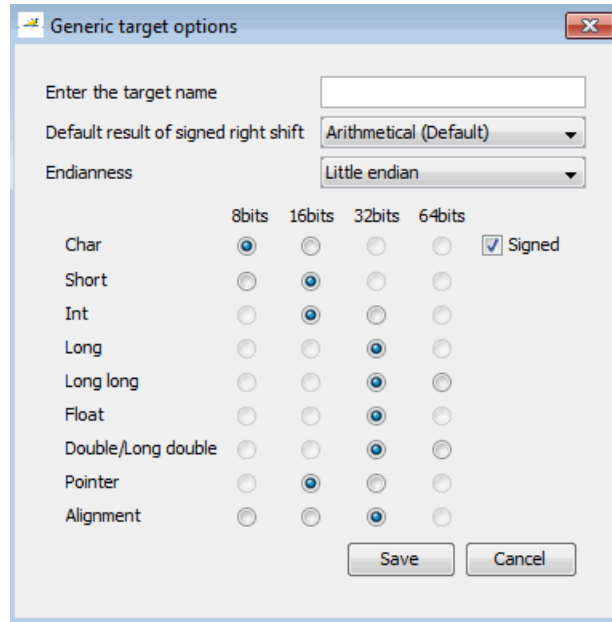
Back Next Finish Cancel

Create a Polyspace Configuration File Template

During a batch run, you may want use different configurations for verification. The software provides the command `PolyspaceSetTemplateCFGFile`, which allows you to apply a configuration defined by a configuration file template. See “MATLAB Functions For Polyspace Batch Runs” on page 4-14.

To create a configuration file template:

- 1** In the Simulink model window, select **Code > Polyspace > Options**. The **Polyspace Model Link** pane opens.
- 2** Click **Configure**. The Project Manager opens, displaying the **Configuration** pane. Use this pane to customize the target and cross compiler.
- 3** From the **Configuration** tree, expand the **Target & Compiler** node.
- 4** In the **Target Environment** section, use the **Target processor type** option to define the size of data types.
 - a** From the drop-down list, select `mcpu...` (Advanced). The Generic target options dialog box opens.



Use this dialog box to create a new target and specify data types for the target. Then click **Save**.

- 5 From the Configuration tree, select **Target & Compiler > Macros**. Use the **Preprocessor definitions** section to define preprocessor macros for your cross-compiler.

To add a macro, in the **Macros** table, click the **+** button. In the new line, enter the required text.

To remove a macro, select the macro and click the **-** button.

Note If you use the LCC cross-compiler, then you must specify the `MATLAB_MEX_FILE` macro.

- 6 Save your changes and close the Project Manager.

- 7** Make a copy of the updated project configuration file, for example, `my_first_code_polyspace.cfg`.
- 8** Rename the copy, for example, `my_cross_compiler.cfg`. This is your new configuration file template.

To make a template the configuration for verification, run the `PolyspaceSetTemplateCFGFile` command in the MATLAB Command Window. For example:

```
PolyspaceSetTemplateCFGFile ('C:\Work\my_cross_compiler.cfg')
```

Specify Header Files for Target Compiler

If you want to verify code generated for a 16-bit target processor, you must specify header files for your 16-bit compiler. The software automatically identifies the compiler from the Simulink model. If the compiler is 16-bit and you do not specify the relevant header files, the software produces an error when you try to run a verification.

Note For a 32-bit or 64-bit target processor, the software automatically specifies the default header file.

To specify header file folders (or header files) for your compiler:

- 1** Open the Polyspace **Configuration** pane. From the Simulink model window, select **Code > Polyspace > Options**. The **Polyspace Model Link** pane opens.
- 2** Click **Configure**. The Project Manager opens, displaying the **Configuration** pane.
- 3** From the **Configuration** tree, expand the **Target & Compiler** node.
- 4** Select **Target & Compiler > Environment Settings**.
- 5** In the **Include folders** (or **Include**) section, specify a folder (or header file) path by doing one of the following:
 - Click the **+** button. Then, in the text field, enter the folder (or file) path.
 - Click the folder button and use the Open file dialog box to navigate to the required folder (or file).

You can remove an item from the displayed list by selecting the item and then clicking **-**.

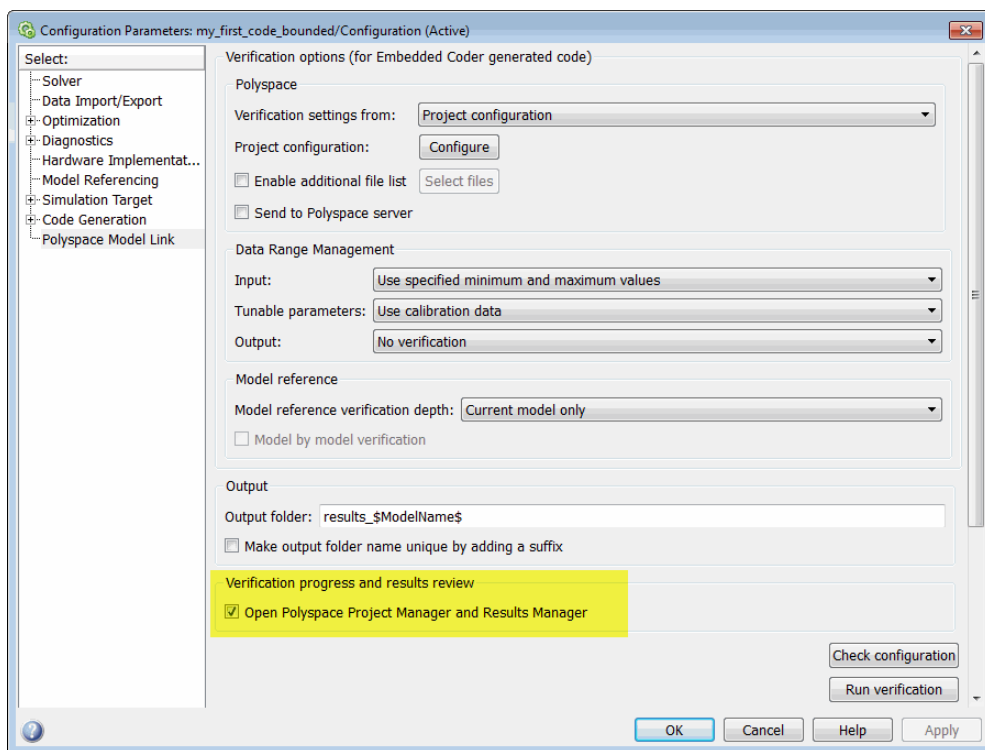
Open Polyspace Project Manager Automatically

You can configure the software to automatically open the Polyspace Project Manager when you launch a verification. This allows you to monitor the progress of your verification from the Project Manager. When verification is complete, you can switch to the Results Manager perspective to review the results.

To configure the Project Manager to open automatically:

- 1 From the model window, select **Code > Polyspace > Options**.

The Polyspace Model Link pane opens.



- 2** In the Verification progress and results review section, select **Open Polyspace Project Manager and Results Manager**.
- 3** Click **Apply** to save your settings.

Remove Polyspace Options From Simulink Model

You can remove Polyspace configuration information from your Simulink model.

For a top model:

- 1 Select **Code > Polyspace > Remove Options from Current Configuration**.
- 2 Save the model.

For a Model block or subsystem:

- 1 Right-click the Model block or subsystem.
- 2 From the context menu, select **Remove Options from Current Configuration**.
- 3 Save the model.

