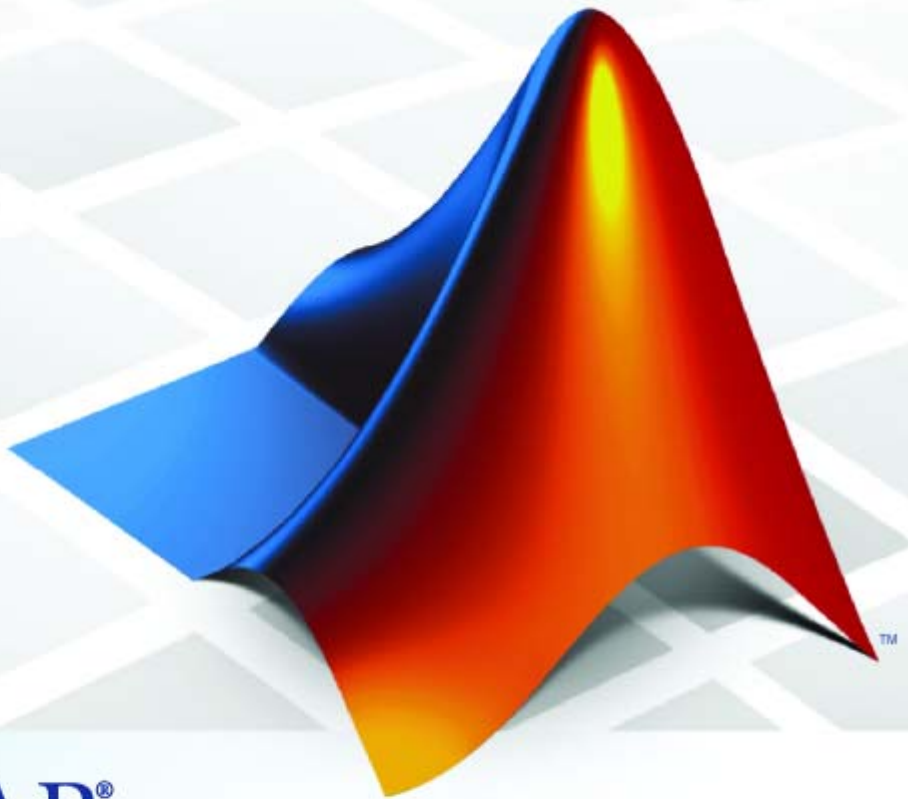


# Simulink® Verification and Validation™ 2 User's Guide



MATLAB®  
& SIMULINK®

## How to Contact The 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.

*Simulink® Verification and Validation™ User's Guide*

© COPYRIGHT 2004–2010 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

The 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

June 2004	First printing	New for Version 1.0 (Release 14)
October 2004	Online only	Revised for Version 1.0.1 (Release 14SP1)
March 2005	Online only	Revised for Version 1.0.2 (Release 14SP2)
April 2005	Second printing	Revised for Version 1.1 (Web release)
September 2005	Online only	Revised for Version 1.1.1 (Release 14SP3)
March 2006	Online only	Revised for Version 1.1.2 (Release 2006a)
September 2006	Online only	Revised for Version 2.0 (Release 2006b)
March 2007	Online only	Revised for Version 2.1 (Release 2007a)
September 2007	Online only	Revised for Version 2.2 (Release 2007b)
March 2008	Online only	Revised for Version 2.3 (Release 2008a)
October 2008	Online only	Revised for Version 2.4 (Release 2008b)
March 2009	Online only	Revised for Version 2.5 (Release 2009a)
September 2009	Online only	Revised for Version 2.6 (Release 2009b)
March 2010	Online only	Revised for Version 2.7 (Release 2010a)



## Getting Started

### 1

<b>Product Overview</b> .....	1-2
<b>System Requirements</b> .....	1-3
Operating System Requirements .....	1-3
Product Requirements .....	1-3

## Requirements Linking and Traceability

### About the Requirements Management Interface

### 2

<b>Creating Links to Requirements with the Requirements Management Interface</b> .....	2-2
<b>What Is a Requirement Link?</b> .....	2-3
<b>Supported Requirements Documents Types and Links</b> .....	2-4
<b>Supported Model Objects for Requirements Linking</b> ..	2-7

### QuickStart Tutorials: Linking to Requirements

### 3

<b>Before You Start: Configuring the RMI for One-Way Linking</b> .....	3-2
--	-----

<b>Linking to Requirements with Selection-Based</b>	
<b>Linking</b> .....	3-4
What Is Selection-Based Linking? .....	3-4
Creating a Link Using Selection-Based Linking .....	3-4
Customizing Selection-Based Linking .....	3-4
<b>Tutorial: Linking to Requirements in Microsoft Word</b>	
<b>Documents</b> .....	3-7
Navigating from a Model to a Requirements Document ...	3-7
Creating a Link from a Model Object to a Microsoft Word	
Requirements Document .....	3-8
Creating a Link from a Model to a Requirements	
Document .....	3-9
Adding Requirement Links to Multiple Model Objects	
Simultaneously .....	3-9
<b>Tutorial: Linking to Requirements in IBM Rational</b>	
<b>DOORS Databases</b> .....	3-11
<b>Creating Requirements Reports</b> .....	3-12

## Creating and Managing Requirements Links

# 4

<b>The Requirements Dialog Box</b> .....	4-2
<b>Tutorial: Managing Requirements Links to Microsoft®</b>	
<b>Excel Workbooks</b> .....	4-3
Before You Create Links .....	4-3
Navigating from a Model Object to Requirements in a	
Microsoft® Excel Workbook .....	4-3
Creating Requirements Links to the Workbook .....	4-4
Changing Requirements Links .....	4-5
Deleting Requirements Links .....	4-6
<b>Tutorial: Creating Links to MuPAD Notebooks</b> .....	4-8

<b>Tutorial: Linking Signal Builder Blocks to Requirements</b> .....	<b>4-10</b>
--	-------------

## Reviewing Requirements Information in a Model

# 5

<b>Highlighting Requirements in a Model</b> .....	<b>5-2</b>
Highlighting a Model Using the Model Editor .....	<b>5-2</b>
Highlighting a Model Using the Model Explorer .....	<b>5-4</b>
<b>Navigating to Requirements from a Model</b> .....	<b>5-5</b>
Navigating from the Model Object .....	<b>5-5</b>
Navigating from a System Requirements Block .....	<b>5-5</b>
<b>Creating and Customizing a Requirements Report</b> ...	<b>5-7</b>
Creating a Default Requirements Report .....	<b>5-7</b>
Reporting on Requirements in Model Blocks .....	<b>5-14</b>
Customizing the Requirements Report .....	<b>5-16</b>
<b>Filtering Requirements</b> .....	<b>5-21</b>
Filtering Requirements with User Tags .....	<b>5-21</b>
Applying a User Tag to a Requirement .....	<b>5-21</b>
Filtering, Highlighting, and Reporting with User Tags ...	<b>5-23</b>
Applying User Tags During Selection-Based Linking ....	<b>5-25</b>
Configuring Requirements Filtering .....	<b>5-27</b>

## Keeping Requirements Information Up to Date

# 6

<b>Checking Requirements Consistency</b> .....	<b>6-2</b>
Running Consistency Checks .....	<b>6-2</b>
Fixing Inconsistent Links .....	<b>6-4</b>
Resolving the Document Path .....	<b>6-7</b>

<b>Deleting Requirement Links from Simulink Objects ..</b>	<b>6-9</b>
Deleting a Single Link from a Simulink Object .....	6-9
Deleting All Links from a Simulink Object .....	6-9
Deleting Links from Multiple Simulink Objects .....	6-10
<b>Managing Requirements in Linked Libraries .....</b>	<b>6-11</b>
Copying Library Blocks with Requirements .....	6-11
Links from Requirements to Library Blocks .....	6-13
Managing Requirements Links Inside Reference Blocks ..	6-14

## Synchronizing a Simulink Model with a DOORS Surrogate Module

# 7

<b>What Is Synchronization? .....</b>	<b>7-2</b>
<b>Advantages of Synchronizing Your Model with a Surrogate Module .....</b>	<b>7-4</b>
<b>Tutorial: Synchronizing a Simulink Model to Create a Surrogate Module .....</b>	<b>7-5</b>
<b>Tutorial: Creating Links Between the Surrogate Module and Formal Module in a DOORS Database During Synchronization .....</b>	<b>7-7</b>
<b>Customizing the Synchronization .....</b>	<b>7-8</b>
DOORS Synchronization Settings .....	7-8
Resynchronizing a Model with a Different Surrogate Module .....	7-10
Customizing the Level of Detail in Synchronization .....	7-11
Tutorial: Resynchronizing to Include All Simulink Objects .....	7-12
<b>Tutorial: Updating the Surrogate Module to Reflect Model Changes .....</b>	<b>7-16</b>



<b>Navigating with the Surrogate Module</b> .....	<b>7-18</b>
Navigating Between Requirements and the Surrogate Module in the DOORS Database .....	<b>7-18</b>
Two-Way Navigation with the Surrogate Module .....	<b>7-19</b>

## **Adding Navigation Controls to IBM Rational DOORS Requirements**

# 8

<b>Why Add Navigation Controls to DOORS Requirements?</b> .....	<b>8-2</b>
 <b>Configuring the Requirements Management Interface for DOORS Software</b> .....	<b>8-3</b>
Before You Begin .....	<b>8-3</b>
Configuring the RMI to Insert Navigation Controls .....	<b>8-3</b>
Manually Installing Additional Files for DOORS Software .....	<b>8-3</b>
 <b>Enabling Two-Way Linking for IBM Rational DOORS Databases</b> .....	<b>8-6</b>
 <b>Inserting Navigation Controls into DOORS Requirements</b> .....	<b>8-8</b>
 <b>Navigating Between a DOORS Requirement and a Model Object</b> .....	<b>8-10</b>
 <b>Troubleshooting Your DOORS Installation</b> .....	<b>8-12</b>
DXL Errors .....	<b>8-12</b>

## Adding Navigation Controls to Microsoft Office Documents

### 9

<b>Why Add Navigation Controls to Microsoft Office Requirements?</b> .....	<b>9-2</b>
<b>Enabling Two-Way Linking for Microsoft Office Documents</b> .....	<b>9-3</b>
<b>Inserting Navigation Controls in Microsoft Office Requirements Documents</b> .....	<b>9-5</b>
<b>Navigating Between a Microsoft Word Requirement and a Model</b> .....	<b>9-6</b>
<b>Troubleshooting Simulink Navigation Controls in Microsoft Office 2007</b> .....	<b>9-7</b>
Saving Requirements Documents to Microsoft Word 2007 Format .....	<b>9-7</b>
Field Codes in Requirements Documents .....	<b>9-8</b>
ActiveX Control Does Not Link to Model Object .....	<b>9-10</b>
Deleting an ActiveX Control from Microsoft® Excel 2007 file .....	<b>9-12</b>

## Creating Custom Types of Requirements Documents

### 10

<b>Why Create a Custom Link Type?</b> .....	<b>10-2</b>
<b>Custom Link Type Registration</b> .....	<b>10-3</b>
<b>Link Properties</b> .....	<b>10-4</b>
<b>Link Type Properties</b> .....	<b>10-5</b>

<b>Creating a Custom Link Requirement Type</b> .....	10-7
Creating a Document Index .....	10-15
<b>Navigating to Simulink Objects from External Documents</b> .....	10-17
Providing Unique Object Identifiers .....	10-17
Using the rmiobjnavigate Function .....	10-17
Determining the Navigation Command .....	10-17
Using the ActiveX Navigation Control .....	10-18
Typical Code Sequence for Establishing Navigation Controls .....	10-18

## Including Requirements Information with Generated Code

# 11

---

## Monitoring Signals in Your Model

### Using Model Verification Blocks

# 12

---

<b>When to Use Model Verification Blocks</b> .....	12-2
--	------

<b>Example: Using the Check Static Lower Bound Block to Check for Out-of-Bounds Signal</b> .....	12-3
--	------

### Using the Verification Manager

# 13

---

<b>Using the Verification Manager</b> .....	13-2
What Is the Verification Manager? .....	13-2
Opening the Verification Manager .....	13-2

Enabling and Disabling Model Verification Blocks Using the Verification Manager .....	13-8
Using Enabling and Disabling Tools in the Verification Manager .....	13-11

## Linking Verification Blocks to Requirements Documents Using the Verification Manager

# 14

## Validating Your Model with Model Coverage

### Using Model Coverage

# 15

<b>Introduction to Model Coverage</b> .....	15-2
What Is Model Coverage? .....	15-2
How Model Coverage Works .....	15-2
Types of Model Coverage .....	15-3
 <b>Model Objects That Receive Model Coverage</b> .....	15-8
Simulink Object Coverage Criteria .....	15-10
 <b>Simulink Optimizations and Model Coverage</b> .....	15-30
Block reduction .....	15-30
Conditional input branch execution .....	15-30
 <b>Analyzing Model Coverage</b> .....	15-32
Model Coverage Analysis Workflow .....	15-32
Creating and Running Test Cases .....	15-32
 <b>Model Coverage for Embedded MATLAB Function Blocks</b> .....	15-37

Types of Model Coverage in Embedded MATLAB Function Blocks .....	15-37
Creating a Model with Embedded MATLAB Function Block Decisions .....	15-39
Understanding Embedded MATLAB Function Block Model Coverage .....	15-43
<b>Colored Simulink Diagram Coverage Display .....</b>	<b>15-56</b>
How Model Coverage Highlighting Works .....	15-56
Enabling the Colored Diagram Display .....	15-56
Displaying Model Coverage with Model Coloring .....	15-57
Accessing Coverage Information for Colored Blocks .....	15-59

## Setting the Model Coverage Options

# 16

<b>Coverage Settings Dialog Box .....</b>	<b>16-2</b>
Coverage Tab .....	16-3
Results Tab .....	16-6
Report Tab .....	16-8
Options Tab .....	16-12

## Understanding Model Coverage Reports

# 17

<b>Types of Coverage Reports .....</b>	<b>17-2</b>
<b>Model Coverage Reports .....</b>	<b>17-3</b>
Coverage Summary .....	17-3
Details .....	17-5
Cyclomatic Complexity .....	17-13
Decisions Analyzed .....	17-15
Conditions Analyzed .....	17-17
MCDC Analysis .....	17-17
Cumulative Coverage .....	17-19
N-Dimensional Lookup Table .....	17-21

Block Reduction .....	17-28
Signal Range Analysis .....	17-30
Signal Size Coverage for Variable-Dimension Signals ....	17-32
Simulink® Design Verifier Coverage .....	17-34
<b>Model Summary Reports .....</b>	<b>17-38</b>
<b>Model Reference Coverage Reports .....</b>	<b>17-39</b>
<b>External MATLAB File Coverage Reports .....</b>	<b>17-40</b>
<b>Subsystem Coverage Reports .....</b>	<b>17-45</b>

## Using Model Coverage Commands

# 18

<b>About Model Coverage Commands .....</b>	<b>18-2</b>
<b>Creating Tests with cvtest .....</b>	<b>18-3</b>
<b>Running Tests with cvsim .....</b>	<b>18-6</b>
<b>Creating HTML Reports with cvhtml .....</b>	<b>18-8</b>
<b>Saving Test Runs to a File with cvsave .....</b>	<b>18-9</b>
<b>Loading Stored Coverage Test Results with cvload ...</b>	<b>18-10</b>
cvload Special Considerations .....	18-10
<b>Coverage Script Example .....</b>	<b>18-11</b>
<b>Using Model Coverage Commands for Referenced</b>	
<b>Models .....</b>	<b>18-12</b>
Introduction .....	18-12
Creating a Test Group with cv.cvtestgroup .....	18-15

Running Tests with cvsimref .....	18-15
Extracting Results from cv.cvdatagroup .....	18-16

# Customizing the Model Advisor

## Overview of the Model Advisor

### 19

<b>Why Use and Customize the Model Advisor?</b> .....	19-2
About the Model Advisor .....	19-2
Customizing the Model Advisor .....	19-2
 <b>Customizing and Using the Model Advisor Workflow</b> ..	19-4
 <b>Before Customizing the Model Advisor</b> .....	19-5

## Authoring Custom Checks

### 20

<b>Authoring Checks Workflow</b> .....	20-2
 <b>Customization File Overview</b> .....	20-3
 <b>Register Checks and Process Callbacks</b> .....	20-6
Create sl_customization Function .....	20-6
Registering Checks and Process Callbacks .....	20-6
Defining Startup and Post-Execution Actions Using Process Callback Functions .....	20-8
 <b>Defining Custom Checks</b> .....	20-11
About Custom Checks .....	20-11
Contents of Check Definitions .....	20-11
Displaying and Enabling Checks .....	20-13
Defining Where Custom Checks Appear .....	20-14

Model Advisor Code Example: Check Definition	
Function .....	20-15
Defining Check Input Parameters .....	20-16
Defining Model Advisor Result Explorer Views .....	20-18
Defining Check Actions .....	20-19
<b>Creating Callback Functions and Results .....</b>	<b>20-22</b>
About Callback Functions .....	20-22
Common Utilities for Authoring Checks .....	20-23
Simple Check Callback Function .....	20-23
Detailed Check Callback Function .....	20-31
Check Callback Function with Hyperlinked Results .....	20-33
Action Callback Function .....	20-37
Formatting Model Advisor Results .....	20-38

## Creating Custom Configurations by Organizing Checks and Folders

# 21

<b>Overview of Creating Custom Configurations .....</b>	<b>21-2</b>
About Creating Custom Configurations .....	21-2
Creating Custom Configurations Workflow .....	21-2
Using the Model Advisor Configuration Editor Versus Customization File .....	21-3
<b>Organizing Checks and Folders Using the Model Advisor Configuration Editor .....</b>	<b>21-4</b>
Overview of the Model Advisor Configuration Editor .....	21-4
Starting the Model Advisor Configuration Editor .....	21-9
How To Organize Checks and Folders Using the Model Advisor Configuration Editor .....	21-10
<b>Organizing Checks and Folders Within a Customization File .....</b>	<b>21-12</b>
Customization File Overview .....	21-12
Register Tasks and Folders .....	21-13
Defining Custom Tasks .....	21-15
Defining Custom Folders .....	21-18
Demo and Code Example .....	21-20



<b>Verifying and Using Custom Configurations</b> .....	<b>21-22</b>
Updating the Environment to Include Your sl_customization File .....	<b>21-22</b>
Verifying Custom Configurations .....	<b>21-22</b>

## Deploying Custom Configurations

# 22

<b>Overview of Deploying Custom Configurations</b> .....	<b>22-2</b>
About Deploying Custom Configurations .....	<b>22-2</b>
Deploying Custom Configurations Workflow .....	<b>22-2</b>
 <b>How to Deploy Custom Configurations</b> .....	 <b>22-3</b>
 <b>Loading and Setting the Default Configuration</b> .....	 <b>22-4</b>

## Function Reference

# 23

<b>Requirements Management Interface</b> .....	<b>23-2</b>
 <b>Model Coverage</b> .....	 <b>23-3</b>
 <b>Model Advisor Customization API</b> .....	 <b>23-5</b>
 <b>Model Advisor Result Template API</b> .....	 <b>23-7</b>
 <b>Model Advisor Formatting API</b> .....	 <b>23-8</b>

## Class Reference

24

<b>Model Coverage</b> .....	24-2
<b>Model Advisor Customization API</b> .....	24-3
<b>Model Advisor Result Template API</b> .....	24-4
<b>Model Advisor Formatting API</b> .....	24-5

## Alphabetical List

25

## Block Reference

26

## Model Advisor Checks

27

<b>Simulink® Verification and Validation Checks</b> .....	27-2
Simulink® Verification and Validation Checks Overview ..	27-2
Modeling Standards Checks Overview .....	27-3
<b>DO-178B Checks</b> .....	27-4
DO-178B Checks Overview .....	27-5
Check safety-related optimization settings .....	27-6
Check safety-related diagnostic settings for solvers .....	27-10
Check safety-related diagnostic settings for sample time ..	27-13
Check safety-related diagnostic settings for signal data ..	27-16
Check safety-related diagnostic settings for parameters ..	27-19

Check safety-related diagnostic settings for data used for debugging .....	27-22
Check safety-related diagnostic settings for data store memory .....	27-24
Check safety-related diagnostic settings for type conversions .....	27-26
Check safety-related diagnostic settings for signal connectivity .....	27-28
Check safety-related diagnostic settings for bus connectivity .....	27-30
Check safety-related diagnostic settings that apply to function-call connectivity .....	27-32
Check safety-related diagnostic settings for compatibility .....	27-34
Check safety-related diagnostic settings for model initialization .....	27-36
Check safety-related diagnostic settings for model referencing .....	27-38
Check safety-related model referencing settings .....	27-41
Check safety-related code generation settings .....	27-43
Check safety-related diagnostic settings for saving .....	27-50
Check for model objects that do not link to requirements ..	27-52
Check for proper usage of Math blocks .....	27-53
Check state machine type of Stateflow charts .....	27-54
Check Stateflow charts for ordering of states and transitions .....	27-56
Check Stateflow debugging settings .....	27-57
Check for proper usage of For Iterator blocks .....	27-59
Check for proper usage of While Iterator blocks .....	27-60
Display model version information .....	27-62
Check for proper usage of blocks that compute absolute values .....	27-63
Check for proper usage of Relational Operator blocks ....	27-65
<b>IEC 61508 Checks .....</b>	<b>27-67</b>
IEC 61508 Checks Overview .....	27-67
Display model metrics and complexity report .....	27-69
Check for unconnected objects .....	27-70
Check for fully defined interface .....	27-71
Check for questionable constructs .....	27-73
Check usage of Stateflow constructs .....	27-75
Check state machine type of Stateflow charts .....	27-79
Check for model objects that do not link to requirements ..	27-81
Display configuration management data .....	27-82

Check usage of Simulink constructs .....	27-83
<b>MathWorks Automotive Advisory Board Checks .....</b>	<b>27-87</b>
MathWorks Automotive Advisory Board Checks	
Overview .....	27-89
Check for difference in font and font sizes .....	27-91
Check transition orientations in flow charts .....	27-93
Check for display of nondefault block attributes .....	27-94
Check for proper labeling on signal lines .....	27-95
Check for propagated labels on signal lines .....	27-97
Check default transition placement in Stateflow charts ..	27-99
Check return value assignments of graphical functions in	
Stateflow charts .....	27-100
Check entry formatting of states in Stateflow charts .....	27-101
Check usage of return values from a graphical function in	
Stateflow charts .....	27-102
Check for pointers in Stateflow charts .....	27-103
Check for event broadcasts in Stateflow charts .....	27-104
Check for MATLAB expressions in Stateflow charts .....	27-105
Check for blocks that do not using one-based indexing ...	27-106
Check for invalid file names .....	27-108
Check for invalid model directory names .....	27-110
Check for blocks that are not discrete .....	27-111
Check for prohibited sink blocks .....	27-112
Check for invalid port positioning and configuration .....	27-113
Check for mismatches between names of ports and	
corresponding signals .....	27-115
Check whether block names do not appear below blocks ..	27-116
Check for systems that mix primitive blocks and	
subsystems .....	27-117
Check whether model has unconnected block input ports,	
output ports, or signal lines .....	27-119
Check for improperly positioned Trigger and Enable	
blocks .....	27-120
Check whether annotations have drop shadows .....	27-121
Check whether tunable parameters specify expressions,	
data type conversions, or indexing operations .....	27-122
Check whether Stateflow events are defined at the chart	
level or below .....	27-124
Check whether Stateflow data objects with local scope are	
defined at the chart level or below .....	27-125
Check interface signals and parameters .....	27-126
Check for exclusive states, default states, and substate	
validity .....	27-127

Check optimization parameters for Boolean data types ...	27-129
Check model diagnostic settings .....	27-130
Check the display attributes of block names .....	27-133
Check icon display attributes for port blocks .....	27-134
Check whether subsystem block names include invalid characters .....	27-135
Check whether Inport and Outport block names include invalid characters .....	27-137
Check whether signal line names include invalid characters .....	27-139
Check whether block names include invalid characters ...	27-141
Check Trigger and Enable block port names .....	27-143
Check for Simulink diagrams that have nonstandard appearance attributes .....	27-144
Check visibility of port block names .....	27-147
Check for direction of subsystem blocks .....	27-149
Check for proper position of constants used in Relational Operator blocks .....	27-150
Check for use of tunable parameters in Stateflow .....	27-151
Check for proper use of Switch blocks .....	27-152
Check for proper use of signal buses and Mux block usage .....	27-153
Check for bitwise operations in Stateflow charts .....	27-155
Check for comparison operations in Stateflow charts .....	27-157
Check for unary minus operations on unsigned integers in Stateflow charts .....	27-158
Check for equality operations between floating-point expressions in Stateflow charts .....	27-159
Check for mismatches between Stateflow ports and associated signal names .....	27-161
Check for proper scope of From and Goto blocks .....	27-162
<b>Requirements Consistency Checks .....</b>	<b>27-163</b>
Identify requirement links with missing documents .....	27-164
Identify requirement links that specify invalid locations within documents .....	27-165
Identify selection-based links having descriptions that do not match their requirements document text .....	27-166
Identify requirement links with inconsistent path types and preferences .....	27-168

**A**

---

<b>Requirements Management Interface</b> .....	<b>A-2</b>
<b>Requirements Management Interface (DOORS Version)</b> .....	<b>A-3</b>
<b>Verification Manager</b> .....	<b>A-3</b>
<b>Model Coverage</b> .....	<b>A-4</b>
<b>Model Advisor Check</b> .....	<b>A-5</b>
<b>Model Advisor Organization</b> .....	<b>A-5</b>

**Index**

---

# Getting Started

---

The Simulink® Verification and Validation™ software uses component tools that contribute to the work of certifying the correct design, implementation, and testing of Simulink® models. Use the following topics to become familiar with the Simulink Verification and Validation software.

