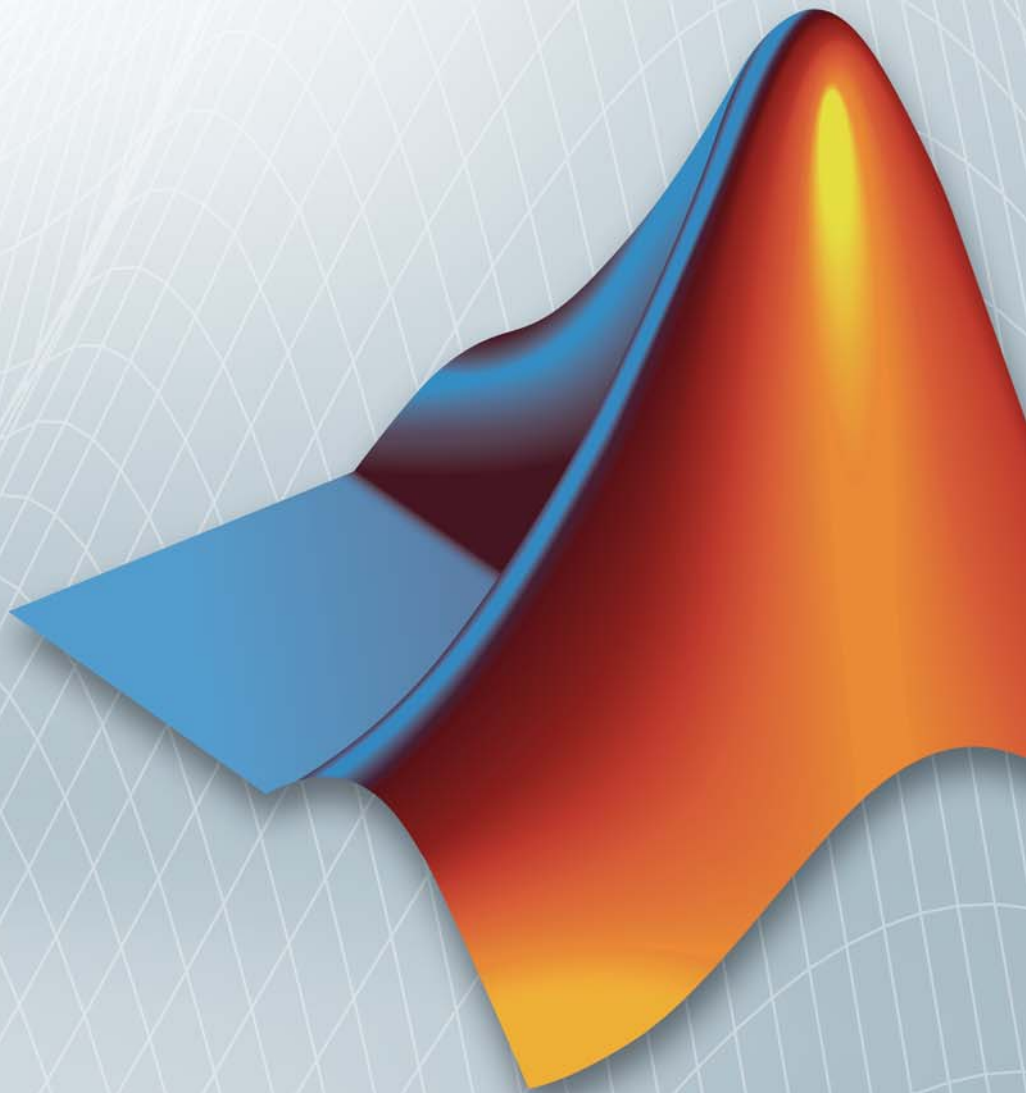


# Polyspace® Model Link Products 5

## User's Guide



## 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)

## Getting Started with Polyspace Model Link Products

### 1

<b>Overview of Polyspace Model Link Products</b> .....	<b>1-2</b>
<b>Getting Started with Model Link Products</b> .....	<b>1-3</b>
Overview .....	1-3
Creating a Simulink Model and Generating Production Code .....	1-4
Starting the Polyspace Verification .....	1-11
Fixing an Error in the Design and the Simulink Model ...	1-13
Base Workspace vs. Polyspace Data Ranges .....	1-18
Setting Data Ranges Using Block Parameters .....	1-26

## Advanced Setup Options

### 2

<b>Advanced Setup</b> .....	<b>2-2</b>
Handwritten Code .....	2-2
Target Production Environment .....	2-5
Creating a Polyspace Configuration File Template .....	2-7
Data Range Management .....	2-9
Main Generation for Model Verification .....	2-11
Annotating Blocks to Justify Known Checks or Coding-Rules Violations .....	2-13

## Polyspace Utilities

### 3

<b>Polyspace Utilities</b> .....	<b>3-2</b>
----------------------------------	------------

Overview of Polyspace Utilities .....	3-2
Configuring Polyspace Project .....	3-4
<b>Polyspace Commands Available in Batch Mode as M-Functions .....</b>	<b>3-6</b>
<b>Archives Files Produced for the Polyspace Verification .....</b>	<b>3-8</b>
Template files located in MATLAB installation directory\polyspace\ .....	3-8
Files used in the model directory .....	3-9
Auto-generated files in the model directory .....	3-9

## Code Generator Specific Information

# 4

<b>Polyspace Model Link SL Product .....</b>	<b>4-2</b>
Overview .....	4-2
Subsystems .....	4-2
Default Options .....	4-2
Data Range Specification .....	4-3
Code Generation Options .....	4-3
Polyspace Analysis Options .....	4-5
<b>Polyspace Model Link TL Product .....</b>	<b>4-12</b>
Overview .....	4-12
Subsystems .....	4-12
Data Range Specification .....	4-12
Lookup Tables .....	4-13
Code Generation Options .....	4-14

## Glossary

# Getting Started with Polyspace Model Link Products

---

- “Overview of Polyspace Model Link Products” on page 1-2
- “Getting Started with Model Link Products” on page 1-3

## Overview of Polyspace Model Link Products

This manual describes how to use Polyspace® for Model-Based Design. The Polyspace Model Link™ SL and Polyspace Model Link TL products allow you to launch a Polyspace C verification from a Simulink® model associated with Embedded Coder™ software, or dSPACE® TargetLink® software.

Polyspace Model Link SL and Polyspace Model Link TL products provide automatic error detection for code generated from Simulink models. It consists of three principal components:

- A Polyspace menu in the Simulink Tools menu.
- A Simulink Polyspace library with associated blocks.
- A “Back to Model” extension in the Run-Time Checks perspective of the Polyspace Verification Environment that allows direct navigation from a runtime error in the auto-generated code to the corresponding Simulink block or Stateflow® chart in the Simulink model.

# Getting Started with Model Link Products

## In this section...

“Overview” on page 1-3

“Creating a Simulink Model and Generating Production Code” on page 1-4

“Starting the Polyspace Verification” on page 1-11

“Fixing an Error in the Design and the Simulink Model” on page 1-13

“Base Workspace vs. Polyspace Data Ranges” on page 1-18

“Setting Data Ranges Using Block Parameters” on page 1-26

## Overview

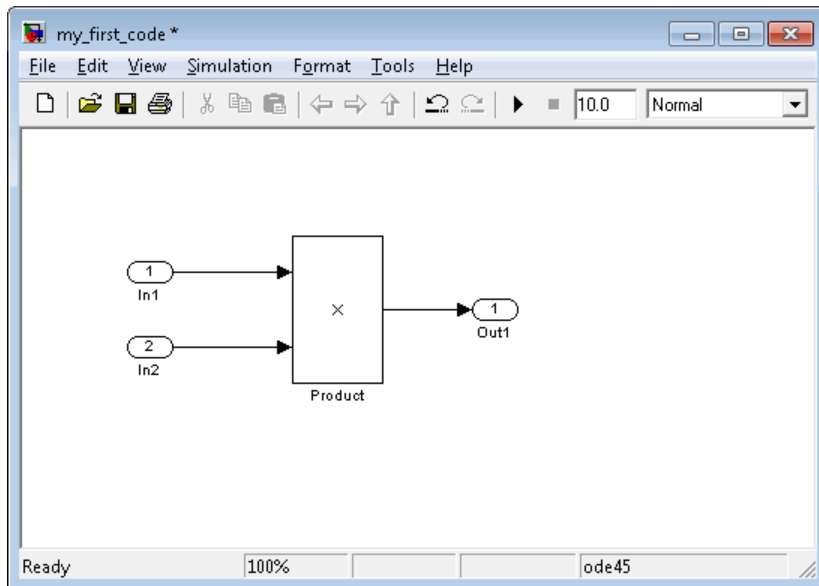
In this section, you will:

- Create a Simulink model and generate production code (For more information, see the *Embedded Coder Getting Started Guide*)
- Start the Polyspace verification

## Creating a Simulink Model and Generating Production Code

To create a Simulink model and generate production code:

- 1 Open MATLAB®, then start Simulink software.
- 2 Create a simple Simulink model, similar to the one below.

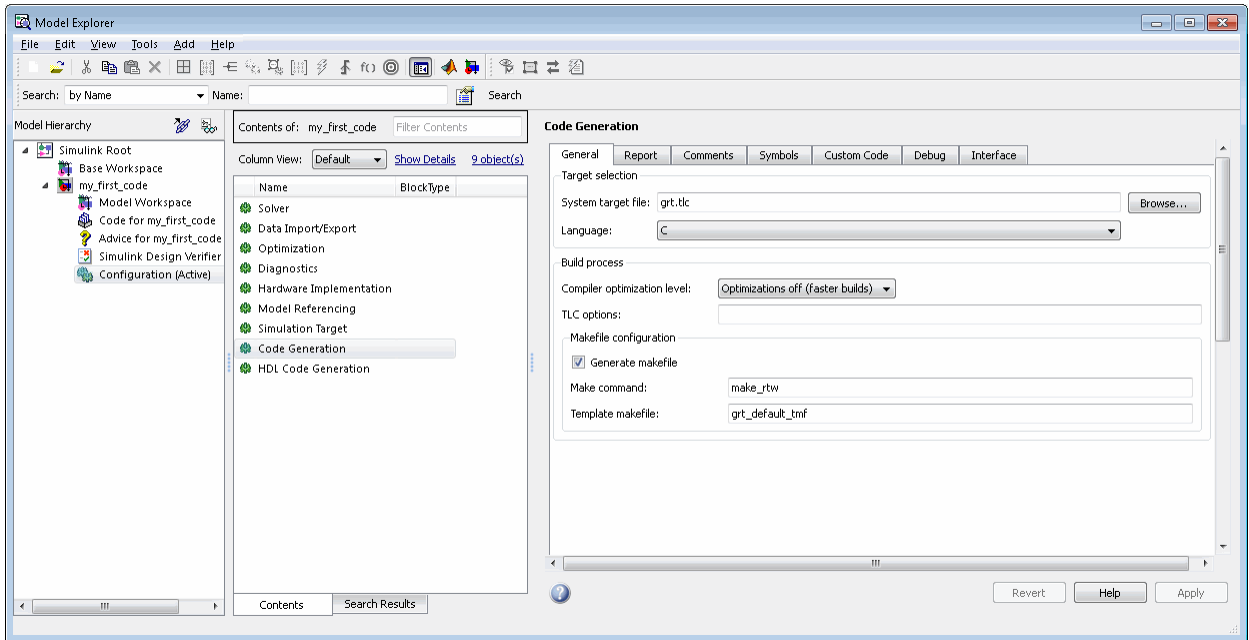


### Create the my\_first\_code model

- 3 Select **File > Save**, then name the model my\_first\_code.
- 4 Select **View > Model Explorer**.

The Model Explorer opens.