Main Generation for Model Verification

When you run a verification using either Polyspace Model Link SL or Polyspace Model Link TL, the software automatically reads the following information from the model:

- `initialize()` functions
- `terminate()` functions
- `step()` functions
- List of parameter variables
- List of input variables

The software then uses this information to generate a main function that:

- 1** Initializes parameters using the Polyspace option `-variables-written-before-loop`.
- 2** Calls initialization functions using the option `-functions-called-before-loop`.
- 3** Initializes inputs using the option `-variables-written-in-loop`.
- 4** Calls the `step` function using the option `-functions-called-in-loop`.
- 5** Calls the `terminate` function using the option `-functions-called-after-loop`.

If the `codeInfo` for the model does not contain the names of the inputs, the software considers all variables as entries, except for parameters and outputs.

For C++ code that is generated with Embedded Coder, the `initialize()`, `step()`, and `terminate()` functions are either class methods or have global scope. These different scopes contain the associated variables.

- For class methods in the generated code, the variables that are written before and in the loop refer to the class members.
- For functions with global scope, the associated variables are also in the global scope.

main for Generated Code

The following example shows the main generator options that the software uses to generate the main function for code generated from a Simulink model.

```
init parameters    \\ -variables-written-before-loop
init_fct()        \\ -functions-called-before-loop
  while(1){       \\ start main loop
    init inputs   \\ -variables-written-in-loop
    step_fct()    \\ -functions-called-in-loop
  }
terminate_fct()   \\ -functions-called-after-loop
```

Polyspace Model Link SL Considerations

In this section...

“Overview” on page 3-27

“Subsystems” on page 3-27

“Default Options” on page 3-27

“Data Range Specification” on page 3-28

“Recommended Polyspace Options for Verifying Generated Code” on page 3-29

“Hardware Mapping Between Simulink and Polyspace” on page 3-33

Overview

The Polyspace Model Link SL product has been tested with Embedded Coder software. For more information, see “Software Installation”.

Subsystems

A dialog will be presented after clicking on the Polyspace for Embedded Coder block if multiple subsystems are present in a diagram. Simply select the subsystem to analyze from the list. The subsystem list is generated from the directory structure from the code that has been generated.

Default Options

When using the Polyspace Model Link SL product, the software sets the following verification options by default:

```
-sources path_to_source_code
-desktop
-D PST_ERRNO
-D main=main_rtwec
-I matlabroot\polyspace\include
-I matlabroot\extern\include
-I matlabroot\rtw\c\libsrc
-I matlabroot\simulink\include
-I matlabroot\sys\lcc\include
```

```
-OS-target no-predefined-OS  
-results-dir results
```

Note *matlabroot* is the MATLAB installation folder.

Data Range Specification

You can constrain inputs, parameters, and outputs to lie within specified data ranges. See “Configure Data Range Settings” on page 3-6.

The software automatically creates a Polyspace Data Range Specification (DRS) file using information from the MATLAB workspace and block parameters.

You can also manually define a DRS file using the Project Manager perspective of the Polyspace verification environment. If you define a DRS file, the software appends the automatically generated information to the DRS file you create. Manually defined DRS information overrides automatically generated information for all variables.

The software supports the automatic generation of data range specifications for the following kinds of generated code:

- Code from standalone models
- Code from configured function prototypes
- Reusable code
- Code generated from referenced models and submodels

The software supports the automatic generation of data range specifications for only the following signal and parameter storage classes:

- SimulinkGlobal
- ExportedGlobal
- Struct (Custom)

Recommended Polyspace Options for Verifying Generated Code

The Polyspace Model Link SL software automatically specifies values for the following verification options:

- `-main-generator`
- `-functions-called-in-loop`
- `-functions-called-before-loop`
- `-functions-called-after-loop`
- `-variables-written-in-loop`
- `-variables-written-before-loop`

In addition, for the option `-server`, the software uses the value specified in the **Send to Polyspace server** check box on the **Polyspace Model Link** pane. These values override the corresponding option values in the **Configuration** pane of the Project Manager.

You can specify other verification options for your Polyspace Project through the Polyspace **Configuration** pane. To open this pane:

- 1** In the Simulink model window, select **Code > Polyspace > Options** . The **Polyspace Model Link** pane opens.
- 2** Click **Configure**. The Project Manager opens, displaying the Polyspace **Configuration** pane.

The following table describes options that you should specify in your Polyspace project before verifying code generated by Embedded Coder software.

| Option | Recommended Value | Comments |
|------------------------------|-------------------|---|
| Target & Compiler | | |
| -D | See Comments | <p>Defines macro compiler flags used during compilation.</p> <p>Use one -D for each line of the Embedded Coder generated <code>defines.txt</code> file.</p> <p>Polyspace Model Link SL does not do this by default.</p> |
| -OS-target | Visual | <p>Specifies the operating system target for Polyspace stubs.</p> <p>This information allows the verification to use system definitions during preprocessing to analyze the included files.</p> |
| -target | i386 | <p>Specifies the target processor type. This allows the verification to consider the size of fundamental data types and the endianness of the target machine.</p> <p>You can configure and specify generic targets. For more information, see “Target Processor Configuration”.</p> |
| -dos | Selected | <p>You must select this option if the contents of the include or source directory comes from a DOS or Windows file system. The option allows the verification to deal with upper/lower case sensitivity and control characters issues. Concerned files are:</p> <ul style="list-style-type: none"> • Header files – All include folders specified (-I option) • Source files – All source files selected for the verification (-sources option) |

| Option | Recommended Value | Comments |
|---------------------------------|-------------------|---|
| Verification Assumptions | | |
| -allow-negative-operations | Selected | <p>Allows a shift operation on a negative number. According to the ANSI® standard, such a shift operation on a negative number is illegal. For example, $-2 \ll 2$. If you select this option, Polyspace considers the operation to be valid. For the given example, $-2 \ll 2 = -8$.</p> |
| -ignore-float-rounding | Selected | <p>Specifies how the verification rounds floats.</p> <p>If this option is not selected, the verification rounds floats according to the IEEE® 754 standard – simple precision on 32-bits targets and double precision on targets that define double as 64-bits.</p> <p>When you select this option, the verification performs exact computation.</p> <p>Selecting this option can lead to results that differ from "real life," depending on the actual compiler and target. Some paths may be reachable (or not reachable) for the verification while they are not reachable (or are reachable) for the actual compiler and target.</p> <p>However, this option reduces the number of unproven checks caused by float approximation.</p> |

| Option | Recommended Value | Comments |
|------------------|-------------------|--|
| Precision | | |
| -0 | 2 | <p>Specifies the precision level for the verification.</p> <p>Higher precision levels provide higher selectivity at the expense of longer verification time.</p> <p>Begin with the lowest precision level. You can then address red errors and gray code before rerunning the Polyspace verification using higher precision levels.</p> <p>Benefits:</p> <p>A higher precision level contributes to a higher selectivity rate, making results review more efficient and hence making bugs in the code easier to isolate.</p> <p>The precision level specifies the algorithms used to model the program state space during verification:</p> <ul style="list-style-type: none"> • -00 corresponds to static interval verification. • -01 corresponds to complex polyhedron model of domain values. • -02 corresponds to more complex algorithms to closely model domain values (a mixed approach with integer lattices and complex polyhedrons). • -03 is suitable only for units smaller than 1,000 lines of code. For such code, selectivity may reach as high as 98%, but verification may take up to an hour per 1,000 lines of code. |

| Option | Recommended Value | Comments |
|--------|---|---|
| -to | <p>C source compliance checking – For C code, when checking coding rule compliance only.</p> <p>C++ source compliance checking – For C++ code, when checking coding rule compliance only.</p> <p>pass0 – When verifying code for the first time.</p> <p>pass4 – When performing subsequent verifications of code.</p> | <p>Specifies the phase after which the verification stops. Each verification phase improves the selectivity of your results, but increases the overall verification time.</p> <p>Improved selectivity can make results review more efficient, and hence make bugs in the code easier to isolate.</p> <p>Begin by running <code>-to pass0</code> (Software Safety Analysis level 0) You can then address red errors and gray code before relaunching verification using higher integration levels.</p> |

Hardware Mapping Between Simulink and Polyspace

The software automatically imports target word lengths and byte ordering (endianess) from Simulink model hardware configuration settings. The software maps **Device vendor** and **Device type** settings on the Simulink **Configuration Parameters > Hardware Implementation** pane to **Target processor type** settings on the Polyspace **Configuration** pane.