- “Product Overview” on page 1-2
- “System Requirements” on page 1-3

## Product Overview

The Simulink Verification and Validation software is a Simulink product that helps you do the following:

- Associate design requirements that you manage using external applications with Simulink model objects that implement the requirements
- Verify proper function of the model by monitoring model signals during extensive testing
- Validate the model, making sure that all possible model decisions are taken through testing.
- Customize the Model Advisor to analyze a model for settings that result in inaccuracies or inefficiencies.

In short, the elements of the Simulink Verification and Validation software give you confidence in the behavior of your Simulink models.



# System Requirements

In this section...
“Operating System Requirements” on page 1-3
“Product Requirements” on page 1-3

## Operating System Requirements

The Simulink Verification and Validation software works with the following operating systems:

- Microsoft® Windows® XP and Windows Vista™
- UNIX® systems (Requirements linking to HTML and TXT documents only)

## Product Requirements

The Simulink Verification and Validation software requires the following MathWorks™ products:

- MATLAB®
- Simulink

If you want to use the Requirements Management Interface with Stateflow® charts, the Simulink Verification and Validation software requires the following MathWorks product:

- Stateflow

The Requirements Management Interface in the Simulink Verification and Validation software allows you to associate requirements with Simulink models and Stateflow charts. The software supports the following applications for documenting requirements:

- Microsoft Word 2000 or later
- Microsoft® Excel® 98 or later
- IBM® Rational® DOORS® 6.0 or later

- Adobe® PDF

# Requirements Linking and Traceability

---

- Chapter 2, “About the Requirements Management Interface”
- Chapter 3, “QuickStart Tutorials: Linking to Requirements”
- Chapter 4, “Creating and Managing Requirements Links”
- Chapter 5, “Reviewing Requirements Information in a Model”
- Chapter 6, “Keeping Requirements Information Up to Date”
- Chapter 7, “Synchronizing a Simulink Model with a DOORS Surrogate Module”
- Chapter 8, “Adding Navigation Controls to IBM Rational DOORS Requirements”
- Chapter 9, “Adding Navigation Controls to Microsoft Office Documents”
- Chapter 10, “Creating Custom Types of Requirements Documents”
- Chapter 11, “Including Requirements Information with Generated Code”



# About the Requirements Management Interface

---

- “Creating Links to Requirements with the Requirements Management Interface” on page 2-2
- “What Is a Requirement Link?” on page 2-3
- “Supported Requirements Documents Types and Links” on page 2-4
- “Supported Model Objects for Requirements Linking” on page 2-7

# Creating Links to Requirements with the Requirements Management Interface

Using the Requirements Management Interface (RMI), you can link Simulink and Stateflow objects to locations in requirements documents.

Use the RMI to:

- Associate Simulink and Stateflow objects with requirements.
- Navigate from a Simulink or Stateflow object to requirements.
- Navigate from an embedded link in a requirements document to the corresponding Simulink or Stateflow object.
- Review requirements links in your model using highlighting and tags that you define.
- Create reports for your Simulink model that show which objects link to which requirements.

## What Is a Requirement Link?

Requirements links are inserted into Simulink models that allow you to navigate from the model to a location inside a requirements document.

Requirements links have the following attributes:

- A description of up to 255 characters.
- A requirements document path name, such as a Microsoft Word file or a module in an IBM Rational DOORS database. (The RMI supports several built-in document formats; you can also register custom types of requirements documents. See “Supported Requirements Documents Types and Links” on page 2-4.)
- A designated location inside the requirements document, such as bookmark, anchor, ID, line number, cell range, or link target.
- Tags that you can define.

## Supported Requirements Documents Types and Links

Depending on the requirements document type, you can link to specific locations in a document using the following options in the Requirements dialog box.

<b>Requirements Document Type</b>	<b>Location Options</b>
Microsoft Word document	<ul style="list-style-type: none"> <li>• <b>Search text</b> — In the <b>Location</b> field, type a string. The RMI links to the first occurrence of that string in the document. This search is not case sensitive.</li> <li>• <b>Named item</b> — In the <b>Location</b> field, type the bookmark name, or from the document index, select an item. The RMI links to the location of that bookmark in the document.</li> <li>• <b>Page/item number</b> — In the <b>Location</b> field, type a page number . The RMI links to the top of that page.</li> </ul>
Microsoft Excel workbook	<ul style="list-style-type: none"> <li>• <b>Search text</b> — In the <b>Location</b> field, type a string. The RMI links to the first occurrence of that string in the workbook. This search is not case sensitive.</li> <li>• <b>Named item</b> — In the <b>Location</b> field, type the name. The RMI links to that named item in the workbook.</li> <li>• <b>Sheet range</b> — In the <b>Location</b> field, type a location in a workbook :               <ul style="list-style-type: none"> <li>▪ Cell number (A1, C13)</li> <li>▪ Range of cells (C5:D7)</li> <li>▪ Range of cells on another worksheet (Sheet1!A1:B4)</li> </ul>               The RMI links to that cell or cells.             </li> </ul>
IBM Rational DOORS database	<p><b>Page/item number</b> — In the <b>Location</b> field, type the unique numeric ID of the target DOORS object. The RMI links to that object.</p>



<b>Requirements Document Type</b>	<b>Location Options</b>
MuPAD® notebook	Named item — In the <b>Location</b> field, type the name of a link target in a MuPAD notebook.
Simulink DocBlock block (RTF format only)	<p>Create links to the RTF file associated with the DocBlock block as you would to a Microsoft Word file:</p> <ul style="list-style-type: none"> <li>• Search text — In the <b>Location</b> field, type a string. The RMI links to the first occurrence of that string in the document. This search is not case sensitive.</li> <li>• Named item — In the <b>Location</b> field, type the bookmark name, or from the document index, select an item. The RMI links to the location of that bookmark in the document.</li> <li>• Page/item number — In the <b>Location</b> field, type a page number . The RMI links to the top of that page.</li> </ul>
Text document	<ul style="list-style-type: none"> <li>• Search text — In the <b>Location</b> field, type a string. The RMI links to the first occurrence of that string within the document. This search is not case sensitive.</li> <li>• Line number — In the <b>Location</b> field, type a line number. The RMI links to that line.</li> </ul>
HTML file	<p>You can link only to a named anchor.</p> <p>For example, in your HTML requirements document, if you define the anchor</p> <pre style="text-align: center;">&lt;A NAME=valve_timing&gt; ...contents... &lt;/A&gt;</pre> <p>in the <b>Location</b> field, enter <code>valve_timing</code> or, from the document index, choose the anchor name.</p>

<b>Requirements Document Type</b>	<b>Location Options</b>
PDF file	<ul style="list-style-type: none"><li>• <b>Named item</b> — In the <b>Location</b> field, type the bookmark name, or, from the document index, select the bookmark name. The RMI links to the location of that bookmark in the document.</li><li>• <b>Page/item number</b> — In the <b>Location</b> field, type a page number. The RMI links to the top of that page.</li></ul>
Web browser URL	<p>The RMI can link to a URL location. In the <b>Document</b> field, type the URL string. When you click the link, the document opens in a Web browser:</p> <ul style="list-style-type: none"><li>• <b>Named item</b> — In the <b>Location</b> field, enter the anchor name.</li></ul>

If you register custom types of requirements documents, the RMI supports those types of documents. For more information, see Chapter 10, “Creating Custom Types of Requirements Documents”.

## Supported Model Objects for Requirements Linking

You can create requirements links in a Simulink model to the following types of objects:

- Simulink blocks, including library-linked blocks and subsystems

---

**Note** You can add requirements to a Model block, but not to its contents.

---

- Simulink block diagrams
- Stateflow states, transitions, and boxes
- Stateflow functions, including:
  - Simulink functions
  - Graphical functions
  - Truth Table functions
  - Embedded MATLAB<sup>®</sup> functions



# QuickStart Tutorials: Linking to Requirements

---

- “Before You Start: Configuring the RMI for One-Way Linking” on page 3-2
- “Linking to Requirements with Selection-Based Linking” on page 3-4
- “Tutorial: Linking to Requirements in Microsoft Word Documents” on page 3-7
- “Tutorial: Linking to Requirements in IBM Rational DOORS Databases” on page 3-11
- “Creating Requirements Reports” on page 3-12

## Before You Start: Configuring the RMI for One-Way Linking

*One-way linking* creates links to requirements in a Simulink model so that you can navigate from the model object to its linked requirement. One-way linking does *not* modify the requirements document to create links from the requirements document to the model. Use one-way linking when you are not authorized to modify the requirements document.

---

**Note** For information about when to use two-way linking, see Chapter 8, “Adding Navigation Controls to IBM Rational DOORS Requirements” and Chapter 9, “Adding Navigation Controls to Microsoft Office Documents”.

---

In the two QuickStart tutorials, before creating a link from a model object to a requirements document, make sure to configure the RMI for one-way linking:

- 1** Open a Simulink model.
- 2** Select **Tools > Requirements > Settings**.
- 3** Click the **Selection Linking** tab.
- 4** If the **Modify documents to include links to models (two-way linking)** option is selected, clear that check box.
- 5** Click the **Filters** tab.
- 6** If the **Filter links by user tags when highlighting and reporting requirements** option is selected, clear that check box.

For this example, you display and report on all requirements in the model. For more information on filtering requirements with user tags, see “Filtering Requirements” on page 5-21.

- 7** Click **Close** to close the Requirements Settings dialog box.

---

**Note** In the Requirements Settings dialog box, the options that you either select or clear go into effect immediately. These settings now apply to whatever model you are working on.

---

## Linking to Requirements with Selection-Based Linking

### What Is Selection-Based Linking?

These tutorials use *selection-based linking* to create links from a model object to a currently selected section or object in the requirements document. Selection-based linking is the easiest way to create requirements links from a model to an external document when you know exactly what text you want to link to.

If you need to search the document or link to a specific bookmark or other designated location in the requirements document, follow the instructions in Chapter 4, “Creating and Managing Requirements Links”.

### Creating a Link Using Selection-Based Linking

Using any model and a requirements document of your own, create a link from the model to the document using selection-based linking:

- 1 In a requirements document, select text or objects to link to.
- 2 Right-click the model object and select **Requirements > Add link to ....**

There are three options that correspond to the three types of requirements documents for which selection-based linking is available:

- Microsoft Word
- Microsoft Excel
- IBM Rational DOORS

### Customizing Selection-Based Linking

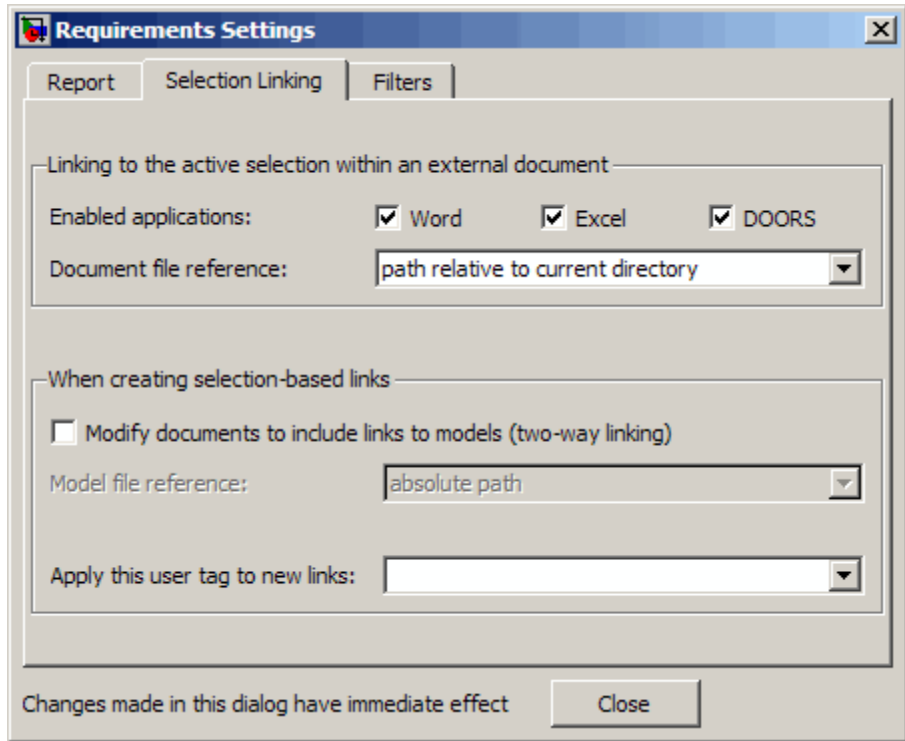
To configure selection-based linking capabilities, in the Model Editor:

- 1 Select **Tools > Requirements > Settings**.

The Requirements Settings dialog box opens.

- 2 Click the **Selection Linking** tab to display the settings for selection-based linking.





- 3 For this tutorial, accept the default settings for selection-based linking, or configure the following settings as desired.

Requirements Settings Selection Linking Option	Description
Enabled applications	Specifies which applications can have selected elements to create links to: <ul style="list-style-type: none"> <li>• Microsoft Word</li> <li>• Microsoft Excel</li> <li>• IBM Rational DOORS</li> </ul>
Document file reference	Specifies how to store the file path to the requirements document:

<b>Requirements Settings Selection Linking Option</b>	<b>Description</b>
	<ul style="list-style-type: none"> <li>• absolute path</li> <li>• path relative to current directory</li> <li>• path relative to model directory (default)</li> <li>• filename only (on MATLAB path)</li> </ul> <p>For information about how the RMI resolves the path to the requirements document, see “Resolving the Document Path” on page 6-7.</p>
<b>Modify documents to include links to models (two-way linking)</b>	<p>Enables the Requirements Management Interface (RMI) to insert navigation controls into the requirements documents, also known as <i>two-way linking</i>. One-way linking, where the links are created only in the Simulink model, not in the requirements document, is the default.</p>
<b>Apply this user tag to new links</b>	<p>Specifies one or more user tags, separated by commas, to be associated automatically with all new selection-based links.</p>

# Tutorial: Linking to Requirements in Microsoft Word Documents

## In this section...

“Navigating from a Model to a Requirements Document” on page 3-7

“Creating a Link from a Model Object to a Microsoft Word Requirements Document” on page 3-8

“Creating a Link from a Model to a Requirements Document” on page 3-9

“Adding Requirement Links to Multiple Model Objects Simultaneously” on page 3-9

This tutorial describes how to create links from objects in a Simulink model to requirements in a Microsoft Word document.

## Navigating from a Model to a Requirements Document

Navigate from the model to the requirements document:

- 1 Open the demo model:

```
slvndemo_fuelsys_officereq
```

- 2 Right-click a blank area of the model and select **Requirements > Design Description Microsoft Word Document**.

This link was created at the top level of the model, not on a model object. The Microsoft Word 2007 document `slvndemo_FuelSys_DesignDescription.docx` opens at the start of the document.

- 3 In the demo model, open the fuel rate controller subsystem.
- 4 Right-click the Airflow calculation block and select **Requirements > 1. “Mass airflow estimation”**.

The Microsoft Word 2007 document `slvndemo_FuelSys_DesignDescription.docx`, which ships with the

Simulink Verification and Validation software, opens, with the header **Mass airflow estimation** selected.

#### 2.1 Mass airflow estimation

**Model Element:** fuelsys/fuelrate controller/Airflow calculation

**Details:** The controller will use engine speed, throttle position and manifold pressure to estimate the mass airflow through the engine. This was very important change.

---

**Note** The Simulink icon next to the header is an ActiveX® control that allows for navigation from the requirement to the associated model object. In this example, you do not insert navigation controls into your requirements documents.

For more information about inserting navigation controls into Microsoft Office documents, see Chapter 9, “Adding Navigation Controls to Microsoft Office Documents”.

---

Keep the demo model and the requirements document open.

## Creating a Link from a Model Object to a Microsoft Word Requirements Document

Create a link from the Airflow calculation subsystem in the slvndemo\_fuelsys\_officereq model to selected text in the requirements document:

**1** In slvndemo\_FuelSys\_DesignDescription.docx, find the section **2.2 Determination of pumping efficiency**.

**2** Select the header text.

**3** Open the demo model:

```
slvndemo_fuelsys_officereq
```

**4** Open the Airflow calculation subsystem.

- 5 Right-click the Pumping Constant block and select **Requirements > Add link to Word selection**.
- 6 To confirm the one-way link, right-click the Pumping Constant block and select **Requirements > 1. “Determination of pumping efficiency”**.

The section **2.2 Determination of pumping efficiency** is displayed, selected in the requirements document.

Keep the demo model and the requirements document open.

## Creating a Link from a Model to a Requirements Document

Create a link from the `slvndemo_fuelsys_officereq` model itself, not from an object in the model:

- 1 In `slvndemo_FuelSys_DesignDescription.docx`, go to the beginning of the document and select the text **Design Description: Fault Tolerant Fuel Control System**.
- 2 In the `slvndemo_fuelsys_officereq` model, right-click a blank area of the model and select **Requirements > Add link to Word selection**.
- 3 Verify that the RMI created the link by right-clicking a blank area of the model and selecting **Requirements**.

## Adding Requirement Links to Multiple Model Objects Simultaneously

You can add links to requirements for a selection of multiple Simulink or Stateflow objects:

- 1 In `slvndemo_FuelSys_DesignDescription.docx`, select the header for one of the requirements.
- 2 In the `slvndemo_fuelsys_officereq` model, select several objects together by holding down the **Shift** key.
- 3 Right-click any one of those blocks and select **Requirements > Add Links to All**.

The Add requirements dialog box opens.

**4** Click **New**.

**5** Under **Selection-based linking**, click **Word**.

The detailed information about the selected requirement is displayed in the dialog box.

**6** Click **Apply** or **OK** to save the new requirements links.

**7** Right-click one of the model objects where you added the requirement and select **Requirements**. On the context menu, you see the header text from the requirements document.

### **Restrictions on Adding Links to Multiple Objects**

If you enable two-way linking by selecting **Modify documents to include links to models**, there are two restrictions to adding requirements links to multiple objects:

- You cannot create selection-based links.
- Any links you create are one-way only, from the model to the requirement.

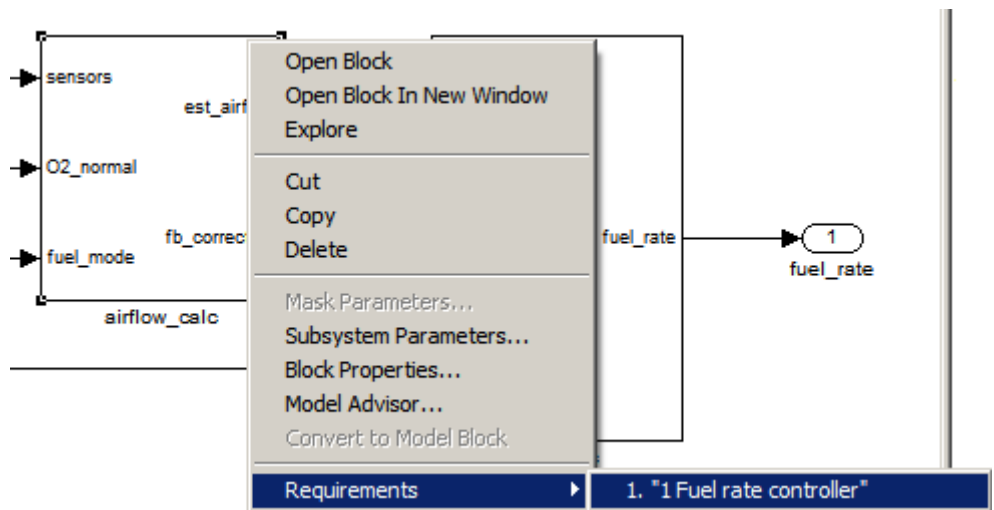
## Tutorial: Linking to Requirements in IBM Rational DOORS Databases

This example describes how to create links from objects in a Simulink model to requirements in an IBM Rational DOORS database.

- 1 Open a DOORS formal module.
- 2 Click to select one of the requirement objects.
- 3 Open the demo model:

```
sldemo_fuelsys
```

- 4 Open the fuel\_rate\_control subsystem.
- 5 Right-click the airflow\_calc subsystem and select **Requirements > Add link to current DOORS object**.
- 6 To confirm the requirement link, right-click the airflow\_calc subsystem and select **Requirements**. In the submenu, the top item is the heading text for the DOORS requirement object.



