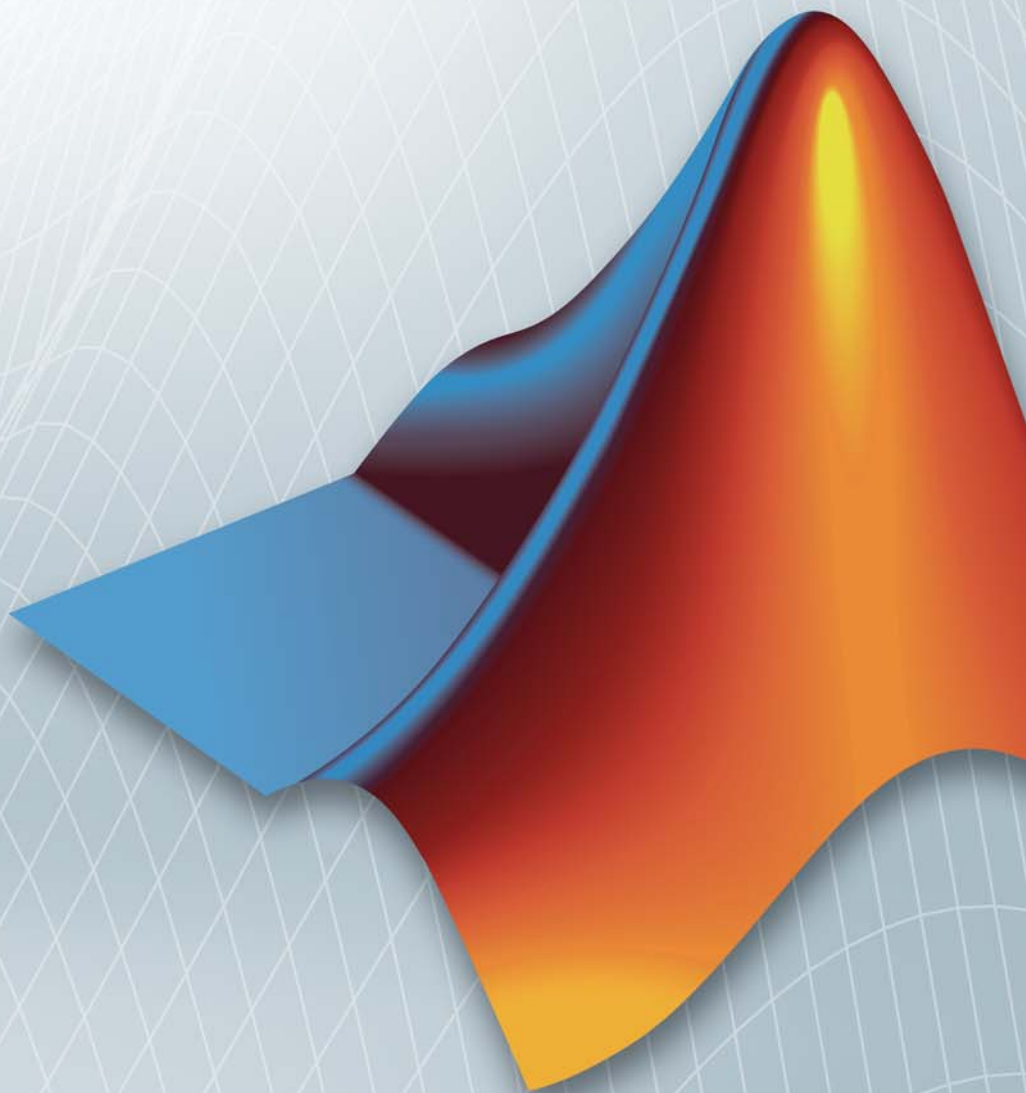


# Polyspace® Model Link Products

## User's Guide

**R2011b**



## How to Contact MathWorks



[www.mathworks.com](http://www.mathworks.com) Web  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab) Newsgroup  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html) Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com) Product enhancement suggestions  
[bugs@mathworks.com](mailto:bugs@mathworks.com) Bug reports  
[doc@mathworks.com](mailto:doc@mathworks.com) Documentation error reports  
[service@mathworks.com](mailto:service@mathworks.com) Order status, license renewals, passcodes  
[info@mathworks.com](mailto: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 1999–2011 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](http://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](http://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)



## Getting Started with Model Link Products

### 1

<b>Product Overview</b> .....	1-2
Polyspace Model Link SL .....	1-2
Polyspace Model Link TL .....	1-2
 <b>Basic Workflow</b> .....	 1-3
 <b>Tutorial: Verifying Code from a Simple Model</b> .....	 1-5
Create Simulink Model and Generate Code .....	1-5
Run Polyspace Verification .....	1-12
View Results in Polyspace Verification Environment .....	1-13
Trace Error to Simulink Model .....	1-16
Specify Signal Ranges .....	1-17
Verify Updated Model .....	1-19

## Configure Model for Code Verification

### 2

<b>Overview of Model Configuration for Code Generation and Verification</b> .....	2-2
 <b>Configuring Simulink Model for Code Verification</b> ...	 2-3
 <b>Recommended Model Settings for Code Verification</b> ..	 2-5
 <b>Checking Simulink Model Settings</b> .....	 2-7
 <b>Specifying Signal Ranges</b> .....	 2-9
Specifying Signal Range through Source Block Parameters .....	2-9

Specifying Signal Range through Base Workspace .....	2-11
<b>Annotating Blocks to Justify Known Checks or Coding-Rules Violations .....</b>	<b>2-14</b>

## Configure Code Verification Options

# 3

<b>Overview of Polyspace Configuration .....</b>	<b>3-2</b>
<b>Including Handwritten Code in Verification .....</b>	<b>3-3</b>
<b>Configuring Data Range Settings .....</b>	<b>3-5</b>
<b>Configuring Polyspace Analysis Options .....</b>	<b>3-7</b>
<b>Configuring Polyspace Project Properties .....</b>	<b>3-9</b>
<b>Creating a Polyspace Configuration File Template ...</b>	<b>3-10</b>
<b>Specifying Header Files for Target Compiler .....</b>	<b>3-13</b>
<b>Specifying Location of Results .....</b>	<b>3-14</b>
<b>Main Generation for Model Verification .....</b>	<b>3-15</b>
<b>Polyspace Model Link SL Considerations .....</b>	<b>3-17</b>
Overview .....	3-17
Subsystems .....	3-17
Default Options .....	3-17
Data Range Specification .....	3-18
Recommended Polyspace Analysis Options for Generated Code .....	3-18
<b>Polyspace Model Link TL Considerations .....</b>	<b>3-25</b>

Overview .....	3-25
Subsystems .....	3-25
Default Options .....	3-25
Data Range Specification .....	3-26
Lookup Tables .....	3-26
Code Generation Options .....	3-27

## Run Code Verification

### 4

<b>Running Verification with Polyspace Model Link SL</b>	
<b>Software</b> .....	4-2
<b>Running Verification with Polyspace Model Link TL</b>	
<b>Software</b> .....	4-4
<b>Monitoring Verification Progress</b> .....	4-7
Client Verifications .....	4-7
Server Verifications .....	4-7
<b>MATLAB Functions For Polyspace Batch Runs</b> .....	4-8
<b>Archive Files for Polyspace Verification</b> .....	4-9
Template File in <i>MATLAB Installation</i>	
<i>folder\polyspace\</i> .....	4-9
Files Used in Model Folder .....	4-9
Auto-Generated Files in Model Folder .....	4-10

## Review Verification Results

### 5

<b>Viewing Results in Polyspace Verification</b>	
<b>Environment</b> .....	5-2
<b>Identifying Errors in Simulink Models</b> .....	5-6





# Getting Started with Model Link Products

---

- “Product Overview” on page 1-2
- “Basic Workflow” on page 1-3
- “Tutorial: Verifying Code from a Simple Model” on page 1-5

## 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 which parts of the model are reliable, and correct design problems that will cause run-time errors in the code. 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 which parts of the model are reliable, and correct design problems that will cause run-time errors in the code. 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.

## Basic Workflow

With Embedded Coder or dSPACE® TargetLink software, you can generate C 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 appropriate.

---

The workflow for using Polyspace Model Link software is:

- 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 “Configuring Polyspace Analysis Options” on page 3-7.

---

- 3** Run Polyspace verification. See:
  - “Running Verification with Polyspace Model Link SL Software” on page 4-2
  - “Running Verification with Polyspace Model Link TL Software” on page 4-4

- 4** View results, analyze errors, locate and fix model faults. See “Viewing 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<sup>®</sup> chart in the Simulink model. See “Identifying Errors in Simulink Models” on page 5-6.

## Tutorial: Verifying Code from a Simple Model

### In this section...

“Create Simulink Model and Generate Code” on page 1-5

“Run Polyspace Verification” on page 1-12

“View Results in Polyspace Verification Environment” on page 1-13

“Trace Error to Simulink Model” on page 1-16

“Specify Signal Ranges” on page 1-17

“Identifying Errors in Simulink Models” on page 5-6

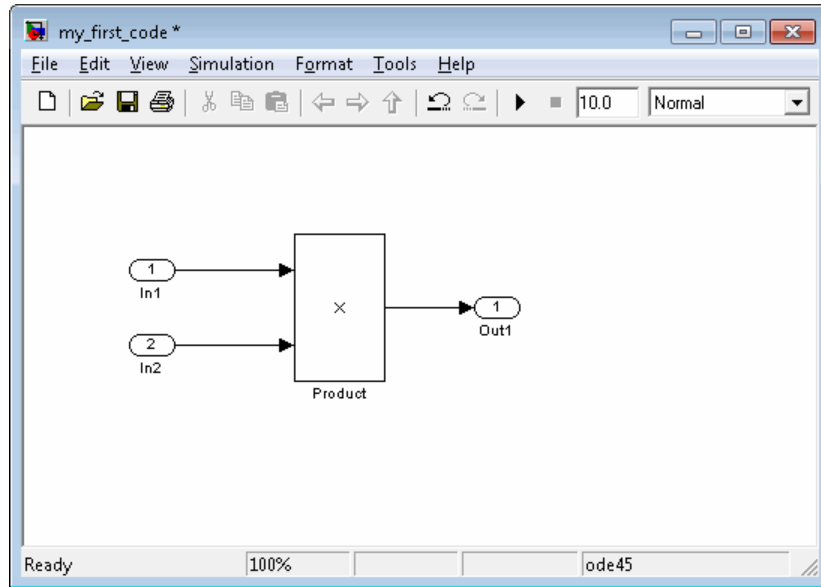
“Specify Signal Ranges” on page 1-17

“Verify Updated Model” on page 1-19

### Create Simulink Model and Generate Code

To create a simple Simulink model and generate code:

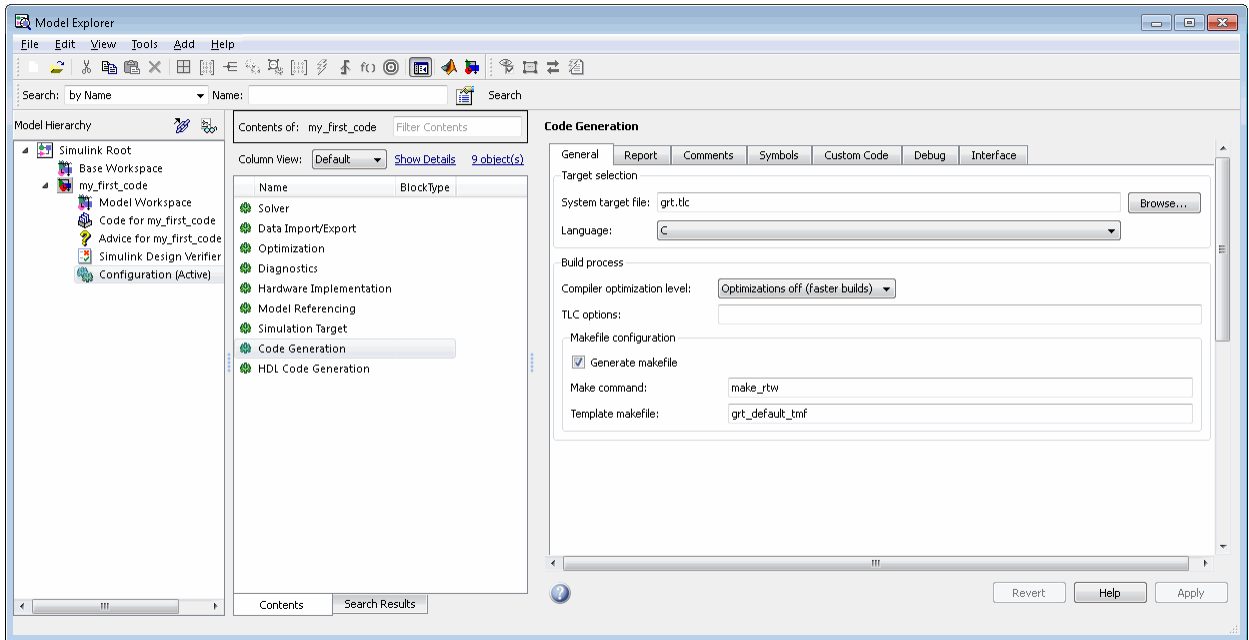
- 1 Open MATLAB®, then start Simulink software.
- 2 Construct the following model.