Note The software creates a generic target for the verification.

Polyspace Model Link TL Considerations

In this section...

“Overview” on page 3-34

“Subsystems” on page 3-34

“Default Options” on page 3-34

“Data Range Specification” on page 3-35

“Lookup Tables” on page 3-36

“Code Generation Options” on page 3-36

Overview

The Polyspace Model Link TL product has been tested with specific releases of the dSPACE Data Dictionary version and TargetLink Code Generator. For more information, see “Polyspace Plug-In Requirements”.

As the Polyspace Model Link TL product extracts information from the dSPACE Data Dictionary, remember to regenerate the code before performing a Polyspace verification.

Subsystems

A dialog will be presented after clicking on the Polyspace for TargetLink block if multiple subsystems are present in a diagram. Simply select the subsystem to analyze from the list.

Default Options

The following default options are set by the tool:

```
-I path to source code
-desktop
-D PST_ERRNO
-I dspace\matlab\TL\SimFiles\Generic
-I dspace\matlab\TL\srcfiles\Generic
-I dspace\matlab\TL\srcfiles\i86\LCC
```

```
-I matlabroot\polyspace\include  
-I matlabroot\extern\include  
-I matlabroot\rtw\c\libsrc  
-I matlabroot\simulink\include  
-I matlabroot\sys\lcc\include
```

Note *dspaceroot* and *matlabroot* are the dSPACE and MATLAB tool installation directories respectively.

Data Range Specification

You can constrain inputs, parameters, and outputs to lie within specified data ranges. See “Configure Data Range Settings” on page 3-6.

The software automatically creates a Polyspace Data Range Specification (DRS) file using the dSPACE Data Dictionary for each global variable. The DRS information is used to initialize each global variable to the range of valid values as defined by the min-max information in the data dictionary. This allows Polyspace software to model every value that is legal for the system during verification. Carefully defining the min-max information in the model allows the verification to be more precise, because only the range of real values is analyzed.

Note Boolean types are modeled having a minimum value of 0 and a maximum of 1.

You can also manually define a DRS file using the Project Manager perspective of the Polyspace Verification Environment. If you define a DRS file, the software appends the automatically generated information to the DRS file you create. Manually defined DRS information overrides automatically generated information for all variables.

DRS cannot be applied to static variables. Therefore, the compilation flags `-D static=` is set automatically. It has the effect of removing the static keyword from the code. If you have a problem with name clashes in the global name

space you may need to either rename one of or variables or disable this option in Polyspace configuration.

Lookup Tables

The tool by default provides stubs for the lookup table functions. This behavior can be disabled from the Polyspace menu. The dSPACE data dictionary is used to define the range of their return values. Note that a lookup table that uses extrapolation will return full range for the type of variable that it returns.

Code Generation Options

From the TargetLink Main Dialog, it is recommended to set the option Clean code and deselect the option Enable sections/pragmas/inline/ISR/user attributes.

When installing the Polyspace Model Link TL product, the `tlcgOptions` variable has been updated with 'PolyspaceSupport', 'on' (see variable in 'C:\dSPACE\Matlab\TL\config\codegen\tl_pre_codegen_hook.m' file).

Run Code Verification

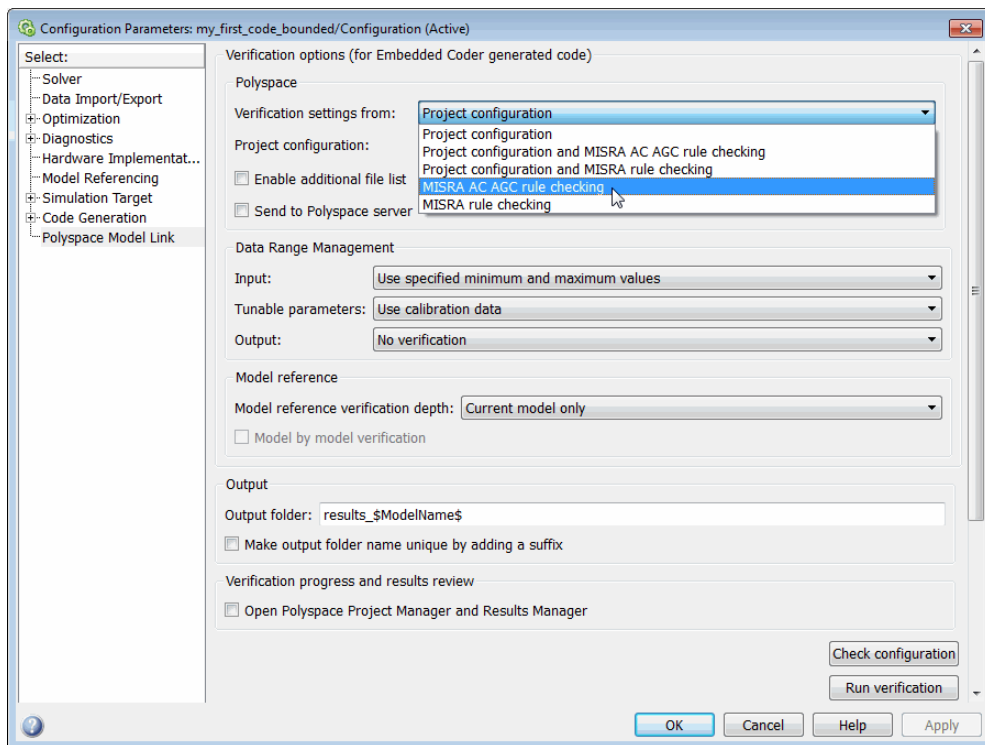
- “Specify Type of Analysis to Perform” on page 4-2
- “Run Verification with Polyspace® Model Link™ SL Software” on page 4-5
- “Run Verification with Polyspace® Model Link™ TL Software” on page 4-8
- “Monitor Verification Progress” on page 4-10
- “Code Generation and Verification with Configured Model” on page 4-12
- “MATLAB Functions For Polyspace Batch Runs” on page 4-14
- “Archive Files for Polyspace Verification” on page 4-15

Specify Type of Analysis to Perform

Before running a verification, you can specify what type of analysis you want to run. You can choose to run code verification, coding rules checking, or both.

To specify the type of analysis to run:

- 1 From the Simulink model window, select **Code > Polyspace > Options**. The **Polyspace Model Link** pane opens.



- 2 In the **Verification settings from** drop-down menu, select the type of analysis you want to perform.

Depending on the type of code generated, different verification settings are available. The following tables describe the different Verification settings.