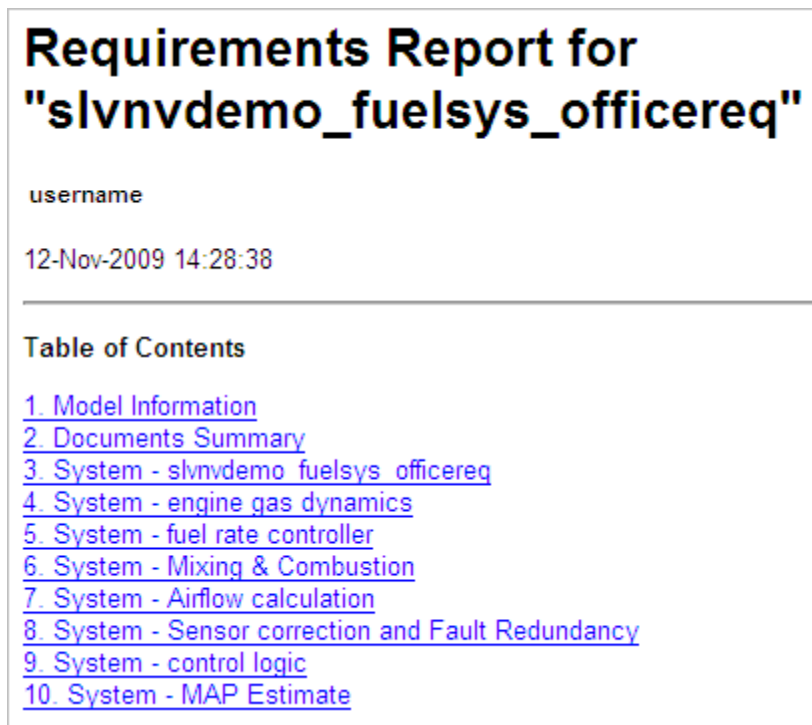
## Creating Requirements Reports

To create the default requirements report for a Simulink model:

- 1 Open a model that has requirements.
- 2 Make sure that your current working folder is writable.
- 3 In the Model Editor, select **Requirements > Generate Report**.

If your model is large and has many requirements links, it takes a few minutes to create the report.

A Web browser window opens with the contents of the report. The following graphic shows the **Table of Contents** for the `slvndemo_fuelsys_officereq` model.



The screenshot shows a web browser window displaying a requirements report. The title is "Requirements Report for 'slvndemo\_fuelsys\_officereq'". Below the title, it shows the username and the date and time: "12-Nov-2009 14:28:38". A horizontal line separates the header from the "Table of Contents" section. The table of contents lists ten items, each with a blue underlined link:

- 1. [Model Information](#)
- 2. [Documents Summary](#)
- 3. [System - slvndemo fuelsys officereq](#)
- 4. [System - engine gas dynamics](#)
- 5. [System - fuel rate controller](#)
- 6. [System - Mixing & Combustion](#)
- 7. [System - Airflow calculation](#)
- 8. [System - Sensor correction and Fault Redundancy](#)
- 9. [System - control logic](#)
- 10. [System - MAP Estimate](#)



A typical requirements report includes:

- Table of contents
- List of tables
- Per-subsystem sections that include:
  - Tables that list objects with requirements and include links to associated requirements documents
  - Graphic images of objects with requirements
  - Lists of objects with no requirements

For detailed information about requirements reports, see “Creating and Customizing a Requirements Report” on page 5-7.



# Creating and Managing Requirements Links

---

- “The Requirements Dialog Box” on page 4-2
- “Tutorial: Managing Requirements Links to Microsoft® Excel Workbooks” on page 4-3
- “Tutorial: Creating Links to MuPAD Notebooks” on page 4-8
- “Tutorial: Linking Signal Builder Blocks to Requirements” on page 4-10

# The Requirements Dialog Box

The Requirements dialog box is a centralized location where you can manage the requirements links associated with a given model object. In this dialog box, you can:

- Create new requirements links.
- Change information about existing requirements links, including adding or changing user tags to filter requirements highlighting and reporting.
- Delete existing requirements links.

You can create multiple requirements links from a single Simulink model object using the Requirements dialog box. You can also link to different requirements documents from the same model object.

Use this dialog box to modify or delete existing links. In this section, you learn how to use the Requirements dialog box to link to Microsoft Excel workbooks and MuPAD notebooks. You also learn how to modify and delete existing requirements links.

# Tutorial: Managing Requirements Links to Microsoft Excel Workbooks

## In this section...

“Before You Create Links” on page 4-3

“Navigating from a Model Object to Requirements in a Microsoft® Excel Workbook” on page 4-3

“Creating Requirements Links to the Workbook” on page 4-4

“Changing Requirements Links” on page 4-5

“Deleting Requirements Links” on page 4-6

## Before You Create Links

The tutorials in this section create one-way links from the model objects to the requirements. Before you create these links, configure the Requirements Management Interface (RMI) for one-way linking by following the steps in “Before You Start: Configuring the RMI for One-Way Linking” on page 3-2.

## Navigating from a Model Object to Requirements in a Microsoft Excel Workbook

**1** Open the demo model:

```
slvndemo_fuelsys_officereq
```

**2** Select **Tools > Requirements > Highlight model** to highlight the model objects with requirements.

**3** Right-click the Test inputs Signal Builder block and select **Requirements > 1. “Normal mode of operation”**.

The `slvndemo_FuelSys_TestScenarios.xlsx` file opens, with the associated cells highlighted.

Keep the demo model and Microsoft Excel requirements document open.

For information about creating requirements links in Signal Builder blocks, see “Tutorial: Linking Signal Builder Blocks to Requirements” on page 4-10.

### Creating Requirements Links to the Workbook

- 1 At the top level of the `slvndemo_fuelsys_officereq` model, right-click the speed sensor block and select **Requirements > Edit/Add Links**.

The Requirements dialog box opens.

- 2 To create a requirements link, click **New**.

- 3 In the **Description** field, enter:

Speed Sensor Failure

You will link the speed sensor block to the **Speed Sensor Failure** information in the Microsoft Excel requirements document.

- 4 At the **Document** field, click **Browse** to locate and open the `slvndemo_FuelSys_TestScenarios.xlsx` file.

The **Document Type** field information changes to **Microsoft Excel**.

- 5 In the workbook, the **Speed sensor failure** information is in cells B22:E22. For the **Location (Type/Identifier)** field, select **Sheet** range and in the second field, enter B22:E22. (The cell range letters are not case sensitive.)

- 6 Click **Apply** or **OK** to create the link.

- 7 To confirm that you created the link, right-click the speed sensor block and select **Requirements > 1. “Speed sensor failure”**.

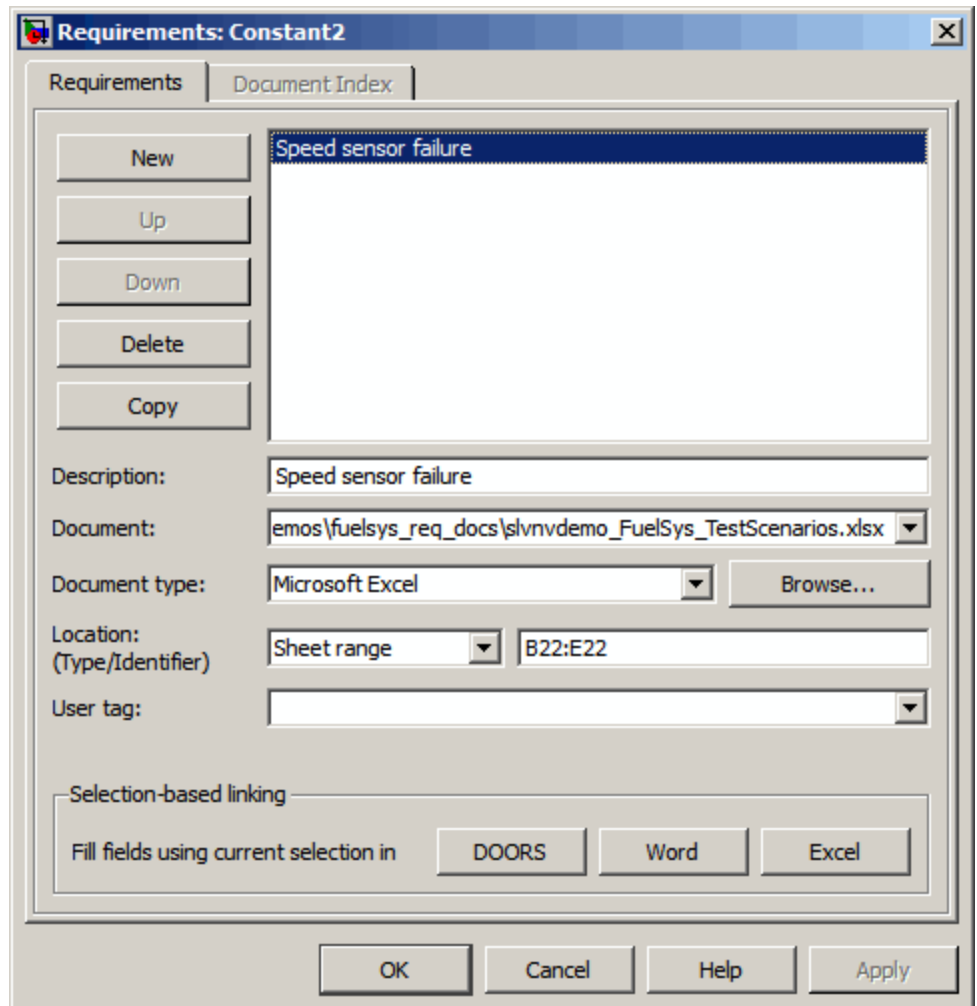
The workbook opens, with cells B22:E22 highlighted.

Keep the demo model and Microsoft Excel requirements document open.

## Changing Requirements Links

- 1 In the slvnvdemo\_fuelsys\_officereq model, right-click the speed sensor block and select **Requirements > Edit/Add Links**.

The Requirements dialog box opens displaying the information about the **Speed sensor failure** link.



**2** In the **Description** field, enter:

Speed sensor test scenario

**3** In the **User tag** field, enter **test**.

User tags allow you to filter requirements for highlighting and reporting. In this example, you include any requirements with the exact user tag **test**.

---

**Note** For more information about user tags, see “Filtering Requirements” on page 5-21.

---

**4** Click **Apply** or **OK** to save the change.

Keep the demo model open.

## Deleting Requirements Links

### Deleting One Requirement Link at a Time

- 1** In the `slvndemo_fuelsys_officereq` model, right-click the speed sensor block and then select **Requirements > Edit/Add Links**.
- 2** In the top pane of the Requirements dialog box, select **Speed sensor failure**.
- 3** Click **Delete**.
- 4** Click **OK** to save the deletion and close the Requirements dialog box.

### Deleting All Requirement Links for a Model Object

- 1** In the `slvndemo_fuelsys_officereq` model, double-click the fuel rate controller subsystem to open it.
- 2** Right-click the Airflow calculation block and select **Requirements**. This block has a link to a requirement, **Mass Airflow Estimation**.



- 3** Right-click the Airflow calculation block and select **Requirements > Delete All Links**.
- 4** In the Delete All Links dialog box, click **OK** to confirm the deletion.

---

**Note** If you select this option for a subsystem block or Stateflow chart whose child objects have requirements links, the RMI does not delete those requirements links.

---

- 5** Right-click the Airflow calculation block and select **Requirements** to confirm that you deleted the link.

# Tutorial: Creating Links to MuPAD Notebooks

This example describes how to create a link from a Simulink model to a MuPAD notebook. You use a model that simulates a nonlinear second-order system with the van der Pol equation.

Before beginning this tutorial, create a MuPAD notebook with one or more link targets. This tutorial uses a MuPAD notebook that includes information about solving the van der Pol equation symbolically and numerically.

---

**Note** You must have the Symbolic Math Toolbox™ installed to open a MuPAD notebook. For information about creating a MuPAD notebook, see “Launching, Opening, and Saving MuPAD Notebooks”.

---

- 1 Open a demo model for the van der Pol equation:

vdp

- 2 Right-click a blank area of the model and select **Requirements > Edit/Add Links**.

In this tutorial, you add the requirement to the model itself, not to a specific block in the model.

- 3 Click **New**.

Microsoft Excel

managing requirements links in

- 4 In the **Document type** drop-down list, select **MuPAD Notebook**.

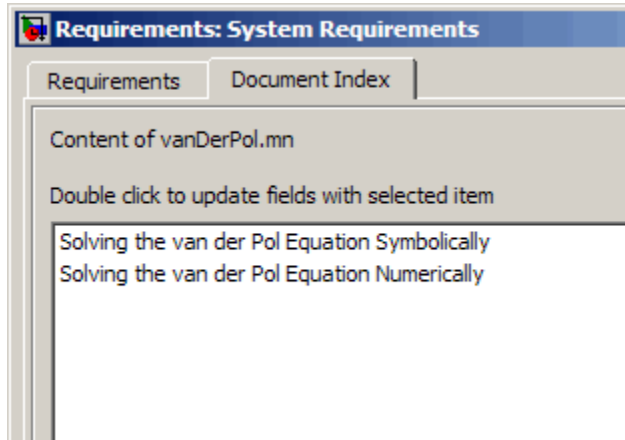
- 5 Click **Browse** to navigate to the notebook that you want to use.

Use a notebook that has link targets in it.

- 6 Make sure the MuPAD notebook is in a saved state. Any link targets created since the last save do not appear in the RMI document index.

- 7 To list the link targets, in the Requirements dialog box, click the **Document Index** tab.

This example shows two link targets.



---

**Note** These link targets are in a MuPAD notebook that was created for the purposes of this tutorial. The **Document Index** tab will display only link targets you have created in your MuPAD notebook.

---

- 8 Select a link target name and click **Apply**.

The **Requirements** tab reopens, displaying the details of the newly created link.

- 9 To confirm that you created the link, right-click a blank area of the model and select **Requirement**. The new link is at the top of the submenu.

# Tutorial: Linking Signal Builder Blocks to Requirements


You can create links from a signal group in a Signal Builder block to a requirements document:

- 1 Open the demo model:

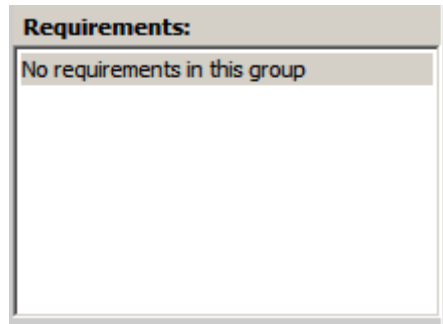
sf\_car

- 2 In the sf\_car model, double-click the User Inputs block.

The Signal Builder dialog box opens, displaying four groups of signals. The Passing Maneuver signal group is the current tab. The RMI associates any requirements links you add to the current signal group.

- 3 At the far-right end of the toolbar, click the **Show verification settings** button . (You may need to expand the Signal Builder dialog box for this button to become visible.)

A **Requirements** pane opens on the right-hand side of the Signal Builder dialog box.



- 4 Place your cursor in the window, right-click, and select **Edit/Add Links**.

The Requirements dialog box opens.

- 5 Click **New**. In the **Description** field, enter User input requirements.
- 6 Browse to a requirements document and click **Open**.

- 7** In the **Location** drop-down list, select **Search text** to link to specified text in the document.
- 8** Next to the **Location** drop-down list, enter User Input Requirements.
- 9** Click **Apply** to create the link.
- 10** To verify that the RMI created the link, in the Model Editor, select the User Inputs block, right-click, and select **Requirements**.  
  
The link to the new requirement is the top menu option.
- 11** Save the sf\_car\_linking model.

---

**Note** Links that you create in this way associate requirements information with individual signal groups, not with the entire Signal Builder block.

In this example, switch to a different tab to link a requirement to another signal group.

---



# Reviewing Requirements Information in a Model

---

- “Highlighting Requirements in a Model” on page 5-2
- “Navigating to Requirements from a Model” on page 5-5
- “Creating and Customizing a Requirements Report” on page 5-7
- “Filtering Requirements” on page 5-21

## Highlighting Requirements in a Model

You can highlight a model so that you see which objects in the model have links to external documents. You highlight a model from the Model Editor or from the Model Explorer.

### In this section...

“Highlighting a Model Using the Model Editor” on page 5-2

“Highlighting a Model Using the Model Explorer” on page 5-4

### Highlighting a Model Using the Model Editor

If you are working in the Model Editor and want to see which model objects have requirements:

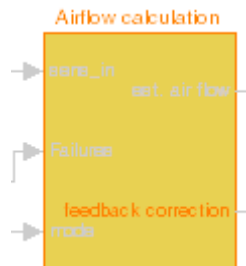
- 1 Open the demo Simulink model:

```
slvndemo_fuelsys_officereq
```

- 2 In the Model Editor, select **Tools > Requirements > Highlight model**.

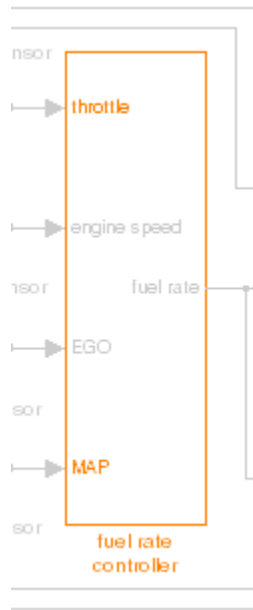
There are two types of highlighting for model objects with requirements:

- A yellow fill for objects that have requirements links for the object itself.



- An orange outline for objects, such as subsystems, whose child objects have requirements links.





Objects that do not have requirements appear dimmed.



- 3** To remove the highlighting from the model, select **Tools > Unhighlight model** or right-click anywhere in the model and select **Remove Highlighting**.

Even if you highlight a model, you can still manage the model and its contents as you do normally.

---

**Note** If your model contains a Model block whose referenced model contains requirements, those requirements are not highlighted. This information is only available in requirements reports. To generate requirements information for referenced models and see highlighted snapshots of those requirements, follow the steps in “Reporting on Requirements in Model Blocks” on page 5-14.

---


### Highlighting a Model Using the Model Explorer

If you are working in the Model Explorer and want to see which model objects have requirements:

- 1 Open the demo Simulink model:

```
slvndemo_fuelsys_officereq
```

- 2 Select **View > Model Explorer**.

- 3 To highlight the model objects with requirements, click the **Highlight items with requirements on model** icon ().

The Model Editor window opens, and all objects and all child objects with requirements are highlighted.

---

**Note** If your model contains a Model block whose referenced model contains requirements, those requirements are not highlighted. This information is only available in requirements reports. To generate requirements information for referenced models and see highlighted snapshots of those requirements, follow the steps in “Reporting on Requirements in Model Blocks” on page 5-14.

---

## Navigating to Requirements from a Model

### In this section...

“Navigating from the Model Object” on page 5-5

“Navigating from a System Requirements Block” on page 5-5

### Navigating from the Model Object

Once you have highlighted a model, you can navigate directly from a highlighted model object to its associated requirement. The external document opens in the appropriate application with the requirements text highlighted.

- 1 Open the demo Simulink model:

```
slvndemo_fuelsys_officereq
```

- 2 Open the fuel rate controller subsystem.

- 3 To open the linked requirement, right-click the Airflow calculation subsystem and select **Requirements > 1. “Mass airflow estimation”**.

The Microsoft Word document, `slvndemo_FuelSys_DesignDescription.docx`, opens, with the section **2.1 Mass airflow estimation** selected.

- 4 Close the requirements document, but leave the demo model open.

### Navigating from a System Requirements Block

The System Requirements block provides a way to gather all the requirements links for objects at a given level in the model hierarchy into one block. The System Requirements block lists the requirements links in the object where you add the block. The System Requirements block does not list requirements associated with child objects.

You can use the links in the System Requirements block to navigate to requirements:

- 1 In the `slvndemo_fuelsys_officereq` model, open the fuel rate controller subsystem.

**2** Open the Airflow calculation subsystem.

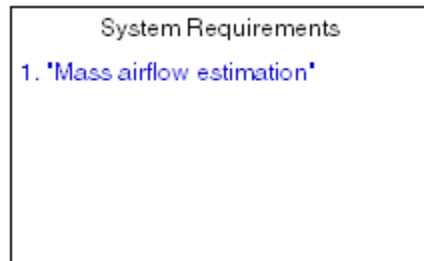
**3** Select **View > Library Browser**

**4** On the **Libraries** pane, click the **Simulink Verification and Validation** library.

This library contains only one block—the System Requirements block.

**5** Drag a System Requirements block into the Airflow calculation subsystem.

After a few seconds, the block includes the **1. “Mass airflow subsystem”** link.



The **Mass airflow estimation** requirement is associated with the Airflow calculation subsystem. The System Requirements block does not list the requirements associated with the Relational Operator and feedback correction blocks. The System Requirements block lists only those requirements associated with the level at which you insert the System Requirements block.

**6** Double-click **1. “Mass airflow subsystem”**.

The Microsoft Word document, `slvnvdemo_FuelSys_DesignDescription.docx`, opens, with the section **2.1 Mass airflow estimation** selected.

**7** Close the requirements document, but leave the demo model open.

# Creating and Customizing a Requirements Report

**In this section...**

“Creating a Default Requirements Report” on page 5-7

“Reporting on Requirements in Model Blocks” on page 5-14

“Customizing the Requirements Report” on page 5-16

## Creating a Default Requirements Report

You can generate a default report with information about all the requirements associated with the model and its objects.

---

**Note** If the model for which you are creating a report contains Model blocks, see “Reporting on Requirements in Model Blocks” on page 5-14.

---

Before you generate the report, add a requirement to a Stateflow chart to see information the requirements report contains about Stateflow charts:

**1** Open the demo model:

```
slvndemo_fuelsys_officereq
```

**2** Open the fuel rate controller subsystem.

**3** Open the Microsoft Word requirements document:

```
matlabroot/toolbox/slvnv/rmidemos/fuelsys_req_docs/slvndemo_FuelSys_RequirementsSpecification.docx
```

**4** Create a link from the control logic Stateflow chart to a location in this document.

**5** Close the requirements document, but keep the demo model open.

To generate the default requirements report for the `slvndemo_fuelsys_officereq` model:

**1** Select **Tools > Requirements > Generate Report**.

The Requirements Management Interface (RMI) searches through all the blocks and subsystems in the model for associated requirements. The RMI generates and displays a complete report in HTML format with the default name `requirements.html`. The report contains the following content:

- “Table of Contents” on page 5-8
- “List of Tables” on page 5-9
- “Model Information” on page 5-10
- “Documents Summary” on page 5-10
- “System” on page 5-11

### **Table of Contents**

The **Table of Contents** lists the major sections of the report. There is one **System** section for the top-level model and one **System** section for each subsystem, Model block, or Stateflow chart.

Click a link to view information about a specific section of the model.