**3** Select **File > Save**, then name the model `my_first_code`.

**4** Select **View > Model Explorer**.

The Model Explorer opens.



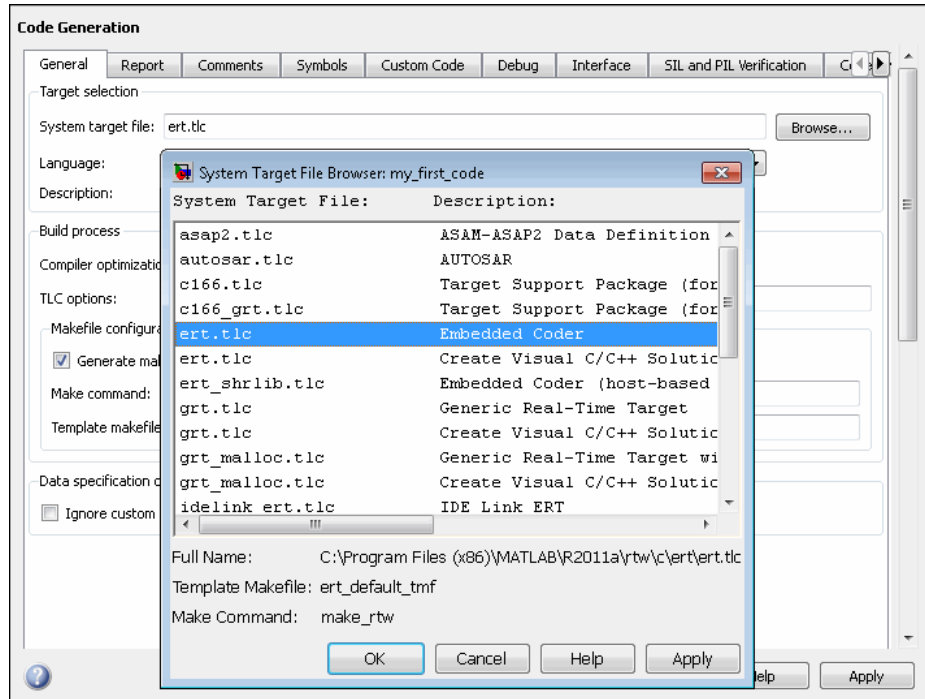
**5** Select **my\_first\_code > Configuration**, in the Model Hierarchy.

**6** Select **Code Generation** in the Configuration.

The Code Generation configuration parameters open.

**7** Select the **General** tab.

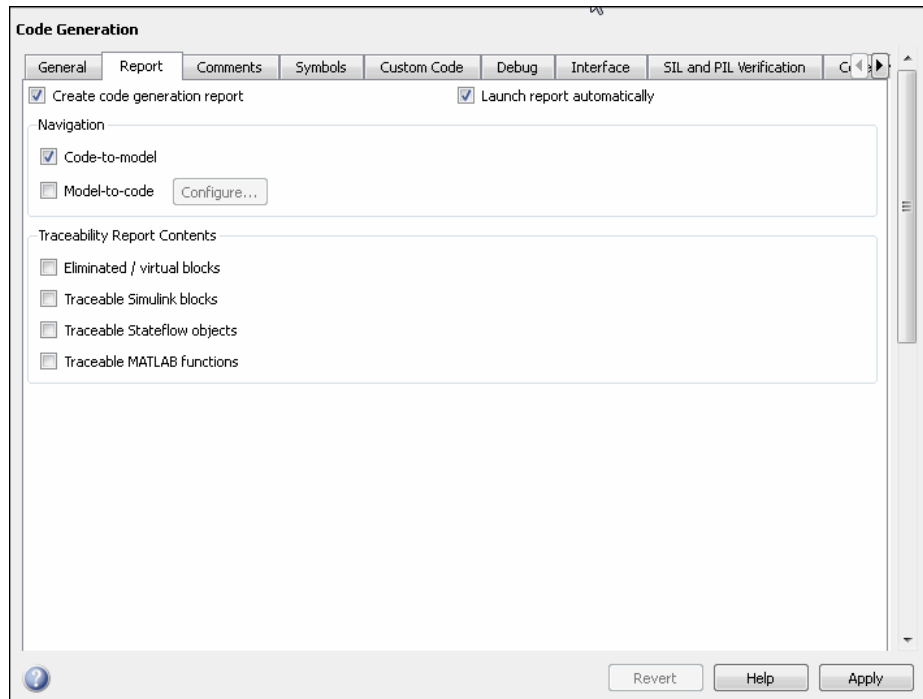
Set the **System target file** to **ert.tlc** (Embedded Coder).



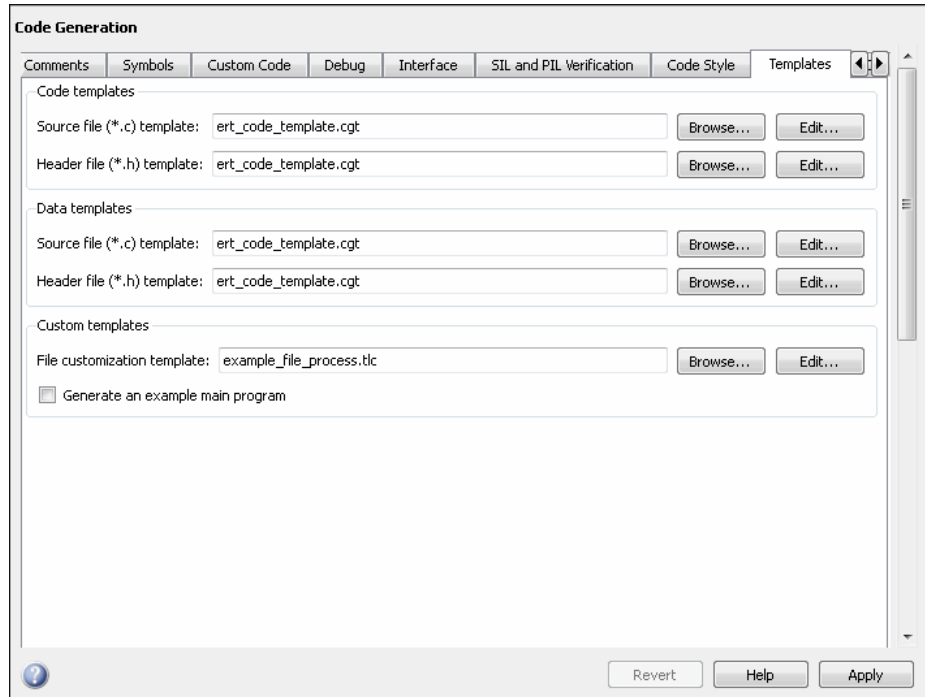
8 Select the **Report** tab.

9 Select **Create code-generation report**, then select **Code-to-model Navigation**.



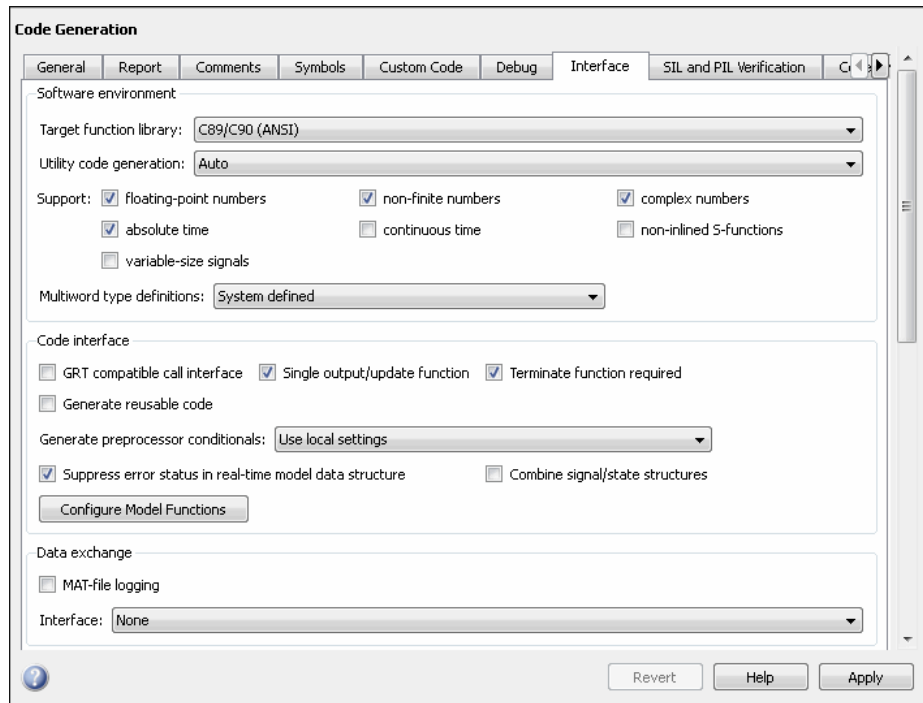


**10** Select the **Templates** tab.



**11** In the Custom templates section, clear **Generate an example main program**.

**12** Select the **Interface** tab.

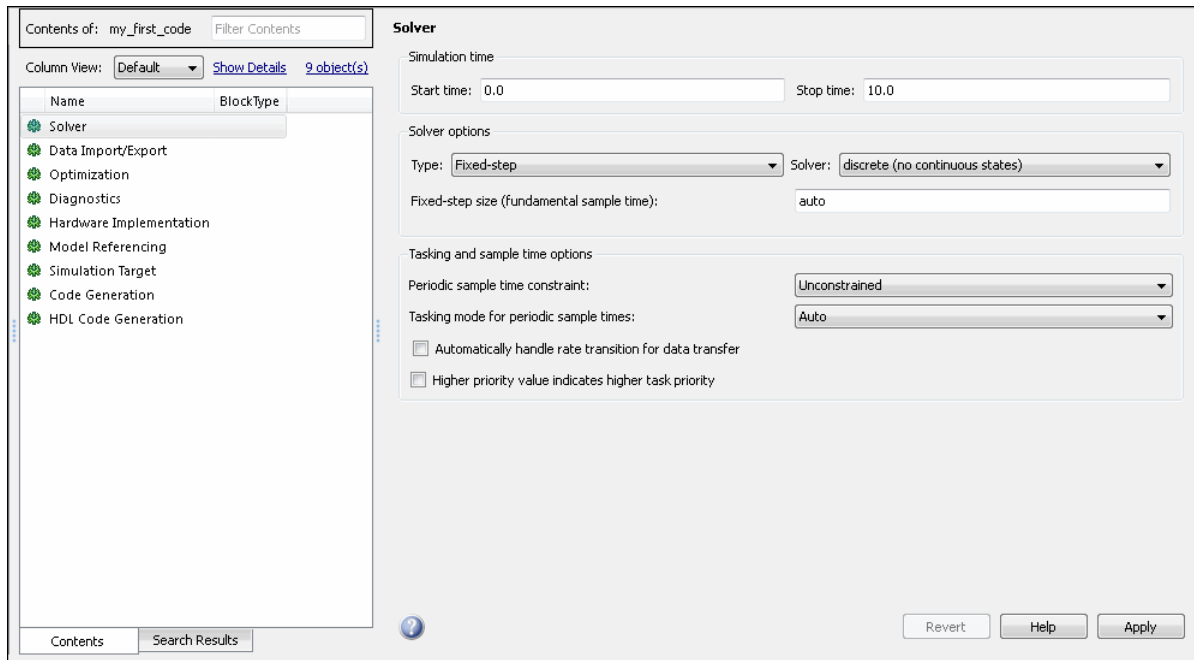


**13** In the Code interface section, select **suppress error status in real-time model data structure**.

**14** Click **Apply** in the lower-right corner of the window.

**15** In the Configuration Preferences, select **Solver**.

The Solver configuration parameters appear.