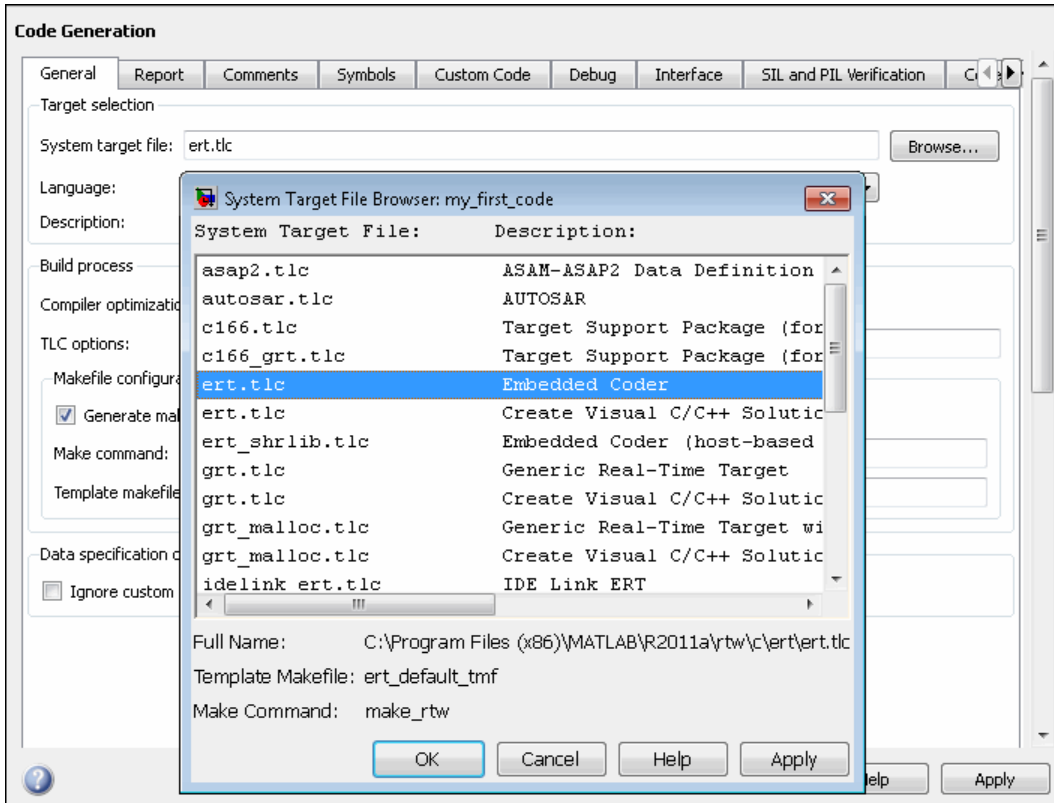
**5** Select `my_first_code > Configuration`, in the Model Hierarchy.

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

The Code Generation configuration parameters open.

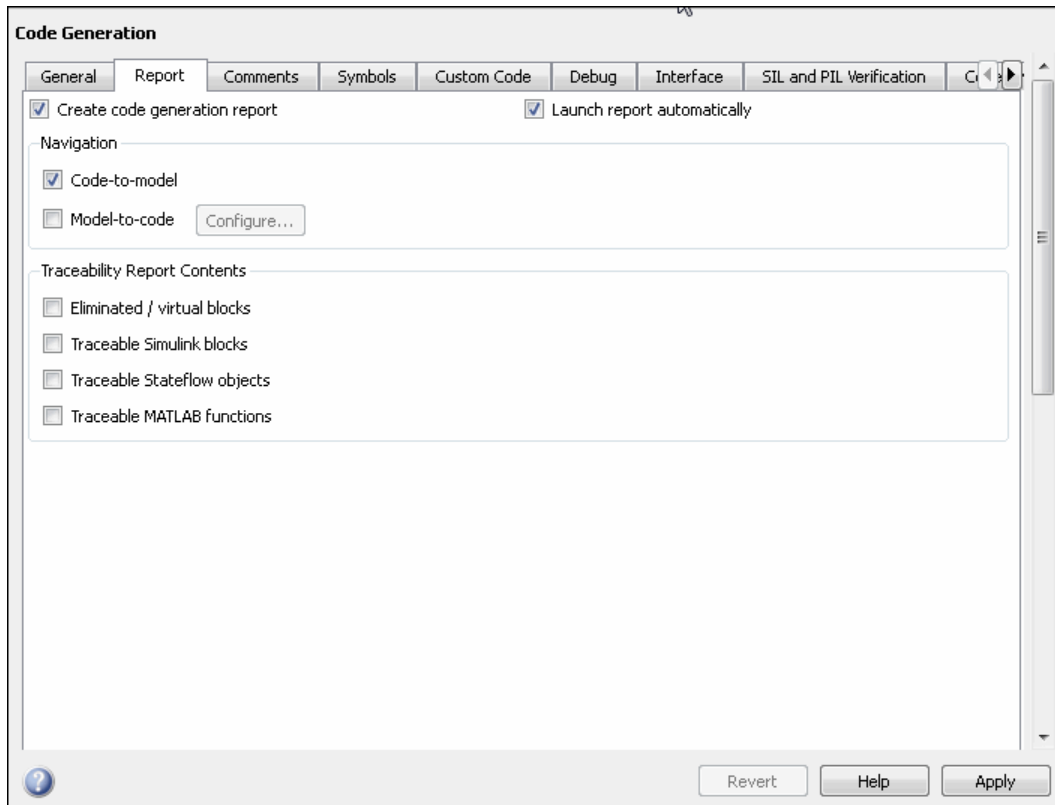
**7** Select the **General** tab.

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



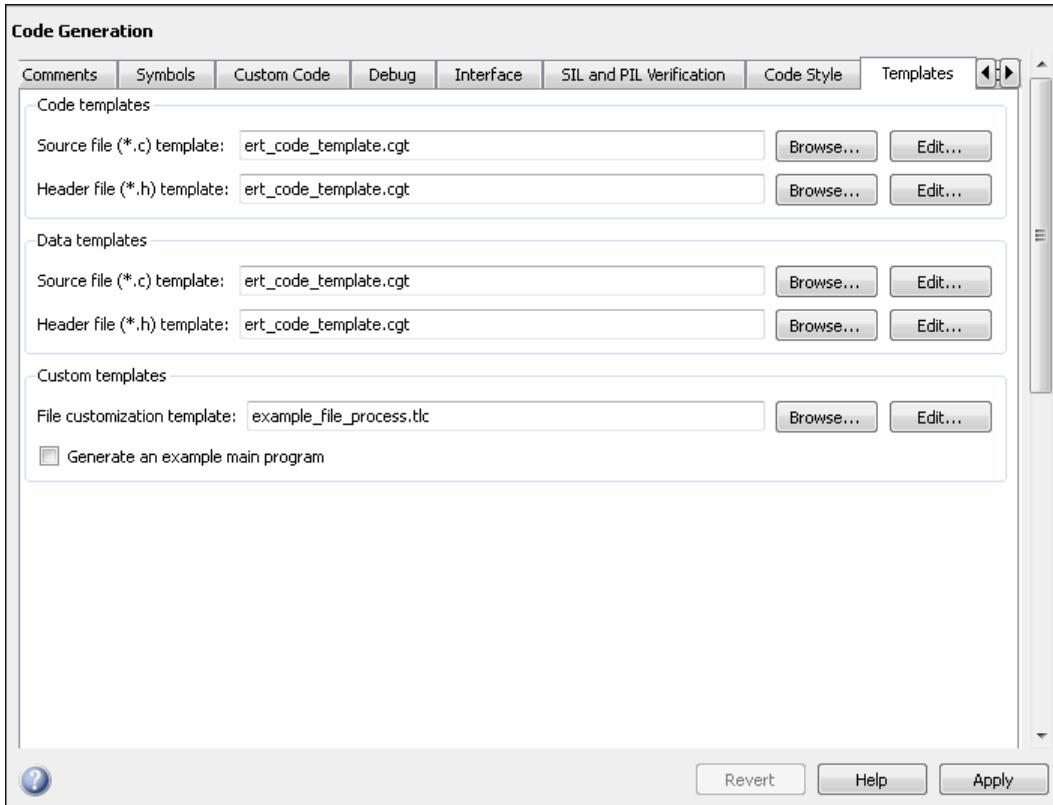
## Change the code generator to Embedded Coder™ software

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



## Set Report Settings

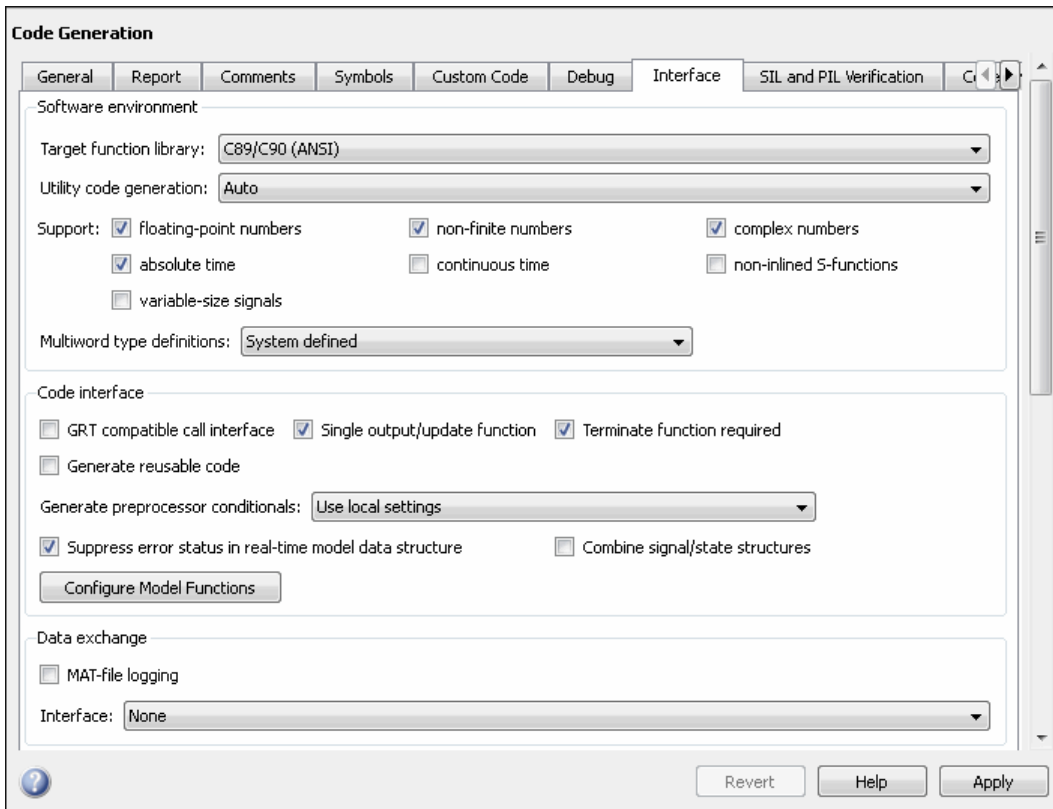
**10** Select the **Templates** tab.