# Requirements Report for "slvndemo\_fuelsys\_officereq"

username

12-Nov-2009 14:28:38

---

## Table of Contents

- [1. Model Information](#)
- [2. Documents Summary](#)
- [3. System - slvndemo\\_fuelsys\\_officereq](#)
- [4. System - engine gas dynamics](#)
- [5. System - fuel rate controller](#)
- [6. System - Mixing & Combustion](#)
- [7. System - Airflow calculation](#)
- [8. System - Sensor correction and Fault Redundancy](#)
- [9. System - control logic](#)
- [10. System - MAP Estimate](#)

## List of Tables

The **List of Tables** includes links to navigate to each table in the report.

### List of Tables

- 1.1. [slvndemo\\_fuelsys\\_officereq Version Information](#)
- 2.1. [Requirements documents linked in model](#)
- 3.1. [slvndemo\\_fuelsys\\_officereq Requirements](#)
- 3.2. [Blocks in "slvndemo\\_fuelsys\\_officereq" that have requirements](#)
- 3.3. [Test inputs : Normal operation signal requirements](#)
- 4.1. [Blocks in "engine gas dynamics" that have requirements](#)
- 5.1. [Blocks in "fuel rate controller" that have requirements](#)
- 6.1. [Blocks in "Mixing & Combustion" that have requirements](#)
- 7.1. [slvndemo\\_fuelsys\\_officereq/fuel rate controller/Airflow calculation Requirements](#)
- 7.2. [Blocks in "Airflow calculation" that have requirements](#)
- 8.1. [Blocks in "Sensor correction and Fault Redundancy" that have requirements](#)
- 9.1. [slvndemo\\_fuelsys\\_officereq/fuel rate controller/control logic Requirements](#)
- 10.1. [slvndemo\\_fuelsys\\_officereq/fuel rate controller/Sensor correction and Fault Redundancy/MAP Estimate Requirements](#)

### Model Information

The **Model Information** contains general information about the model, such as when the model was created and when the model was last modified.

### Chapter 1. Model Information

Table 1.1. slvndemo\_fuelsys\_officereq Version Information

<i>ModelVersion</i>	1.157	<i>ConfigurationManager</i>	none
<i>Created</i>	Tue Jun 02 16:11:43 1998	<i>Creator</i>	The MathWorks Inc.
<i>LastModifiedDate</i>	Thu Oct 22 11:35:35 2009	<i>LastModifiedBy</i>	username

### Documents Summary

The **Documents Summary** section lists all the requirements documents to which objects in the slvndemo\_fuelsys\_officereq model link.



## Chapter 2. Documents Summary

Table 2.1. Requirements documents linked in model

ID	Document paths stored in the model	Last modified	# links
DOC1	<a href="#">fuelsys_req_docs\slvmdemo FuelSys DesignDescription.docx</a>	03-Nov-2009 12:26:52	8
DOC2	<a href="#">fuelsys_req_docs\slvmdemo FuelSys RequirementsSpecification.docx</a>	03-Nov-2009 12:26:52	7
DOC3	<a href="#">fuelsys_req_docs\slvmdemo FuelSys TestScenarios.xlsx</a>	03-Nov-2009 12:26:52	2

To view a requirements document in its native application, in the Document paths stored in the model column, click the link to the requirements document.

The document IDs—in this example, **DOC1**, **DOC2**, and **DOC3**—are short names for the requirements document. In other parts of the report, you can click the short name links to open the document associated with a document ID, as listed in the previous table.

The right-most column lists the total number of links to each document.

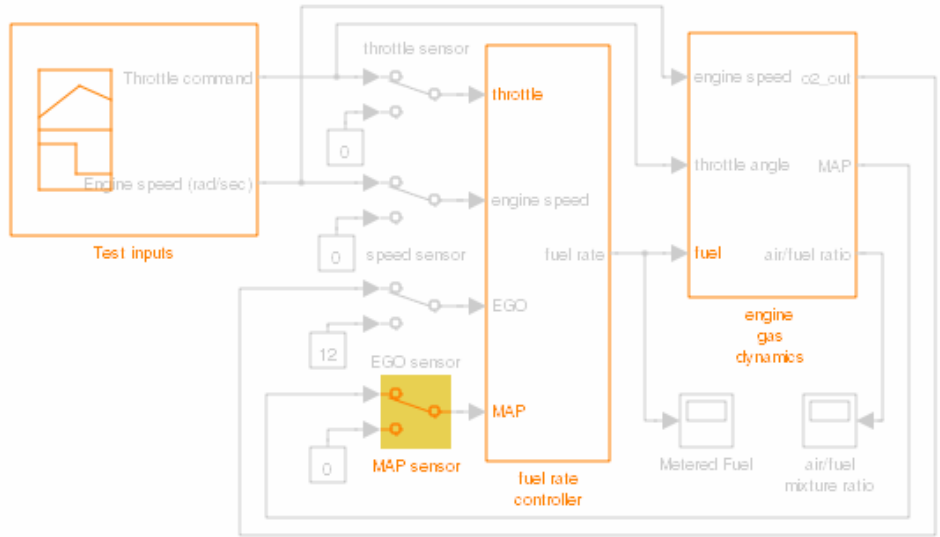
### System

Each **System** section includes:

- An image of the model or model object. The objects with requirements are highlighted.

Chapter 3. System - slvndemo\_fuelsys\_officereq

Fault-Tolerant Fuel Control System



Copyright 1997-2009 The MathWorks, Inc.

- A list of requirements associated with the model or model object. In this example, click **DOC1** to open the requirements document associated with the slvndemo\_fuelsys\_officereq model.

Table 3.1. slvndemo\_fuelsys\_officereq Requirements

Link#	Description	Target (document name and location ID)
1	Label: Design Description Microsoft Word Document	<a href="#">DOC1</a>

- A list of blocks in the top-level model that have requirements. In this example, only the MAP sensor block has a requirement at the top

level. Click **DOC3, at “Simulink requirement item 2”** to open the requirements document associated with the MAP sensor block.

**Table 3.2. Blocks in "slvndemo\_fuelsys\_officereq" that have requirements**

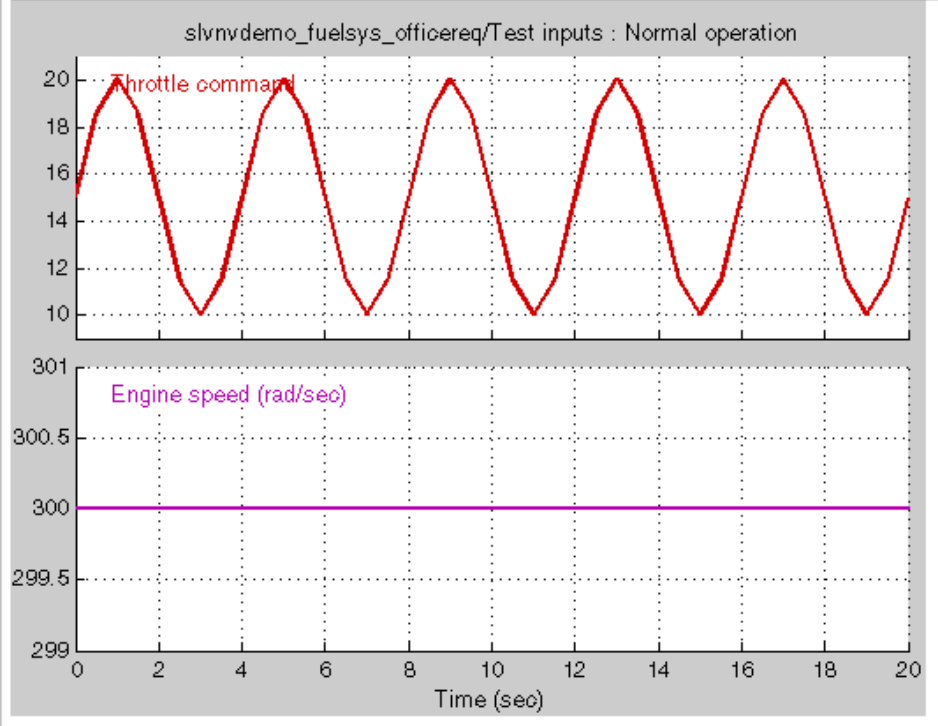
Name	Requirements
MAP sensor	<p>Link#1 label: "The System detects the Manifold Absolute Pressure sensor short to ground or open circuit by monitoring when the signal is below a calibratable threshold. The failure is detected within 100mSec of the occurrence and the MAP sensor reading is reverted to an estimated throttle position based on engine speed and throttle position."</p> <p>Target: <a href="#">DOC3, at 'Simulink requirement item 2'</a></p>

The preceding table does not include these blocks in the top-level model because:

- The fuel rate controller and engine gas dynamics subsystems are in dedicated chapters of the report.
- The report lists Signal Builder blocks separately, in this example, in Table 3.3.
- A list of requirements associated with each signal group in any Signal Builder block, and a graphic of that signal group. In this example, the Test inputs Signal Builder block in the top-level model has one signal group that has a requirement link. Click **DOC3, at “Simulink requirement item 3”** to open the requirements document associated with this signal group in the Test inputs block.

Table 3.3. Test inputs : Normal operation signal requirements

Link#	Description	Target (document name and location ID)
1	Label: Normal mode of operation	<a href="#">DOC3 at 'Simulink requirement item 3'</a>



### Reporting on Requirements in Model Blocks

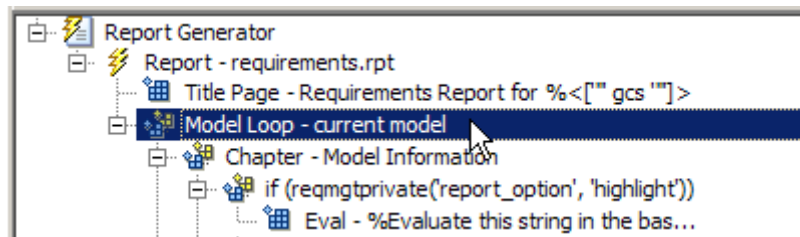
If your model contains Model blocks that reference external models, the default report does not include information about requirements in the referenced models. To generate a report that includes requirements information for referenced models, you must have a license for the Simulink® Report Generator™ software. The report includes the same information and graphics for referenced models as it does for the top-level model.

If you have a Simulink Report Generator license, take the following steps before generating a requirements report:

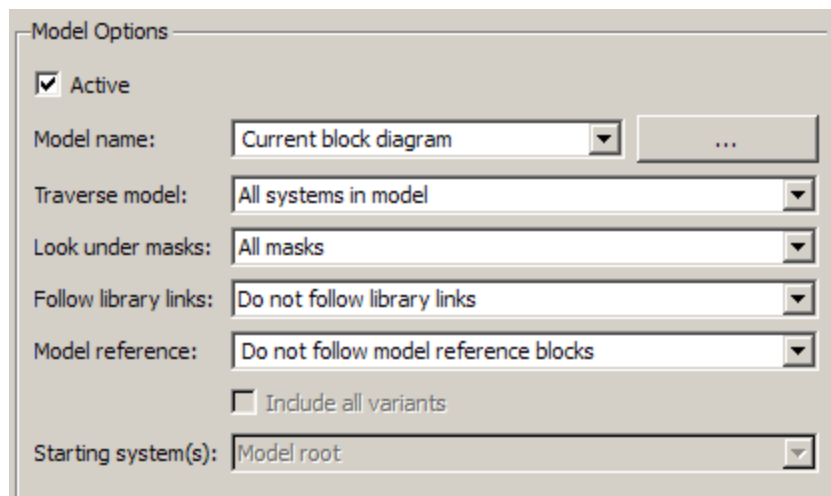
- 1 Open the model for which you want to create a requirements report.
- 2 To open the template for the default requirements report, at the MATLAB command prompt, enter

```
setedit requirements
```


- 3 In the Simulink Report Generator software window, in the far-left pane, click the **Model Loop** component.



- 4 On the far-right pane, locate the **Model reference** field. If you cannot see the drop-down arrow for that field, expand the pane.



- 5 In the **Model reference** field drop-down list, select **Follow all model reference blocks**.

- 6 To generate a requirements report for the open model that includes information about referenced models, click the Report icon .

### Customizing the Requirements Report

The Requirements Management Interface (RMI) uses the Simulink Report Generator software to generate the requirements report. You can customize the requirements report using the RMI or using the Simulink Report Generator software:

- “Customizing a Requirements Report Using the RMI Settings” on page 5-16
- “Customizing the Report Using the Simulink® Report Generator Software” on page 5-18

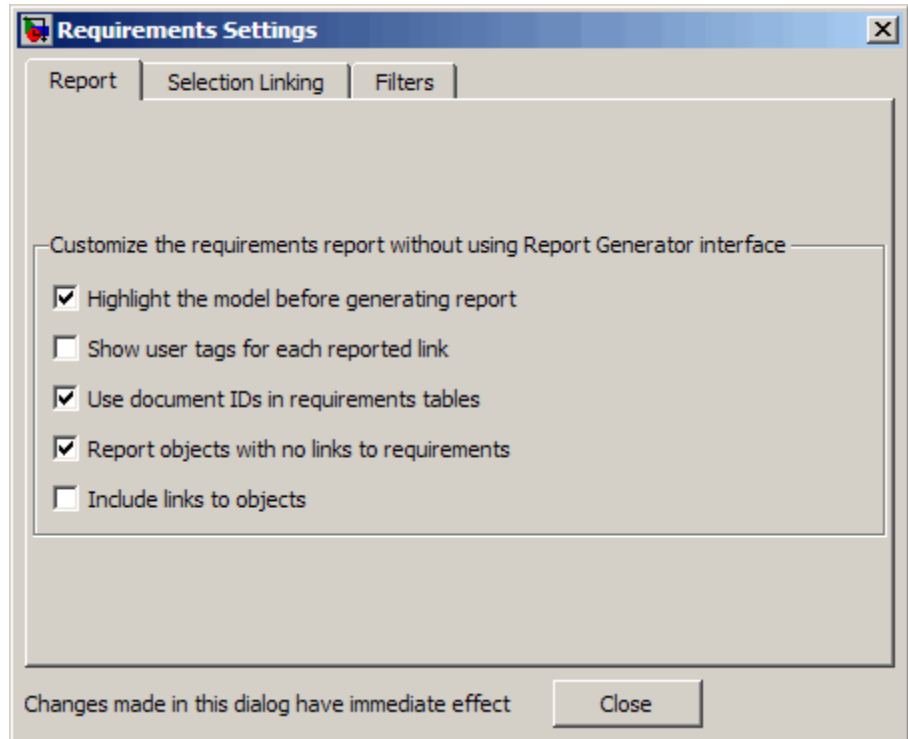
### Customizing a Requirements Report Using the RMI Settings

To customize the requirements report using the RMI settings:

- 1 In the Model Editor, select **Tools > Requirements > Settings**.

The Requirements Settings dialog box opens.

- 2 Click the **Report** tab.



In the Requirements Settings dialog box, you select options that specify the contents that you want in the report.

<b>Requirements Settings Report Option</b>	<b>Description</b>
<b>Highlight the model before generating report</b>	Enables highlighting of Simulink objects with requirements in the report graphics.
<b>Show user tags for each reported link</b>	Lists the user tags, if any, for each reported link.

<b>Requirements Settings Report Option</b>	<b>Description</b>
<b>Use document IDs in requirements tables</b>	Uses a document ID, if available, instead of a path name in the tables of the requirements report. This capability prevents long path names to requirements documents from cluttering the report tables.
<b>Report objects with no links to requirements</b>	Includes hypertext links from the report to model objects that have no requirements.
<b>Include links to objects</b>	Includes hypertext links from the report to model objects that have requirements.

**3** For this example, select the following options:

- **Highlight the model before generating the report** — Before generating the report, the RMI highlights all objects with requirements in the Model Editor. In addition, the graphics of the model in the report are highlighted.
- **Show user tags for each reported link** — The report lists the user tags (if any) associated with each requirement.

**4** To close the selected options and close the Requirements Settings dialog box, click **Close**.

**5** Generate a new requirements report by selecting **Tools > Requirements > Generate Report**.

### **Customizing the Report Using the Simulink Report Generator Software**

If you have a license for the Simulink Report Generator software, you can further modify the default requirements report.

At the MATLAB command prompt, enter the following command:

```
setedit requirements
```



The Report Explorer GUI opens the requirements report template that the RMI uses when generating a requirements report. The report template contains Simulink Report Generator components that define the structure of the requirements report.

If you click a component in the middle pane, the options you can specify for that component appear in the right-hand pane. For detailed information about using a particular component to customize your report, click **Help** at the bottom of the right-hand pane.

In addition to the standard report components, Simulink Report Generator provides components specific to the RMI in the Requirements Management Interface category.

<b>Simulink Report Generator Component</b>	<b>Report Information</b>
Missing Requirements Block Loop	Applies all child components to blocks that have no requirements
Missing Requirements System Loop	Applies all child components to systems that have no requirements
Requirements Block Loop	Applies all child components to blocks that have requirements
Requirements Documents Table	Inserts a table that lists requirements documents
Requirements Signal Loop	Applies all child components to signal groups with requirements
Requirements Summary Table	Inserts a property table that lists requirements information for blocks with associated requirements
Requirements System Loop	Applies all child components to systems with requirements
Requirements Table	Inserts a table that lists system and subsystem requirements

To customize the requirements report, you can:

- Add or delete components.
- Move components up or down in the report hierarchy.
- Customize components to specify how the report presents certain information.

For more information, see *Simulink Report Generator User's Guide*.

## Filtering Requirements

### In this section...

“Filtering Requirements with User Tags” on page 5-21

“Applying a User Tag to a Requirement” on page 5-21

“Filtering, Highlighting, and Reporting with User Tags” on page 5-23

“Applying User Tags During Selection-Based Linking” on page 5-25

“Configuring Requirements Filtering” on page 5-27

### Filtering Requirements with User Tags

*User tags* are user-defined keywords that you associate with specific requirements. With user tags, you can highlight a model or generate a requirements report for a model:

- Highlight or report only those requirements that have a specific user tag
- Highlight or report only those requirements that have one of several user tags
- Do not highlight and report requirements that have a specific user tag

### Applying a User Tag to a Requirement

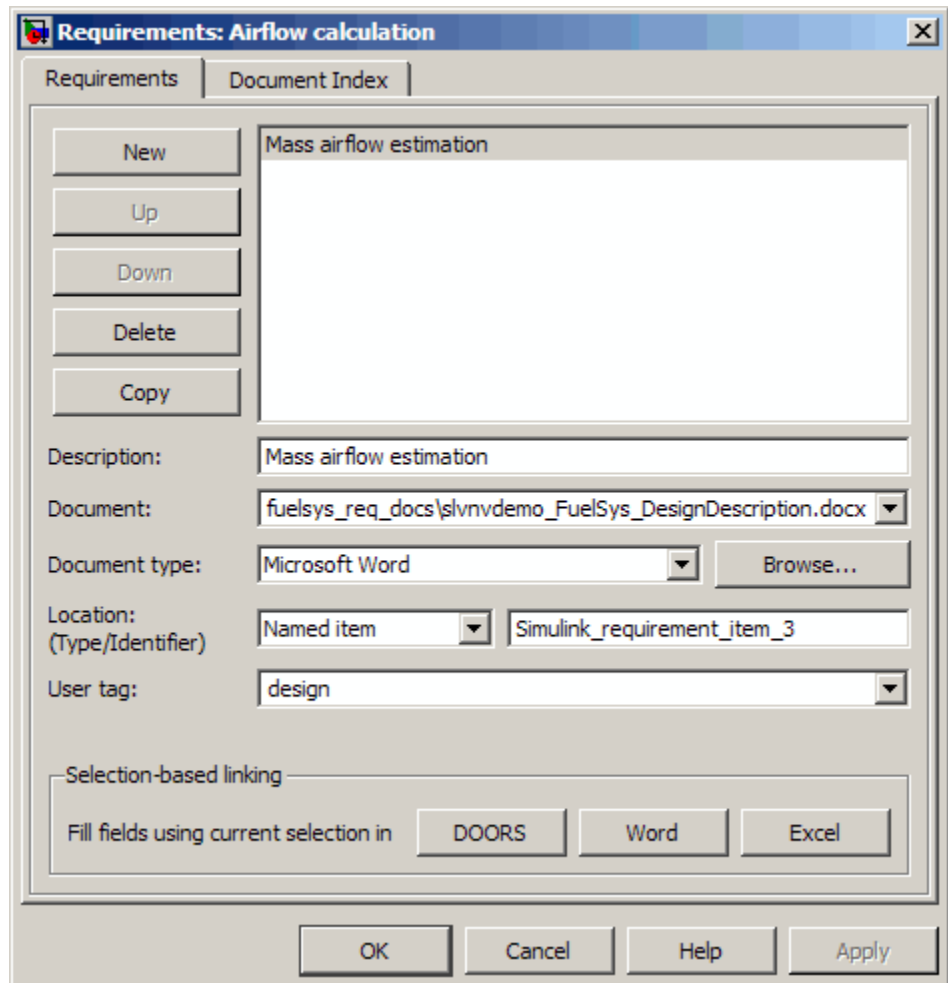
Apply one or more user tags to a newly created requirement:

- 1 Open the demo model:

```
slvndemo_fuelssystem_officereq
```

- 2 Open the fuel rate controller subsystem.
- 3 To open the requirements document, right-click the Airflow calculation subsystem and select **Requirements > Edit/Add links**.

The Requirements dialog box opens, with the details about the requirement that you created.



- 4** In the **User tag** field, enter one or more keywords, separated by commas, that the RMI can use to filter requirements. In this example, after `design`, enter a comma, followed by the user tag `test` to specify a second user tag for this requirement.

User tags:

- Are not case sensitive.

- Can consist of multiple words. If, for example, you enter design requirement, the entire phrase constitutes the user tag. Separate user tags with commas.

**5** Click **Apply** or **OK** to save the changes.

## **Filtering, Highlighting, and Reporting with User Tags**

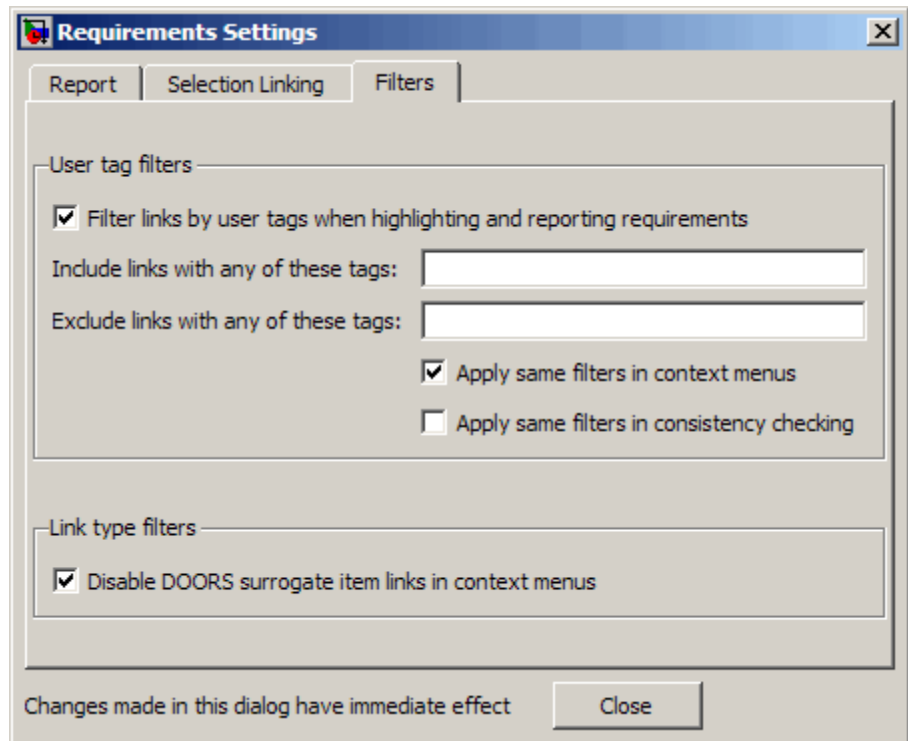
The slvndemo\_fuelsys\_officereq model includes several requirements with the user tag design. This section describes how to highlight only those model objects that have the user tag, test:

**1** In the Model Editor, remove any highlighting from the slvndemo\_fuelsys\_officereq model by selecting **Tools > Requirements > Unhighlight model**.