C Code Verification Settings

| Verification Setting | Description |
|--|--|
| Project configuration | Run code verification using the options specified in the Project configuration. |
| Project configuration and MISRA AC AGC rule checking | Check compliance with the MISRA AC-AGC rule set, and run code verification using the options specified in the Project configuration. |
| Project configuration and MISRA rule checking | Check compliance with all MISRA C coding rules, and run code verification using the options specified in the Project configuration. |
| MISRA AC AGC rule checking | Check compliance with the MISRA AC-AGC rule set. Verification stops after rules checking. |
| MISRA rule checking | Check compliance with all MISRA C coding rules. Verification stops after rules checking. |

C++ Code Verification Settings

| Verification Setting | Description |
|---|---|
| Project configuration | Run code verification using the options specified in the Project configuration. |
| Project configuration and MISRA C++ rule checking | Check compliance with the MISRA C++ coding rules, and run code verification using the options specified in the Project configuration. |
| Project configuration and JSF C++ rule checking | Check compliance with all JSF C++ coding rules, and run code verification using the options specified in the Project configuration. |

C++ Code Verification Settings (Continued)

| Verification Setting | Description |
|-------------------------|--|
| MISRA C++ rule checking | Check compliance with the MISRA C++ coding rules. Verification stops after rules checking. |
| JSF C++ rule checking | Check compliance with all JSF C++ coding rules. Verification stops after rules checking. |

3 Click **Apply** to save your settings.

Run Verification with Polyspace Model Link SL Software

To start a Polyspace verification of:

- Code generated from the top model, from the Simulink model window, select **Code > Polyspace > Polyspace for Embedded Coder > Verify Generated Code**.
- All model reference code associated with the top model, from the model window, select **Code > Polyspace > Polyspace for Embedded Coder > Verify Generated Model Reference Code**.
- Model reference code associated with a specific Model block or subsystem, right-click the Model block or subsystem. From the context menu, select **Polyspace > Polyspace for Embedded Coder**.

Note You can also start verification from the **Polyspace Model Link** pane by clicking **Run verification**.

When the verification starts, messages appear in the MATLAB Command window:

```
### Starting Polyspace verification for Embedded Coder
### Creating results folder C:\PolySpace_Results\results_my_first_code for system my_first_code
### Checking Polyspace Model-Link Configuration:
### Parameters used for code verification:
System                : my_first_code
Results Folder        : C:\PolySpace_Results\results_my_first_code
Additional Files       : 0
Remote                : 0
Model Reference Depth : Current model only
Model by Model         : 0
DRS input mode         : DesignMinMax
DRS parameter mode    : None
DRS output mode        : None
### Writing DRS table in C:\PolySpace_Results\results_my_first_code\my_first_code_drs.txt
...
```

Polyspace verification of my_first_code project.
Starting at 11/04/2011, 12h35.

Follow the progress of the verification in the MATLAB Command window. If you are running a server verification, you can follow the later stages of the verification through the Polyspace spooler (Queue Manager).

The software writes all status messages to a log file in the results folder, for example Polyspace_R2012b_my_first_code_05_16_2012-18h40.log:

```
<polyspace-c R2012b PID5864 PGID5864>
```

Polyspace verification of my_first_code project.
Starting at 05/16/2012, 18h40.

Options used with Verifier:

```
-polyspace-version=CC-8.4.0.1 (R2012b)  
-date=16/05/2012  
-from=scratch  
-context-sensitivity=[none]  
-enum-type-definition=defined-by-standard  
-lang=C  
-allow-negative-operand-in-shift=true  
-max-processes=4  
-variables-written-in-loop=custom=my_first_code_U  
-author=ausser  
-dialect=none  
-results-dir=C:\Work\results_my_first_code\my_first_code  
-scalar-overflows-behavior=truncate-on-error  
-target=mcpu  
-data-range-specifications=C:\Work\results_my_first_code\my_first_code\my_first_code_drs.txt  
-double-is-64bits=true  
-big-endian=true  
-functions-called-before-loop=[my_first_code_initialize]  
-verif-version=1.0  
-main-generator=true  
-O=-O2  
-variables-written-before-loop=none  
-prog=my_first_code  
-scalar-overflows-checks=signed
```

```
-D1=CLASSIC_INTERFACE=0
-D2=HAVESTDIO
-D3=INTEGER_CODE=0
-D4=MAT_FILE=0
-D5=MODEL=my_first_code
-D6=MT=0
...
```

If you want to stop a client verification:

- From the Simulink model window, select **Code > Polyspace > Stop Local Verification**.
- Right-click a Model block or subsystem. From the context menu, select **Polyspace > Stop Local Verification**.

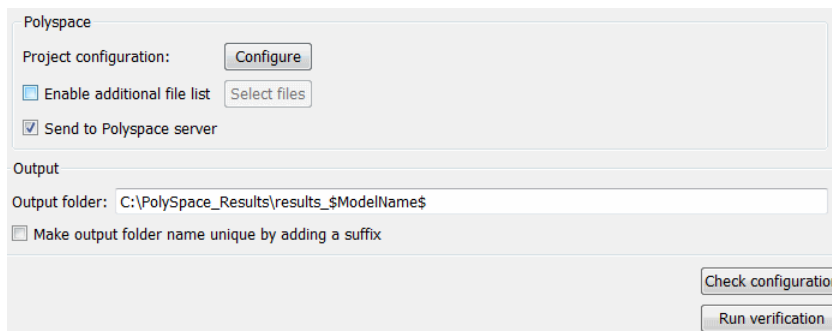
If you want to stop a server verification, use the Polyspace Queue Manager (Spooler). See “Stop Server Verification Before It Completes”.

Run Verification with Polyspace Model Link TL Software

To start the Polyspace verification:

- 1 In the Simulink model window select **Code > Polyspace > Polyspace for TargetLink**.

The **Configuration Parameters > Polyspace Model Link** pane opens.



- 2 Click **Run verification** to start the verification.

Messages appear in the MATLAB Command window:

```
### Polyspace Model-Link for Embedded Coder
### Version MBD-5.7.0.6 (R2011a)
### Preparing code verification
### Creating results folder
### Analysing subsystem: my_first_code
### Locating generated source files:
   H:\Documents\MATLAB\my_first_code_ert_rtw\ert_main.c ok
   H:\Documents\MATLAB\my_first_code_ert_rtw\my_first_code.c ok
### Generating DRS table
   my_first_code_U.In1 min max init
   my_first_code_U.In2 min max init
### Computing code verification options
   ...
### Starting code verification
```

The exact messages depend on the code generator you use. However, the messages always have the same format:

- Name of code generator
- Version number of the plug-in
- List of source files
- DRS (Data Range Specification) information.

Follow the progress of the verification in the MATLAB Command window. If you are running a server verification, you can follow the later stages of the verification through the Polyspace spooler (Queue Manager).

Note Verification of a 3,000 block model will take approximately one hour to verify, or about 15 minutes for each 2,000 lines of generated code.

Monitor Verification Progress

| In this section... |
|-------------------------------------|
| “Client Verifications” on page 4-10 |
| “Server Verifications” on page 4-10 |

Client Verifications

For client verifications, you can follow the progress of the verification in the MATLAB Command window. The software also saves all status messages to a log file in the results folder. For example:

```
Polyspace_R2012b_my_first_code_05_16_2012-18h40.log
```

Server Verifications

For server verifications, you can follow the initial stages of the verification in the MATLAB Command window.

Once the compilation phase is complete, you can follow the progress of the verification using either the Polyspace Queue Manager (Spooler), or the Polyspace Project Manager.