**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** From the Simulink model window, select **Tools > Code Generation > Build Model** to generate code.

**19** Save your Simulink model.

## Run Polyspace Verification

To start the Polyspace verification:

**1** From the Simulink model window, select **Tools > Polyspace > Run Verification**.

The verification starts, and you see messages 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
```

**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 Run-Time Checks perspective of the Polyspace verification environment.

To view your results:

**1** From the Simulink model window, select **Tools > Polyspace > Open results**.

After a few seconds, the Run-Time Checks perspective of the Polyspace verification environment opens.

# 1 Getting Started with Model Link Products

The screenshot displays the Polyspace IDE interface with several panels open. The main window shows the source code for `example.c` at line 157, column 5, with a comment: `Recursion( ex ); // always encounters a division by zero`. The `Recursion` function is defined as follows:

```
144 if ( *depth <= 50 )
145 {
146     Recursion( *depth );
147 }
148
149 }
150
151 static void Recursion_caller( void )
152 {
153     int x = random_int();
154
155     if ( (x > -4) && (x < -1) )
156     {
157         Recursion( ex ); // always encounters a division by zero
158     }
159
160
161     x = 10;
162     if ( random_int() >= 0 )
163     {
164         Recursion( ex ); /* never encounters a division by zero
165     }
166 }
167
```

The `Run-Time Checks` panel on the left shows a list of checks for `Demo_C` (coverage: 93%, unprocessed: 1/48). The `Check Review` panel shows details for the `Recursion` check, including its classification, status, and comment. The `Review Statistics` panel shows the following data:

Coding review progress	Count	Progr...
Red NTC justified / to justify	0/4	0
Red justified / to justify	0/8	0
Gray justified / to justify	0/6	0
Orange justified / to justify	0/18	0
Software reliability indicator	259/297	87

The `Source` panel shows the current file `example.c` with the source code. The `Call Hierarchy` panel shows the following calls:

Calls	Line
example.Recursion_caller	151
pst_stubs_0.random_int	152
example.Recursion	157
pst_stubs_0.random_int	162
example.Recursion	164
example.RTE	238

The `Variable Access` panel shows the following variables:

Variables	Detailed
Demo_C	
initialisations.arr	pointer to
initialisations.current_data	pointer to
initialisations.first_payload	int 32
initialisations.second_payload	int 32
initialisations.tab	array(0..9)
single_file_analysis.output_v1	int 8
single_file_analysis.output_v6	int 32

Labels with arrows point to the following panels:

- Run-time checks
- Source code
- Variable access
- Call hierarchy

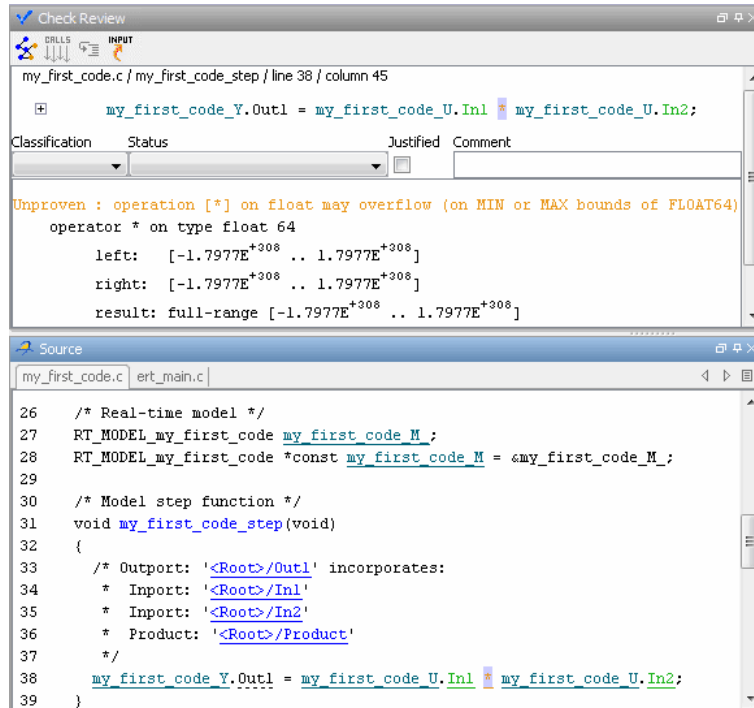
2 Type CTRL-N to go to the first error.

The screenshot shows the 'Run-Time Checks' window with a table of procedural entities. The table has columns for error counts (red exclamation mark, orange X, orange question mark, green checkmark), percentage, and line number. The 'my\_first\_code' entity is expanded to show several checks, with 'OVFL.1' highlighted in blue.

Procedural entities	!	X	?	✓	%	Line
my_first_code	0	0	1	23	96	
ert_main.c					0	1
my_first_code.c			1	4	80	1
_init_globals()					0	1
my_first_code_initialize()				2	100	42
my_first_code_step()			1	2	67	31
NIV.0				1		38
OVFL.1			1			38
NIV.2				1		38
my_first_code_terminate()					0	58
stdio.h					0	1
__polyspace__stdstubs.c				9	100	1
__polyspace_main.c				10	100	1

**3** Click the orange OVFL check.

The Check Review 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 inputs 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

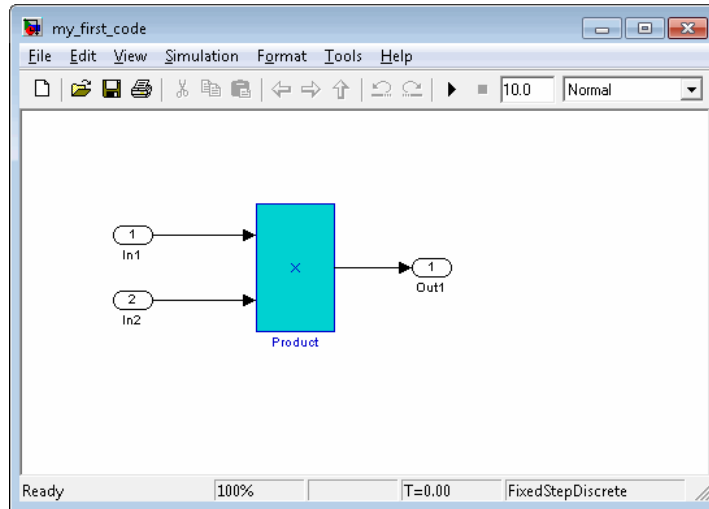
To fix this overflow issue, you must return to the Simulink model.

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 identified a potential bug. This could be a flaw in either:

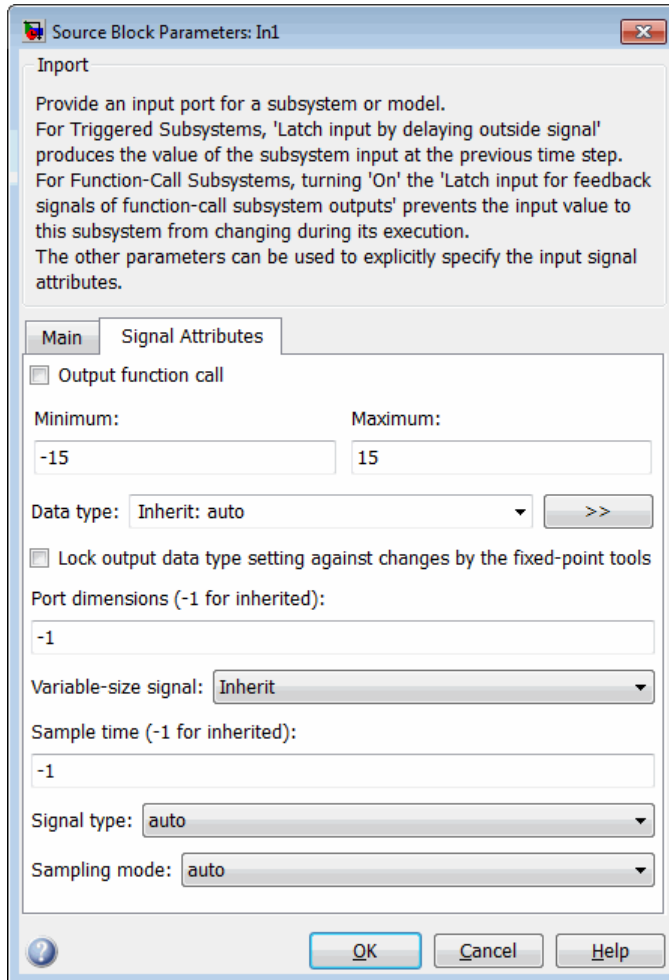
- **Design** — If the model should be robust against the full signal range, it is a design bug. In this case, you must change the model to accommodate the full signal range. For example, you could saturate the output of the previous block, or bound the signal with a Switch block.
- **Specifications** — If the model is designed to work within 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, and the check will turn 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.
- 2 The Source Block Parameters dialog box opens.



- 3 Select the **Signal Attributes** tab.
- 4 Set the **Minimum** value for the signal to -15.
- 5 Set the **Maximum** value for the signal to 15.

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