**2** Select **Tools > Requirements > Settings**.

The Requirements Settings dialog box opens.

**3** Click the **Filters** tab.



**4** To enable filtering with user tags, click the **Filter links by user tags when highlighting and reporting requirements** option.

**5** To include only those requirements that have the user tag, **test**, enter **test** in the **Include links with any of these tags** field.

**6** Click **Close**.

**7** In the Model Editor, select **Tools > Requirements > Highlight model**.

The RMI highlights only those model objects whose requirements have the user tag **test**, for example, the MAP sensor.

**8** Reopen the Requirements Settings dialog box to the **Filters** tab.

**9** In the **Include links with any of these tags** field, delete **test**. In the **Exclude links with any of these tags** field, add **test**.

In the model, the highlighting changes to exclude objects whose requirements have the `test` user tag; the MAP sensor and Test inputs blocks are no longer highlighted.

- 10** In the Model Editor, select **Tools > Requirements > Generate Report**.

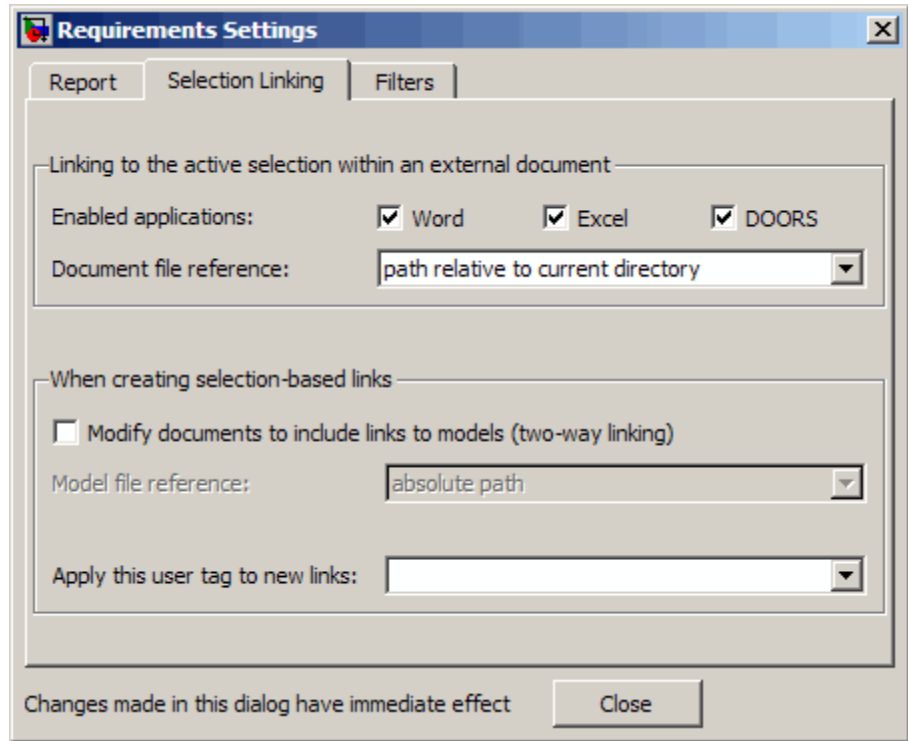
The report does not include information about objects whose requirements have the `test` user tag.

## **Applying User Tags During Selection-Based Linking**

When creating a succession of requirements links, you can apply the same user tags to all links automatically. This capability, also known as *selection-based linking*, is available only when you are creating links to selected objects in the requirements documents.

When creating selection-based links, specify one or more user tags to apply to requirements:

- 1** In the Model Editor, select **Tools > Requirements > Settings**.
- 2** Select the **Selection Linking** tab.



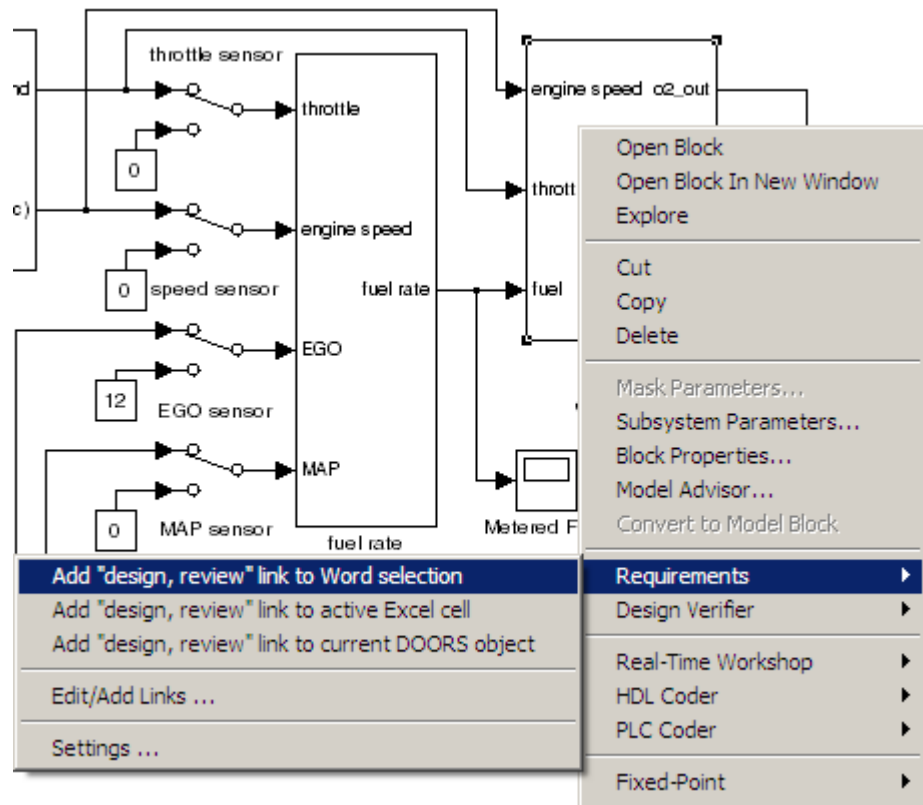
- 3 In the **Apply this user tag to new links** field, enter one or more user tags, separated by commas.

The RMI applies these user tags to all selection-based requirements links that you create.

- 4 Click **Close** to close the Requirements Settings dialog box.
- 5 In a requirements document, select the desired requirement text.
- 6 Right-click a model object and select **Requirements**.

The selection-based linking options identify which user tags the RMI applies to the link that you create. In the following example, you can apply the user tags **design** and **review** to the link that you create to your selected text.





## Configuring Requirements Filtering

In the Requirements Settings dialog box, the **Filters** tab has the following options for filtering requirements in a model.

<b>Option</b>	<b>Description</b>
<b>Filter links by user tags when highlighting and reporting requirements</b>	Enables filtering for highlighting and reporting, based on specified user tags.
<b>Include links with any of these tags</b>	Includes information about all requirements that have any of the specified user tags. Separates multiple user tags with commas.
<b>Exclude links with any of these tags</b>	Excludes information about all requirements that have any of the specified user tags. Separates multiple user tags with commas or spaces.
<b>Apply same filters in context menus</b>	Disables link labels in context menus if any of the specified filters are satisfied, for example if a requirement has a designated user tag.
<b>Apply same filters in consistency checking</b>	Includes or excludes requirements with specified user tags when running a consistency check between a model and its associated requirements documents.
<b>Under Link type filters, Disable DOORS surrogate item links in context menus</b>	Disables links to IBM Rational DOORS surrogate items from the context menus when you right-click a model object. This option does not depend on current user tag filters.

# Keeping Requirements Information Up to Date

---

- “Checking Requirements Consistency” on page 6-2
- “Deleting Requirement Links from Simulink Objects” on page 6-9
- “Managing Requirements in Linked Libraries” on page 6-11

## Checking Requirements Consistency

### In this section...

“Running Consistency Checks” on page 6-2

“Fixing Inconsistent Links” on page 6-4

“Resolving the Document Path” on page 6-7

### Running Consistency Checks

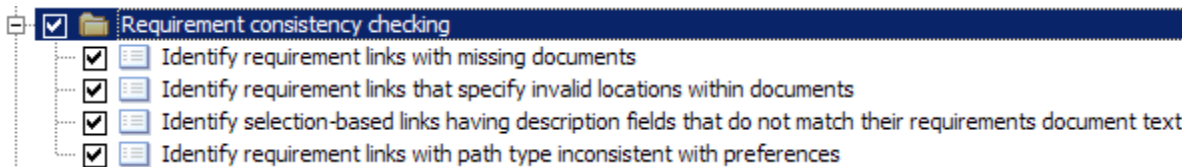
If you change your model or your requirements documents, over time, the requirements links can become out of date. To make sure that every requirements link has a matching target in a requirements document, run the Model Advisor Requirements consistency checks, for example:

- 1 Open the demo model:

```
slvndemo_fuelsys_officereq
```

- 2 Open the Model Advisor to run a consistency check by selecting **Tools > Requirements > Consistency checking**.

In the **Requirement consistency checking** category, all the checks are selected. For this tutorial, keep all the checks selected.



These checks identify the following problems with your model requirements.

<b>Consistency Check</b>	<b>Problem Identified</b>
<b>Identify requirement links with missing documents</b>	The Model Advisor cannot find the requirements document.
<b>Identify requirement links that specify invalid locations within documents</b>	The Model Advisor cannot find the designated location in the requirement document.
<b>Identify selection-based links having description fields that do not match their requirements document text</b>	The <b>Description</b> field for the link does not match the requirements document text. When you create selection-based links, the RMI saves the selected text in the link <b>Description</b> field.
<b>Identify requirement links with path type inconsistent with preferences</b>	The path for the requirements document does not match the <b>Document file reference</b> field in the Requirements Settings dialog box <b>Selection Linking</b> tab.  For information about how the RMI resolves the path to the requirements document, see “Resolving the Document Path” on page 6-7.

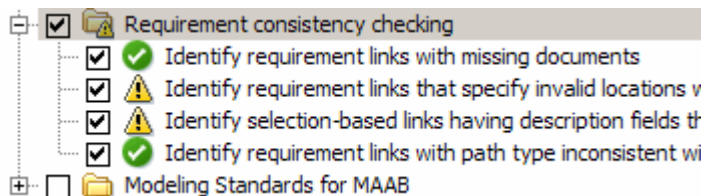
The Model Advisor checks to see if any applications that have link targets are running:

- If your model has links to Microsoft Word or Microsoft Excel documents, the consistency check requires that you close all instances of those applications. If you have one of these applications open, it displays a warning and does not continue the checks. The consistency checks must verify up-to-date stored copies of the requirements documents.
  - If your model has links to DOORS requirements, you must be logged in to the DOORS software.
- 3** For this tutorial, make sure that you close both Microsoft Word and Microsoft Excel.

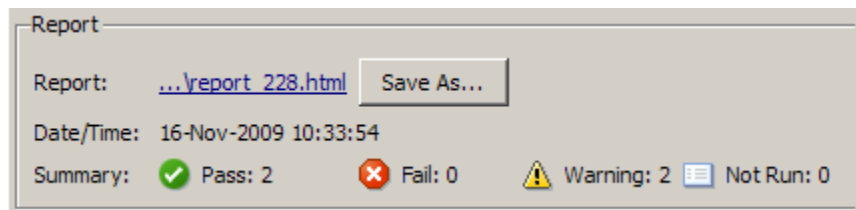
### 4 Click **Run Selected Checks**.

After the check is complete:

- The green circles with the check mark indicate that two checks passed.
- The yellow triangles with the exclamation point indicate that two checks generated warnings.



The right-hand pane shows that two checks passed and two checks had warnings. The pane includes a link to the HTML report.



Keep the Model Advisor open. The next section describes how to interpret and fix inconsistent links.

## Fixing Inconsistent Links

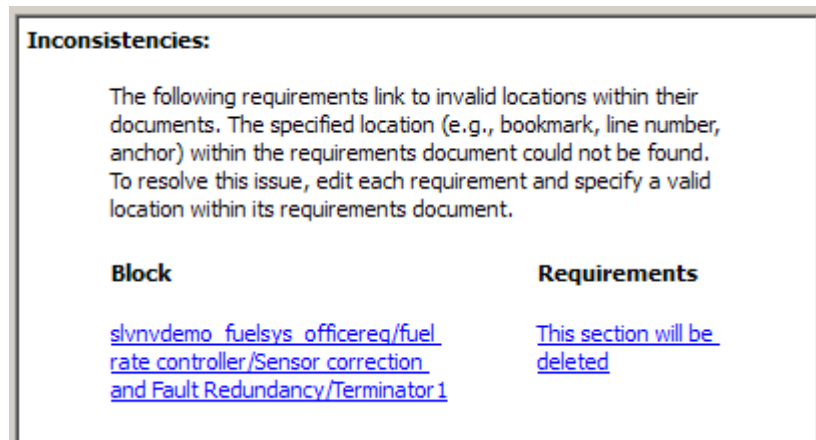
In “Running Consistency Checks” on page 6-2, two requirements consistency checks generate the following warnings in the `slvnvdemo_fuelsys_officereq` model:

- “Requirement Link Specifies an Invalid Location in the Requirements Document” on page 6-5
- “Link Description Field Does Not Match the Requirements Document Text” on page 6-6

## Requirement Link Specifies an Invalid Location in the Requirements Document

To fix the warning about attempting to link to an invalid location in a requirements document:

- 1 In the Model Advisor, select **Identify requirement links that specify invalid locations within documents** to display the description of the warning.



This check identifies a link that specifies a location that does not exist in the Microsoft Word requirements document, `slvnvdemo_FuelSys_DesignDescription.docx`. The link originates in the Terminator1 block. In this example, the target location in the requirements document was deleted after the requirement was created.

- 2 Get more information about this link:
  - a To navigate to the Terminator1 block, under **Block**, click the hyperlink.
  - b To open the Requirements dialog box for this link, under **Requirements**, click the hyperlink.
- 3 Fix the problem from the Requirements dialog box, do one of the following:
  - In the **Location** field, specify a valid location in the requirements document.

- Delete the requirements link by selecting the link and clicking **Delete**.
- 4** In the Model Advisor, select the **Requirement consistency checking** category of checks.
  - 5** Click **Run Selected Checks** again, and verify that the warning no longer occurs.

### Link Description Field Does Not Match the Requirements Document Text

To fix the warning about the **Description** field not matching the requirement text:

- 1** In the Model Advisor, click **Identify selection-based links having description fields that do not match their requirements document text** to display the description of the warning.

**Inconsistencies:**

The following selection-based links have descriptions that differ from their corresponding selections in the requirements documents. If this reflects a change in the requirements document, click **Update** to replace the current description in the selection-based link with the text from the requirements document (the external description).

<b>Block</b>	<b>Current description</b>	<b>External description</b>	
<a href="#">slvrvdemo_fuelsys_officereq/fuel_rate_controller/Sensor correction and Fault Redundancy/MAP Estimate</a>	<a href="#">Manifold pressure failure</a>	Manifold pressure failure mode	<a href="#">Update</a>

This check identified mismatching text between the MAP Estimate block requirement and the requirements document:

- The **Description** field in the MAP Estimate block link is **Manifold pressure failure**.
  - The requirement text in slvrvdemo\_FuelSys\_DesignDescription.docx is **Manifold pressure failure mode**.
- 2** Get more information about this link:





### Relative (Partial) Path Example

Current MATLAB folder	C:\work\scratch
Model file	C:\work\models\controllers\pid.mdl
Document link	..\reqs\pid.html
Documents searched for (in order)	C:\work\reqs\pid.html C:\work\models\reqs\pid.html

### Relative (No) Path Example

Current MATLAB folder	C:\work\scratch
Model file	C:\work\models\controllers\pid.mdl
Requirements document	pid.html
Documents searched for (in order)	C:\work\scratch\pid.html <MATLAB path dir>\pid.html C:\work\models\controllers\pid.html

### Absolute Path Example

Current MATLAB folder	C:\work\scratch
Model file	C:\work\models\controllers\pid.mdl
Requirements document	C:\work\reqs\pid.html
Documents searched for	C:\work\reqs\pid.html

## Deleting Requirement Links from Simulink Objects

### In this section...

“Deleting a Single Link from a Simulink Object” on page 6-9

“Deleting All Links from a Simulink Object” on page 6-9

“Deleting Links from Multiple Simulink Objects” on page 6-10

### Deleting a Single Link from a Simulink Object

If you have an obsolete link to a requirement, delete it from the model object.

To delete a single link to a requirement from a Simulink model object:

- 1 Right-click a model object and select **Requirements > Edit/Add Links**.
- 2 In the top-most pane of the Requirements dialog box, select the link that you want to delete.
- 3 Click **Delete**.
- 4 Click **Apply** or **OK** to complete the deletion.

### Deleting All Links from a Simulink Object

To delete all links to requirements from a Simulink model object:

- 1 Right-click the model object and select **Requirements > Delete All Links**
- 2 Click **OK** to confirm the deletion.

This action deletes all requirements at the top level of the object. For example, if you delete requirements for a subsystem, this action does not delete any requirements for objects inside the subsystem; it only deletes requirements for the subsystem itself. To delete requirements for child objects inside a subsystem, Model block, or Stateflow chart, you must navigate to each child object and perform these steps for each object from which you want to delete requirements.

### **Deleting Links from Multiple Simulink Objects**

To delete all links to requirements from a group of Simulink model objects:

- 1** Select the model objects whose requirements links you want to delete.
- 2** Right-click one of the objects and select **Requirements > Delete All**.
- 3** Click **OK** to confirm the deletion.

This action deletes all requirements at the top level of each object. It does not delete requirements for any child objects inside subsystems, Model blocks, or Stateflow charts.

## Managing Requirements in Linked Libraries

In this section...
“Copying Library Blocks with Requirements” on page 6-11
“Links from Requirements to Library Blocks” on page 6-13
“Managing Requirements Links Inside Reference Blocks” on page 6-14

Simulink allows you to create your own block libraries. Creating a block library allows you to reuse the functionality of a block or subsystem in multiple models.

When you use a library block in a Simulink model, the new block is called a *reference block*. The reference block is connected to the library block using a *library link*.

If you change a library block, any reference block that is linked to the library block updates with those changes when its parent model is opened.

---

**Note** For more information about reference blocks and library links, see “Working with Block Libraries” in the Simulink documentation.

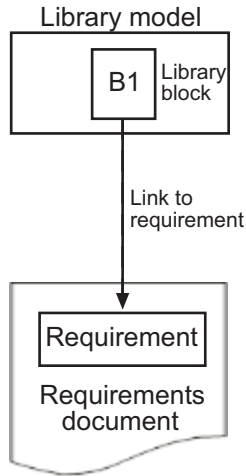
---

Library blocks can have links to requirements in external documents that the Requirements Management Interface (RMI) supports. The following sections describe how those requirements are maintained between the library block and the reference block.

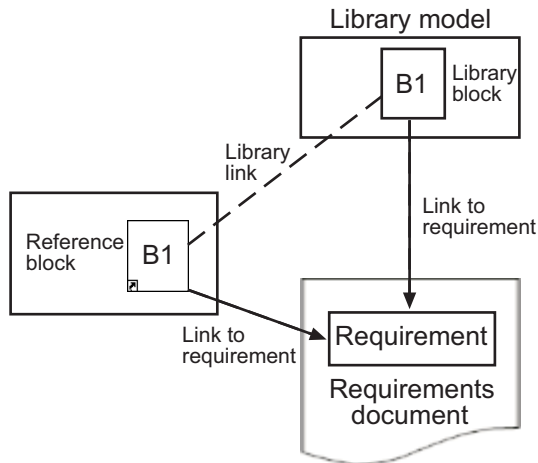
### Copying Library Blocks with Requirements

You can create a link to a requirement from a library block. After you drag that library block to a model, Simulink updates the reference block with the characteristics of the library block, including any requirements links.

For example, consider the library block B1 in the following graphic. B1 has a link to a requirement in a requirements document.



If you drag library block B1 to a model, you create a reference block that contains the characteristics of B1, including any requirements links.



You can navigate from the reference block to the linked requirements document in the same way that you navigate from the library block to the requirements document.

---

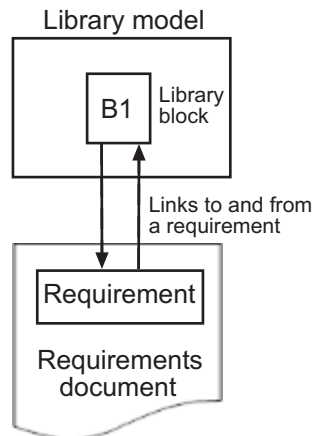
**Note** For more information about navigating from a block to a linked requirement, see “Navigating to Requirements from a Model” on page 5-5.

---

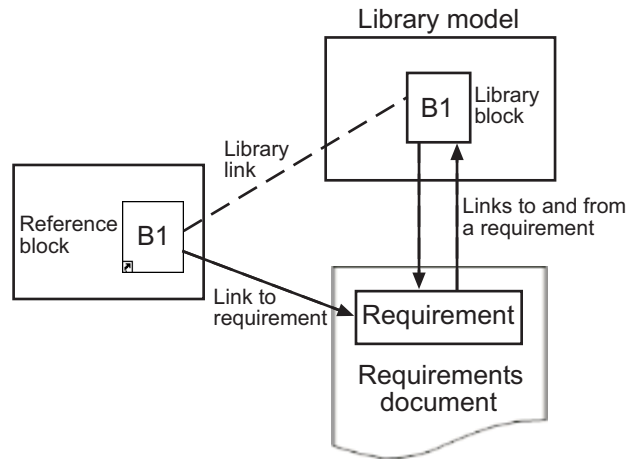
## Links from Requirements to Library Blocks

If you have a requirement that links to a library block and you drag that library block to a model, the requirement does not link to the reference block; the requirement continues to link *only* to the library block.

For example, consider the situation where you have established two-way linking between a library block (B1 in the following graphic) and a requirement.



When you use library block B1 in a model, Simulink also copies the requirement link to the reference block, as in the following graphic. However, the link from the requirement still points to library block B1, not to the reference block.