Monitor Progress from Polyspace Queue Manager

To open the Polyspace Queue Manager, do one of the following:

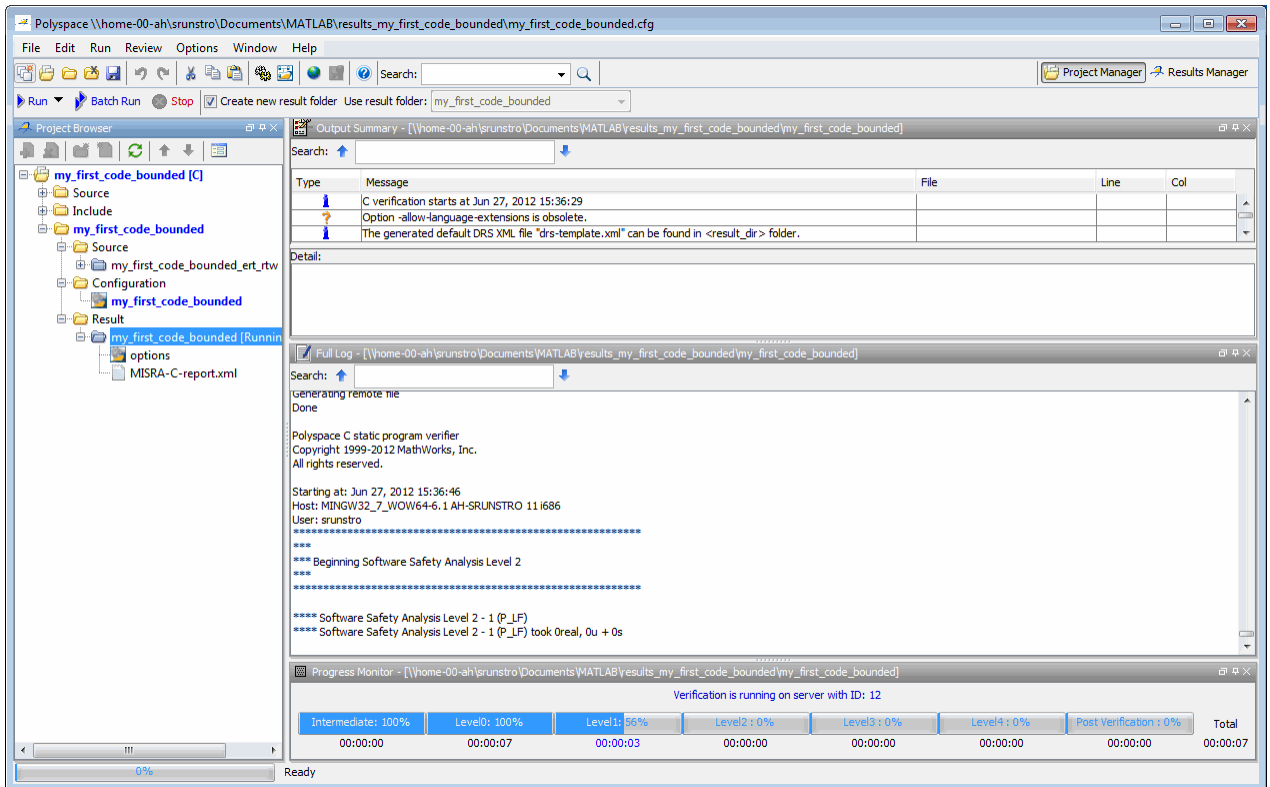
- From the Simulink model window, select **Code > Polyspace > Open Spooler**.
- Right-click a Model block or subsystem. From the context menu, select **Polyspace > Open Spooler**.

For more information, see “Verification Job Management”.

Monitor Progress from Polyspace Project Manager

If you configure the Polyspace Model Link options to **Open the Polyspace Project Manager and Results Manager**, you can monitor the progress of a server verification from the Project Manager.

The Project Manager opens automatically at the end of the compilation phase. The Progress Monitor window at the bottom of the Project Manager displays the progress of the verification.



For more information on monitoring progress from the Project Manager, see “Monitor Progress of Verification”.

For more information on configuring the software to open the Project Manager, see “Open Polyspace Project Manager Automatically” on page 3-22.

Code Generation and Verification with Configured Model

You can generate Embedded Coder code from the configured model `psdemo_model_link_sl`. You can then run a Polyspace verification on the generated code.

To open `psdemo_model_link_sl` in the Simulink model window:

- 1 In the MATLAB Command Window, enter `psdemo_model_link_sl`.

This command opens the `psdemo_model_link_sl` model that is compatible with your version of MATLAB (either `psdemo_model_link_sl`, `psdemo_model_link_sl_v1`, or `psdemo_model_link_sl_v2`).

To generate code and start the Polyspace verification:

- 1 Double-click the Re-install the demo block to generate the legacy code related to the S-function.
- 2 If you want to apply data ranges to the input parameters, double-click the green block Use input constraints. To remove the data range constraints, double-click the orange block Worst case inputs.
- 3 Right-click the subsystem controller.
- 4 From the context-menu, select **C/C++ Code > Build This Subsystem**.
- 5 In the Build code for Subsystem dialog box, click **Build** to generate code. When the code generation is complete, the code generation report opens.
- 6 Server verification is specified by default. If you want to specify client verification, select **Code > Polyspace > Options**. When the **Configuration Parameters > Polyspace Model Link** pane opens, clear the **Send to Polyspace server** check box. Then click **OK**.
- 7 Right-click the subsystem controller. From the context menu, select **Polyspace > Polyspace for Embedded Coder**. The verification starts.

To monitor the progress of the verification:

- If you specified server verification, select **Code > Polyspace > Open Spooler**. Use the Polyspace Queue Manager (Spooler) to monitor progress.
- If you specified client verification, you can monitor progress from the Command Window.

Once the verification is complete, to display the results:

- 1** Select **Code > Polyspace > Open Results > For Generated Code**.
- 2** In the Polyspace environment, select **File > Open Result**.
- 3** Use the Open Results dialog box to navigate to the specified results folder, for example, `C:\Polyspace_Results\controller`.
- 4** Select the results file, for example, `RTE_px_controller_LAST_RESULTS.rte`. Then click **Open**. The software displays the results in the Results Manager perspective.

MATLAB Functions For Polyspace Batch Runs

In addition to `pslinkrun` and `pslinkoptions`, you can run the following commands in the Command Window.

| Command | Description |
|--|--|
| <code>PolySpaceSpooler</code> | Open the Polyspace Queue Manager (Spooler), which allows you to manage server verifications. |
| <code>PolySpaceViewer</code> | Open Polyspace verification environment Results Manager perspective. |
| <code>PolySpaceSetTemplateSECFile</code> | Set the template file, for example, during a batch run. |
| <code>PolySpaceGetTemplateCFFile</code> | Get the currently selected template file (empty by default). |
| <code>PolySpaceReconfigure</code> | In case of a Polyspace release update without enabling the MATLAB plug-in. |
| <code>PolySpaceAnnotation</code> | Annotate block in Simulink model, which then appears in Polyspace results. You can annotate either run-time checks or coding rule violations, and provide a classification, status, and comment for each annotation. |
| <code>ver</code> | Display version numbers of MathWorks® products, including Polyspace Model Link products. |

Archive Files for Polyspace Verification

In this section...

“Template File in *MATLAB Installation folder*\polyspace\” on page 4-15

“Files Used in Model Folder” on page 4-15

“Auto-Generated Files in Model Folder” on page 4-16

Template File in *MATLAB Installation folder*\polyspace\

When a verification is first performed, the software creates a copy of the file `cfg\templateEmbeddedCoder.cfg` in the local model folder, `model_folder\model_name-polyspace.cfg`. The software does not create a copy in subsequent verifications.

The file `cfg\templateEmbeddedCoder.cfg` contains the template Polyspace configuration settings to support the TargetLink code generator. The `templateTargetLink.cfg` file can be updated with site specific settings, to facilitate verification of new models.

You can use the MATLAB command `PolyspaceSetTemplateCFGFile(config_filename)` to change the name and location of the file that contains the template configuration. For example, when you run Polyspace verification as part of an automated process, which specifies the template configuration file, erases the local copy in the model folder, and starts the verification.

Files Used in Model Folder

- `model-name-polyspace.cfg` — As mentioned above this file is copied from the MATLAB `installation_folder\polyspace\cfg\templateEmbeddedCoder.cfg` file the first time a verification is run on a model. It is subsequently modified by the Project Configuration block, or the Configure button in the option

in the Polyspace Analyzer dialog. It contains the Polyspace settings for verifying the current model.