## Templates Tab

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

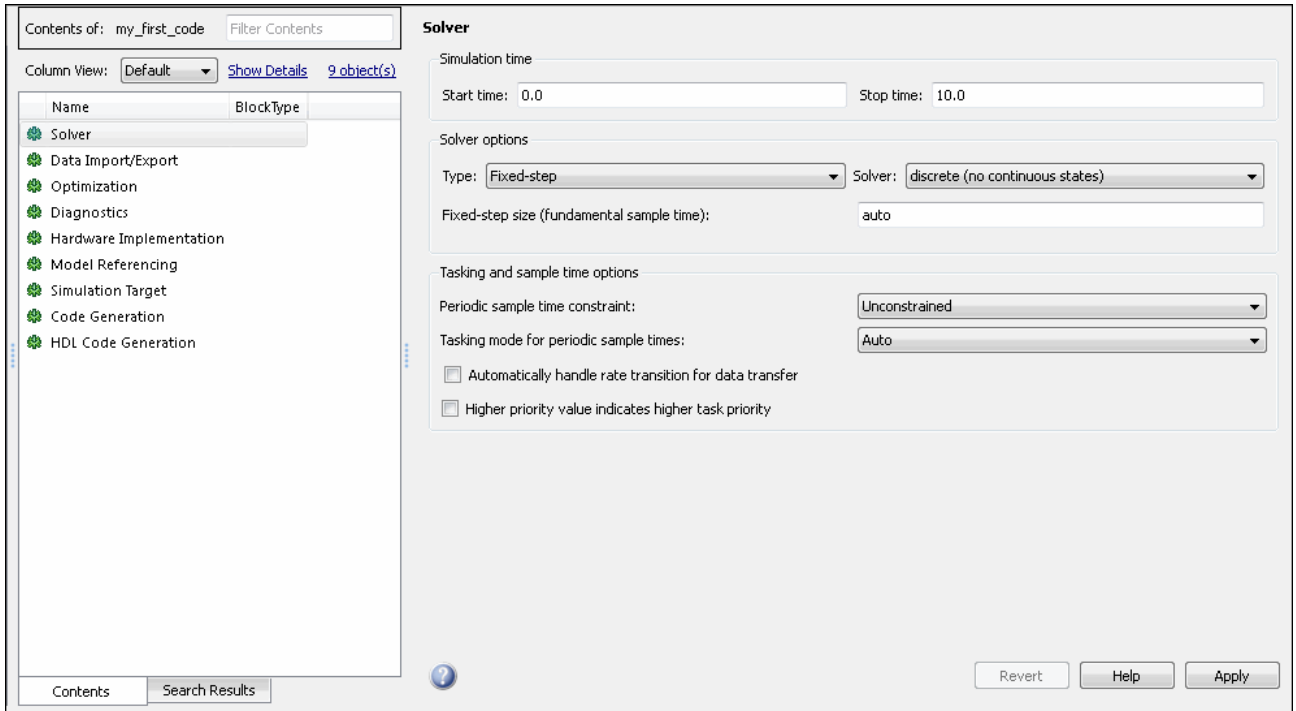
**12** Select the **Interface** tab.



## 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.



## Choose Fixed-step Solver

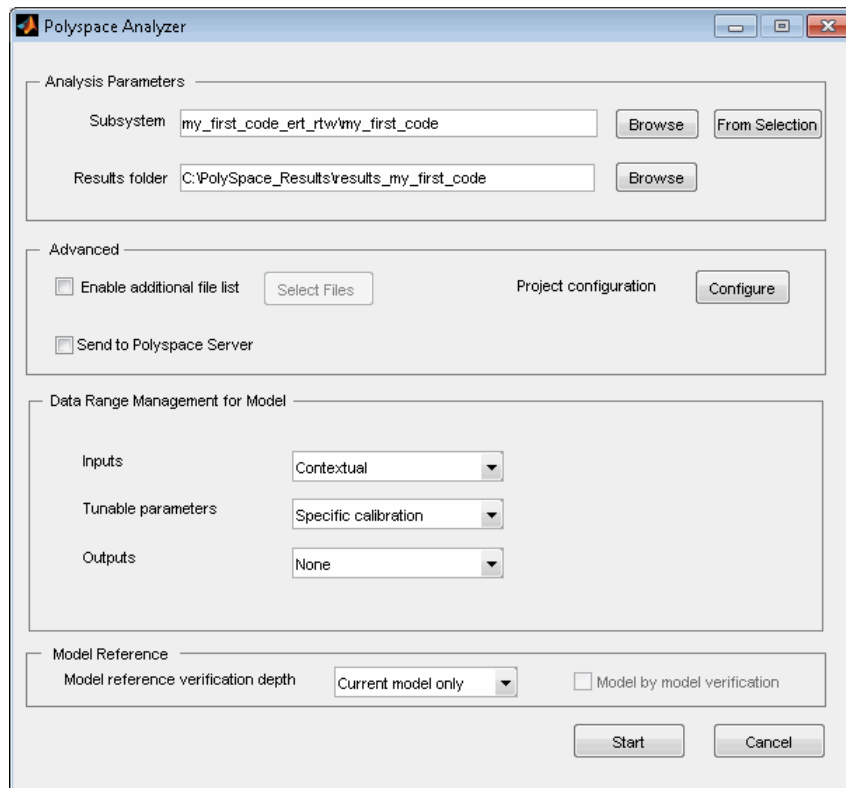
- 16** In the Solver options section, set the solver **Type** to **Fixed-step**. Then, set the **Solver** to **discrete (no continuous states)**.
- 17** Click **Apply**.
- 18** In the Simulink Model Window, select **Tools > Code Generation > Build Model** to generate the production code.
- 19** Save your Simulink model.

## Starting the Polyspace Verification

To Start the Polyspace verification:

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

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 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.

---

## **Fixing an Error in the Design and the Simulink Model**

After the 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 User’s Guide*.

---

To view your results:

- 1** In the Simulink model window select **Tools > Polyspace > Polyspace Utilities > Open results**.

After a few seconds, the Run-Time Checks perspective of the Polyspace Verification Environment opens.

# 1 Getting Started with Polyspace® Model Link Products

The screenshot displays the Polyspace IDE interface with several key components:

- Review Details:** Shows a warning for an unproven operation on a float type. The error message is: "Unproven : operation [\*] on float may overflow (on MIN or MAX bounds of FLO... operator \* on type float 64". It provides numerical ranges for the left and right operands and the full-range result.
- Review Statistics:** A table summarizing the review progress for various coding rules.
- Source Code:** The main editor window showing the C code for `my_first_code_step`.
- Call Hierarchy:** A tree view showing the call stack, with `my_first_code.my_first_code_step` at line 31 and `__polyspace_main.main` at line 93.
- Variable Access:** A table showing the number of reads and writes for various variables.
- Run-Time Checks:** A panel on the left showing the status of various procedural entities.

Coding review progress	Count	Progress
Orange F-OVFL justified / to justify	0/1	0
Red justified / to justify	0/0	100
Gray justified / to justify	0/0	100
Orange justified / to justify	0/1	0
Software reliability indicator	23/24	95

Calls	Line
my_first_code.my_first_code_step	31
__polyspace_main.main	93

Variables	# Re...	# Write	W. T.	R. T.
my_first_code				
ert_main.OverrunFlag	0	0		
my_first_code.my_first_code_M	1	1		
my_first_code.my_first_code_M_	0	2		
my_first_code._init_globals				
my_first_code.my_first_code_M_	0	1		
my_first_code.my_first_code_				

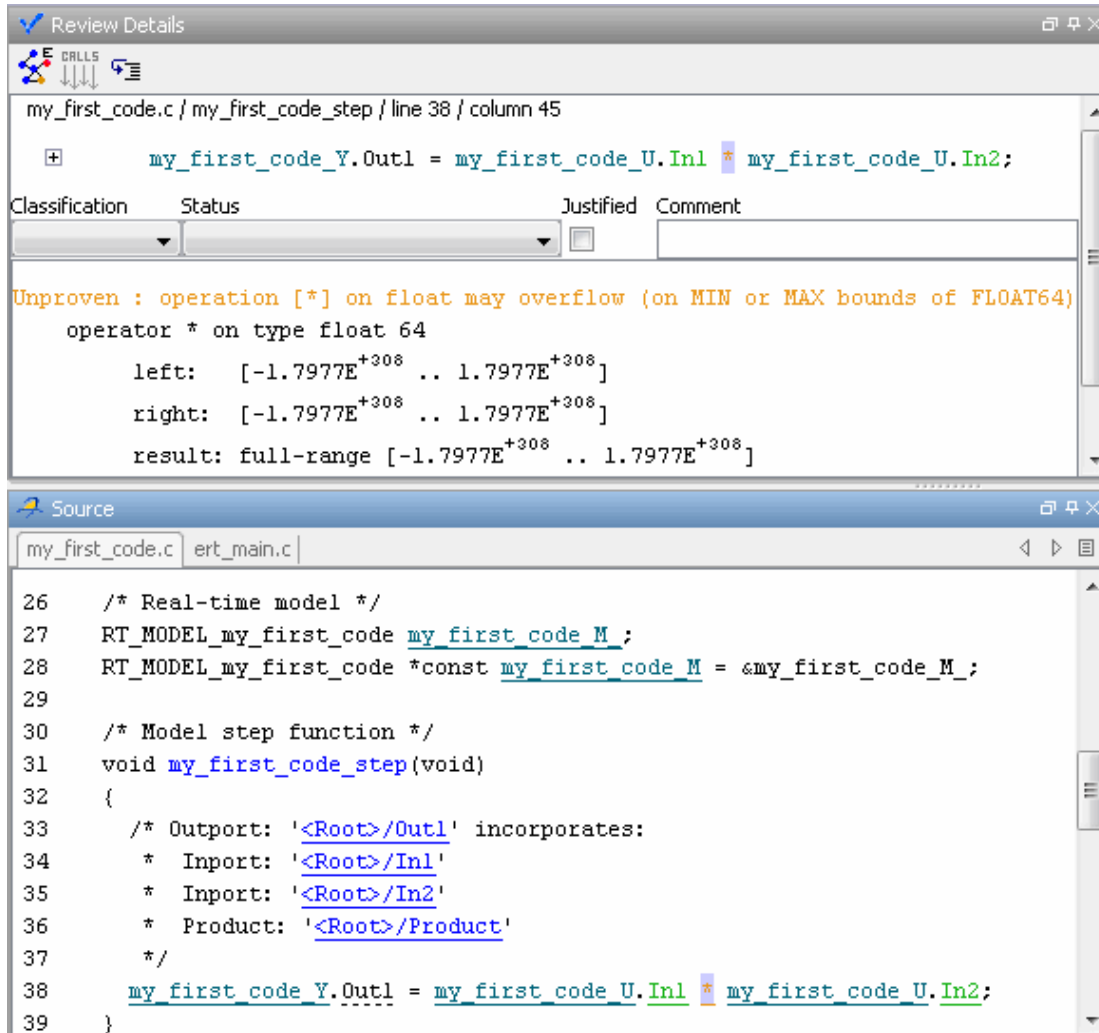
2 Type CTRL-N to go to the next error.

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

### Orange Check in Polyspace® Run-Time Checks

- 3 Click on the orange check.

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

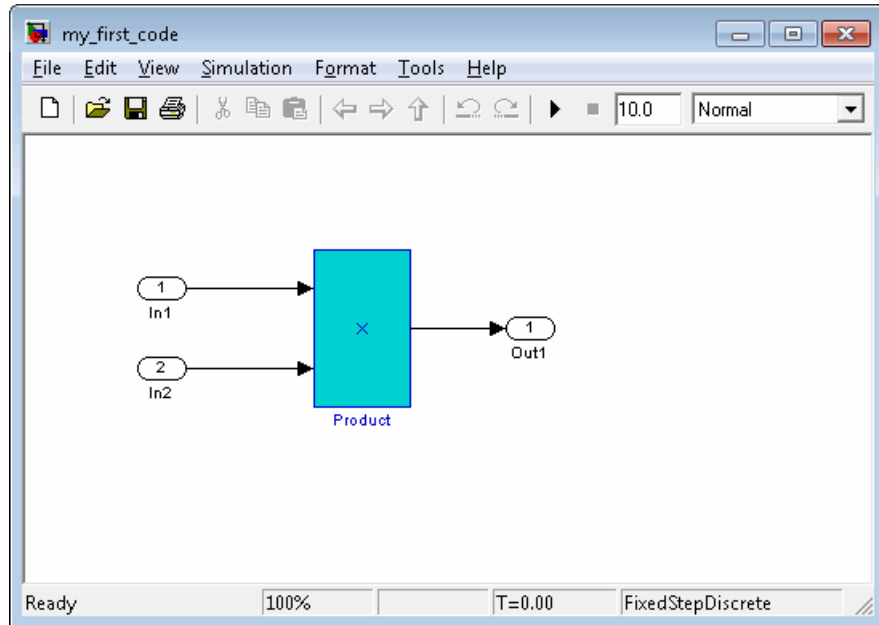


### Potential overflow in the code

This orange check shows a potential overflow of the two entries. Polyspace software assumes that the values for entries are full range, and their multiplication can overflow.

- 4 To fix this issue, you must return to the model. To go back to the model, 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.



### Model with Highlighted Block

- 5 You now must fix the defect in the model. For example, you may come to one of the following conclusions:
- **It is a bug in the design**— The developer should saturate the output, providing this functionally makes sense bound the entries in the model, by adding blocks which will test the input values, and bound them accordingly.
  - **It is a bug in the specifications** — The developer should bound the entries, by giving them a range in Simulink software that Polyspace verification can take the ranges into account and turns the code green.