To create a link from the requirement to the reference block, follow the steps described in Chapter 8, “Adding Navigation Controls to IBM Rational DOORS Requirements” and Chapter 9, “Adding Navigation Controls to Microsoft Office Documents”.

### Managing Requirements Links Inside Reference Blocks

If you create a link to a requirement from inside a reference block, you can push that requirement link to the library block so that other models can use the updated library block, including the requirement.

The workflow for this task is as follows:

- 1 Disable the library link between the reference block and the library block.

You cannot modify the reference block without first disabling the library link.

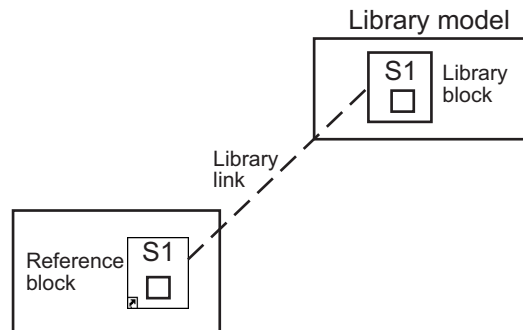


**2** Create the link from inside the reference block to the requirements document.

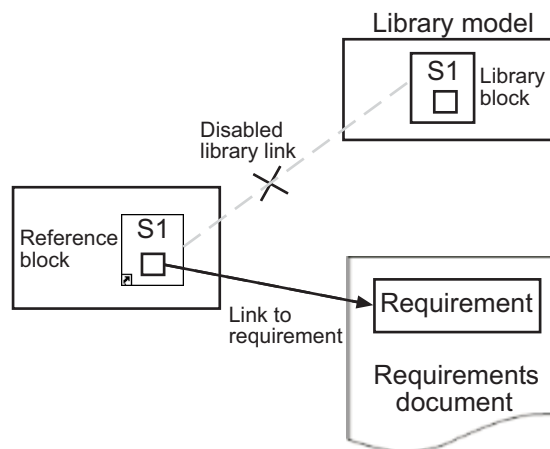
**3** Restore the library link between the reference block and the library block.

If you see a message that the library is locked, you must unlock the library before you can push the changes to the library block.

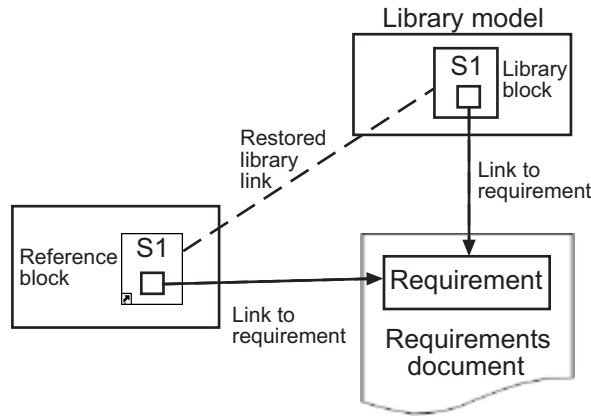
For example, suppose that within a library you have a subsystem S1, as in the following graphic. You drag S1 to a model, creating a new subsystem.



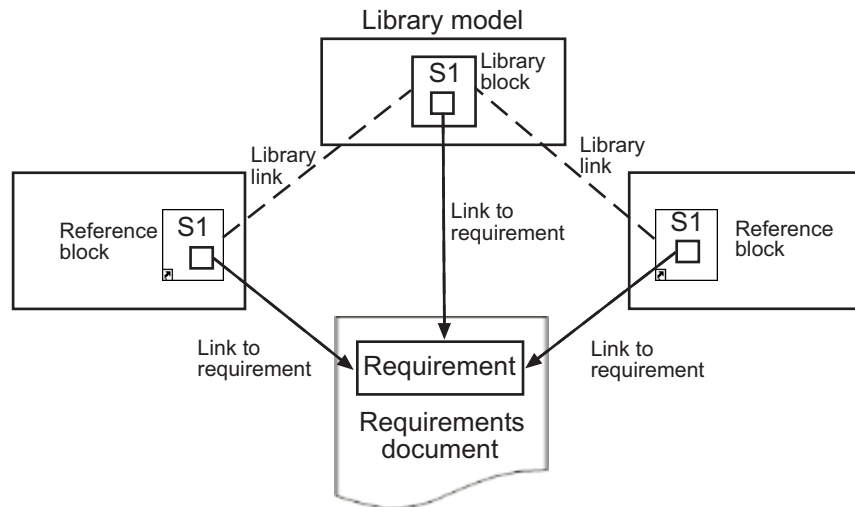
Then you disable the library link and create a link inside the reference block to a requirement.



If you then restore the library link between the library block and the reference block, Simulink pushes the new requirement link to the library block S1, along with any other changes you made to the reference block. The following graphic shows the new link from inside the library block S1 to the requirement.



Now suppose you reuse S1, which now has a requirement link, in another model. The new subsystem contains a link to the same requirement, as shown in the following graphic.



# Synchronizing a Simulink Model with a DOORS Surrogate Module

---

- “What Is Synchronization?” on page 7-2
- “Advantages of Synchronizing Your Model with a Surrogate Module” on page 7-4
- “Tutorial: Synchronizing a Simulink Model to Create a Surrogate Module” on page 7-5
- “Tutorial: Creating Links Between the Surrogate Module and Formal Module in a DOORS Database During Synchronization” on page 7-7
- “Customizing the Synchronization” on page 7-8
- “Tutorial: Updating the Surrogate Module to Reflect Model Changes” on page 7-16
- “Navigating with the Surrogate Module” on page 7-18

### What Is Synchronization?

Synchronization is a user-initiated process that creates or updates a DOORS surrogate module. A *surrogate module* is a DOORS formal module that is a representation of a Simulink model hierarchy.

When you synchronize a model for the first time, the DOORS software creates a surrogate module. The surrogate module contains a representation of the model, depending on your synchronization settings. (To learn how to customize the links and level of detail in the synchronization, see “Customizing the Synchronization” on page 7-8.)

If you create new objects links or remove existing objects links, keep your surrogate module up to date by resynchronizing. The updated surrogate module reflects any changes in the requirements links since the previous synchronization.

---

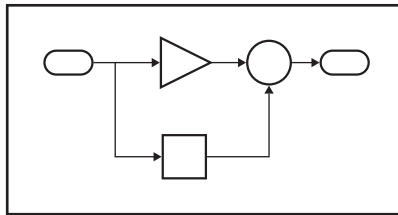
**Note** The RMI and DOORS software both use the term *object*. In the RMI, and in this document, the term *object* refers to a Simulink model or block, or to a Stateflow chart or its contents.

In the DOORS software, *object* refers to numbered elements in modules. The DOORS software assigns each of these objects a unique object ID. In this document, these objects are referred to as *DOORS objects*.

---

You use standard DOORS capabilities to navigate between the Simulink objects in the surrogate module and requirements in other formal modules. The surrogate module facilitates navigation between the Simulink model object and the requirements, as the following diagram illustrates.

Objects in a Simulink Model



DOORS Surrogate Module

Object ID	Block Name
200	1 Model
202	1.1 Subsystem
203	1.1.1 Block
204	1.1.2 Block
205	1.1.3 Block
206	1.2 Subsystem
207	1.2.1 Block
208	1.3 Block

DOORS Formal Module(s) with Requirements

Object ID	Requirement
D1	1 <b>Requirement Name</b>
D2	◀ 1.1 Requirement text ...
	...
D3	1.2 Requirement text ...

A surrogate module is a representation of a Simulink model hierarchy.

Enter requirements in the DOORS formal module and link them to objects in the DOORS surrogate module, so you can navigate from requirements to Simulink objects.

# Advantages of Synchronizing Your Model with a Surrogate Module

Synchronizing your Simulink model with a surrogate module offers the following advantages:

- You can navigate from a requirement to a Simulink object without modifying the requirements modules.
- You avoid cluttering your requirements modules with inserted navigation objects.
- The DOORS database contains complete information about requirements links. You can review requirements links and verify traceability, even if the Simulink software is not running.
- You can use DOORS reporting features to analyze requirements coverage.
- You can separate the requirements tracking work from the Simulink model developers' work, as follows:
  - Systems engineers can establish requirements links to models without using the Simulink software.
  - Model developers can capture the requirements information using synchronization and store it with the model.
- You can resynchronize a model with a new surrogate module, updating any model changes, or specifying different synchronization options.

## Tutorial: Synchronizing a Simulink Model to Create a Surrogate Module

The first time that you synchronize your model with the DOORS software, the DOORS software creates a surrogate module.

In this tutorial, you synchronize the `sf_car` model with the DOORS software.

---

**Note** Before you begin, make sure you know how to create links from a Simulink model object to a requirement in a DOORS database. For a tutorial on creating links to DOORS requirements, see “Tutorial: Linking to Requirements in IBM Rational DOORS Databases” on page 3-11.

---

- 1** To create a surrogate module, start the DOORS software and open a project. If the DOORS software is not already running, start the DOORS software and open a project.
- 2** Open the `sf_car` model.
- 3** Rename the model to `sf_car_doors`, and save the model in a writable folder.
- 4** Create links to a DOORS formal module from two objects in `sf_car_doors`:
  - The transmission subsystem
  - The engine torque block inside the Engine subsystem
- 5** In the Model Editor, select **Tools > Requirements > Synchronize with DOORS**.

The DOORS synchronization settings dialog box opens.

- 6** For this tutorial, accept the default synchronization options.

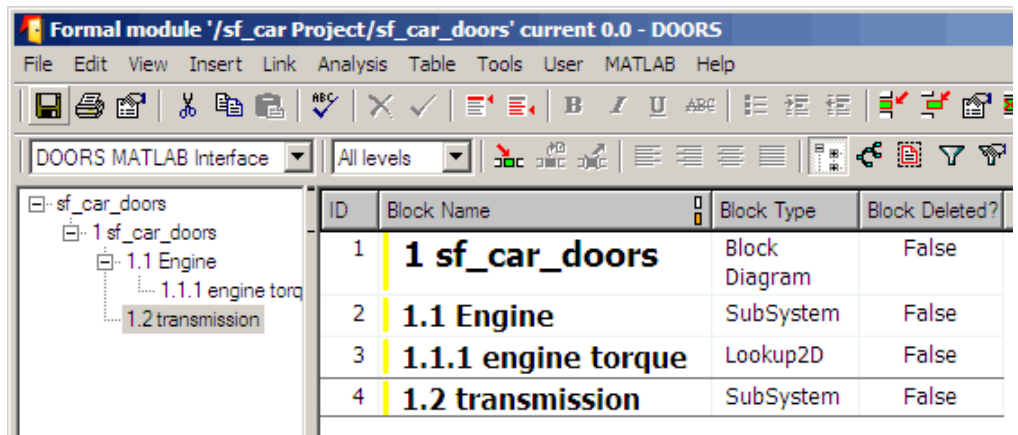
The default option, `None`, creates objects in the surrogate module for the model and any model objects with links to DOORS requirements.

For more information about the options, see “Customizing the Synchronization” on page 7-8.

- 7 Click **Synchronize** to create and open a surrogate module for all DOORS requirements that have links to objects in the sf\_car\_doors model.

After synchronization with the None option, the surrogate module contains:

- A top-level object for the model (sf\_car\_doors)
- Objects that represent model objects with links to DOORS requirements (transmission, engine torque), and their parent objects (Engine).



The screenshot displays the DOORS MATLAB Interface for a formal module named '/sf\_car Project/sf\_car\_doors' current 0.0 - DOORS. The interface includes a menu bar (File, Edit, View, Insert, Link, Analysis, Table, Tools, User, MATLAB, Help) and a toolbar with various icons. Below the toolbar, there is a dropdown menu for 'DOORS MATLAB Interface' and another for 'All levels'. The main area is divided into two panes. The left pane shows a hierarchical tree view of the surrogate module structure:

- sf\_car\_doors
  - 1 sf\_car\_doors
    - 1.1 Engine
      - 1.1.1 engine torque
      - 1.2 transmission

The right pane contains a table with the following data:

ID	Block Name	Block Type	Block Deleted?
1	<b>1 sf_car_doors</b>	Block Diagram	False
2	<b>1.1 Engine</b>	SubSystem	False
3	<b>1.1.1 engine torque</b>	Lookup2D	False
4	<b>1.2 transmission</b>	SubSystem	False

- 8 Save the surrogate module and the model.

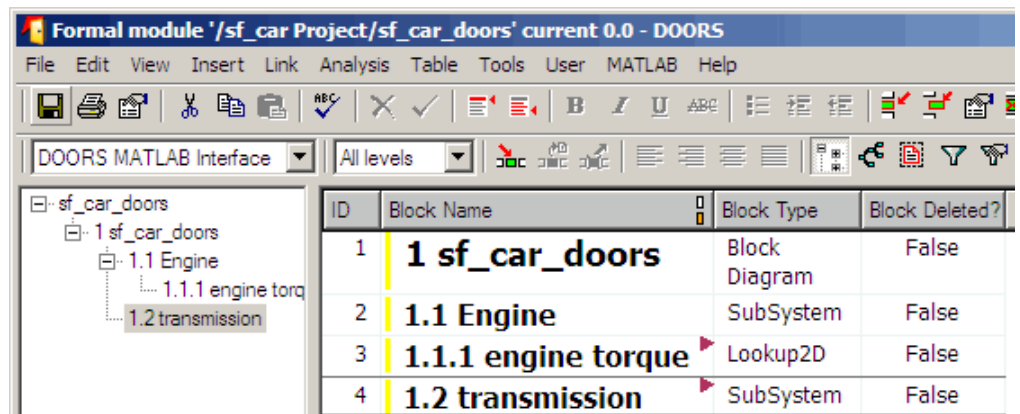


## Tutorial: Creating Links Between the Surrogate Module and Formal Module in a DOORS Database During Synchronization

Copy link information from a Simulink model to a surrogate module in a DOORS database:

- 1 Open the sf\_car\_doors model.
- 2 In the Model Editor, select **Tools > Requirements > Synchronize with DOORS**.
- 3 In the DOORS synchronization settings dialog box, select two options:
  - **Update links during synchronization**
  - **from Simulink to DOORS**.

When you select these options, the RMI creates links from the DOORS surrogate module to the formal module. These links correspond to links from the Simulink model to the formal module. In this example, the DOORS software copies the links from the engine torque block and transmission subsystems to the formal module, as indicated by the red triangles.



The screenshot shows the DOORS MATLAB Interface. On the left, a tree view displays the hierarchy of the sf\_car\_doors model, including sub-modules like 1.1 Engine and 1.2 transmission. On the right, a table lists the blocks in the formal module, with red triangles indicating links from the surrogate module.

ID	Block Name	Block Type	Block Deleted?
1	<b>1 sf_car_doors</b>	Block Diagram	False
2	<b>1.1 Engine</b>	SubSystem	False
3	<b>1.1.1 engine torque</b>	Lookup2D	False
4	<b>1.2 transmission</b>	SubSystem	False

## Customizing the Synchronization

In this section...
“DOORS Synchronization Settings” on page 7-8
“Resynchronizing a Model with a Different Surrogate Module” on page 7-10
“Customizing the Level of Detail in Synchronization” on page 7-11
“Tutorial: Resynchronizing to Include All Simulink Objects” on page 7-12

### DOORS Synchronization Settings

When you synchronize your Simulink model with a DOORS database, you can:

- Customize the level of detail for your surrogate module.
- Update links in the surrogate module or in the model to ensure consistency of requirements links among the model, and the surrogate and formal modules.

The DOORS synchronization settings dialog box provides the following options during synchronization.

DOORS Settings Option	Description
DOORS surrogate module path and name	Specifies a unique DOORS path to a new or an existing surrogate module.  For information about how the RMI resolves the path to the requirements document, see “Resolving the Document Path” on page 6-7.
Extra mapping additionally to objects with links	Determines the completeness of the Simulink model representation in the DOORS surrogate module. <b>None</b> specifies synchronizing only those Simulink objects that have linked requirements, and their parent objects. For more information about these synchronization options, see “Customizing the Level of Detail in Synchronization” on page 7-11.

<b>DOORS Settings Option</b>	<b>Description</b>
<b>Update links during synchronization</b>	Specifies updating any unmatched links the RMI encounters during synchronization, as designated in the <b>Copy unmatched links</b> and <b>Delete unmatched links</b> options.
<b>Copy unmatched links</b>	<p>During synchronization, selecting the following options has the following results:</p> <ul style="list-style-type: none"> <li>• <b>from Simulink to DOORS:</b> For links between the model and the formal module, the RMI creates matching links between the DOORS surrogate and formal modules.</li> <li>• <b>from DOORS to Simulink:</b> For links between the DOORS surrogate and formal modules, the RMI creates matching links between the model and the DOORS modules.</li> </ul>
<b>Delete unmatched links</b>	<p>During synchronization, selecting the following options has the following results:</p> <ul style="list-style-type: none"> <li>• <b>Remove unmatched in DOORS:</b> For links between the formal and surrogate modules, if there is not a corresponding link between the model and the DOORS modules, the RMI deletes the link in DOORS.  This option is available only if you select the <b>from Simulink to DOORS</b> option.</li> <li>• <b>Remove unmatched in Simulink:</b> For links between the model and the DOORS modules, if there is not a corresponding link between the formal and surrogate modules, the RMI deletes the link from the model.  This option is available only if you select the <b>from DOORS to Simulink</b> option.</li> </ul>

DOORS Settings Option	Description
Save DOORS surrogate module	After the synchronization, saves all changes to the surrogate module and updates the version of the surrogate module in the DOORS database.
Save Simulink model (recommended)	After the synchronization, saves all changes to the model. If you use a version control system, selecting this option changes the version of the model.

### Resynchronizing a Model with a Different Surrogate Module

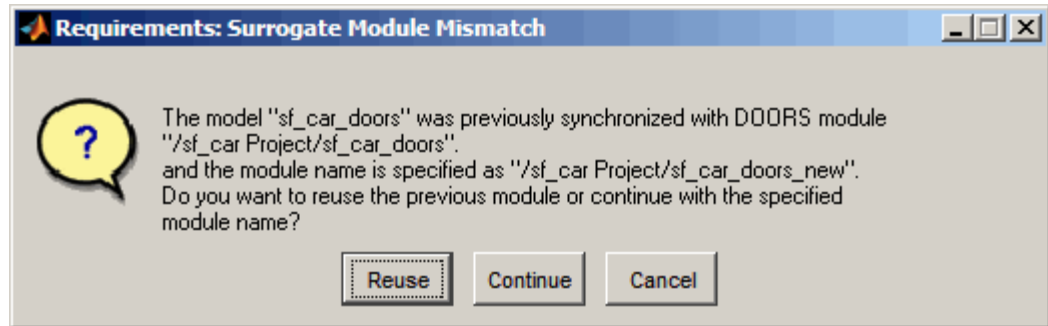
You can synchronize the same Simulink model with a new DOORS surrogate module. For example, you might want the surrogate module to contain only objects that have requirements to DOORS, rather than all objects in the model. In this case, you can change the synchronization options to reduce the level of detail in the surrogate module:

- 1 In the DOORS synchronization settings dialog box, change the **DOORS surrogate module path and name** to the path and name of the new surrogate module in the DOORS database.
- 2 Specify a module with either a relative path (starting with ./) or a full path (starting with /).

The software appends relative paths to the current DOORS project. Absolute paths must specify a project and a module name.

When you synchronize a model, the RMI automatically updates the **DOORS surrogate module path and name** with the actual full path. The RMI saves the unique module ID with the module.

- 3 If you select a new module path or if you have renamed the surrogate module, and you click **Synchronize**, the Requirements: Surrogate Module Mismatch dialog box opens.



- 4 Click **Continue** to create a new surrogate module with the new path or name.

## Customizing the Level of Detail in Synchronization

You can customize the level of detail in a surrogate module so that the module reflects the full or partial Simulink model hierarchy.

In “Tutorial: Synchronizing a Simulink Model to Create a Surrogate Module” on page 7-5, you synchronized the model with the **Extra mapping additionally to objects with links** option set to **None**. As a result, the surrogate module contains only Simulink objects that have requirement links, and their parent objects. Additional synchronization options, described in this section, can increase the level of surrogate detail. Increasing the level of surrogate detail can slow down synchronization.

The **Extra mapping additionally to objects with links** option can have one of the following values. Each subsequent option adds additional Simulink objects to the surrogate module. You choose **None** to minimize the surrogate size or **Complete** to create a full representation of your model. The **Complete** option adds all Simulink objects to the surrogate module, creating a one-to-one mapping of the Simulink model in the surrogate module. The intermediate options provide more levels of detail.

Drop-Down List Option	Description
None (Recommended for better performance)	Maps only Simulink objects that have requirements links and their parent objects to the surrogate module.
Minimal - Non-empty unmasked subsystems and Stateflow charts	Adds all nonempty Stateflow charts and unmasked Simulink subsystems to the surrogate module.
Moderate - Unmasked subsystems, Stateflow charts, and superstates	Adds Stateflow superstates to the surrogate module.
Average Nontrivial Simulink blocks, Stateflow charts and states	Adds all Stateflow charts and states and Simulink blocks, except for trivial blocks such as ports, bus objects, and data-type converters, to the surrogate module.
Extensive - All unmasked blocks, subsystems, states and transitions	Adds all unmasked blocks, subsystems, states, and transitions to the surrogate module.
Complete - All blocks, subsystems, states and transitions	Copies <i>all</i> blocks, subsystems, states, and transitions to the surrogate module.

### Tutorial: Resynchronizing to Include All Simulink Objects

This tutorial shows how you can include *all* Simulink objects in the DOORS surrogate module. Before you start these steps, make sure you have completed the tutorials “Tutorial: Synchronizing a Simulink Model to Create a Surrogate Module” on page 7-5 and “Tutorial: Creating Links Between the Surrogate Module and Formal Module in a DOORS Database During Synchronization” on page 7-7.

- 1 Open the `sf_car_doors` model that you synchronized in “Tutorial: Synchronizing a Simulink Model to Create a Surrogate Module” on page 7-5 and again in “Tutorial: Creating Links Between the Surrogate Module and Formal Module in a DOORS Database During Synchronization” on page 7-7.
- 2 In the Model Editor, select **Tools > Requirements > Synchronize with DOORS**.

The DOORS synchronization settings dialog box opens.

- 3** Resynchronize with the same surrogate module, making sure that the **DOORS surrogate module path and name** specifies the surrogate module path and name that you used in “Tutorial: Synchronizing a Simulink Model to Create a Surrogate Module” on page 7-5.

For information about how the RMI resolves the path to the requirements document, see “Resolving the Document Path” on page 6-7.

- 4** Update the surrogate module to include *all* objects in your model. To do this, under **Extra mapping additionally to objects with links**, from the drop-down list, select Complete - All blocks, subsystems, states and transitions.

- 5** Click **Synchronize**.

After synchronization, the DOORS surrogate module for the `sf_car_doors` model opens with the updates. All Simulink objects and all Stateflow objects in the `sf_car_doors` model are now mapped in the surrogate module.