- `polyspace_additional_file_list.txt` — This file is created if the Advanced option, Select Files is used in the Polyspace Analyzer dialog box. This option allows files that are not part of the model to be analyzed together with the model. For example these files could contain custom lookup table code, custom stubs, device driver code etc. The Enable additional file list option needs to be set together with configuring the list of extra files to analyze.

Auto-Generated Files in Model Folder

These files are generated from the model for each verification when it is started, and do not need archiving:

- `model_name_drs.txt` — The DRS information extracted automatically from the model.
- `polyspace_include_dir_list.txt` — List of compilation include directories extracted from the mode.
- `polyspace_file_list.txt` — List of file contained in the model to analyze
- `model_name_last_parameter.txt` — The last set of parameters used in the Polyspace Analyzer dialog box.

Review Verification Results

- “View Results in Polyspace Verification Environment” on page 5-2
- “Identify Errors in Simulink Models” on page 5-5

View Results in Polyspace Verification Environment

When a verification completes, you can view the results using the Results Manager perspective of the Polyspace verification environment.

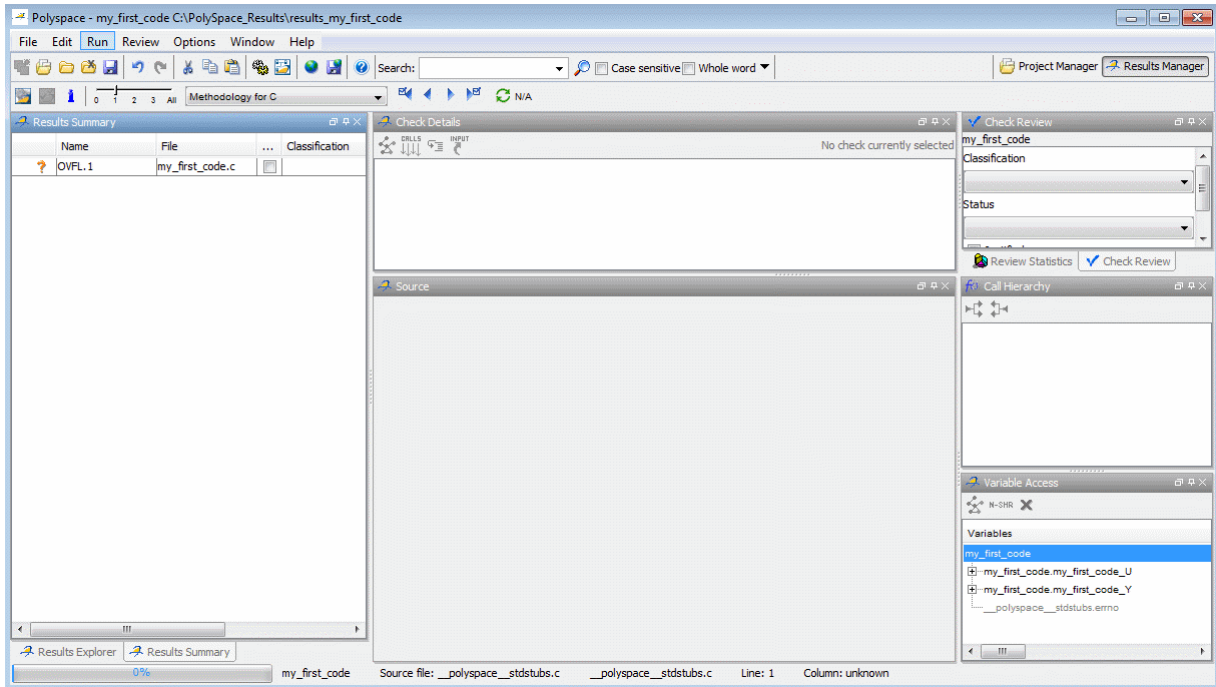
To view your results:

- 1 From the Simulink model window, select **Code > Polyspace > Open Results**.

Note If you set **Model reference verification depth** to All and selected **Model by model verification**, the Select the Result Folder to Open in Polyspace dialog box opens. The dialog box displays a hierarchy of referenced models from which the software generates code. To view the verification results for code generated from a specific model, select the model from the hierarchy. Then click **OK**.

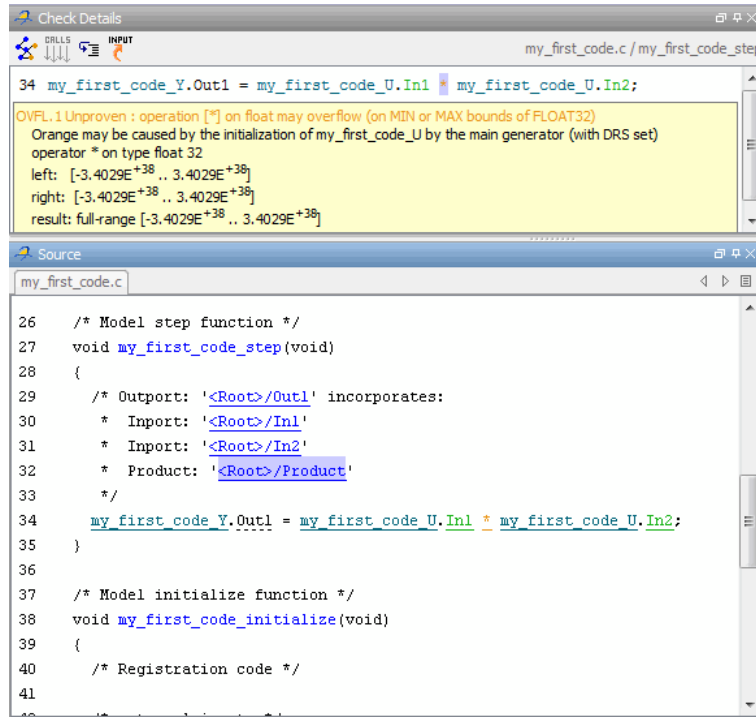
You can also open results through a Model block or subsystem. From the Simulink model window, right-click the Model block or subsystem, and from the context menu, select **Polyspace > Open Results**.

After a few seconds, the Results Manager perspective of the Polyspace verification environment opens.



2 On the **Results Summary** tab, click any check to review additional information.

In this example, the **Check Details** pane shows information about the orange check, and the **Source** pane shows the source code containing the orange check.



For more information on reviewing run-time checks, see “Run-Time Error Review”.

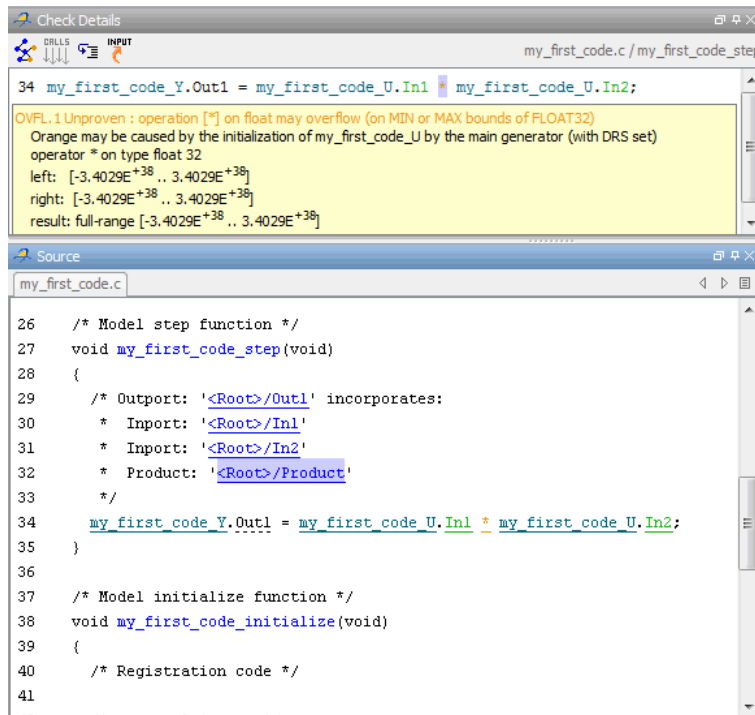
For information on specific checks, see “Run-Time Check Reference”.