## **Verify Updated Model**

After changing your model, you must re-generate code and run verification again.

To regenerate code and relaunch verification:

- 1** From the Simulink model window, select **Tools > Code Generation > Build Model**.

The software generates code for the updated model.

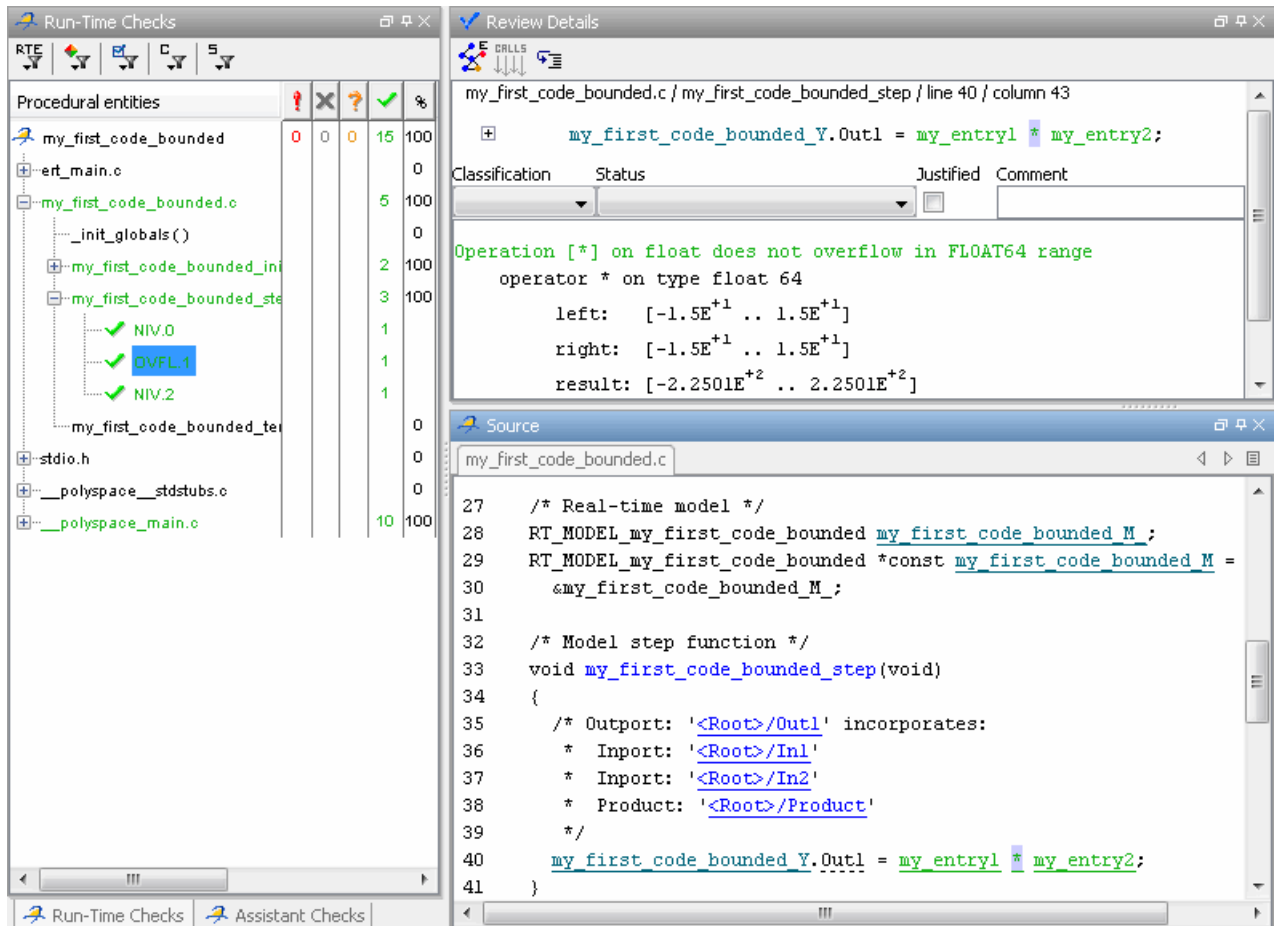
- 2** Select **Tools > Polyspace > Run Verification**.

The software verifies the generated code.

- 3** Select **Tools > Polyspace > Open results**.

Verification results open in the Polyspace verification environment.

- 4** Examine the results in the Run-Time Checks perspective.



The OVFL check is now green. Polyspace verification has confirmed that no Runtime Errors are present in the model.

# Configure Model for Code Verification

---

- “Overview of Model Configuration for Code Generation and Verification” on page 2-2
- “Configuring Simulink Model for Code Verification” on page 2-3
- “Recommended Model Settings for Code Verification” on page 2-5
- “Checking Simulink Model Settings” on page 2-7
- “Specifying Signal Ranges” on page 2-9
- “Annotating Blocks to Justify Known Checks or Coding-Rules Violations” on page 2-14

# Overview of Model Configuration for Code Generation and Verification

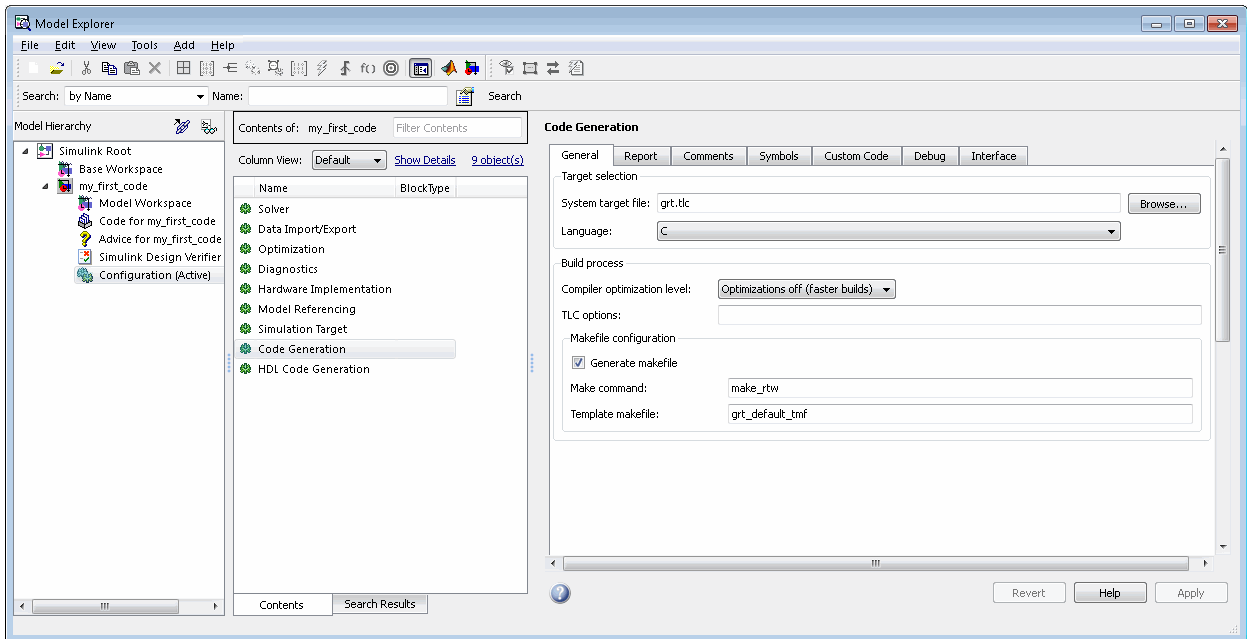
To facilitate Polyspace code verification and the review of results:

- There are certain settings that you should apply to your model before generating code. See “Recommended Model 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 “Checking Simulink Model Settings” on page 2-7
- You can constrain signals in your model to lie within specified ranges. See “Specifying Signal Ranges” on page 2-9.
- You can highlight blocks that you know contain checks or coding rule violations. See “Annotating Blocks to Justify Known Checks or Coding-Rules Violations” on page 2-14.

# Configuring Simulink Model for Code Verification

To configure a Simulink model for code generation and verification:

- 1 Open Model Explorer for your model.



- 2 In the Configuration for your model, select **Code Generation**.

The Code Generation configuration parameters open.

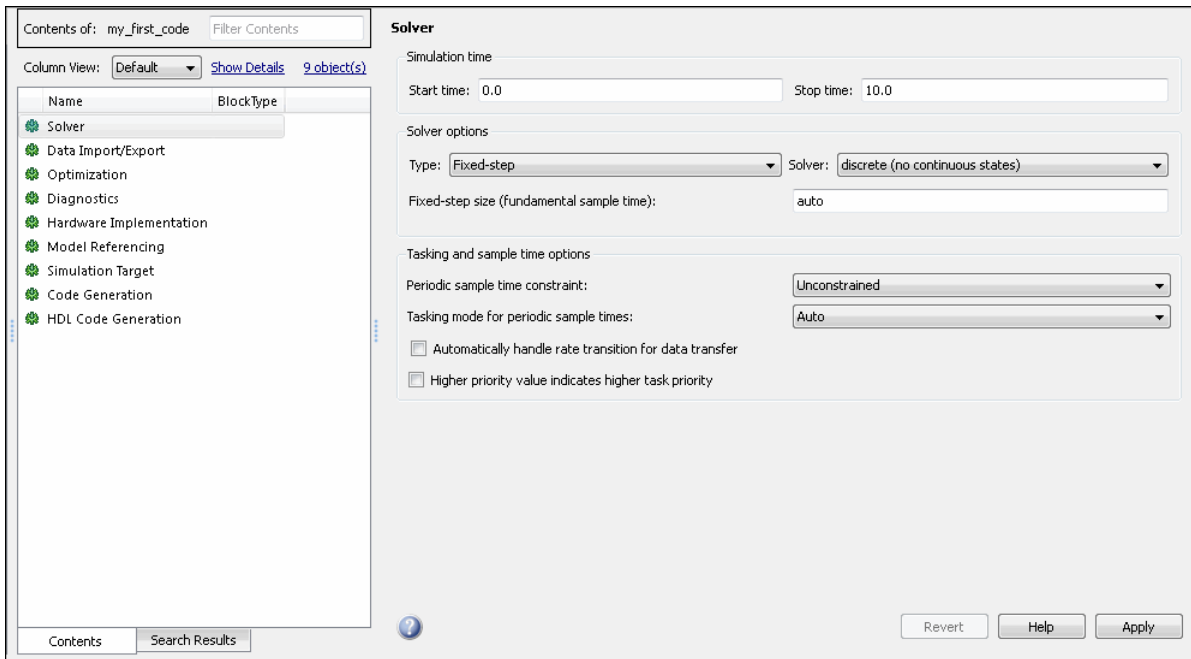
- 3 In the **General** tab, set the **System target file** to **ert.tlc** (Embedded Coder).
- 4 In the **Report** tab, select:
  - **Create code-generation report**
  - **Code-to-model** Navigation.
- 5 In the **Templates** tab, clear **Generate an example main program**.

**6** In the **Interface** tab, select **suppress error status in real-time model data structure**.

**7** Click **Apply** to save your changes.

**8** In the Configuration for your model, select **Solver**.

The Solver configuration parameters appear.



**9** In the Solver options section, set:

- **Type** to Fixed-step.
- **Solver** to discrete (no continuous states).

**10** Click **Apply** to save your changes.

**11** Save your model.



## Recommended Model Settings for Code Verification

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

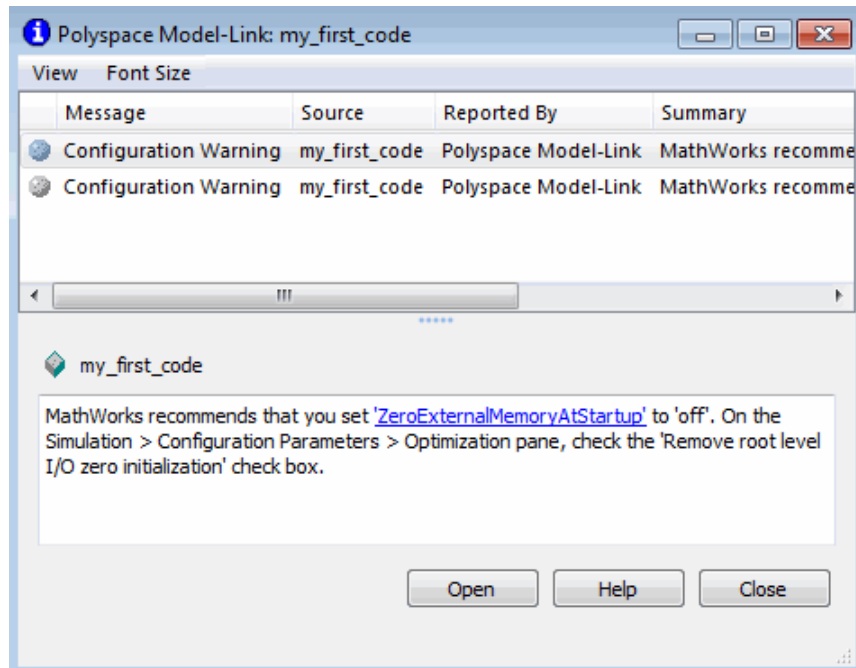
Parameter	Recommended value for Polyspace verification	How you specify value in Configuration Parameters dialog box	If you do not use recommended value, Polyspace Model Link SL generates ...
InitFltsAndDbls ToZero	'on'	Select check box <b>Optimization &gt; Use memset to initialize floats and doubles to 0.0</b>	Warning
InlineParams	'on'	Select check box <b>Optimization &gt; Signals and Parameters &gt; Inline parameters</b>	Warning
MatFileLogging	'off'	Clear check box <b>Code Generation &gt; Interface &gt; MAT-file logging</b>	Warning
MultiInstanceERT Code	'off'	Clear check box <b>Code Generation &gt; Interface &gt; Generate reusable code</b>	Warning
Solver	'FixedStepDiscrete'	Select discrete (no continuous states) from <b>Solver &gt; Solver</b> drop-down list	Warning
SystemTargetFile	'ert.tlc'	Specify ert.tlc (for Embedded Coder) in <b>Code Generation &gt; System target file</b>	Error

<b>Parameter</b>	<b>Recommended value for Polyspace verification</b>	<b>How you specify value in Configuration Parameters dialog box</b>	<b>If you do not use recommended value, Polyspace Model Link SL generates ...</b>
TargetLang	'C'	Select C from <b>Code Generation &gt; Language</b> drop-down list	Error
ZeroExternalMemoryAtStartup	'off' when <b>Configuration Parameters &gt; Polyspace Model Link &gt; Data Range Management &gt; Output is Global assert</b>	Clear check box <b>Optimization &gt; Remove root level I/O zero initialization</b>	Warning

## Checking 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 **Tools > 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 Model 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.

---