ID	Block Name	Block Type	Block Deleted?
1	<b>1 sf_car_doors</b>	Block Diagram	False
2	<b>1.1 Engine</b>	SubSystem	False
5	<b>1.1.1 Ti</b>	Inport	False
6	<b>1.1.2 throttle</b>	Inport	False
7	<b>1.1.3 Integrator</b>	Integrator	False
8	<b>1.1.4 Sum</b>	Sum	False
3	<b>1.1.5 engine torque</b>	Lookup2D	False
9	<b>1.1.6 engine + impeller inertia</b>	Gain	False
10	<b>1.1.7 Ne</b>	Outport	False
11	<b>1.2 Mux</b>	Mux	False
12	<b>1.3 User Inputs:Passing Maneuver</b>	Signal Group	False
13	<b>1.4 User Inputs:Gradual Acceleration</b>	Signal Group	False
14	<b>1.5 User Inputs:Hard</b>	Signal Group	False

6 Scroll through the surrogate module. Notice that the objects with requirements (the engine torque block and transmission subsystem) retain their links to the DOORS formal module, as indicated by the red triangles.

7 Save the surrogate module.

### Detailed Information About The Surrogate Module You Created

Notice the following information about the surrogate module that you created in “Tutorial: Resynchronizing to Include All Simulink Objects” on page 7-12:



- The name of the surrogate module is `sf_car_doors`, as you specified in the DOORS synchronization settings dialog box.
- DOORS object headers are the names of the corresponding Simulink objects.
- The **Block Type** column identifies each object as a particular block type or a subsystem.
- If you delete a previously synchronized object from your Simulink model and then resynchronize, the **Block Deleted** column reads **true**. Otherwise, it reads **false**.

These objects are not deleted from the surrogate module. The DOORS software retains these surrogate module objects so that the RMI can recover these links if you later restore the model object.

- Each Simulink object has a unique ID in the surrogate module. For example, the ID for the surrogate module object associated with the Mux block in the preceding figure is 11.
- Before the complete synchronization, the surrogate module contained the transmission subsystem, with an ID of 3. After the complete synchronization, the transmission object retains its ID (3), but is listed farther down in the surrogate module. This order reflects the model hierarchy. The transmission object in the surrogate module retains the red arrow that indicates that it links to a DOORS formal module object.

## Tutorial: Updating the Surrogate Module to Reflect Model Changes

If you change your model after synchronization, the RMI does not display a warning message. If you want the surrogate module to reflect changes to the Simulink model, resynchronize your model.

In this tutorial, you add a new block to the `sf_car_doors` model, and later delete it, resynchronizing after each step:

- 1 In the `sf_car_doors` model, make a copy of the vehicle mph (yellow) & throttle % Scope block and paste it into the model. The name of the new Scope block is vehicle mph (yellow) & throttle %1.
- 2 Select **Tools > Requirements > Synchronize with DOORS**.
- 3 In the DOORS settings dialog box, leave the **Extra mapping additionally to objects with links** option set to Complete - All blocks, subsystems, states, and transitions. Click **Synchronize**.

After the synchronization, the surrogate module includes the new block.

89	<b>1.10.6 Ti</b>	Output	False
90	<b>1.10.7 Tout</b>	Output	False
91	<b>1.11 vehicle mph (yellow) &amp; throttle %0</b>	Scope	False
92	<b>1.12 vehicle mph (yellow) &amp; throttle %01</b>	Scope	True

- 4 In the `sf_car_doors` model, delete the newly added Scope block and resynchronize.

The block that you delete appears at the bottom of the list of objects in the surrogate module. Its entry in the **Block Deleted** column reads True.

89	<b>1.10.6 Ti</b>	Outport	False
90	<b>1.10.7 Tout</b>	Outport	False
91	<b>1.11 vehicle mph (yellow) &amp; throttle %0</b>	Scope	False
92	<b>1.12 vehicle mph (yellow) &amp; throttle %01</b>	Scope	True

- 5** Delete the copied object (vehicle mph (yellow) & throttle %0) and resynchronize the model.
- 6** Save the surrogate module.
- 7** Save the sf\_car\_doors model.

## Navigating with the Surrogate Module

In this section...
“Navigating Between Requirements and the Surrogate Module in the DOORS Database” on page 7-18
“Two-Way Navigation with the Surrogate Module” on page 7-19

### Navigating Between Requirements and the Surrogate Module in the DOORS Database

The surrogate module and the requirements in the formal module are both in the DOORS database. When you synchronize your model, the DOORS software creates links between the surrogate module objects and the requirements in the DOORS database.

Navigating between the requirements and the surrogate module allows you to review the requirements that have links to the model without starting the Simulink software.

To navigate from the surrogate module transmission object to the requirement in the formal module:

- 1 In the surrogate module object for the transmission subsystem, right-click the right-facing red arrow.

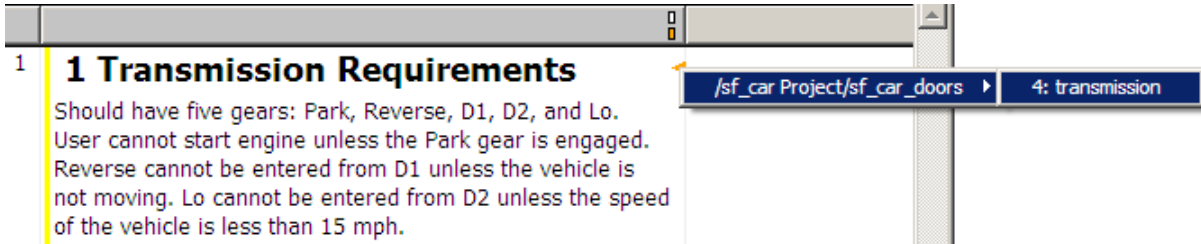
65	1.9.3.7 up_th	Output	False
4	1.10 transmission	/sf_car Project/sf_car Requirements ▶ 1: Transmission Requirements: Shoul	
66	1.10.1 Ne	Inport	False

- 2 Select the requirement name.

The formal module opens, at the Transmission Requirements object.

To navigate from the requirement in the formal module to the surrogate module:

- 1 In the Transmission Requirements object in the formal module, right-click the left-facing orange arrow.



- 2 Select the object name.

The surrogate module for sf\_car\_doors opens, at the object associated with the transmission subsystem.

## Two-Way Navigation with the Surrogate Module

Two-way navigation allows you to navigate from Simulink objects to DOORS requirements and from DOORS requirements to the model. If you synchronize your model, you using the surrogate module as an intermediary for the navigation in both directions. The surrogate module allows two-way navigation to remain available even if you remove the direct link from the model object to the DOORS formal module.

### Navigating from a Simulink Object to a Requirement via the Surrogate Module

To navigate from the transmission subsystem in the sf\_car\_doors model to a requirement in the DOORS formal module:

- 1 In the sf\_car\_doors model, right-click the transmission subsystem and select **Requirements > 1. “DOORS Surrogate Item”**. (The direct link to the DOORS formal module is also available.)

The surrogate module opens, at the object associated with the transmission subsystem.

- 2 To display the individual requirement, in the surrogate module, right-click the right-facing red arrow and select the requirement.

The formal module opens, at Transmission Requirements.

### **Navigating from a Requirement to the Model via the Surrogate Module**

To navigate from the Transmission Requirements requirement in the formal module to the transmission subsystem in the `sf_car_doors` model:

- 1** In the formal module, in the Transmission Requirements object, right-click the left-facing orange arrow.
- 2** Select the path to the linked surrogate object: `/sf_car Project/sf_car_doors > 4. transmission`.

The surrogate module opens, at the transmission object.

- 3** In the surrogate module, select **MATLAB > Select item**.

The linked object is highlighted in `sf_car_doors`.

# Adding Navigation Controls to IBM Rational DOORS Requirements

---

- “Why Add Navigation Controls to DOORS Requirements?” on page 8-2
- “Configuring the Requirements Management Interface for DOORS Software” on page 8-3
- “Enabling Two-Way Linking for IBM Rational DOORS Databases” on page 8-6
- “Inserting Navigation Controls into DOORS Requirements” on page 8-8
- “Navigating Between a DOORS Requirement and a Model Object” on page 8-10
- “Troubleshooting Your DOORS Installation” on page 8-12

## Why Add Navigation Controls to DOORS Requirements?

IBM Rational DOORS software is a requirements management application that you use to capture, track, and manage requirements. The Requirements Management Interface (RMI) allows you to link objects in a Simulink model to requirements managed by external applications, including the DOORS software.

When you create a link from a model object to a DOORS requirement, the RMI stores information about the DOORS object in the model. This information allows you to navigate from the model to the associated requirement.

You can configure the RMI to insert a navigation control in the DOORS database. This control allows you to navigate from the DOORS requirement to the model object.

Inserting navigation controls into both the model and the DOORS database is *two-way linking*. You must have write access to the DOORS database for two-way linking to work.

---

**Note** One-way linking is the default. For more information, see “Tutorial: Linking to Requirements in IBM Rational DOORS Databases” on page 3-11.

---



# Configuring the Requirements Management Interface for DOORS Software

## In this section...

“Before You Begin” on page 8-3

“Configuring the RMI to Insert Navigation Controls” on page 8-3

“Manually Installing Additional Files for DOORS Software” on page 8-3

## Before You Begin

If you plan to use DOORS software with the RMI, install additional files to establish communication between the DOORS application and the Simulink software. The sections that follow describe the installation and configuration procedures.

## Configuring the RMI to Insert Navigation Controls

To enable communication between your Simulink model and your DOORS installation, run the RMI setup script. At the MATLAB Command Window, enter the following command:

```
rmi setup
```

This command registers ActiveX controls. This registration configures your DOORS installation to communicate with the Simulink software. Two-way linking with Microsoft Office applications also requires this registration.

## Manually Installing Additional Files for DOORS Software

The setup script automatically copies all the required DOORS files to the correct folders. However, the script may fail because of file permissions in your DOORS installation. If the script fails, change the file permissions on the DOORS installation folders and rerun the script.

Alternative, you can install the additional files into the folders specified manually, as described in the following steps:

- 1 If the DOORS software is running, close the application.
- 2 Copy the following files from `matlabroot\toolbox\slvnn\reqmgt` to the `<doors_install_dir>\lib\dx1\addins` folder.

```
addins.idx
addins.hlp
```

If you have not modified the files, replace any existing versions of the files; otherwise, merge the contents of both files into a single file.

- 3 Copy the following files from `matlabroot\toolbox\slvnn\reqmgt` to the `<doors_install_dir>\lib\dx1\addins\dmi` folder.

```
dmi.hlp
dmi.idx
dmi.inc
runsim.dxl
selblk.dxl
```

Replace any existing versions of these files.

- 4 Open the `<doors_install_dir>\lib\dx1\startup.dxl` file. In the user-defined files section, add the following include statement:

```
#include <addins/dmi/dmi.inc>
```

If you upgrade from Version 7.1 to a later version of the DOORS software, perform these additional steps:

- a In your DOORS installation folder, navigate to the `...\lib\dx1\startupFiles` subfolder.
- b In a text editor, open the `copiedFromDoors7.dxl` file.
- c Add `//` before this line to comment it out:

```
#include <addins/dmi/dmi.inc>
```

- d Save and close the file.

- 5 Start the DOORS and MATLAB software.
- 6 Run the setup script:

rmi setup

## Enabling Two-Way Linking for IBM Rational DOORS Databases

By default, the RMI does not insert navigation controls into requirements documents. If you want the RMI to insert navigation controls when you create a link from a model object to a requirement, you must enable two-way linking.

The RMI can insert navigation controls into the following applications:

- IBM Rational DOORS
- Microsoft Excel
- Microsoft Word

Enable two-way linking:

- 1 Open the Simulink demo model:

```
sldemo_fuelsys
```

---

**Note** You can modify requirements settings only from the Model Editor. Even though you have a model open, any settings you change persist for all models you open subsequently.

---

- 2 Select **Tools > Requirements > Settings**.

The Requirements Settings dialog box opens.

- 3 Click the **Selection Linking** tab.

- 4 Select the **Modify documents to include links to models (two-way linking)** option.

When you select this option, every time you create a selection-based link from a model object to a requirement, the RMI inserts navigation controls at the designated location. Using this option requires write access to the requirements document.

- 5 Set the **Model file reference** option to none (on MATLAB path).

For this exercise, you save a copy of the demo model on the MATLAB path.

If you are adding requirements to a model that is not on the MATLAB path, select **absolute**, to indicate an absolute path to the model.

- 6 In the **Apply this user tag to new links** field, enter one or more user tags to apply to the two-way links that you create.

For more information about user tags, see “Filtering Requirements with User Tags” on page 5-21.

- 7 Click **Close** to close the Requirements Settings dialog box.

Keep the `sldemo_fuelsys` model open.

## Inserting Navigation Controls into DOORS Requirements

With two-way linking enabled, the RMI inserts a navigation control into both the model and the requirement. For this tutorial, you need a formal module that contains requirements. The examples in the tutorial show a demo model used for the purposes of illustration.


- 1 Open the Simulink demo model:


```
sldemo_fuelsys
```

- 2 Rename the `sldemo_fuelsys` model and save it in a writable folder on the MATLAB path.
- 3 Start the DOORS software and open a formal module that contains requirements.
- 4 Select the requirement that you want to link to by left-clicking that requirement in the DOORS database.
- 5 In the `sldemo_fuelsys` model, select an object in the model.

This example creates a requirement from the `fuel_rate_control` subsystem.

- 6 Right-click the model object and select **Requirements > Add link to current DOORS object**.

The RMI creates the link for the `fuel_rate_control` subsystem. It also inserts a new DOORS object into the formal module—a Simulink reference object (  ) that enables you to navigate from the requirement to the model.

ID	
1	<b>1 Fuel rate controller requirements</b> The controller will use engine speed, throttle position and manifold pressure to airflow through the engine.
2	 [Simulink reference: sldemo_fuelsys/fuel_rate_control (SubSystem)]

- 7 Close the model.

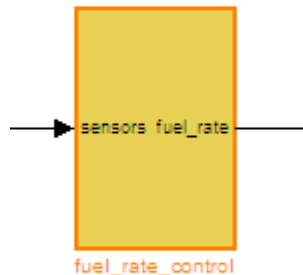
The next section describes how to navigate the two-way links that you created.

## Navigating Between a DOORS Requirement and a Model Object

In “Inserting Navigation Controls into DOORS Requirements” on page 8-8, you created a two-way link between a DOORS requirement and the `fuel_rate_control` subsystem in the `sldemo_fuelsys` model. Navigate the links in both directions:

- 1 With the `sldemo_fuelsys` model closed, go to the DOORS requirement in the formal module.
- 2 Left-click the Simulink reference object that you inserted to select it.
- 3 Select **MATLAB > Select item**.

Your version of the `sldemo_fuelsys` model opens, with the `fuel_rate_control` subsystem highlighted.



- 4 Log in to the DOORS software.
- 5 Navigate from the model to the DOORS requirement. In the Model Editor, right-click the `fuel_rate_control` subsystem and select **Requirements > 1. “<requirement name>”** where `<requirement name>` is the name of the DOORS requirement that you created.

The DOORS formal module opens with the requirement object and its child objects highlighted in red.



## 1 Fuel rate controller requirements

The controller will use engine speed, throttle position and manifold pressure airflow through the engine.

[Simulink reference: sldemo\_fuelsys\_test/fuel\_rate\_control (SubSystem)]



## Troubleshooting Your DOORS Installation

### **DXL Errors**

If you try to synchronize your Simulink model to a DOORS project, you may see the following errors:

- E- DXL: <Line:2> incorrectly concatenated tokens
- E- DXL: <Line:2> undeclared variable (dmiRefreshModule)
- I- DXL: all done with 2 errors and 0 warnings

If you see these errors, exit the DOORS software, rerun the `rmi setup` command at the MATLAB command prompt, and restart the DOORS software.

# Adding Navigation Controls to Microsoft Office Documents

---

- “Why Add Navigation Controls to Microsoft Office Requirements?” on page 9-2
- “Enabling Two-Way Linking for Microsoft Office Documents” on page 9-3
- “Inserting Navigation Controls in Microsoft Office Requirements Documents” on page 9-5
- “Navigating Between a Microsoft Word Requirement and a Model” on page 9-6
- “Troubleshooting Simulink Navigation Controls in Microsoft Office 2007” on page 9-7

## Why Add Navigation Controls to Microsoft Office Requirements?

You can use the Microsoft Word and Microsoft Excel applications to capture, track, and manage requirements. The Requirements Management Interface (RMI) allows you to link objects in a Simulink model to requirements managed by external applications.

When you create a link from a model to a requirement in a Microsoft Office document, the RMI stores information about the requirement in the model. With this information, you can navigate from the model to the associated requirement.

You can configure the RMI to insert a navigation reference in the Microsoft Office document. With this control, you can navigate from the requirement to the model object.

Inserting a navigation reference into both the model and a requirements document is *two-way linking*. You must have write access to the requirements document for two-way linking to work.

---

**Note** One-way linking is the default. For more information, see “Tutorial: Linking to Requirements in Microsoft Word Documents” on page 3-7.

---

## Enabling Two-Way Linking for Microsoft Office Documents

By default, the RMI does not insert navigation controls into requirements documents. So that the RMI inserts navigation controls into the requirements document when you create a link from a model object to a requirement, you must enable two-way linking.

The RMI can insert navigation controls into the following applications:

- IBM Rational DOORS
- Microsoft Excel
- Microsoft Word

Inserting navigation controls into Microsoft Office documents for two-way linking requires that ActiveX is enabled. If you cannot navigate using the links from the Microsoft Office document, you may also need to take the steps described in “ActiveX Control Does Not Link to Model Object” on page 9-10.

Enable two-way linking:

- 1** Open the demo model:

```
slvndemo_fuelsys_officereq
```

---

**Note** You can modify requirements settings only from the Model Editor. Even though you have a model open, any settings you change persist for all models you open subsequently.

---

- 2** Select **Tools > Requirements > Settings**.

The Requirements Settings dialog box opens.

- 3** Click the **Selection Linking** tab.

- 4** Select the **Modify documents to include links to models (two-way linking)** option.

When you select this option, every time you create a link from a model object to a requirement, the RMI inserts navigation controls into the designated location in the requirements document. If you do not have write access to the requirements document, save the requirements document that include the controls with a new file name.

- 5 For this exercise, you save a copy of the demo model on the MATLAB path. Set the **Model file reference** option to none (on MATLAB path).

If you are adding requirements to a model that is not on the MATLAB path, select **absolute**, to indicate an absolute path to the model.

- 6 To specify one or more user tags to apply to the two-way links that you create, in the **Apply this user tag to new links** field, enter the tags.

For more information about user tags, see “Filtering Requirements with User Tags” on page 5-21.

- 7 Click **Close** to close the Requirements Settings dialog box.

Keep the `slvndemo_fuelsys_officereq` model open.

## Inserting Navigation Controls in Microsoft Office Requirements Documents

Use selection-based linking to create a requirement from the `slvndemo_fuelsys_officereq` model to a requirements document. With two-way linking enabled, the RMI inserts a navigation control into both the model and the requirement.

- 1 Open the Microsoft Word requirements document:

```
matlabroot/toolbox/slvnv/rmidemos/fuelsys_req_docs/  
slvndemo_FuelSys_RequirementsSpecification.docx
```

- 2 Select the **Throttle Sensor** header.
- 3 In the `slvndemo_fuelsys_officereq` model, open the engine gas dynamics subsystem.
- 4 Right-click the Throttle & Manifold subsystem and select **Requirements > Add link to Word selection**.
- 5 The RMI inserts an ActiveX control into the requirements document.

### 1.1.6. Throttle Sensor

## Navigating Between a Microsoft Word Requirement and a Model

In “Inserting Navigation Controls in Microsoft Office Requirements Documents” on page 9-5, you created a two-way link between a Microsoft Word requirement and the Throttle & Manifold subsystem in the slvnvdemo\_fuelsys\_officereq demo model. Navigate these links in both directions:

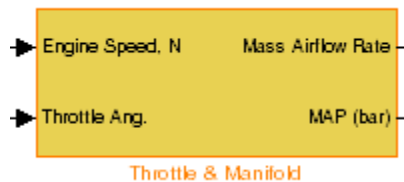
- 1 In the slvnvdemo\_fuelsys\_officereq model, right-click the Throttle & Manifold subsystem and select **Requirements > 1. “Throttle Sensor”**.

The requirements document opens, and the header in the requirements document is highlighted.

### 1.1.6. Throttle Sensor

- 2 In the requirements document, next to **Throttle Sensor**, double-click the navigation control.

The engine gas dynamics subsystem opens, with the Throttle & Manifold subsystem highlighted.



---

**Note** To ensure that the navigation controls work, in the Microsoft Office Trust Center, enable ActiveX controls.

---



# Troubleshooting Simulink Navigation Controls in Microsoft Office 2007

## In this section...

“Saving Requirements Documents to Microsoft Word 2007 Format” on page 9-7

“Field Codes in Requirements Documents” on page 9-8

“ActiveX Control Does Not Link to Model Object” on page 9-10

“Deleting an ActiveX Control from Microsoft® Excel 2007 file” on page 9-12

## Saving Requirements Documents to Microsoft Word 2007 Format

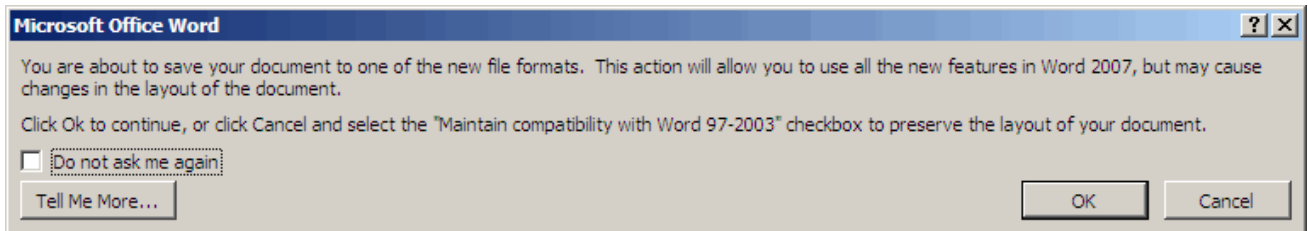
If you create a requirements document with an earlier version of Microsoft Word than Word 2007, the two-way links automatically work. If you open a document created in an earlier version and then save it in Microsoft Word 2007 format, make sure that the two-way links continue to work:

- 1 In the Microsoft Word window, in the upper-left corner, click the **Microsoft Office Button**.



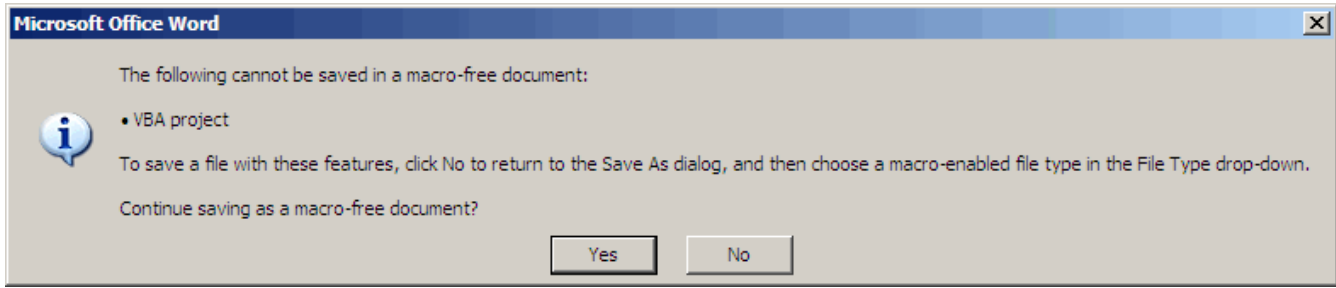
- 2 Select **Save As > Word Document**.