Note When you run verification on a Polyspace server, the software automatically downloads your results to the client when verification completes. You can also download server results manually. For more information, see “Download Results from Server to Client”.

Identify Errors in Simulink Models

Polyspace Model Link products allow you to trace run-time checks in your verification results directly to your Simulink model.

Consider the following example, where the **Check Details** pane shows information about an orange check, and the **Source** pane shows the source code containing the orange check.



This orange check shows a potential overflow issue when multiplying the signals from the inports In1 and In2. To fix this issue, you must return to the model.

To trace this run-time check to the model:

- 1 Click the blue underlined link ([<Root>/Product](#)) immediately before the check in the **Source** pane. The Simulink model opens, highlighting the block with the error.
- 2 Examine the model to find the cause of the check.

In this example, the highlighted block multiplies two full-range signals, which could result in an overflow. This could be a flaw in:

- Design — If the model is supposed to be robust for the full signal range, then the issue is a design flaw. In this case, you must change the model to accommodate the full signal range. For example, you could saturate the output of the previous block, or bound the signal with a Switch block.
- Specifications — If the model is supposed to work for specific input ranges, you can provide these ranges using block parameters or the base workspace. The verification will then read these ranges from the model. See “Specify Signal Ranges” on page 2-9.

Applying either solution should address the issue and cause the orange check to turn green.

If your operating system is Windows Vista™ or Windows® 7, you may encounter problems with the trace-back functionality if one of the following conditions apply:

- User Account Control (UAC) is enabled.
- You do not have administrator privileges.

If you have a MATLAB session running and your model is open, a possible workaround is:

- 1 Open a DOS window in administrator mode.
- 2 Go to your MATLAB installation folder.
- 3 From the bin folder, enter `matlab -regserver`.
- 4 Click the link again.

Functions

pslinkrun

Purpose Run Polyspace verification from MATLAB command line

Syntax

```
resultsFolder = pslinkrun
resultsFolder = pslinkrun(system)
resultsFolder = pslinkrun(system,opts)
resultsFolder = pslinkrun(system,opts,topModelRef)
```

Description `resultsFolder = pslinkrun` runs a Polyspace verification on generated code from the current system and returns the location of the results folder. It uses the configuration options associated with the current system. The current system, or model, is the system returned by the command `bdroot`.

`resultsFolder = pslinkrun(system)` runs a verification on the code generated from the model or subsystem specified by `system`. This verification uses the configuration options associated with `system`.

`resultsFolder = pslinkrun(system,opts)` verifies `system` using the configuration options from the options object `opts`.

`resultsFolder = pslinkrun(system,opts,topModelRef)` uses `topModelRef` to specify which generated code to verify.

Input Arguments

system - Model or system

`bdroot` (default) | model or system name

Model or system that you want to verify, specified as a string with the model or system name in single quotes. The default value is the system returned by `bdroot`.

Example: `resultsFolder = pslinkrun('demoC')`. `demoC` is the name of a model.

Data Types

char

opts - Configuration options

options associated with system (default) | Polyspace Model Link options object

Configuration options for the verification, specified as an options object or the options already associated with the model or system. The function pslinkoptions creates an options object.

Example: pslinkrun('demoC', opts_demo). demoC is the name of a model and opts_demo is an options object.

topModelRef - Indicator for model reference verification

false (default) | true

Indicator for model reference verification, specified as true or false.

- If topModelRef is false (default), the software verifies all code generated from the top model including all the referenced models.
- If topModelRef is true, Polyspace verifies only code generated from models referenced by the top model.

This verification choice is equivalent to choosing between **Verify Generated Code** and **Verify Generated Model Reference Code** in the Simulink Polyspace options.

Data Types

logical

Output Arguments

resultsFolder - Variable for location of the results folder

string

Variable for location of the results folder, specified as a string. The default value of this variable is results_\$. This value can be changed in the configuration options using pslinkrun.

Data Types

char

Examples

Run a Polyspace Verification from the Command Line

Use a Simulink model to generate code, set configuration options, and then run a verification from the command line.

Open the Simulink model `datatypedemo`:

```
open('datatypedemo')
```

From the Simulink menu, select **Code > C/C++ Code > Code Generation Options**, and then select the **Code Generation** pane.

Change the **System target file** to `ert.tlc` for Embedded Coder.

Clear the **Generate makefile** option.

Click **Apply**.

Select **Code > C/C++ Code > Build Model** to build the model and generate code.

From the MATLAB command line, create a Polyspace options object from the model:

```
opts = pslinkoptions('datatypedemo');
```

Change the configuration to run the verification in client mode, and check MISRA C coding rules:

```
opts.SendToPolyspaceServer = false;  
opts.VerificationSettings = 'PrjConfigAndMisra'
```

```
opts =
```

```
          ResultDir: 'results_$(ModelName$'  
VerificationSettings: 'PrjConfigAndMisra'  
  OpenProjectManager: 0  
  AddSuffixToResultDir: 0  
EnableAdditionalFileList: 0  
  AdditionalFileList: {0x1 cell}
```



```
SendToPolyspaceServer: 0
    InputRangeMode: 'DesignMinMax'
    ParamRangeMode: 'None'
    OutputRangeMode: 'None'
    ModelRefVerifDepth: 'Current model only'
ModelRefByModelRefVerif: 0
CxxVerificationSettings: 'PrjConfig'
```

Run a verification:

```
results = pslinkrun('datatypedemo',opts)
```

See Also

`pslinkoptions` | “MATLAB Functions For Polyspace Batch Runs”
on page 4-14 |

pslinkoptions

Purpose Creates options object to customize Polyspace verification from MATLAB command line

Syntax
`opts = pslinkoptions(codegen)`
`opts = pslinkoptions(model)`

Description `opts = pslinkoptions(codegen)` returns an options object with the configuration options for code generated by `codegen`.

`opts = pslinkoptions(model)` returns an options object with the configuration options for the Simulink model `model`.

Input Arguments

codegen - Code generator

`'ec' | 'tl'`

Code generator, specified as either `'ec'` for Embedded Coder or `'tl'` for TargetLink. Each argument creates a Polyspace options object with configuration options specific to that code generator.

For a description of all configuration options and their values, see Configuration Options on page 6-7.

Example: `embedded_coder_optsObj = pslinkoptions('ec')`

Example: `target_link_optsObj = pslinkoptions('tl')`

Data Types

`char`

model - Simulink model

`model name`

Simulink model, specified by the model name. Creates a Polyspace options object with the configuration options of that model. If no options have been set, the object has all default configuration options. If a code generator has been set, the object has the default options for that code generator.

For a description of all configuration options and their values, see Configuration Options on page 6-7.

Example: `model_optsObj = pslinkoptions('my_model')`

Data Types

char

Output Arguments

opts - Polyspace configuration options

options object

Polyspace configuration options, returned as an options object. The object is used with `pslinkrun` to run a verification from the MATLAB command line.

The following table provides possible values and a description for each configuration option. Depending on the code generator, the object will have different configuration options. The value in curly brackets {} is the default.