## Specifying 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 “Specifying Signal Range through Source Block Parameters” on page 2-9.
- Constraining signals through the base workspace. See “Specifying 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.

---

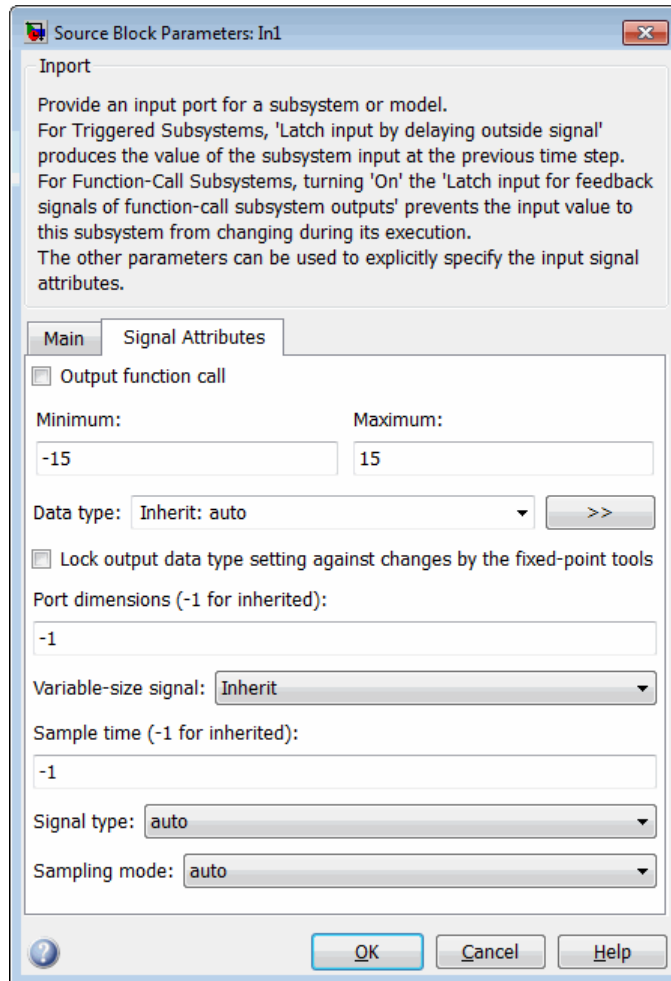
### Specifying Signal Range through Source Block Parameters

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 “Specifying 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, for example, In1.
- 2** The Source Block Parameters dialog box opens.



- 3 Select the **Signal Attributes** tab.
- 4 Set the **Minimum** value for the signal to -15.
- 5 Set the **Maximum** value for the signal to 15.
- 6 Click **OK**.

## Specifying 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.

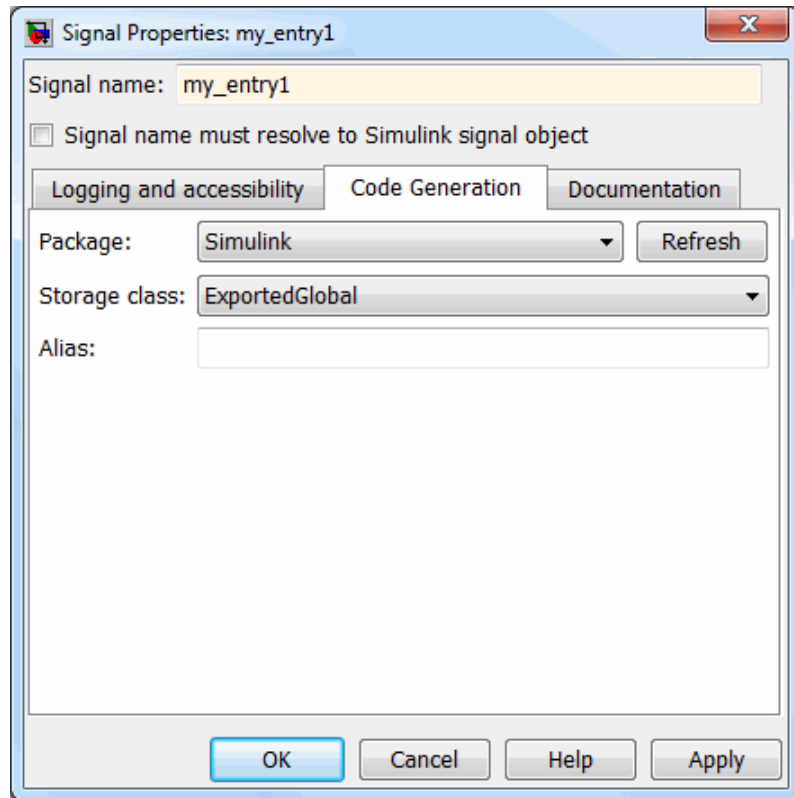
---

**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 “Specifying Signal Range through Source Block Parameters” on page 2-9.

---

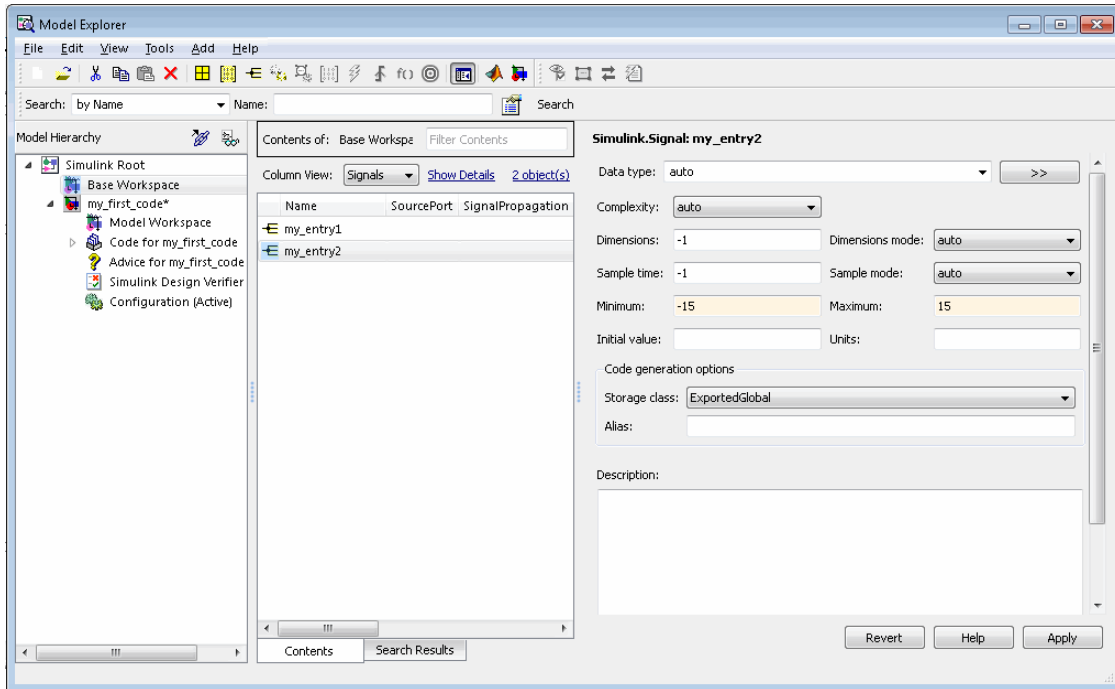
To specify an input signal range through the base workspace:

- 1** Configure the signal to use the `ExportedGlobal` storage class:
  - a** Right-click the signal. From the context menu, select **Signal 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.
- 2** Using Model Explorer, specify the signal range:
- a** Select **View > Model Explorer** to open Model Explorer.
  - b** In the Model Hierarchy, 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**.

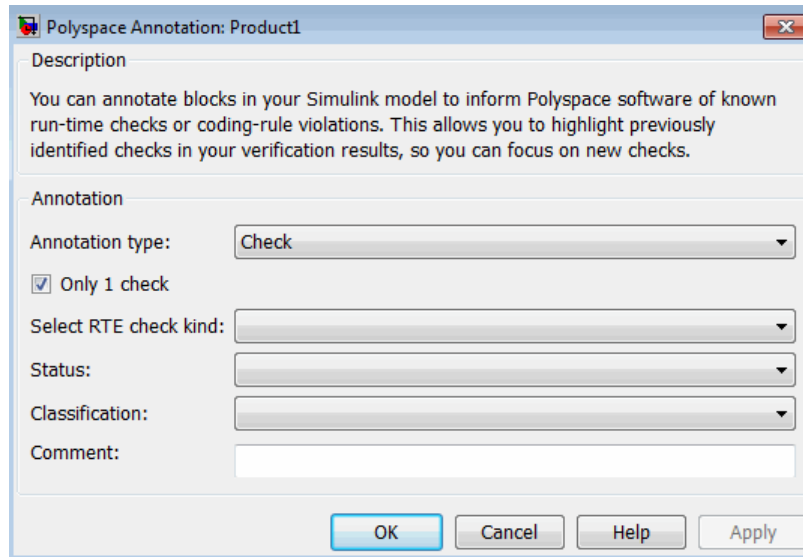
# Annotating Blocks to Justify Known Checks or Coding-Rules Violations

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 that you can focus on new checks when reviewing verification results.

The Polyspace Run-Time Checks 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.



- 3 From the **Annotation type** drop-down list, select one of the following:

- Check — To indicate run-time error
- MISRA — To indicate coding rule violation

**4** If you want to highlight only one check, select **Only 1 check** and the relevant run-time check (or MISRA rule) from the **Select RTE check kind** (or **Select MISRA 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

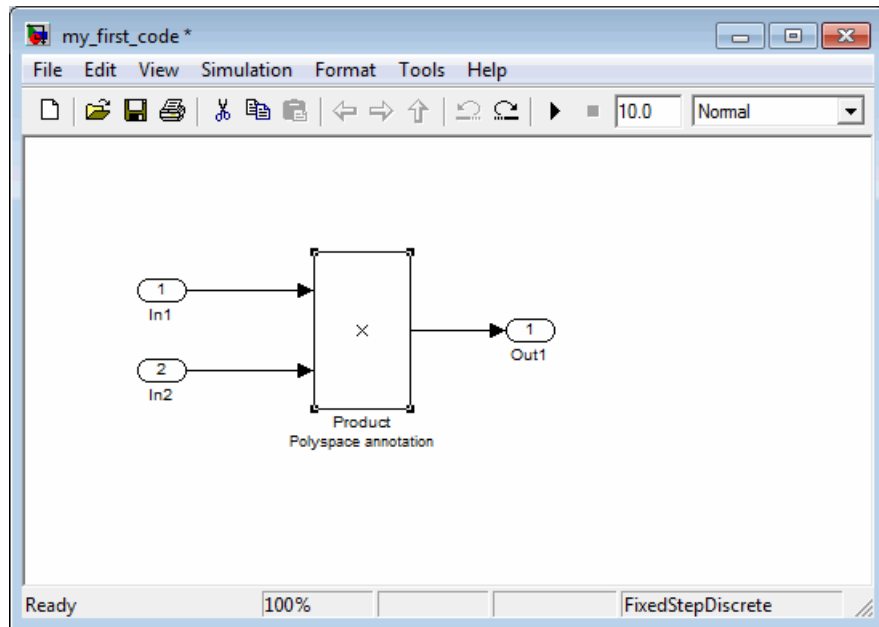
**6** Select a **Classification** to describe the severity of the issue:

- 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 to the block.

## 2 Configure Model for Code Verification



# Configure Code Verification Options

---

- “Overview of Polyspace Configuration” on page 3-2
- “Including Handwritten Code in Verification” on page 3-3
- “Configuring Data Range Settings” on page 3-5
- “Configuring Polyspace Analysis Options” on page 3-7
- “Configuring Polyspace Project Properties” on page 3-9
- “Creating a Polyspace Configuration File Template” on page 3-10
- “Specifying Header Files for Target Compiler” on page 3-13
- “Specifying Location of Results” on page 3-14
- “Main Generation for Model Verification” on page 3-15
- “Polyspace Model Link SL Considerations” on page 3-17
- “Polyspace Model Link TL Considerations” on page 3-25

## Overview of Polyspace Configuration

You do not have to manually create a Polyspace project or specify Polyspace analysis 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:

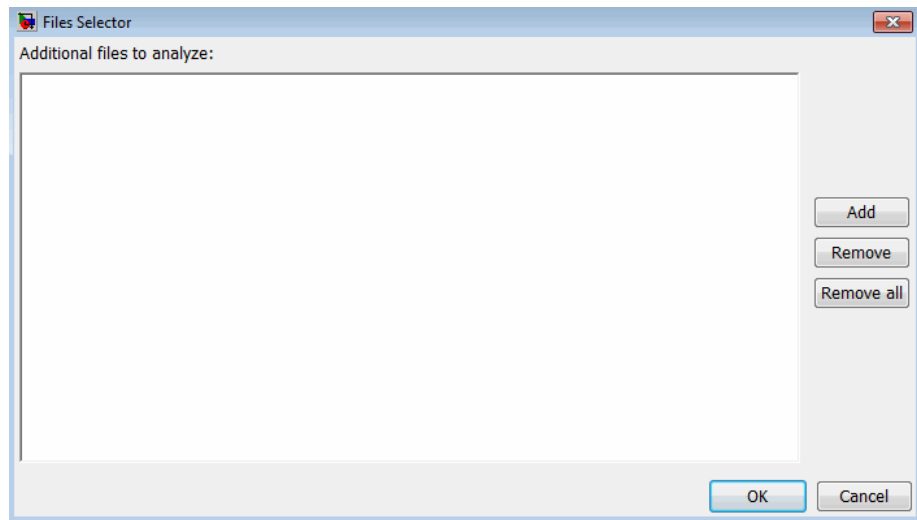
- You may incorporate separately created code within the code generated from your Simulink model. See “Including Handwritten Code in Verification” on page 3-3.
- By default, the Polyspace verification is contextual and treats tunable parameters as constants. You can specify a verification that considers robustness, including tunable parameters that lie within a range of values. See “Configuring Data Range Settings” on page 3-5.
- You may customize the analysis options for your verification. For example, to specify the target environment or adjust precision settings. See “Configuring Polyspace Analysis Options” on page 3-7 and “Recommended Polyspace Analysis Options for Generated Code” on page 3-18.
- You may create specific configurations for batch runs. See “Creating a Polyspace Configuration File Template” on page 3-10.
- If you want to verify code generated for a 16-bit target processor, you must specify header files for your 16-bit compiler. See “Specifying Header Files for Target Compiler” on page 3-13.

## Including 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 **Tools > Polyspace > Options**. The Configuration Parameters dialog box opens, displaying the Polyspace Model Link SL 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 appropriate folder
  - Add the required files.

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

---