## **Base Workspace vs. Polyspace Data Ranges**

After you examine the model, you can see a block whose signal ranges are not in the expected range.

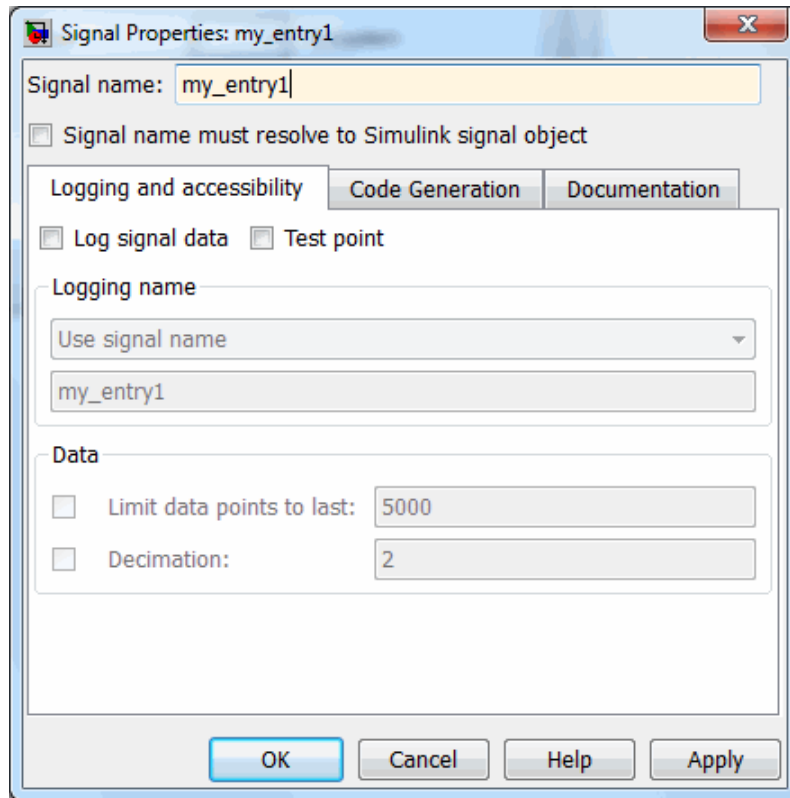
- If its block is supposed to be robust against this range, it is a design bug. Should the previous block be saturated? Should the signal be bounded with a “switch” block? It is up to the developer to decide the appropriate change in the model
- If the range is an input range of the model, the developer may wish to give this information to the Simulink model, so that Polyspace tools can use that range as an entry.

## **Creating Signals with ExportedGlobal Storage Class**

Before creating bounded signals in the base workspace, you must configure your Simulink model to use input signals with the `ExportedGlobal` storage class.

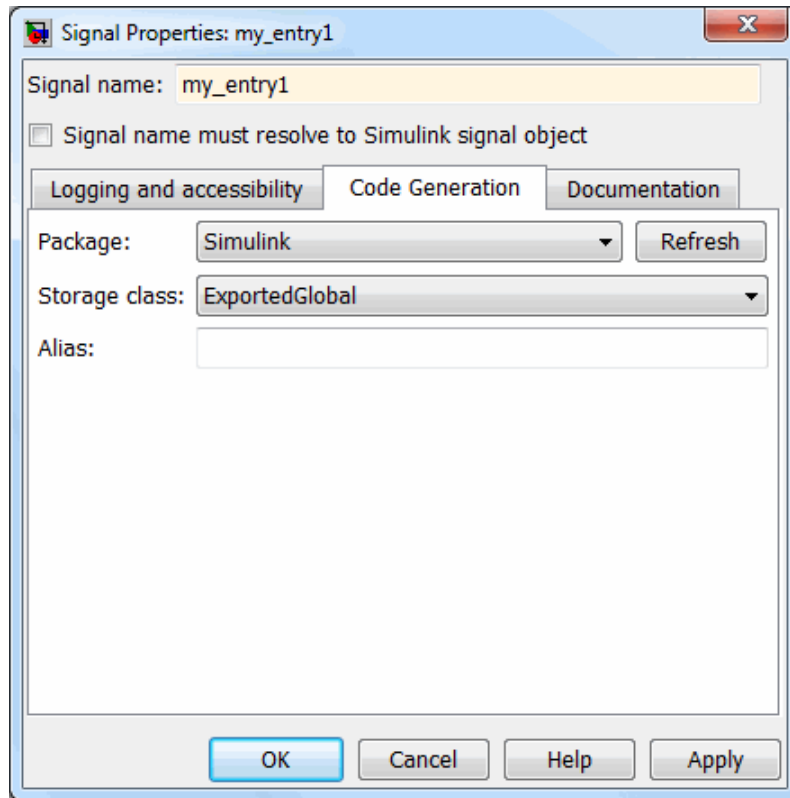
To set the storage class:

- 1** Right-click the signal-line from the **In1** block, then select Signal Properties.  
The Signal Properties dialog box opens.



**2** In the **Signal name** field, enter `my_entry1`.

**3** Select the Code Generation tab.



- 4 In the **Packages** drop-down menu, select Simulink.
- 5 In the **Storage class** drop-down menu, select ExportedGlobal.
- 6 Click **OK** to save your changes and close the dialog box.
- 7 Repeat steps 1–6 to create signal my\_entry2 for the In2 block signal.

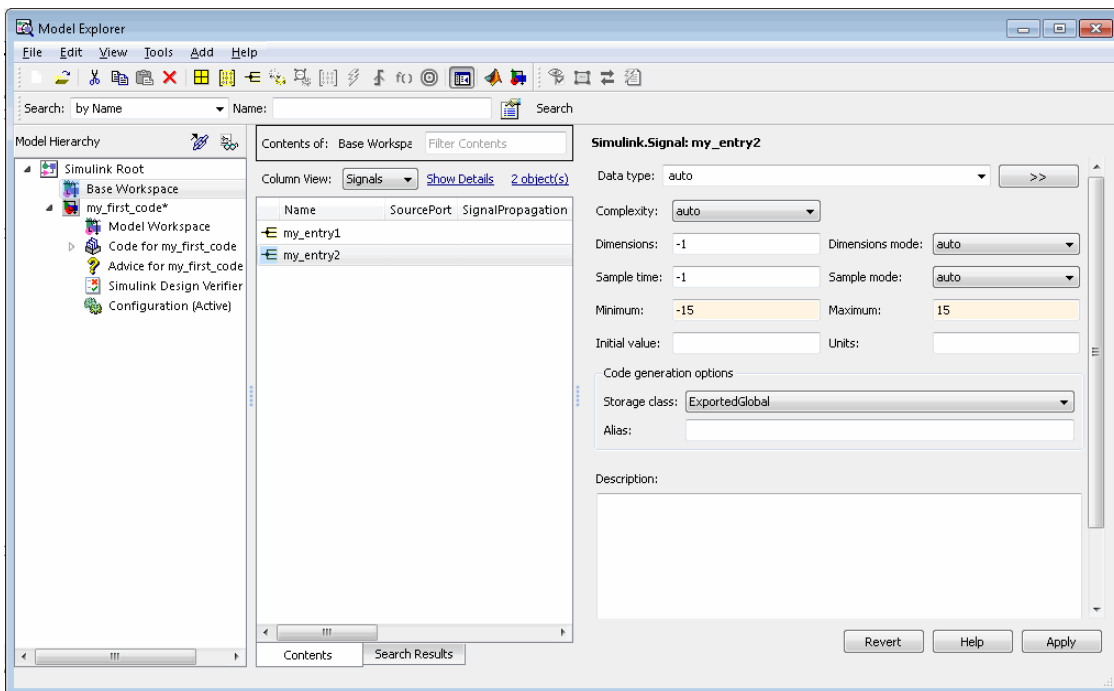
## Setting Signal Ranges

Since your signals now have the ExportedGlobal storage class, you can set ranges for the signals in the base workspace.

To set the ranges for your signals in the base workspace:



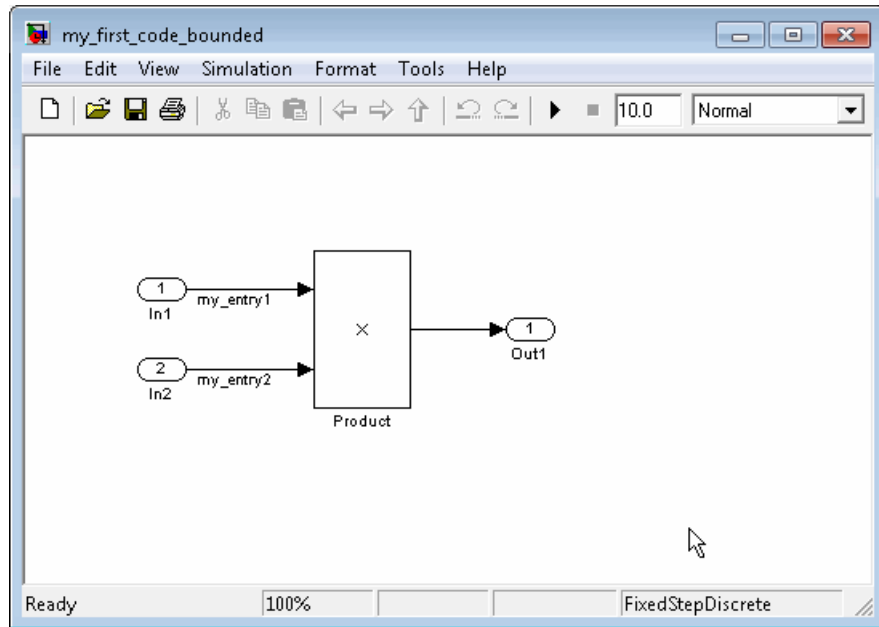
- 1 Select **View > Model Explorer** to open Model Explorer.
- 2 In the Model Hierarchy, select **Base Workspace**.
- 3 Create two signals:
  - my\_entry1
  - my\_entry2
- 4 Set the **Minimum** value for each signal to -15.
- 5 Set the **Maximum** value for each signal to 15.
- 6 Set the storage class for each signal to ExportedGlobal.



## Signals in the Base Workspace

- 7 Click **Apply**.

Your model should now look like the following example, with signals on entries:



## Model with ExportedGlobal signals my\_entry1 and my\_entry2

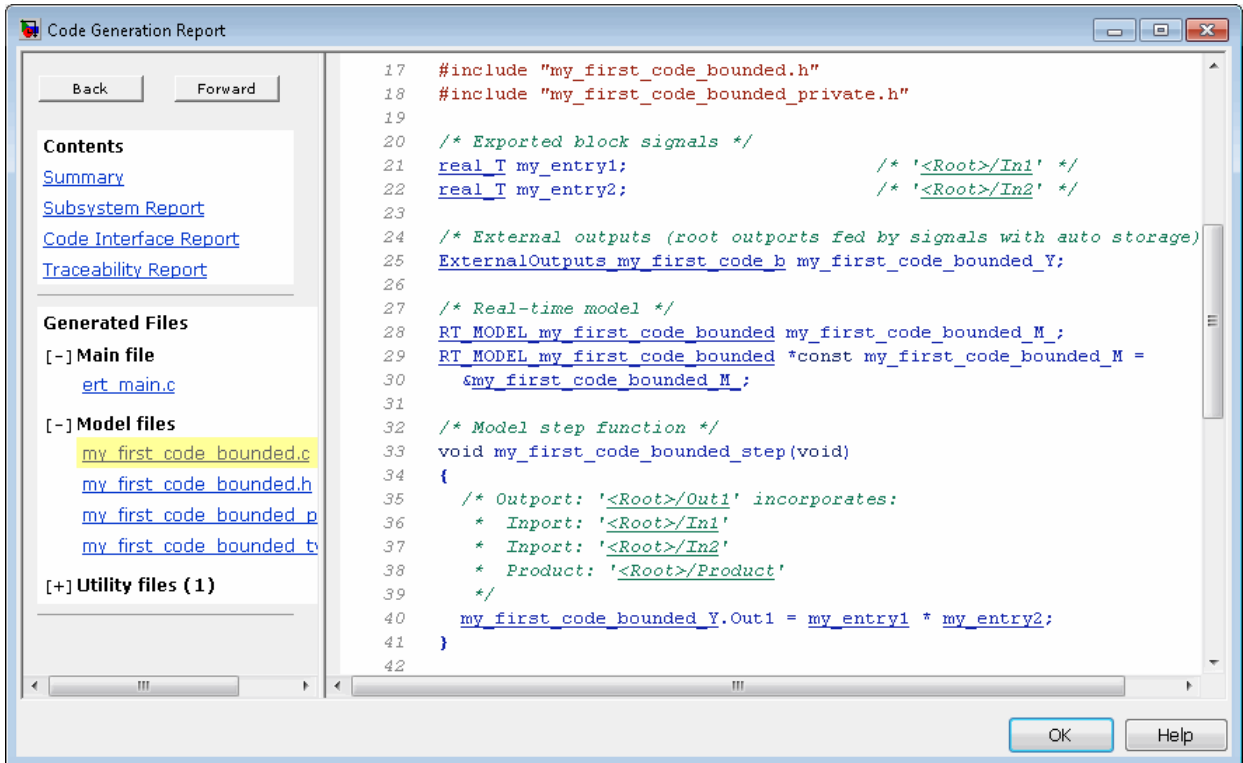
8 Save your model as my\_first\_code\_bounded.