Configuration Options

| Configuration Option | Values | Description |
|----------------------|--|---|
| ResultDir | {'C:\Polyspace\Results\results_\$(ModelName)'} Specify the location of the results folder. | Can be either an absolute path or a path relative to the current folder. |
| VerificationSettings | {'PrjConfig'} 'PrjConfigAndMisraAGC' 'PrjConfigAndMisraAGC' 'MisraAGC' 'Misra' | Specify checking of coding rules for C verification: 'PrjConfig' – Inherit all options from project configuration and run complete verification. 'PrjConfigAndMisraAGC' – Inherit all options from project configuration, enable MISRA AC AGC rule checking, and run complete verification. 'PrjConfigAndMisra' – Inherit all options from project configuration, enable MISRA C rule checking, and run complete verification. |

pslinkoptions

Configuration Options (Continued)

| Configuration Option | Values | Description |
|--|---|--|
| | | <p>'MisraAGC' – Enable MISRA AC AGC rule checking, and run compilation phase only.</p> <p>'Misra' – Enable MISRA C rule checking, and run compilation phase only.</p> |
| CxxVerificationSettings <i>Only for Polyspace Model Link SL</i> | {'PrjConfig'} {'PrjConfigAndMisraCxx'} {'PrjConfigAndJSF'} {'MisraCxx'} {'JSF'} | <p>Specify checking of coding rules for C++ verification: 'PrjConfig' – Inherit all options from project configuration and run complete verification.</p> <p>'PrjConfigAndMisraCxx' – Inherit all options from project configuration, enable MISRA C++ rule checking, and run complete verification.</p> <p>'PrjConfigAndJSF' – Inherit all options from project configuration, enable JSF rule checking, and run complete verification.</p> <p>'MisraCxx' – Enable MISRA C++ rule checking, and run compilation phase only.</p> <p>'JSF' – Enable JSF rule checking, and run compilation phase only.</p> |
| OpenProjectManager | {false} true | Open Polyspace Project Manager to monitor verification progress. When verification is complete, you can switch to the Results Manager perspective to review the results. |
| AddSuffixToResultDir | {false} true | Modify location of results folder by appending a unique number to the folder name instead of overwriting an existing folder. |

Configuration Options (Continued)

| Configuration Option | Values | Description |
|---|--------------------------------|---|
| EnableAdditionalFileList | {false} true | Specify whether additional files must be verified. You can specify these additional files through the AdditionalFileList option |
| AdditionalFileList | {0x1 cell} | List additional files to verify. |
| SendToPolyspaceServer | false {true} | Select client or server verification. |
| InputRangeMode | {'DesignMinMax' 'FullRange'} | Specify whether to use data ranges defined in blocks and workspace or treat inputs as full-range values. |
| ParamRangeMode | {'None' 'DesignMinMax'} | Specify whether to use constant values of parameters specified in the code, or use a range defined in blocks and workspace. |
| OutputRangeMode | {'None' 'DesignMinMax'} | Specify whether to apply assertions to outputs (using a range defined in blocks and workspace). |
| AutoStubLUT <i>Only for Polyspace Model Link TL.</i> | {false} true | Specify whether to include Lookup Table code in the verification. |

pslinkoptions

Configuration Options (Continued)

| Configuration Option | Values | Description |
|--|--|--|
| ModelRefVerifDepth <i>Only for Polyspace Model Link SL.</i> | {'Current model only'} '1' '2' '3' 'All' | Specify verification of generated code with respect to model reference hierarchy levels. |
| ModelRefByModelRefVerif <i>Only for Polyspace Model Link SL</i> | {false} true | Specify whether to verify code from models within model reference hierarchies jointly or separately. |

Examples

Use a Simulink model to create and edit an options objects

Open the Simulink model `datatypedemo`:

```
open('datatypedemo')
```

From the MATLAB command line, create a Polyspace options object from the model:

```
model_optsObj = pslinkoptions('datatypedemo')
```

```
model_optsObj =
```

```
ResultDir: 'results_ $ModelName$'  
VerificationSettings: 'PrjConfig'  
OpenProjectManager: 0  
AddSuffixToResultDir: 0  
EnableAdditionalFileList: 0  
AdditionalFileList: {0x1 cell}  
SendToPolyspaceServer: 1  
InputRangeMode: 'DesignMinMax'  
ParamRangeMode: 'None'  
OutputRangeMode: 'None'
```

```
ModelRefVerifDepth: 'Current model only'  
ModelRefByModelRefVerif: 0  
AutoStubLUT: 0  
CxxVerificationSettings: 'PrjConfig'
```

The model does not have a specific code generator set, so all configuration options appear.

Change the results folder name and set the configuration to open the Polyspace Project Manager:

```
model_optsObj.ResultDir = 'results_1_$ModelName$';  
model_optsObj.OpenProjectManager = 1;
```

```
model_optsObj =
```

```
ResultDir: 'results_1_$ModelName$'  
VerificationSettings: 'PrjConfig'  
OpenProjectManager: 1  
AddSuffixToResultDir: 0  
EnableAdditionalFileList: 0  
AdditionalFileList: {0x1 cell}  
SendToPolyspaceServer: 1  
InputRangeMode: 'DesignMinMax'  
ParamRangeMode: 'None'  
OutputRangeMode: 'None'  
ModelRefVerifDepth: 'Current model only'  
ModelRefByModelRefVerif: 0  
AutoStubLUT: 0  
CxxVerificationSettings: 'PrjConfig'
```

Create and edit an options object for Embedded Coder at the command line

Create a Polyspace options object called `new_optsObj` with Embedded Coder parameters:

```
new_optsObj = pslinkoptions('ec')
```

pslinkoptions

```
new_optsObj =  
  
        ResultDir: 'results_$(ModelName$'  
    VerificationSettings: 'PrjConfig'  
        OpenProjectManager: 0  
    AddSuffixToResultDir: 0  
EnableAdditionalFileList: 0  
    AdditionalFileList: {0x1 cell}  
    SendToPolyspaceServer: 1  
        InputRangeMode: 'DesignMinMax'  
        ParamRangeMode: 'None'  
        OutputRangeMode: 'None'  
    ModelRefVerifDepth: 'Current model only'  
    ModelRefByModelRefVerif: 0  
    CxxVerificationSettings: 'PrjConfig'
```

Change the `SendToPolyspaceServer` value of your object to run the verification in client mode:

```
new_optsObj.SendToPolyspaceServer = false
```

```
new_optsObj =  
  
        ResultDir: 'results_$(ModelName$'  
    VerificationSettings: 'PrjConfig'  
        OpenProjectManager: 0  
    AddSuffixToResultDir: 0  
EnableAdditionalFileList: 0  
    AdditionalFileList: {0x1 cell}  
    SendToPolyspaceServer: 0  
        InputRangeMode: 'DesignMinMax'  
        ParamRangeMode: 'None'  
        OutputRangeMode: 'None'  
    ModelRefVerifDepth: 'Current model only'  
    ModelRefByModelRefVerif: 0  
    CxxVerificationSettings: 'PrjConfig'
```


Change the configuration to check for both run-time errors and MISRA C coding rule violations:

```
new_optsObj.VerificationSettings = 'PrjConfigAndMisra'
```

```
new_optsObj =
```

```
                ResultDir: 'results_$modelName$'  
    VerificationSettings: 'PrjConfigAndMisra'  
      OpenProjectManager: 0  
    AddSuffixToResultDir: 0  
EnableAdditionalFileList: 0  
    AdditionalFileList: {0x1 cell}  
    SendToPolyspaceServer: 0  
      InputRangeMode: 'DesignMinMax'  
      ParamRangeMode: 'None'  
      OutputRangeMode: 'None'  
    ModelRefVerifDepth: 'Current model only'  
    ModelRefByModelRefVerif: 0  
    CxxVerificationSettings: 'PrjConfig'
```

See Also

[pslinkrun](#) | “MATLAB Functions For Polyspace Batch Runs” on page 4-14 |