**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**.



## Configuring 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.

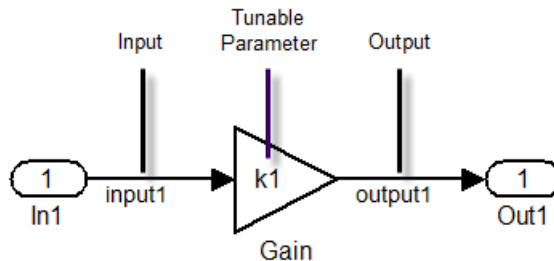
For more information, see “Choosing Robustness or Contextual Verification” in the *Polyspace Products for C/C++ User’s Guide*.

---

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

---

Polyspace Model Link SL allows 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 **Tools > Polyspace > Options**. The Configuration Parameters dialog box opens, displaying the Polyspace Model Link SL pane.

- 2 In the Data Range Management for Model section, specify how you want the verification to treat :
  - a **Input** — Select one of the following:
    - **Contextual (Default)** — Apply data ranges defined in blocks or base workspace to increase the precision of the verification. See “Specifying Signal Ranges” on page 2-9.
    - **Robustness** — Assume all inputs are full-range values (min...max)
  - b **Tunable parameters** — Select one of the following:
    - **Specific calibration (Default)** — Use value of constant parameter specified in code.
    - **Generic calibration** — Use a parameter range defined in the block or base workspace. See “Specifying Signal Ranges” on page 2-9. If no range defined, use full range (min...max).
  - c **Output** — Select one of the following:
    - **None (Default)** — No assertion ranges on outputs.
    - **Global assert** — Use assertion ranges on outputs.

---

**Note** The `Global assert` mode is incompatible with the Automatic Orange Tester.

---

In general, you should use the following combinations:

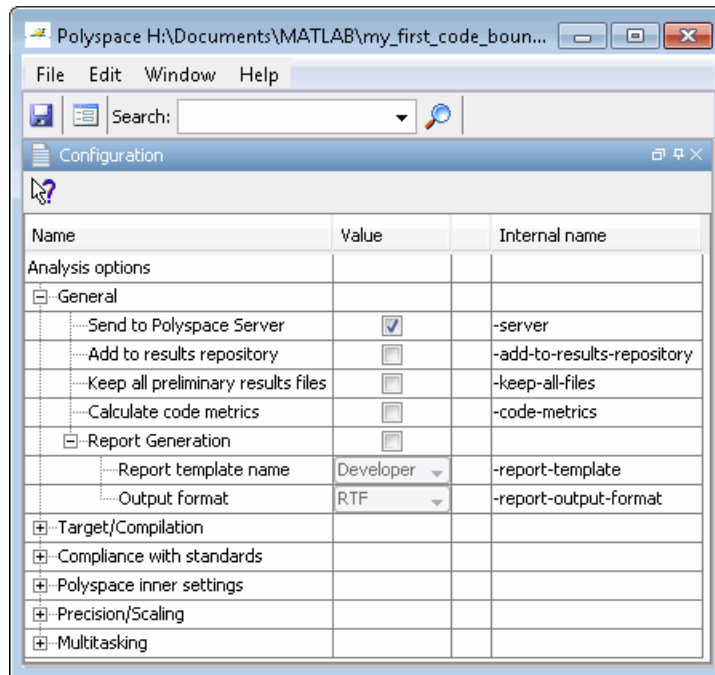
- **Contextual mode** for inputs and **Specific calibration mode** for tunable parameters to maximize verification precision
- **Robustness mode** for inputs and **Generic calibration mode** for tunable parameters to verify the worst cases of program execution

## Configuring Polyspace Analysis Options

Polyspace Model Link software supports a simplified version of the Polyspace Project Manager, which allows you to customize the analysis 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 Project Manager:

- 1 From the Simulink model window, select **Tools > Polyspace > Configure Project**.



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

- `-OS-target` — Set to `no-predefined-OS`
- `-results-dir` — Set to `results`

The software also configures other options automatically, but the settings depend on the code generator used. See “Polyspace Model Link SL Considerations” on page 3-17 and “Polyspace Model Link TL Considerations” on page 3-25.

### 2 Set other options as needed for your application.


For a list of recommended options for verifying generated code, see “Recommended Polyspace Analysis Options for Generated Code” on page 3-18.

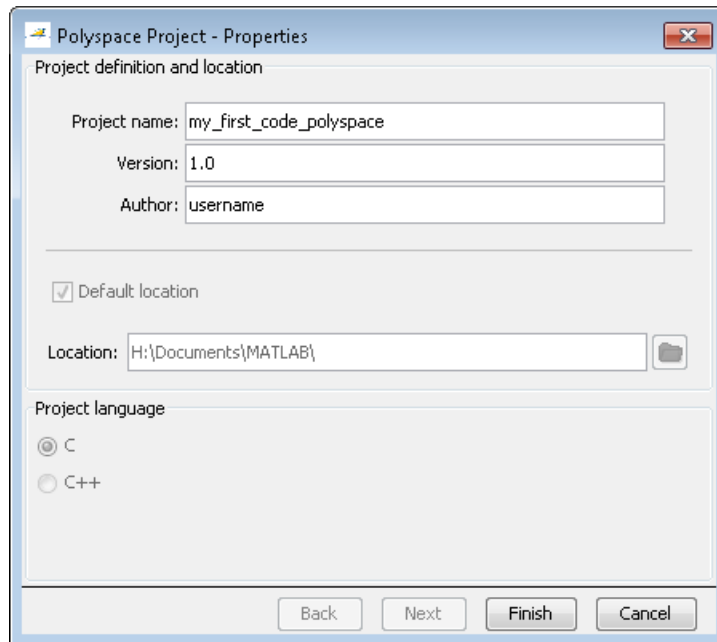
For descriptions of all analysis options, see “Option Descriptions for C Code” in the *Polyspace Products for C/C++ Reference*.

## Configuring 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 **Tools > Polyspace > Configure Project**.

- 2 On the Project Manager toolbar, click the **Project properties** icon .



Polyspace Project - Properties

Project definition and location

Project name: my\_first\_code\_polyspace

Version: 1.0

Author: username

Default location

Location: H:\Documents\MATLAB\

Project language

C

C++

Back Next Finish Cancel

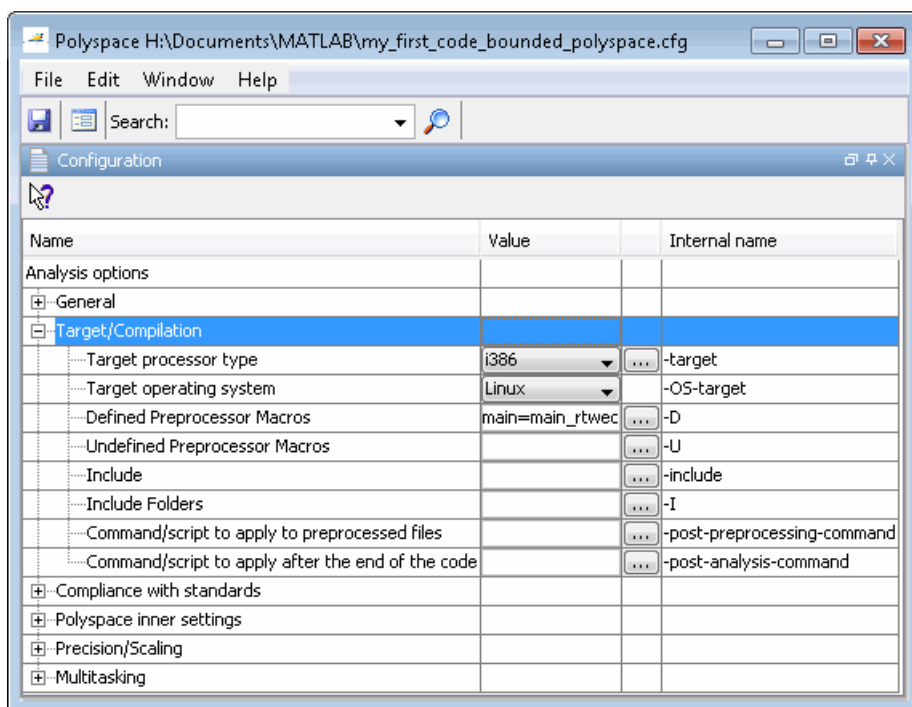
## Creating a Polyspace Configuration File Template

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 4-8.

To create a configuration file template:

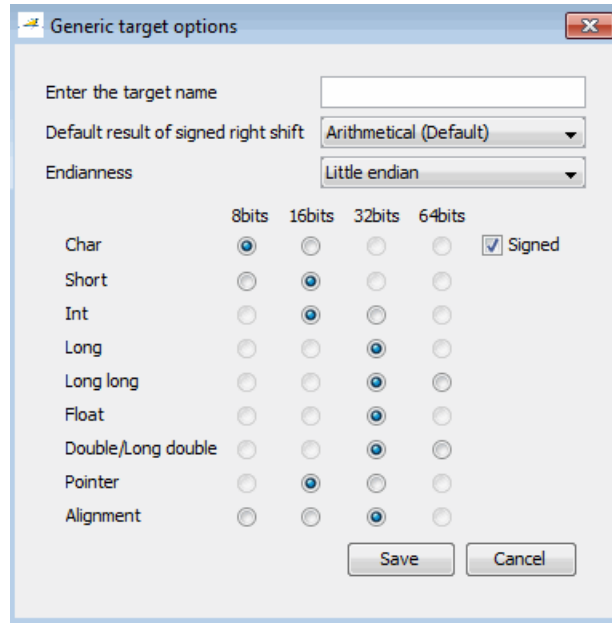
- 1 In the Simulink model window, select **Tools > Polyspace > Configure project**.

The Project Manager opens, allowing you to customize the target and cross compiler.



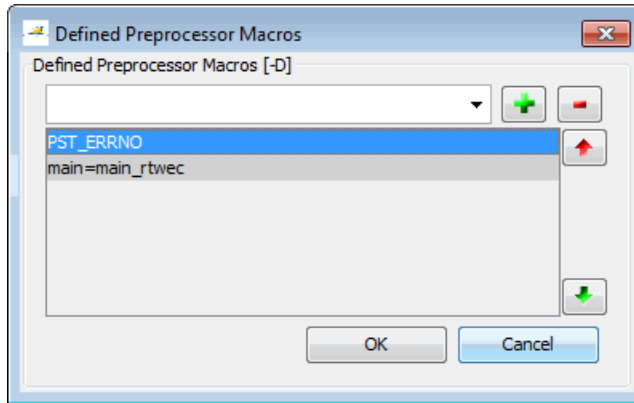
- 2 The **Target processor type (-target)** option defines the size of data types.

- a From the drop-down list in the Value column, 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**.

- 3 The **Defined Preprocessor Macros (-D)** option allows you to define preprocessor macros for your cross-compiler. Click the ... button. The Defined Preprocessor Macros dialog box opens.



To add a macro, in the field under **Defined Preprocessor Macros (-D)** enter the required text. Click the + button. The software adds your macro to the displayed list.

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.

---

When you have added or removed the required macros, click **OK**.

- 4 Save your changes and close the Project Manager.
- 5 Make a copy of the updated project configuration file, for example, `my_first_code_polyspace.cfg`.
- 6 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')
```