## Re-Generate Code and Launch the Polyspace Verification Again

To regenerate the code and relaunch the Polyspace verification:

- 1 In the Simulink Model Window, select **Tools > Code Generation > Build Model** to regenerate the code

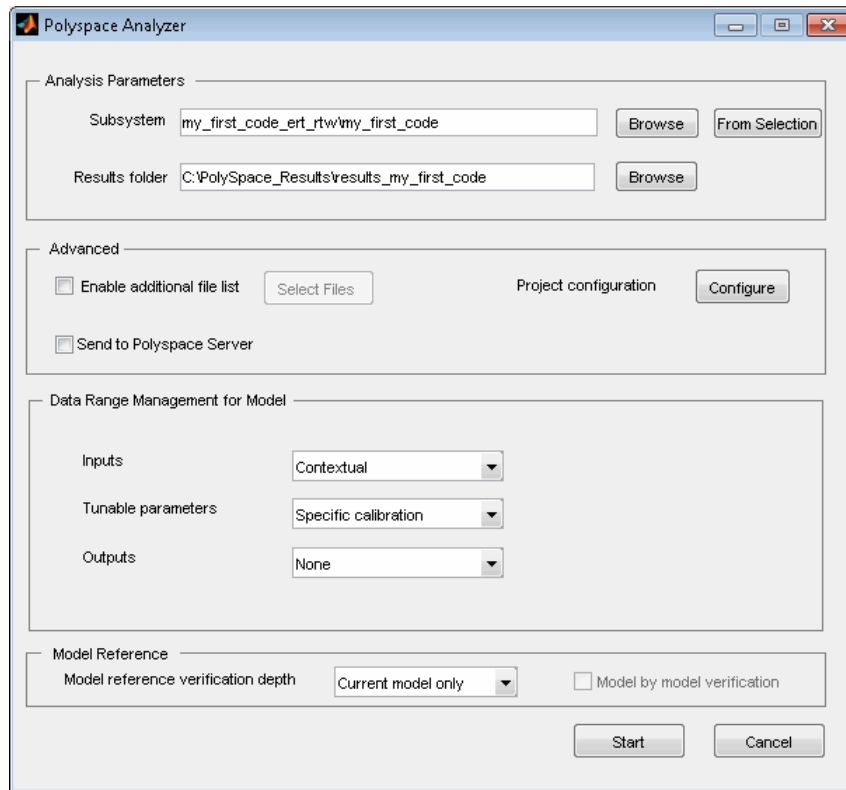
The entries are no longer part of a structure, they are separated global each.



### Html report generator from Embedded Coder™ Software

**2** Select **Tools > Polyspace > Polyspace for Embedded Coder**.

The Polyspace Analyzer dialog box opens.



## Polyspace Analyzer Dialog Box

- 3 In Results folder field enter `results_my_first_code_bounded`.
- 4 In the Subsystem field, enter `my_first_code_bounded`.
- 5 Click **Start** to start the verification.
- 6 Once verification is complete, select **Tools > Polyspace > Polyspace Utilities > Open results** to view your results.
- 7 Examine the generated files in the Polyspace Run-Time Checks perspective:

The screenshot displays the Polyspace Run-Time Checks interface. On the left, the 'Run-Time Checks' window shows a tree of procedural entities. A table below the tree provides a summary of the checks:

Entity	Failures	Warnings	Errors	Checks	Percentage
my_first_code_bounded	0	0	0	15	100
ert_main.c	0	0	0	0	0
my_first_code_bounded.c	0	0	0	5	100
_init_globals()	0	0	0	0	0
my_first_code_bounded_init	0	0	0	2	100
my_first_code_bounded_step	0	0	0	3	100
NIV.0	0	0	0	1	100
OVFL.1	0	0	0	1	100
NIV.2	0	0	0	1	100
my_first_code_bounded_term	0	0	0	0	0
stdio.h	0	0	0	0	0
__polyspace_stdstubs.c	0	0	0	0	0
__polyspace_main.c	0	0	0	10	100

The 'Review Details' window shows a message for the operation `my_first_code_bounded_Y.Out1 = my_entry1 * my_entry2;` at line 40, column 43. The message states: "Operation [\*] on float does not overflow in FLOAT64 range". A table below the message provides details:

Classification	Status	Justified	Comment
		<input type="checkbox"/>	

The 'Source' window shows the following code snippet:

```

27  /* Real-time model */
28  RT_MODEL_my_first_code_bounded my_first_code_bounded_M;
29  RT_MODEL_my_first_code_bounded *const my_first_code_bounded_M =
30    &my_first_code_bounded_M;
31
32  /* Model step function */
33  void my_first_code_bounded_step(void)
34  {
35    /* Output: '<Root>/Out1' incorporates:
36     * Inport: '<Root>/In1'
37     * Inport: '<Root>/In2'
38     * Product: '<Root>/Product'
39     */
40    my_first_code_bounded_Y.Out1 = my_entry1 * my_entry2;
41  }

```

### Detail of generated files viewed in Polyspace® Run-Time Checks perspective

Everything is green. Polyspace verification has confirmed that no Runtime Errors are present in the model.

## Can You Find More Bugs in the Model?

To answer this question, we need to know more about the tool, such as:

- Which windows in the Run-Time Checks perspective contain what information
- Which colors hide which messages
- How to find bugs using the Polyspace Verification Environment

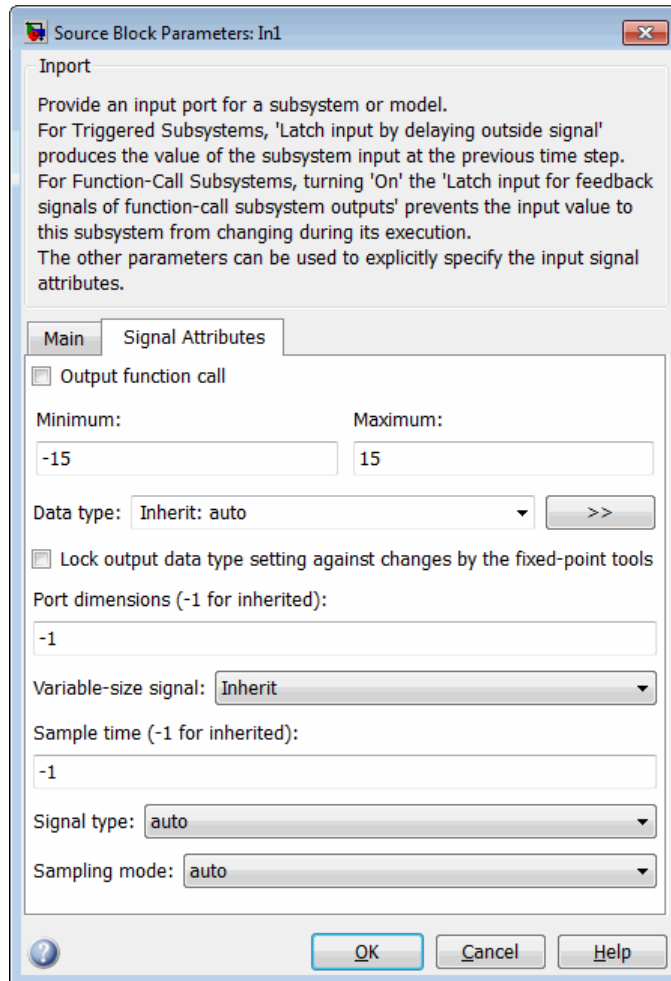
For more information, see “Reviewing Verification Results” in the *Polyspace Products for C User’s Guide*.

## Setting Data Ranges Using Block Parameters

In addition to providing data ranges in the base workspace, you can specify data ranges using block parameters. This method is often easier than creating signal objects in the base workspace.

To specify data ranges using source block parameters:

- 1 Double-click the **In1** 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.





# Advanced Setup Options

---

## Advanced Setup

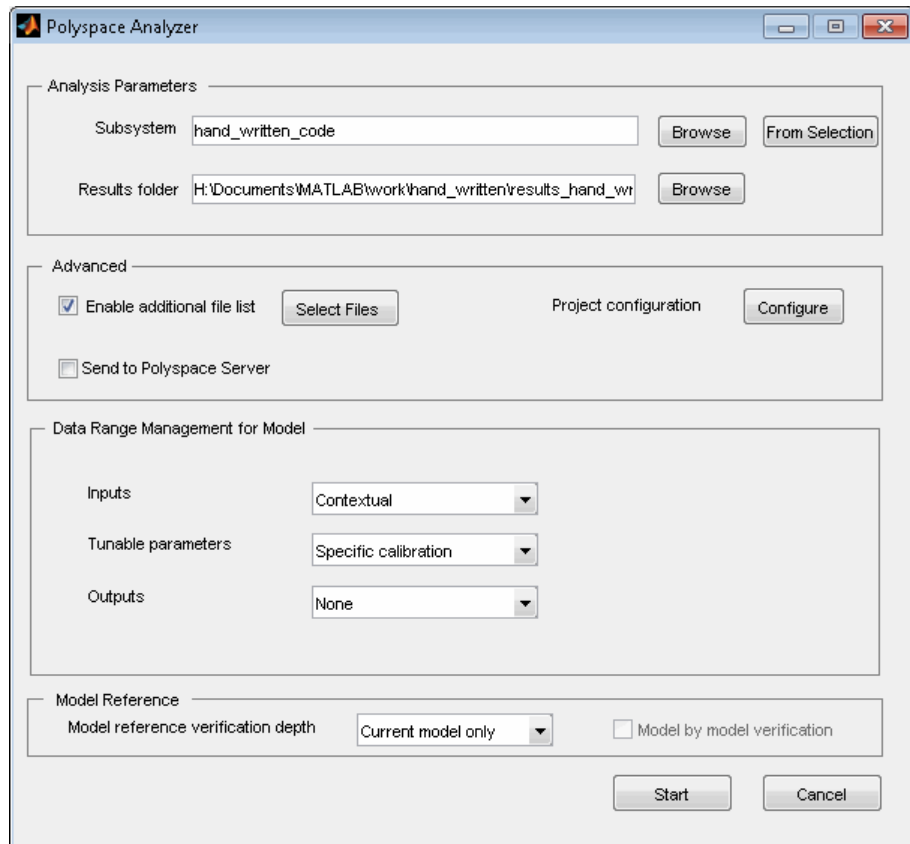
In this section...
“Handwritten Code” on page 2-2
“Target Production Environment” on page 2-5
“Creating a Polyspace Configuration File Template” on page 2-7
“Data Range Management” on page 2-9
“Main Generation for Model Verification” on page 2-11
“Annotating Blocks to Justify Known Checks or Coding-Rules Violations” on page 2-13

### Handwritten Code

Files such as S-function wrappers are, by default, not part of the Polyspace verification. They should be added manually.