## Specifying 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 necessary 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 files (or folders containing header files) for your compiler:

- 1** Open the Polyspace Configuration pane: from the Simulink model window, select **Tools > Polyspace > Configure Project**. Selecting **Tools > Polyspace > Options > Project Configuration > Configure** also opens this pane.
- 2** On the Configuration pane, expand the **Target/Compilation** node.
- 3** To specify files, click the ... button for the **Include (-include)** option. The Include dialog box opens. If you want to specify folders, click the ... button for the **Include Folders (-I)** option. In this case, the Include Folders dialog box opens
- 4** Specify a header file (or folder) path by doing one of the following:
  - In the text field, enter the file (or folder) path and click **+**.
  - Click button and use the Select a file (or folder) to include dialog box to navigate to the required header file (or folder).

The software displays the list of specified header files (or folders) in the dialog window. You can remove a header file (or folder) from the displayed list by selecting the file (or folder) and clicking **-**.

### Specifying 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 **Tools > 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`.

## Main Generation for Model Verification

When you run a verification using Polyspace Model Link SL, 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 with the following behavior:

- 1** It initializes parameters using the Polyspace option `-variables-written-before-loop`.
- 2** It calls initialization functions using the option `-functions-called-before-loop`.
- 3** It initializes inputs using the option `-variables-written-in-loop`.
- 4** It calls the step function using the option `-functions-called-in-loop`.
- 5** It 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 more information on the main generator, see “Main Generator Behavior for Polyspace Software”.

### Main for Generated Code

The following example shows how to use the main generator options to generate a main 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 3-17

“Subsystems” on page 3-17

“Default Options” on page 3-17

“Data Range Specification” on page 3-18

“Recommended Polyspace Analysis Options for Generated Code” on page 3-18

## Overview

The Polyspace Model Link SL product has been tested with Embedded Coder software — see the Installation Guide for more information.

## 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 Analysis 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 directory.

---

### Data Range Specification

The software automatically creates a Polyspace Data RangeSpecification (DRS) file using information from the MATLAB workspace and block parameters. This DRS information is used to initialize each global variable to the range of valid values, as defined by the min-max information in the workspace.

The main sources of information are Simulink.signals and Simulink.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.

For more information, see “Configuring Data Range Settings” on page 3-5.

### Recommended Polyspace Analysis Options for Generated Code

This section describes recommended Analysis options for verifying code generated with Embedded Coder software. MathWorks recommends setting these options in your Polyspace project before verifying generated code.

If you have Polyspace Model Link SL software, you can specify the Analysis options for your Polyspace Project by selecting **Tools > Polyspace > Configure Project** in the Simulink model window.

---

**Note** For the following options, the values that you specify through the **Configuration Parameters > Polyspace Model Link SL** pane override the values specified in the **Configuration** pane:

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

Option	Recommended Value	Comments
<b>Polyspace Project – Properties</b>		
-I	See Comments	<p>Specifies the name of a folder to include when compiling C sources. You can specify only one folder for each I, but can use the option multiple times.</p> <p>A script to automatically determine -I based on buildInfo is available</p>
<b>Target/Compilation Options</b>		

Option	Recommended Value	Comments
-d	See Comments	<p>Defines macro compiler flags used during compilation.</p> <p>Use one <code>-d</code> 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 appropriate system definitions during preprocessing in order to analyze the included files properly.</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 <i>Setting Up Project for Generic Target Processors</i> in the <i>Polyspace Products for C User's Guide</i>.</p>
<b>Compliance with standards Options</b>		
-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"> <li>• <b>Header files</b> – All include folders specified (<code>-I</code> option)</li> <li>• <b>Source files</b> – All source files selected for the verification (<code>-sources</code> option)</li> </ul>



Option	Recommended Value	Comments
-allow-negative-operand-in-shift	Selected	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, $-2 \ll 2 = -8$ .
-misra2	[all-rules   file_name]	Specifies that the software checks coding rules in conformity to MISRA-C:2004. All MISRA checks are included in the log file of the verification. Options: <ul style="list-style-type: none"> <li>• <b>all-rules</b> – Checks all available MISRA C® rules. Any violation of MISRA C rules is considered a warning.</li> <li>• <b>filename</b> – Specifies an ASCII file containing a list of MISRA® rules to check.</li> </ul>
-includes-to-ignore	<MSVC dir>\VC\include	Specifies files or folders that are excluded from MISRA rules checking (all files and subfolders within the selected folder). This option is useful when you have non-MISRA C conforming include headers.
<b>Polyspace inner settings Options</b>		
-variables-written-before-loop	public	Specifies how the generated main initializes global variables.  By selecting public, every variable except static and const variables are assigned a "random" value, representing the full range of possible values.

<b>Option</b>	<b>Recommended Value</b>	<b>Comments</b>
<code>-functions-called</code> <code>-in-loop</code>	unused	<p>Specifies how the generated main calls functions.</p> <p>By selecting unused, every function is called by the generated main unless it is called elsewhere by the code undergoing verification.</p>
<code>-ignore-float-rounding</code>	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>
<b>Precision/Scaling Options</b>		

Option	Recommended Value	Comments
-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>MathWorks recommends you begin with the lowest precision level. You can then address red errors and gray code before relaunching 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"> <li>• -00 corresponds to static interval verification.</li> <li>• -01 corresponds to complex polyhedron model of domain values.</li> <li>• -02 corresponds to more complex algorithms to closely model domain values (a mixed approach with integer lattices and complex polyhedrons).</li> <li>• -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.</li> </ul>
-to	<b>c-compile</b> – When checking MISRA	Specifies the phase after which the verification stops. Each verification phase

Option	Recommended Value	Comments
	<p>compliance only.<b>pass0</b> – When verifying code for the first time.</p> <p><b>pass4</b> – When performing subsequent verifications of code.</p>	<p>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>MathWorks recommends you begin by running -to pass0 (Software Safety Analysis level 0) You can then address red errors and gray code before relaunching verification using higher integration levels.</p>

# Polyspace Model Link TL Considerations

## In this section...

“Overview” on page 3-25

“Subsystems” on page 3-25

“Default Options” on page 3-25

“Data Range Specification” on page 3-26

“Lookup Tables” on page 3-26

“Code Generation Options” on page 3-27

## Overview

The Polyspace Model Link TL product has been tested with the some release of the dSPACE Data Dictionary version and TargetLink Code Generator - see the Installation Guide for more information.

As the Polyspace Model Link TL product extracts information from the dSPACE Data Dictionary remember to regenerate the code before performing a Polyspace verification. This ensures that the Data Dictionary has been correctly updated.

## 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 *dspaceroot*\matlab\TL\SimFiles\Generic
- I *dspaceroot*\matlab\TL\srcfiles\Generic
- I *dspaceroot*\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

The tool automatically creates Polyspace Data RangeSpecification (DRS) information using the dSPACE Data Dictionary for each global variable. This 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 its verification. Further the Boolean types are modeled having a minimum value of 0 and a maximum of 1. Defining the min-max information carefully in the model can help Polyspace verification to be more precise significantly because only range of reels values are analyzed.

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

---

- “Running Verification with Polyspace Model Link SL Software” on page 4-2
- “Running Verification with Polyspace Model Link TL Software” on page 4-4
- “Monitoring Verification Progress” on page 4-7
- “MATLAB Functions For Polyspace Batch Runs” on page 4-8
- “Archive Files for Polyspace Verification” on page 4-9

## Running Verification with Polyspace Model Link SL Software

To start Polyspace verification from the Simulink model window:

- 1 Select **Tools > Polyspace > Run Verification**.

---

**Note** You can also start verification from the Configuration Parameters dialog box, by clicking **Run verification** on the Polyspace Model Link pane.

---

The verification starts, and 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
```

- 2 Follow the progress of the verification in the MATLAB Command window, and later using the Polyspace spooler (Queue Manager) if you are performing a server verification.

The software writes all status messages to a log file in the results folder, for example: Polyspace\_R2011b\_my\_first\_code\_05\_26\_2011-17h26.log:

```
<polyspace-c R2011b PID9284 PGID9284>
```

Polyspace verification of my\_first\_code project.  
Starting at 05/26/2011, 17h26.

Options used with Verifier:

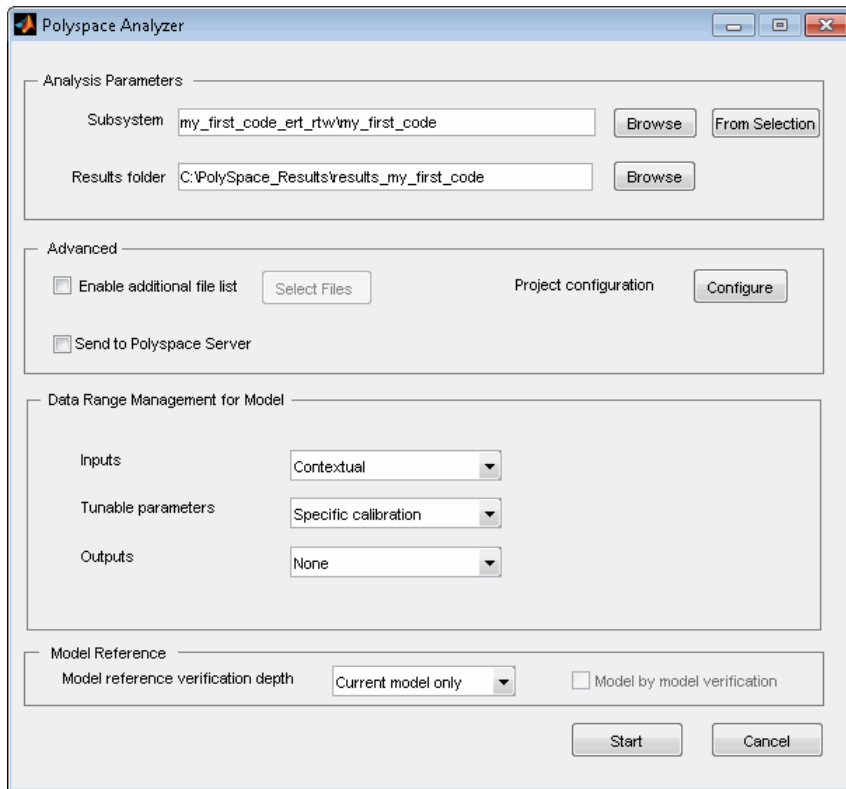
```
-polyspace-version=CC-8.2.0.1 (R2011b)
-author=johnr
-from=scratch
-date=26/05/2011
-enum-type-definition=defined-by-standard
-lang=C
-allow-negative-operand-in-shift=true
-max-processes=4
-discard-asm=true
-variables-written-in-loop=custom=my_first_code_U
-results-dir=C:\PolySpace_Results\results_my_first_code
-target=i386
-scalar-overflows-behavior=truncate-on-error
-data-range-specifications=C:\Work\work5\my_first_code_polyspace_drs.txt
-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=__restrict__=
-D2=example=include_head
-D3=main=main_rtwec
-D4=MODEL=my_first_code
-D5=NUMST=1
...
```

## Running Verification with Polyspace Model Link TL Software

To start the Polyspace verification:

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

The Polyspace Analyzer dialog box opens.



**Polyspace Analyzer Dialog Box**

---

**Note** The subsystem field is automatically populated with the name of the current subsystem, and the results directory is automatically set to `results_subsystem_name`. If more than one subsystem is present in the model, a subsystem selection dialog opens.

---

**2** Click **Start** to start the verification.

The verification starts, and 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.

**3** You can follow the progress of the verification in the MATLAB Command window, and later using the Polyspace spooler (Queue Manager) if you are performing a server verification.

---

**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.

---

## Monitoring Verification Progress

In this section...
“Client Verifications” on page 4-7
“Server Verifications” on page 4-7

### 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_R2011b_my_first_code_05_26_2011-17h26.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 the Polyspace Queue Manager (Spooler).

To open the Polyspace Queue Manager:

- 1 From the Simulink model window, select **Tools > Polyspace > Open Spooler**.

For more information, see “Managing Verification Jobs Using the Polyspace Queue Manager” in the *Polyspace Products for C/C++ User’s Guide*.

## MATLAB Functions For Polyspace Batch Runs

You can run the following commands in the Command Window.

Command	Description
<code>PolyspaceForEmbeddedCoder</code>	Start Polyspace verification of code generated by Embedded Coder software
<code>PolyspaceForTargetLink</code>	Start Polyspace verification of code generated by TargetLink
<code>PolyspaceSpooler</code>	Inspect the queue of the remotely sent verification over the server
<code>PolyspaceViewer</code>	Open Polyspace verification environment Run-Time Checks perspective
<code>PolyspaceSetTemplateCFGFile</code>	Select a template file, for example, during a batch run
<code>PolyspaceGetTemplateCFGFile</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>	Displays the Polyspace For Model-Link Version number along with other MathWorks product information

### Example with Embedded Coder

Suppose you open a Simulink model with the name `example.mdl` and generate code. To start a verification, in the MATLAB Command window, enter `PolyspaceForEmbeddedCoder('example')`.



# Archive Files for Polyspace Verification

## In this section...

“Template File in *MATLAB Installation folder*\polyspace\” on page 4-9

“Files Used in Model Folder” on page 4-9

“Auto-Generated Files in Model Folder” on page 4-10

## 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

---

- “Viewing Results in Polyspace Verification Environment” on page 5-2
- “Identifying Errors in Simulink Models” on page 5-6

## Viewing Results in Polyspace Verification Environment

When a verification completes, you can view the results using the Run-Time Checks perspective of the Polyspace verification environment.

---

**Note** If you perform a server verification, you must download your results from the server before you can view them. For more information, see “Downloading Results from Server to Client” in the *Polyspace Products for C/C++ User’s Guide*.

---

To view your results:

- 1** From the Simulink model window, select **Tools > Polyspace > Open results**.

After a few seconds, the Run-Time Checks perspective of the Polyspace verification environment opens.

Check details

Review statistics

The screenshot displays the Polyspace Verification Environment interface. The main window shows the 'Run-Time Checks' panel on the left, listing various checks such as 'Close\_To\_Zero', 'Non\_Infinite\_Loop', 'Pointer\_Arithmetic', 'RTE', 'Recursion', 'Recursion\_caller', 'Square\_Root', 'Square\_Root\_conv', 'Unreachable\_Code', and 'get\_oil\_pressure'. The 'Check Review' panel in the center shows details for a specific check: 'Recursion( ex ); // always encounters a division by zero' at line 157, column 5. The 'Review Statistics' panel on the right provides a summary of coding review progress, including counts for Red, Gray, and Orange NTCs, and a software reliability indicator. The 'Source' panel shows the code snippet for 'Recursion\_caller' at line 157. The 'Call Hierarchy' panel lists the calls to 'Recursion\_caller' from other files. The 'Variable Access' panel shows the variables accessed in the current context.

Run-time checks

Source code

Variable access

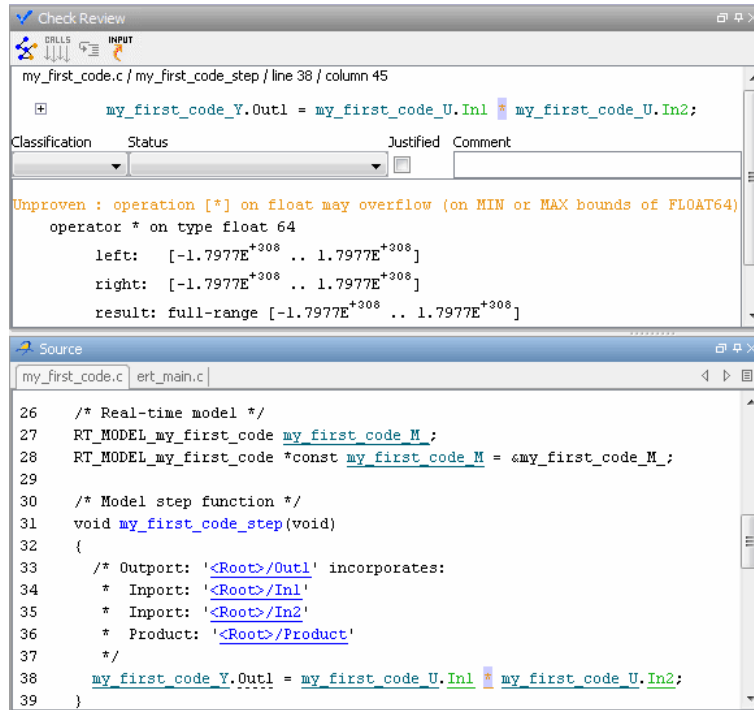
Call hierarchy

2 To navigate to the next error, type **CTRL-N**.

Procedural entities	!	X	?	✓	%	Line
my_first_code	0	0	1	23	96	
ert_main.c					0	1
my_first_code.c			1	4	80	1
_init_globals()					0	1
my_first_code_initialize()				2	100	42
my_first_code_step()			1	2	67	31
NIV.0				1		38
DcPL.1			1			38
NIV.2				1		38
my_first_code_terminate()					0	58
stdio.h					0	1
__polyspace_stdstubs.c				9	100	1
__polyspace_main.c				10	100	1

**3** Click any check to review additional information.

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



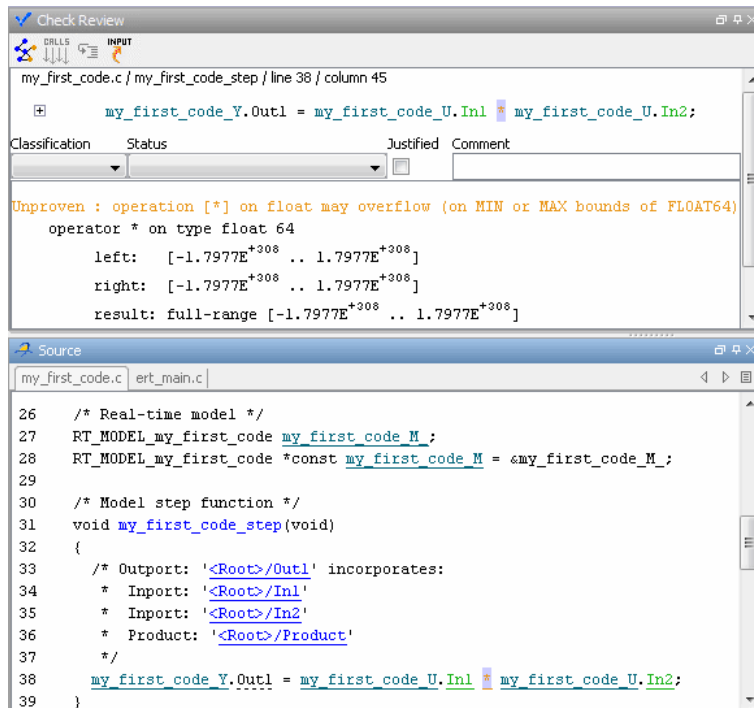
For more information on reviewing run-time checks, see “Reviewing Verification Results” in the *Polyspace Products for C/C++ User’s Guide*.

For information on specific checks, see “Colored Source Code for C” in the *Polyspace Products for C/C++ Reference*.

## Identifying 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 Review Details pane shows information about an orange check, and the Source pane shows the source code containing the orange check.



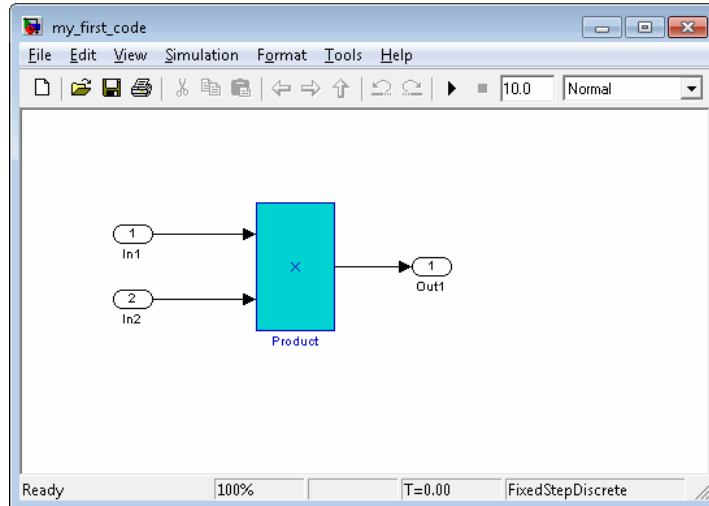
This orange check shows a potential overflow issue when multiplying the signals from the inports In1 and In2. To fix this issue, you must return to the model.

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 either:

- **Design** — If the model should be robust against the full signal range, it is a design bug. 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 designed to work within 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 “Specifying Signal Ranges” on page 2-9.

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