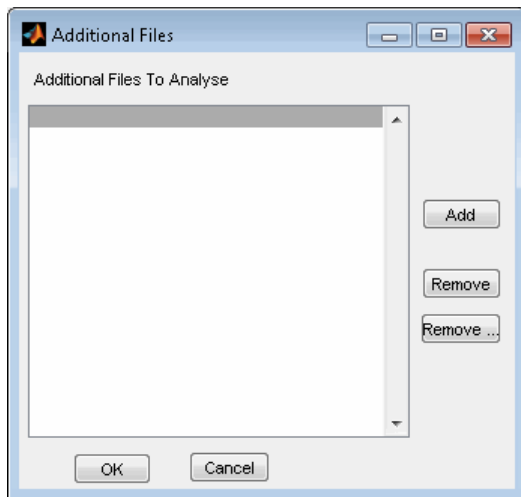
To add a file manually:

- 1 When starting the Polyspace verification, browse and add c-files to your verification:



- 2 Select additional files by ticking “Enable additional file list,” then click on “Select Files”.

A C File browser appears to add files to the Polyspace verification.



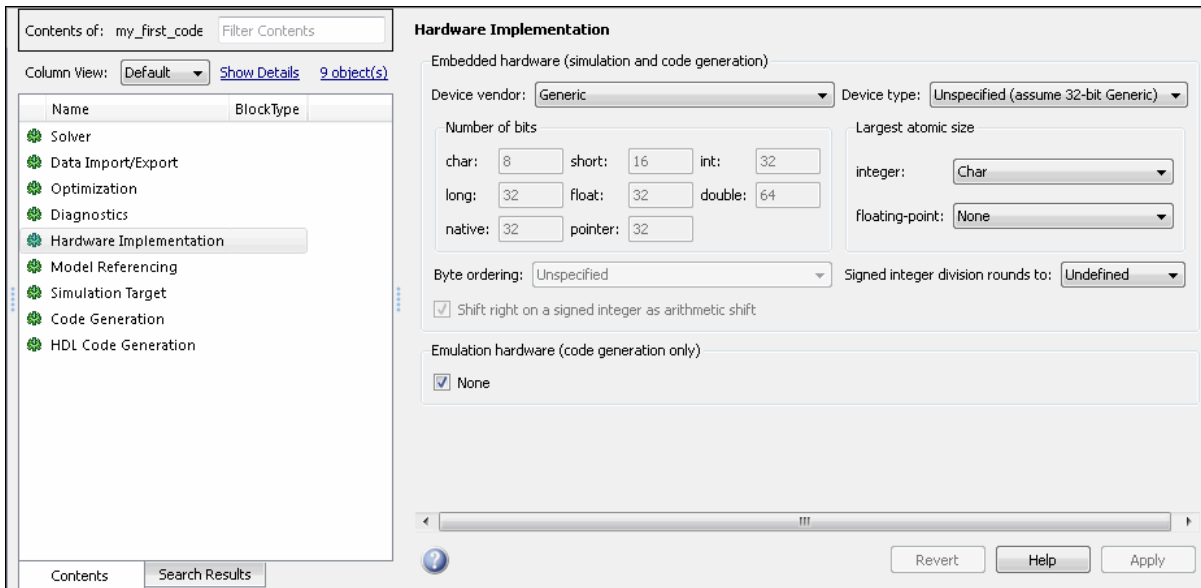
**3** Select the appropriate c file and then start the verification.

## Target Production Environment

In Simulink software, you need to configure the target and cross-compiler specificities.

These parameters include:

- Size of the types for char, short, int (see Hardware implementation of the model explorer)



### Target selection in Simulink® Configuration Parameters

- Cross compiler flag (-D), and library include (-I), implicitly defined when - for instance - the cross compiler is setup via the “mex -setup” command.

```
Command Window
>> mex -setup
Please choose your compiler for building external interface (MEX) files:

Would you like mex to locate installed compilers [y]/n?

Select a compiler:
[1] Lcc C version 2.4.1 in C:\MATLAB\R2006b\sys\lcc
[2] Microsoft Visual C/C++ version 6.0 in C:\Program Files\Microsoft Visual S
[0] None

Compiler: 1

Please verify your choices:

Compiler: Lcc C 2.4.1
Location: C:\MATLAB\R2006b\sys\lcc

Are these correct?([y]/n):

Trying to update options file: C:\Documents and Settings\Marc Lalo\Applicatio
From template: C:\MATLAB\R2006b\bin\win32\mexopts\lccopts.bat

Done . . .

>> |
```

### Cross compiler settings in MATLAB® Command Window

Polyspace settings work exactly the same way, you will need to perform the following tasks (they will be detailed step by step in the next sections).

- 1 define the same parameters for your cross compiler and target.
- 2 save this in a template Polyspace configuration file and set this template to be the default configuration file for every Polyspace verification.

Why does this matter?

- For the Polyspace verification, an overflow on an integer type does not mean the same when the size of an integer is 16 bits or 32 bits.

- Polyspace software needs the cross compiler header files, as they contain definitions of types, macros, used by the application, whether the application made of generated code or hand written code.

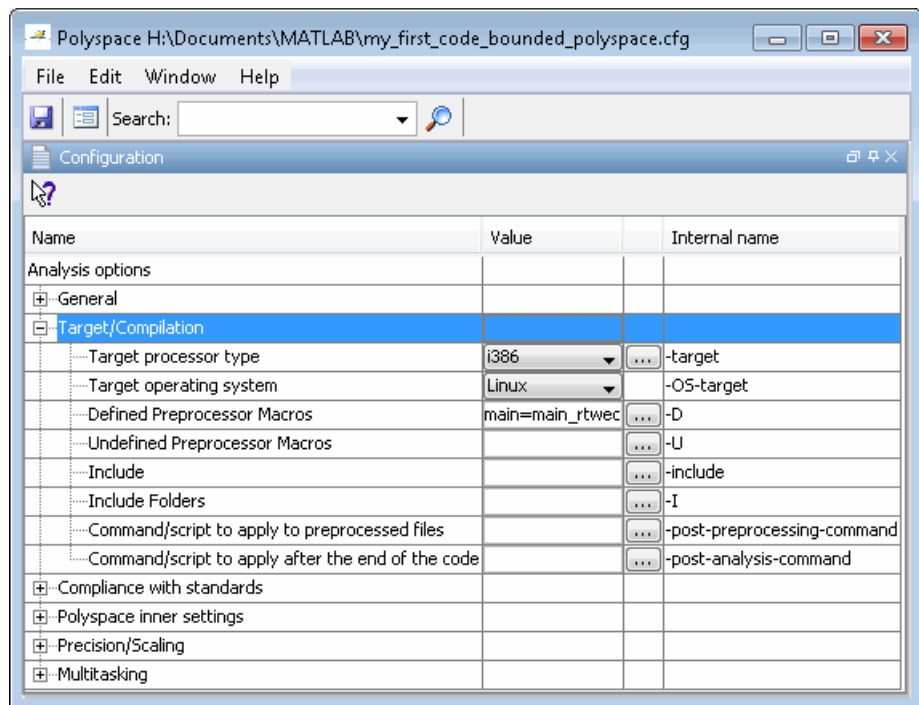
For more information, refer to and “Option Descriptions” in the *Polyspace Products for C Reference*.

## Creating a Polyspace Configuration File Template

To Create a configuration file template:

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

The Project Manager perspective of the Polyspace Verification Environment interface opens, allowing you to customize the target and cross compiler.



**Target and cross compiler settings in Polyspace® tools**

- 2 The `-target` option defined the size of types. You can configure a custom target by selecting `mcpu (advanced)` at the bottom of the drop-down list
- 3 You can configure cross compiler settings by clicking on the `-D` options.

Target/Compilation			
Target processor type	i386	...	-target
Target operating system	no-predefined-OS	...	-OS-target
Defined Preprocessor Macros	main=main_rtvec, __restrict__=	...	-D
Undefined Preprocessor Macros		...	-U

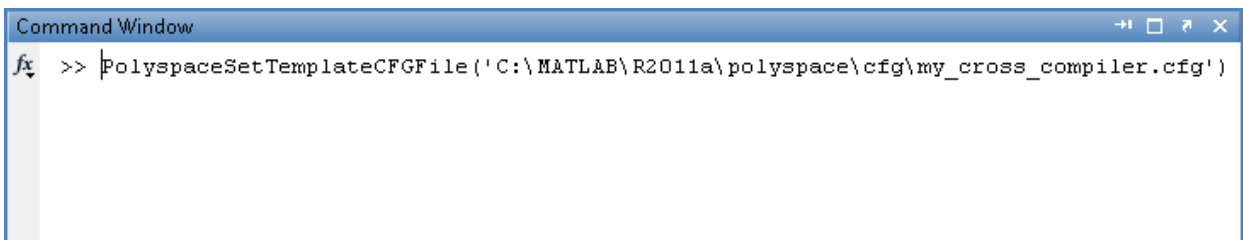
---

**Note** `MATLAB_MEX_FILE` is a directive option that is needed when the LCC cross-compiler is specified. Defining templates can be use in all subsequent verification.

---

- 4 Save the configuration file and close the interface.
- 5 Copy the file in `<matlabroot>/polyspace/cfg` directory.
- 6 Rename it in `my_cross_compiler.cfg` (It could be any other name).
- 7 Type in the MATLAB command window:

```
PolyspaceSetTemplateCFGFile
('C:\MATLAB\R2011a\polyspace\cfg\my_cross_compiler.cfg')
```



### Create a template configuration file

This configuration file can now be used as a template for all subsequent verification.



## Data Range Management

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

- **Contextual Verification** – Prove software 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 software works under all conditions, including “abnormal” conditions for which it was not designed. This can be thought of as “worst case” verification.

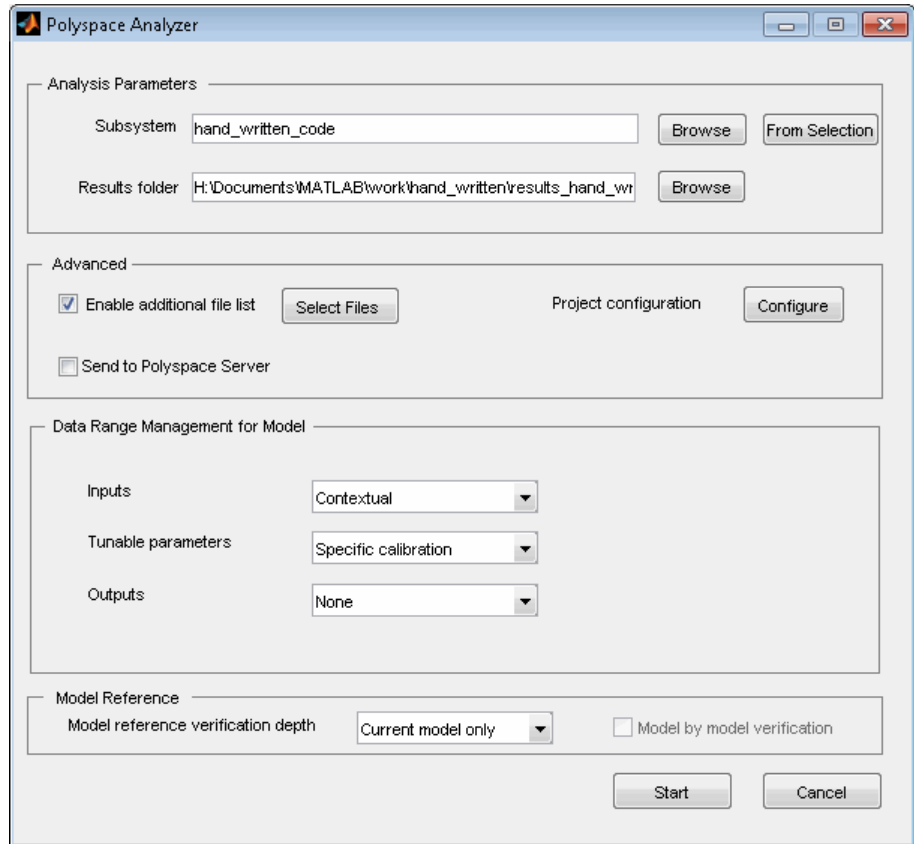
For more information, see “Choosing Robustness or Contextual Verification” in the *Polyspace Products for C User’s Guide*.

Polyspace Model Link SL allows you to run either contextual or robustness verification by specifying how the verification handles data ranges on model inputs, outputs, and tunable parameters within the model.

To specify data range settings for your model:

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

The Polyspace Analyzer dialog box opens.



**2** In the Data Range Management for Model section, specify how the verification handles **Inputs**:

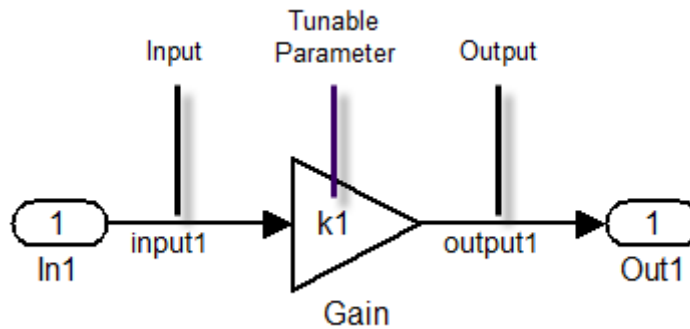
- **Contextual** (Default) – Uses data ranges defined in blocks or workspace to increase the precision of the verification.
- **Robustness** – Assumes all Inputs are full-range values (`min...max`).

**3** Specify how the verification handles **Tunable parameters**:

- **Specific calibration** (Default) – Use the constant parameter value specified in the code.
- **Generic calibration** – Use a parameter range defined in the block or workspace. If no range is defined, use full range (`min...max`).

#### 4 Specify how the verification handles Outputs:

- **None** (Default) – No assertion ranges on outputs.
- **Global assert** – Use assertion ranges on outputs.



#### Data Range Management for Model

---

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

---

In general, you should use contextual inputs and specific calibration for parameters to maximize verification precision, and robustness inputs and generic calibration for to verify the worst cases of program execution.

---

**Note** Data Range Management settings require Simulink Version 7.4 (R2009b) or later.

---

### 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 Polyspace option `-functions-called-before-loop`.
- 3** It initializes inputs using the Polyspace option `-variables-written-in-loop`.
- 4** It calls the step function using the Polyspace option `-functions-called-in-loop`.
- 5** It calls the terminate function using the Polyspace 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
```

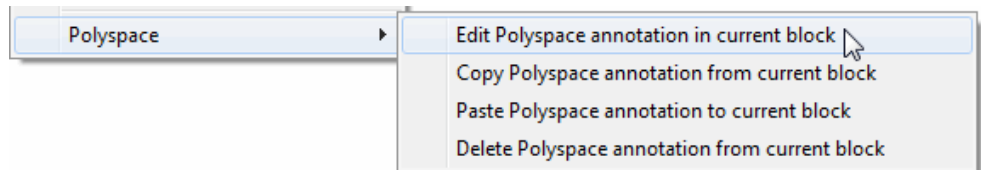
## Annotating Blocks to Justify Known Checks or Coding-Rules Violations

You can place annotations on individual blocks in your Simulink model that 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 your verification results.

The Polyspace Run-Time Checks perspective displays the information that you provide with block annotations, and marks the checks as Justified.

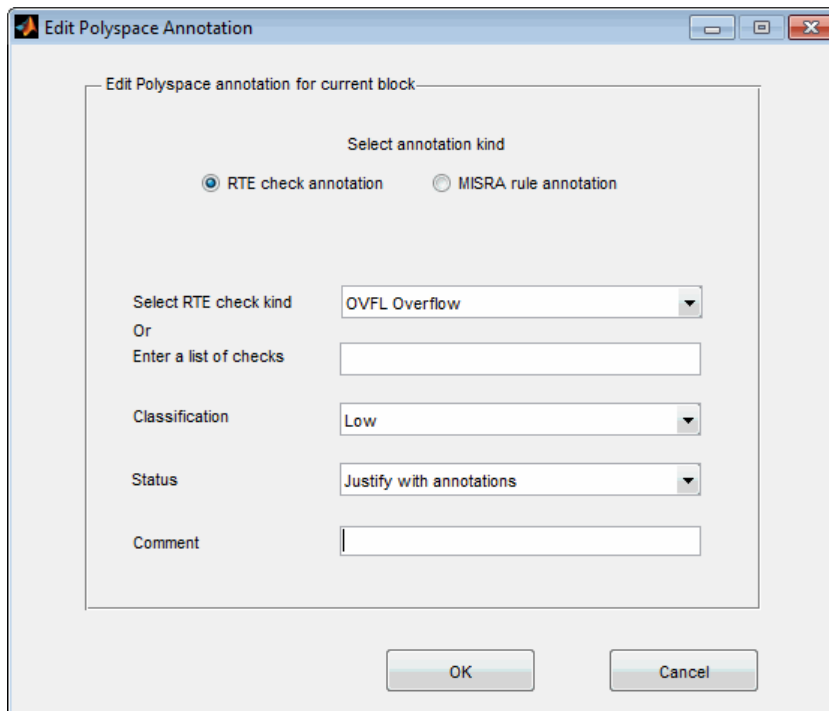
To annotate a block:

- 1 In the Simulink model window, Right-click the block you want to annotate.



- 2 Select **Polyspace > Edit Polyspace annotation in current block**.

The Edit Polyspace Annotation dialog box opens.



**3** Select the type of annotation:

- **RTE check annotation**
- **MISRA rule annotation**

**4** Select the type of RTE check or the coding rule you are commenting.

**5** Select a **Classification** to describe the seriousness of the issue:

- High
- Medium
- Low
- Not a defect

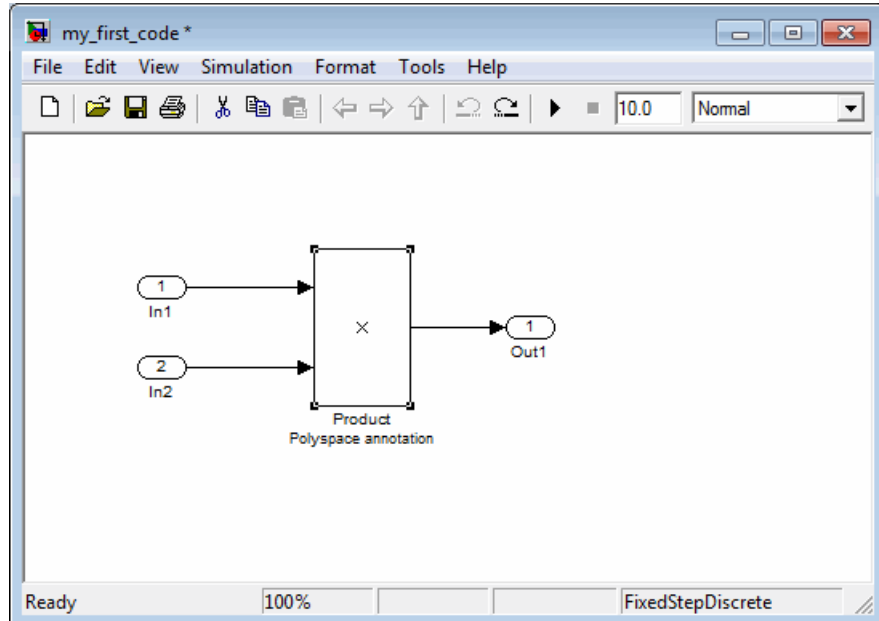
**6** Select a **Status** to describe how you intend to address the issue:

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

**7** In the comment box, enter additional information about the check.

**8** Click **OK**.

The Polyspace annotation is added to the block.







# Polyspace Utilities

---

- “Polyspace Utilities” on page 3-2
- “Polyspace Commands Available in Batch Mode as M-Functions” on page 3-6
- “Archives Files Produced for the Polyspace Verification” on page 3-8

## Polyspace Utilities

### In this section...

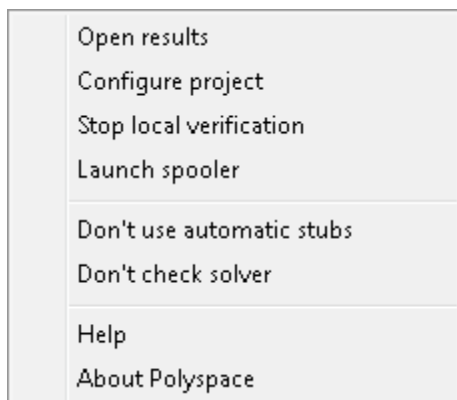
“Overview of Polyspace Utilities” on page 3-2

“Configuring Polyspace Project” on page 3-4

### Overview of Polyspace Utilities

The Polyspace utilities menu allows you to access the options in the Polyspace Library directly from the Simulink model window.

To access the utilities menu, select **Tools > Polyspace > Polyspace Utilities** from the model window.



The Utilities menu contains the following options:

- **Open results** – Opens the Run-Time Checks perspective with the last available results. You can then navigate directly from the verification results to elements in the Simulink model. If the verification ran on the server, you must download your results before selecting this option. Do not change the proposed directory during download.

**Configure project** – Opens the Polyspace configuration dialog, for more information see the next section, “Configuring Polyspace Project” on page 3-4.

- **Stop local verification** – Stops a verification running on the local machine. If the verification is run on the server, this option only works during the compilation phase before the verification is sent to the server. However, you can click the **Launch spooler** button and stop the verification from the spooler dialog.
- **Launch spooler** – Opens the Polyspace spooler. For more information, see “Running Verifications on Polyspace Server” in the *Polyspace Products for C User’s Guide*.
- **Don’t use automatic stubs** – Specifies that the verification will not generate stubs. By default, Polyspace verification stubs all functions. Using this option allows you to use manual stubbing. For more information, see “Stubbing” in the *Polyspace Products for C User’s Guide*.
- **Don’t check solver** – Specifies that the model be verified regardless of the type of solver selected. To ensure optimal precision and performance, the software checks that the model uses a fixed-step discrete solver. Selecting this option disables this check.
- **Help** – Opens the Polyspace documentation.

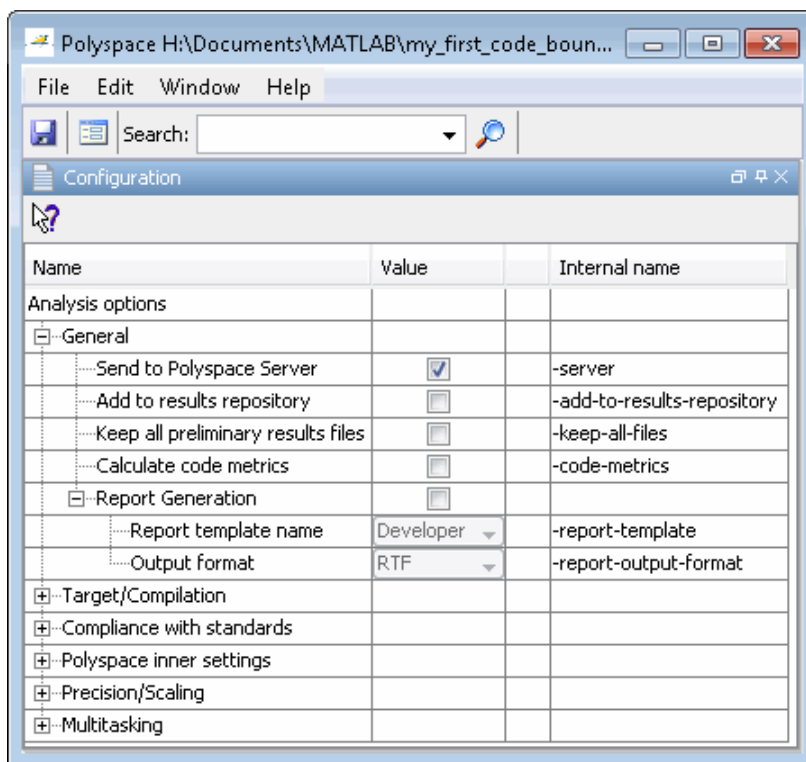
## Configuring Polyspace Project

Selecting “Configure project” opens a simplified version of the Polyspace Project Manager that allows you to customize your project Configuration. For example, you can set options such as the target processor type, target operating system, and compilation flags.


The first time you open the configuration, the following options are set:

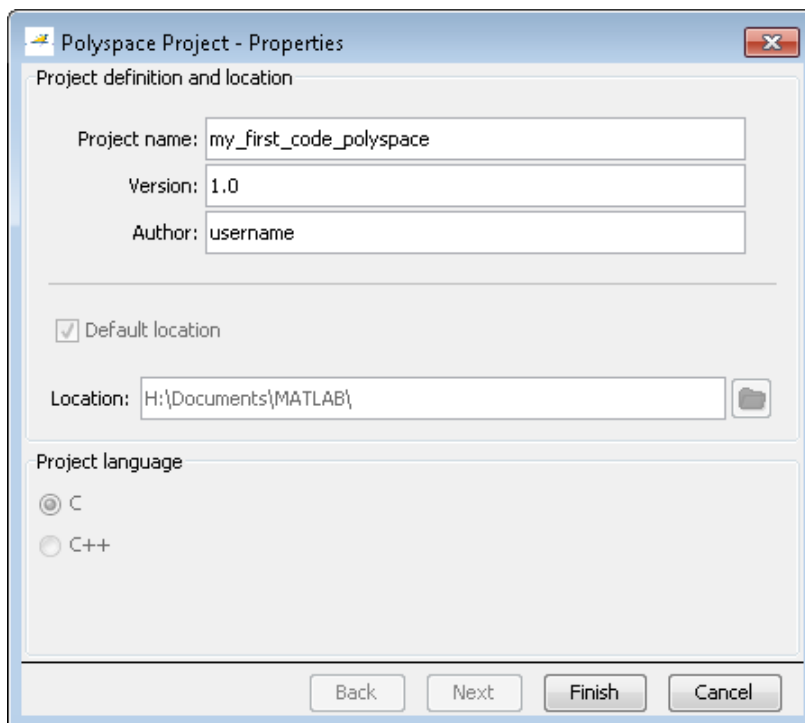
```
-OS-target no-predefined-OS
-results-dir results
```

Other options are automatically configured depending on the code generator you use. For more information, see Chapter 4, “Code Generator Specific Information”.



**Polyspace Configuration Dialog Box**

Select the **Project properties** icon  in the toolbar to open the Project – Properties dialog box.



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

## Polyspace Commands Available in Batch Mode as M-Functions

You can also run the following commands from the command line.

Command	Description
PolyspaceForEmbeddedCoder	Launch Polyspace verification on code generated by Embedded Coder software
PolyspaceForTargetLink	Launch Polyspace on code generated by TargetLink
PolyspaceSpooler	Inspect the queue of the remotely sent verification over the server
PolyspaceViewer	Launch Polyspace Verification Environment Run-Time Checks perspective
PolyspaceSetTemplateCFGFile	Select a template file in batch mode
PolyspaceGetTemplateCFGFile	Get the currently selected template file (empty by default)
PolyspaceReconfigure	In case of a Polyspace release update without enabling the MATLAB plug-in
PolyspaceAnnotation	Add Polyspace annotation directly to block in Simulink model. Annotation 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.
ver	Displays the Polyspace For Model-Link Version number along with other MathWorks product information

### Example with Embedded Coder product:

Suppose that you open a Simulink model with the name `example.mdl`.

Enter `PolyspaceForEmbeddedCoder('example')` in the MATLAB Command window.

The verification starts.

## Archives Files Produced for the Polyspace Verification

### In this section...

“Template files located in MATLAB installation directory\polyspace\” on page 3-8

“Files used in the model directory” on page 3-9

“Auto-generated files in the model directory” on page 3-9

### Template files located in MATLAB installation directory\polyspace\

When a verification is first performed the tool copies the following two files into the local model directory. On subsequent verifications the files are not copied again meaning it is OK to model the copies in the model directory.

- `cfg\templateEmbeddedCoder.cfg` — This file is copied to the `model_directory/model_name-polyspace.cfg` at the start of the first verification of the model. It contains the template Polyspace configuration settings to support the TargetLink code generator. The `templateTargetLink.cfg` file can be updated with site specific settings, to ease verification of new models.

A MATLAB command exists to change the name/location of the file which contains the template configuration:

```
PolyspaceSetTemplateCFGFile(config_filename)
```

This is most useful when the Polyspace verification is started as part of an automated process. Here the process would set the template configuration file to use, erase the local copy in the model directory and then start the Polyspace verification.

- `stub\ppcom_ec.sh` — This file is copied to the `model_directory/ppcom_ec.sh` at the start of the first verification of a model. The file is not recopied for subsequent verifications. It is used to stub lookup table types (only interpolation, not extrapolation) to improve the accuracy of verification results.



## Files used in the model directory

- `model-name-polyspace.cfg` — As mentioned above this file is copied from the MATLAB installation directory\`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.
- `ppcom_ec.sh` — The Polyspace Embedded Coder post preprocessing command.
- `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 the model directory

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.



# Code Generator Specific Information

---

- “Polyspace Model Link SL Product” on page 4-2
- “Polyspace Model Link TL Product” on page 4-12

## Polyspace Model Link SL Product

In this section...
“Overview” on page 4-2
“Subsystems” on page 4-2
“Default Options” on page 4-2
“Data Range Specification” on page 4-3
“Code Generation Options” on page 4-3
“Polyspace Analysis Options” on page 4-5

### 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_rtvec  
-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 “Data Range Management” on page 2-9.

## Code Generation Options

This section describes recommended configuration parameter settings for Simulink® Coder™ software. MathWorks recommends using these options for optimum use of Polyspace verification.

In the **Code Generation** tab:

- 1** Select **Generate HTML report**.
- 2** Select **Include hyperlinks to model**.

---

**Note** If you do not set this option, navigation from Polyspace results to the model will not work.

---

- 3 Set the system target file to be an appropriate `ert.tlc` (use the browse button to locate).

This is an indication that the code generator is Embedded Coder software (and not just Simulink Coder software, used for rapid prototyping).

In the **Solver** tab:

- 1 Set the solver **Type** to *Fixed-step*,
- 2 Set the **Solver** to *discrete* (no continuous state).

This specifies that the code is generated for a target, and not for a simulation based on continuous timing.

In the **Interface panel** tab:

- 1 Ensure that **Generate reusable code** is not selected.

Setting this option will generate more warnings in the Polyspace results.

## Polyspace Analysis Options

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 > Polyspace Utilities > Configure Project** in the Simulink model window.

Option	Recommended Value	Comments
<b>Polyspace Project – Properties</b>		
-prog	<session identifier>	Specifies the application name. Use characters valid for Unix file names.  This information is specified in the Polyspace Project – Properties dialog box as Project name.  For example, New_Project.
-author	<your name>	Specifies the name of the author of the verification. This information is specified in the Polyspace Project – Properties dialog box.
-desktop	<b>Selected</b> – When checking MISRA Compliance (code analysis).  <b>Not Selected</b> – When checking runtime errors (code verification)	Specifies whether verification occurs on the Polyspace Client or Polyspace Server. MathWorks recommends using the Client to perform MISRA analysis and the Server for code verification.
-I	See Comments	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.  A script to automatically determine -I based on buildInfo is available

Option	Recommended Value	Comments
-results-dir	<results dir>	<p>Specifies the folder in which Polyspace software saves verification results. You can specify a relative path. However, be careful if you plan to launch verification remotely over a network, or if you plan to copy the project configuration file using the "Save as" option.</p>
<b>General Options</b>		
-sources -sources-list-file	See Comments	<p>Specifies a list of source files to be verified. You must enclose the list of source files in double-quotes, separated by commas.</p> <ul style="list-style-type: none"> <li>• -sources "file1[ file2[ ...]]" (Linux and Solaris™) -</li> <li>• -sources "file1[,file2[, ...]]" (Windows®, Linux and Solaris)</li> <li>• -sources-list-file file_name (not a graphical option)</li> </ul> <p>You can specify multiple files using UNIX® standard wildcards.</p> <p>The software compiles the source files in the order in which you specify them.</p> <p>If you do not specify any files, the software verifies all files in the source directory in alphabetical order.</p> <hr/> <p><b>Note</b> The specified files must have valid extensions:            *. (c C cc cpp CPP cxx CXX)</p>



Option	Recommended Value	Comments
		A script to automatically determine <code>-sources</code> based on <code>buildInfo</code> is available.
-verif-version	1.0	Specifies the version identifier of the verification. You can use this option to identify multiple verifications of the same project. This information is identified in the GUI as the Version.
<b>Target/Compilation Options</b>		
-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>		

Option	Recommended Value	Comments
-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 (-I option)</li> <li>• <b>Source files</b> – All source files selected for the verification (-sources option)</li> </ul>
-misra2	[all-rules   file_name]	<p>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:</p> <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	<p>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.</p>
<p><b>Polyspace inner settings Options</b></p>		

Option	Recommended Value	Comments
-variables-written-before-loop	public	<p>Specifies how the generated main initializes global variables.</p> <p>By selecting public, every variable except static and const variables are assigned a "random" value, representing the full range of possible values</p>
-functions-called-in-loop	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>
-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>
<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>

Option	Recommended Value	Comments
-from	scratch	<p>Specifies the phase from which verification starts.</p> <p>You can use this option on an existing verification to elaborate on the results that you have already obtained. For example, if a verification ran <code>-to pass1</code>, you can restart the verification <code>-from pass1</code> to reduce verification time.</p> <p>Note the following:</p> <ul style="list-style-type: none"> <li>• This option can be used only for client verifications. All server verifications start from <code>scratch</code>.</li> <li>• All options other than <code>scratch</code> can be used only if the previous verification was launched using the option <code>-keep-all-files</code>.</li> <li>• You cannot use this option if you modify the source code between verifications.</li> </ul>
-to	<p><b>c-compile</b> – When checking MISRA compliance only.<b>pass0</b></p> <p>– When verifying code for the first time.</p> <p><b>pass4</b> – 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>MathWorks recommends you 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>

# Polyspace Model Link TL Product

In this section...
“Overview” on page 4-12
“Subsystems” on page 4-12
“Data Range Specification” on page 4-12
“Lookup Tables” on page 4-13
“Code Generation Options” on page 4-14

## 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.

## 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 — see for more information. 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.

## 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.

---

### **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).



**Atomic**

In computer programming, atomic describes a unitary action or object that is essentially indivisible, unchangeable, whole, and irreducible.

**Atomicity**

In a transaction involving two or more discrete pieces of information, either all of the pieces are committed or none are.

**Batch mode**

Execution of Polyspace from the command line, rather than via the Graphical User Interface.

**Category**

One of four types of orange check: *potential bug*, *inconclusive check*, *data set issue* and *basic imprecision*.

**Certain error**

See "red check."

**Check**

A test performed by Polyspace during a verification and subsequently colored red, orange, green or gray in the Run-Time Checks perspective.

**Code verification**

The Polyspace process through which code is tested to reveal definite and potential runtime errors and a set of results is generated for review.

**Dead Code**

Code which is inaccessible at execution time under all circumstances due to the logic of the software executed prior to it.

**Development Process**

The process used within a company to progress through the software development lifecycle.

**Green check**

Code has been proven to be free of runtime errors.

**Gray check**

Unreachable code; dead code.

**Imprecision**

Approximations are made during a Polyspace verification, so data values possible at execution time are represented by supersets including those values.

**mcpu**

Micro Controller/Processor Unit

**Orange check**

A warning that represents a possible error which may be revealed upon further investigation.

**Polyspace Approach**

The manner of use of Polyspace to achieve a particular goal, with reference to a collection of techniques and guiding principles.

**Precision**

An verification which includes few inconclusive orange checks is said to be precise

**Progress text**

Output from Polyspace during verification to indicate what proportion of the verification has been completed. Could be considered as a “textual progress bar”.

**Red check**

Code has been proven to contain definite runtime errors (every execution will result in an error).

**Review**

Inspection of the results produced by a Polyspace verification.

**Scaling option**

Option applied when an application submitted to Polyspace proves to be bigger or more complex than is practical.

**SelectivityPolyspace**

The ratio (green checks + gray checks + red checks) / (total amount of checks)

**Unreachable code**

Dead code.

**Verification**

The Polyspace process through which code is tested to reveal definite and potential runtime errors and a set of results is generated for review.