You see the following dialog box.



**3** Click **OK**.

You then see the following dialog box.




**4** Click **Yes** to save the current document in Microsoft Word 2007 format, with a .docx extension.

## Field Codes in Requirements Documents

If your Microsoft Word requirements document displays the field codes in addition to, or instead of, the ActiveX icon, clear the **Show field codes instead of their values** option in Microsoft Word 2007.

The following graphic shows a requirements document created in Microsoft Word 2003, with the field codes (CONTROL mwSimulink1.SLRefButton \s) displayed.

***Determination of pumping efficiency***{CONTROL  
mwSimulink1.SLRefButton \s }   
**Requirement ID:** REQ2  
**Model Element:** fuelsys/fuel rate controller/Airflow calculat  
**Details:** The airflow calculation will use a calibratib  
pumping efficiency of the engine based on  
manifold pressure.

The following graphic shows a requirements document created in Microsoft Word 2007, with the field codes (CONTROL mwSimulink1.SLRefButton) displayed.

## Primary Requirements

Requirement text. Requirement text. Requirement text.  
 Requirement text. Requirement text. Requirement text.  
 Requirement text. Requirement text. Requirement text. { CONTROL mwSimulink1.SLRefButton }  
 Requirement text. Requirement text. Requirement text.  
 Requirement text. Requirement text. Requirement text.  
 Requirement text. Requirement text. Requirement text.

To hide the field codes and display the ActiveX icon:

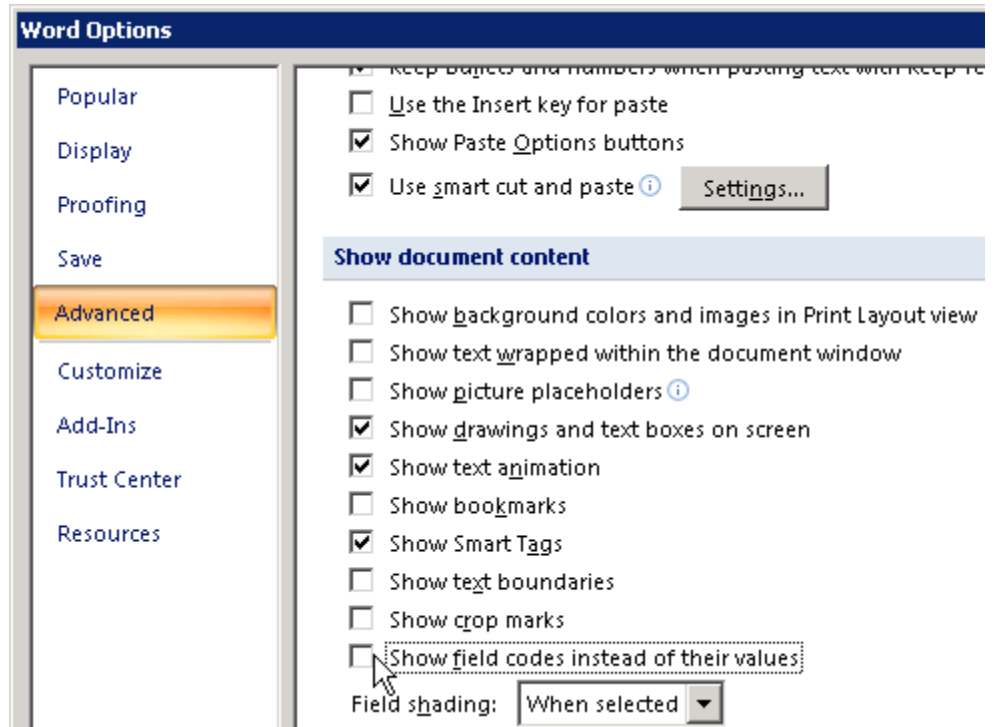
- 1 In the Microsoft Word window, in the upper-left corner, click the **Microsoft Office Button**.



- 2 In the pane that opens, at the bottom, click **Word Options**.



- 3 In the left-hand portion of the pane, click **Advanced**.
- 4 In the **Advanced** pane, scroll to the **Show document content** section and clear the **Show field codes instead of their values** option.



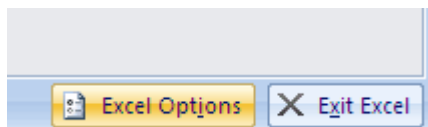
### ActiveX Control Does Not Link to Model Object

If you click an ActiveX control that links to a Simulink or Stateflow object, and the object does not open, do one of the following:

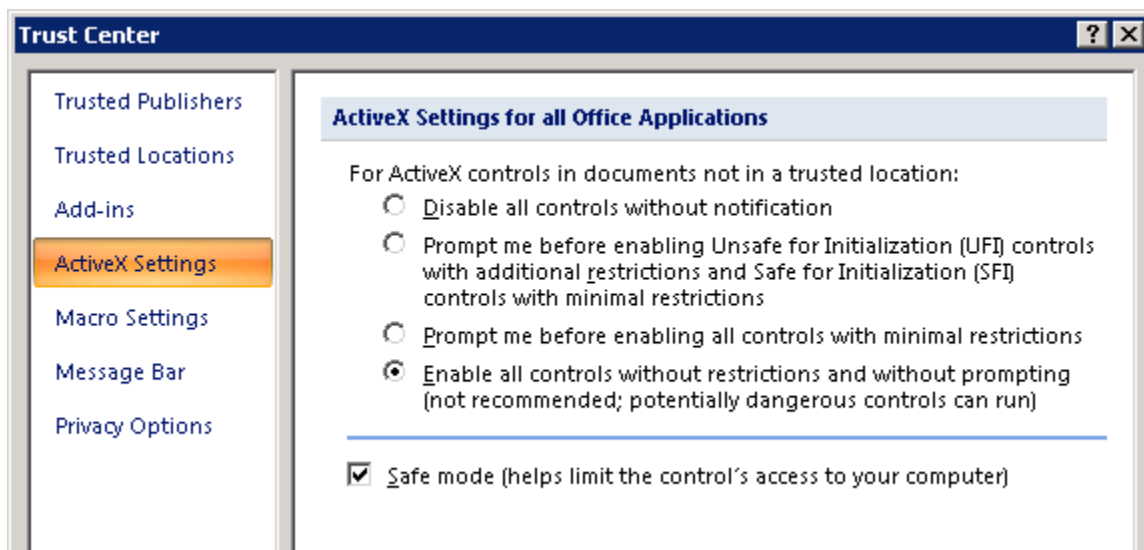
- Store your requirements documents in trusted locations, as described in the Microsoft Office 2007 documentation. The Trust Center does not check files for ActiveX controls stored in trusted locations, so you can maintain your Trust Center restrictions.
- Enable ActiveX controls:
  - 1 In the Microsoft Word or Microsoft Excel window, in the upper-left corner, click the **Microsoft Office Button**.



- 2 In the pane that opens, at the bottom, click **Word Options** or **Excel Options**, depending on which program you are running.



- 3 In the left-hand portion of the pane, click **Trust Center**.
- 4 In the **Trust Center** pane, click **Trust Center Settings**.
- 5 In the **Trust Center** pane, on the right, select **ActiveX Settings**.



- 6 Select the setting that you want for ActiveX controls:
  - **Prompt me for enabling all controls with minimum restrictions** to decide each time you click an ActiveX control if you want to enable all controls.

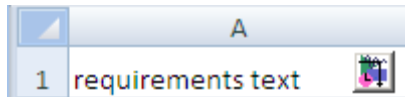
- **Enable all controls without restrictions and without prompting** to enable all ActiveX controls whenever you open the document.

7 Close and then restart the application for the settings to take effect.

## Deleting an ActiveX Control from Microsoft Excel 2007 file

Use the following procedure to remove an ActiveX control from your Microsoft Excel 2007 file.

- 1 Your document may have an ActiveX control in a worksheet cell:



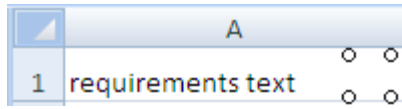
In the Microsoft Excel window, in the upper-left corner, click the **Microsoft Office Button**.



- 2 In the pane that opens, at the bottom, click **Excel Options**.
- 3 In the Excel Options dialog box, in the left-hand pane, click **Popular**.
- 4 On the **Popular** pane, in the **Top options for working with Excel** section, select **Show Developer tab in the Ribbon**.
- 5 Click **OK**.
- 6 In the Ribbon, on the **Developer** tab, select **Design Mode**.

When you select **Design Mode**, the ActiveX control is no longer visible in the cell.

- 7 Click where the ActiveX control was, and you see four handles showing the location of the control.



**8** Select **Home > Cut** to delete the control.





# Creating Custom Types of Requirements Documents

---

- “Why Create a Custom Link Type?” on page 10-2
- “Custom Link Type Registration” on page 10-3
- “Link Properties” on page 10-4
- “Link Type Properties” on page 10-5
- “Creating a Custom Link Requirement Type” on page 10-7
- “Navigating to Simulink Objects from External Documents” on page 10-17

## Why Create a Custom Link Type?

In addition to linking to built-in types of requirements documents, you can register custom requirements document types with the Requirements Management Interface (RMI). Then you can create requirement links to these types of documents.

Custom link types let you define how you:

- Open and navigate to a document
- Browse for a document
- View an index of a document's contents

When you define a custom link type, you create MATLAB functions that perform these operations. The RMI invokes the registered code:

- When navigating to a document with the new link type that you created.
- When browsing for a document or displaying the index of a document within the Requirements dialog box.

Using the external interfaces supported by the MATLAB software, you can interact with external applications and run programs from the command shell. You can also use the built-in Web browser and text editor to display custom variants of HTML and text files without installing external applications.

With custom link types, you can:

- Link to requirement items in commercial requirement tracking software
- Link to in-house database systems
- Link to document types that the RMI does not support

## Custom Link Type Registration

You register custom link types with a unique MATLAB function name. The function must exist on the MATLAB path and must not require any input arguments. The function must return a single output argument that is an instance of the requirements link type class. You can register your link type with the following MATLAB command:

```
rmi register mytargetfilename
```

*mytargetfilename* is the name of the MATLAB function, *mytargetfilename.m*.

Once you register a link type, it appears in the Requirements dialog box as an entry in the **Document type** drop-down list. A file in your preference folder contains the list of registered link types, so you can restore it in new MATLAB sessions. You can remove a link type with the following MATLAB command:

```
rmi unregister mytargetfilename
```

When you create links using custom link types, the software saves the registration name in the model. When you attempt to navigate to a link, the RMI resolves the link type against the registered list. If the software cannot find the link type, you see an error message.

### Link Properties

Requirements links are the data structures, saved in the Simulink model, that identify a specific location within a document. You get and set the links on a block using the `rmi` command. The RMI encapsulates link information in a MATLAB structure array. Each element of the array is a single requirement link.

Links and link types work together to perform navigation and manage requirements. The document and ID fields of links uniquely identify the linked item in external documents. The RMI passes both of these strings to the navigation command when you navigate a link from the model.

## Link Type Properties

Link type properties define how links are created, identified, navigated to, and stored within the requirement management tool. The following table describes each of these properties.

Property	Description
Registration	The name of the function that creates the link type. The RMI stores this name in the Simulink model.
Label	A string to identify this link type. In the Requirements dialog box, this string appears on the <b>Document type</b> drop-down list for a Simulink or Stateflow object.
IsFile	<p>A Boolean property that indicates if the linked documents are files within the computer file system. If a document is a file:</p> <ul style="list-style-type: none"> <li>• The software uses the standard method for resolving the path.</li> </ul> <p>For information about how the RMI resolves the path to the requirements document, see “Resolving the Document Path” on page 6-7.</p> <ul style="list-style-type: none"> <li>• In the Requirements dialog box, when you click <b>Browse</b>, the file selection dialog box opens.</li> </ul>
Extensions	An array of file extensions. Use these file extensions as filter options in the Requirements dialog box when you click <b>Browse</b> . The file extensions infer the link type based on the document name. If you registered more than one link type for the same file extension, the link type that you registered takes first priority.
LocDelimiters	A string containing the list of supported navigation delimiters. The first character in the ID of a requirement specifies the type of identifier. For example, an identifier can refer to a specific page number (#4), a named bookmark (@my_tag), or some searchable text (?search_text). The valid location delimiters determine the possible entries in the Requirements dialog box <b>Location</b> drop-down list.

<b>Property</b>	<b>Description</b>
NavigateFcn	<p>The MATLAB callback you invoke when you click a link. The function has two input arguments: the document field and the ID field of the link:</p> <pre>feval(LinkType.NavigateFcn, Link.document, Link.id)</pre>
ContentsFcn	<p>The MATLAB callback you invoke when you click the <b>Document Index</b> tab in the Requirements dialog box. This function has a single input argument that contains the full path of the resolved function or, if the link type is not a file, the <b>Document</b> field contents.</p> <p>The function returns three outputs:</p> <ul style="list-style-type: none"><li>• Labels</li><li>• Depths</li><li>• Locations</li></ul>
BrowseFcn	<p>The MATLAB callback you invoke when you click <b>Browse</b> in the Requirements dialog box. This function is not necessary when the link type is a file. The function takes no input arguments and returns a single output argument that identifies the selected document.</p>

## Creating a Custom Link Requirement Type

In this example, you implement a custom link type to a hypothetical document type, a text file with the extension `.abc`. Within a document, the requirement items are identified with a special text string, `Requirement::`, followed by a single space and then the requirement item inside quotation marks (`"`).

Create a document index containing a list of all the requirement items. When navigating from the Simulink model to the requirements document, the document opens in the MATLAB Editor at the line of the requirement that you want.

To create a custom link requirement type:

- 1 Write a function that implements the custom link type and save it on the MATLAB path. In this example, the file is `rmicustabcinterface.m`, containing the function, `rmicustabcinterface`, that implements the ABC files shipping with your installation. You can view it here, or at the MATLAB prompt, type `edit rmicustabcinterface`.

```
function linkType = rmicustabcinterface
%RMICUSTABCINTERFACE - Example custom requirement link type
%
% This file implements a requirements link type that maps
% to "ABC" files.
% You can use this link type to map a line or item within an
% ABC file to a Simulink or Stateflow object.
%
% You must register a custom requirement link type before
% using it. Once registered, the link type will be reloaded in
% subsequent sessions until you unregister it. The following
% commands perform registration and registration removal.
%
% Register command:  >> rmi register rmicustabcinterface
% Unregister command: >> rmi unregister rmicustabcinterface
%
% There is an example document of this link type contained in
% the requirement demo directory to determine the path to the
% document invoke:
%
```

```
% >> which demo_req_1.abc

% Copyright 1984-2005 The MathWorks, Inc.
% $Revision: 1.1.4.4 $ $Date: 2009/08/04 14:34:12 $

% Create a default (blank) requirement link type
linkType = ReqMgr.LinkType;
linkType.Registration = mfilename;

% Label describing this link type
linkType.Label = 'ABC file (for demonstration)';

% File information
linkType.IsFile = 1;
linkType.Extensions = {'.abc'};

% Location delimiters
linkType.LocDelimiters = '>@';
linkType.Version = ''; % not needed

% Uncomment the functions that are implemented below
linkType.NavigateFcn = @NavigateFcn;
linkType.ContentsFcn = @ContentsFcn;

function NavigateFcn(filename,locationStr)
if ~isempty(locationStr)
    findId=0;
    switch(locationStr(1))
    case '>'
        lineNum = str2num(locationStr(2:end));
        openFileToLine(filename, lineNum);
    case '@'
        openFileToItem(filename,locationStr(2:end));
    otherwise
        openFileToLine(filename, 1);
    end
end
end
```



```

function openFileToLine(fileName, lineNumber)
    if lineNumber > 0
        err = javachk('mwt', 'The MATLAB Editor');
        if isempty(err)
            editor = com.mathworks.mlservices.MLEditorServices;
            editor.openDocumentToLine(fileName, lineNumber);
        end
    else
        edit(fileName);
    end
end

function openFileToItem(fileName, itemName)
    reqStr = ['Requirement:: "' itemName '"'];
    lineNumber = 0;
    fid = fopen(fileName);
    i = 1;
    while lineNumber == 0
        lineStr = fgetl(fid);
        if ~isempty(strfind(lineStr, reqStr))
            lineNumber = i;
        end;
        if ~ischar(lineStr), break, end;
        i = i + 1;
    end;
    fclose(fid);
    openFileToLine(fileName, lineNumber);
end

function [labels, depths, locations] = ContentsFcn(filePath)
    % Read the entire file into a variable
    fid = fopen(filePath, 'r');
    contents = char(fread(fid));
    fclose(fid);

    % Find all the requirement items
    fList1 = regexp(contents, '\nRequirement:: "(.*?)"', 'tokens');

    % Combine and sort the list
    items = [fList1{:}];
end

```

```
items = sort(items);
items = strcat('@',items);

if (~iscell(items) && length(items)>0)
    locations = {items};
    labels = {items};
else
    locations = [items];
    labels = [items];
end

depths = [];
```

---

**Note** To view these files for the built-in link types, see the following files in *matlabroot\toolbox\slvnv\reqmgt\private*:

```
linktype_rmi_doors.m
linktype_rmi_excel.m
linktype_rmi_html.m
linktype_rmi_pdf.m
linktype_rmi_text.m
linktype_rmi_url.m
linktype_rmi_word.m
```

- 
- 2** To register the custom link type ABC, type the following MATLAB command:

```
rmi register rmicustabcinterface
```

The ABC file type appears on the Requirements dialog box drop-down list of document types.

- 3** Create a text file with the *.abc* extension containing several requirement items marked by the `Requirement:: string`. For your convenience, an example file ships with your installation. The example file is

demo\_req\_1.abc and resides in *matlabroot*\toolbox\slvkv\rmidemos.  
demo\_req\_1.abc contains the following content:

```
Requirement:: "Altitude Climb Control"
```

```
Altitude climb control is entered whenever:  
|Actual Altitude- Desired Altitude | > 1500
```

```
Units:
```

```
Actual Altitude - feet
```

```
Desired Altitude - feet
```

```
Description:
```

```
When the autopilot is in altitude climb  
control mode, the controller maintains a  
constant user-selectable target climb rate.
```

```
The user-selectable climb rate is always a  
positive number if the current altitude is  
above the target altitude. The actual target  
climb rate is the negative of the user  
setting.
```

```
<END "Altitude Climb Control">
```

```
Requirement:: "Altitude Hold"
```

```
Altitude hold mode is entered whenever:  
|Actual Altitude- Desired Altitude | <  
30*Sample Period*(Pilot Climb Rate / 60)
```

```
Units:
```

```
Actual Altitude - feet
```

```
Desired Altitude - feet
```

```
Sample Period - seconds
```

```
Pilot Climb Rate - feet/minute
```

Description:

The transition from climb mode to altitude hold is based on a threshold that is proportional to the Pilot Climb Rate.

At higher climb rates the transition occurs sooner to prevent excessive overshoot.

<END "Altitude Hold">

Requirement:: "Autopilot Disable"

Altitude hold control and altitude climb control are disabled when autopilot enable is false.

Description:

Both control modes of the autopilot can be disabled with a pilot setting.

<END "Autopilot Disable">

Requirement:: "Glide Slope Armed"

Glide Slope Control is armed when Glide Slope Enable and Glide Slope Signal are both true.

Units:

Glide Slope Enable - Logical

Glide Slope Signal - Logical

Description:

ILS Glide Slope Control of altitude is only enabled when the pilot has enabled this mode and the Glide Slope Signal is true. This indicates the Glide Slope broadcast signal has been validated by the on board receiver.

<END "Glide Slope Armed">

Requirement:: "Glide Slope Coupled"

Glide Slope control becomes coupled when the control is armed and (Glide Slope Angle Error > 0) and Distance < 10000

Units:

Glide Slope Angle Error - Logical

Distance - feet

Description:

When the autopilot is in altitude climb control mode the controller maintains a constant user selectable target climb rate.

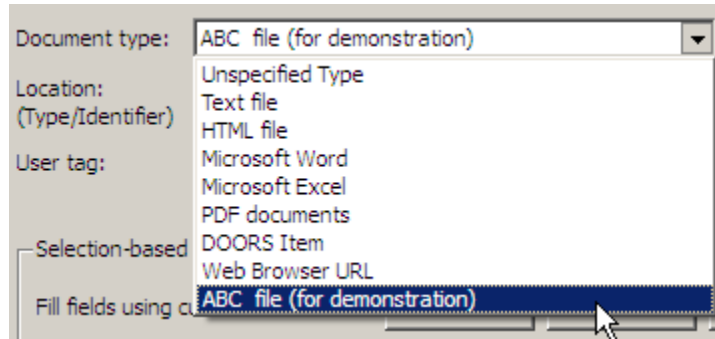
The user-selectable climb rate is always a positive number if the current altitude is above the target altitude the actual target climb rate is the negative of the user setting.

<END "Glide Slope Coupled">

- 4** Open the model `aero_dap3dof`.
- 5** Right-click the Reaction Jet Control subsystem and select **Requirements > Edit/Add Links**.

The Requirements dialog box opens.

- 6 Click **New** to add a new requirement link. The **Document type** drop-down list now contains the ABC file (for demonstration) option.



- 7 Set **Document type** to ABC file (for demonstration) and browse to the demo\_req\_1.abc file. The browser shows only the files with the .abc extension.

- 8 To define a particular location in the requirements document, use the **Location** field.

In this example, the rmicustabcinterface function specifies two types of location delimiters for your requirements:

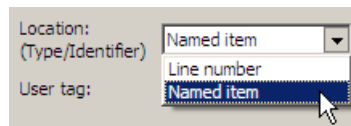
- > — Line number in a file
- @ — Named item, such as a bookmark, function, or HTML anchor

---

**Note** The rmi reference page describes other types of requirements location delimiters.

---

The **Location** drop-down list contains these two types of location delimiters whenever you set **Document type** to ABC file (for demonstration).



- 9** Select **Line number**. Enter the number 26, which corresponds with the line number for the Altitude Hold requirement in `demo_req_1.abc`.
- 10** In the **Description** field, enter Altitude Hold, to identify the requirement by name.
- 11** Click **Apply**.
- 12** Verify that the Altitude Hold requirement links to the Reaction Jet Control subsystem. Right-click the subsystem and select **Requirements > 1. "Altitude Hold"**.

## Creating a Document Index

A *document index* is a list of all the requirements in a given document. To create a document index, MATLAB uses file I/O functions to read the contents of a requirements document into a MATLAB variable. Then the RMI extracts the list of requirement items.

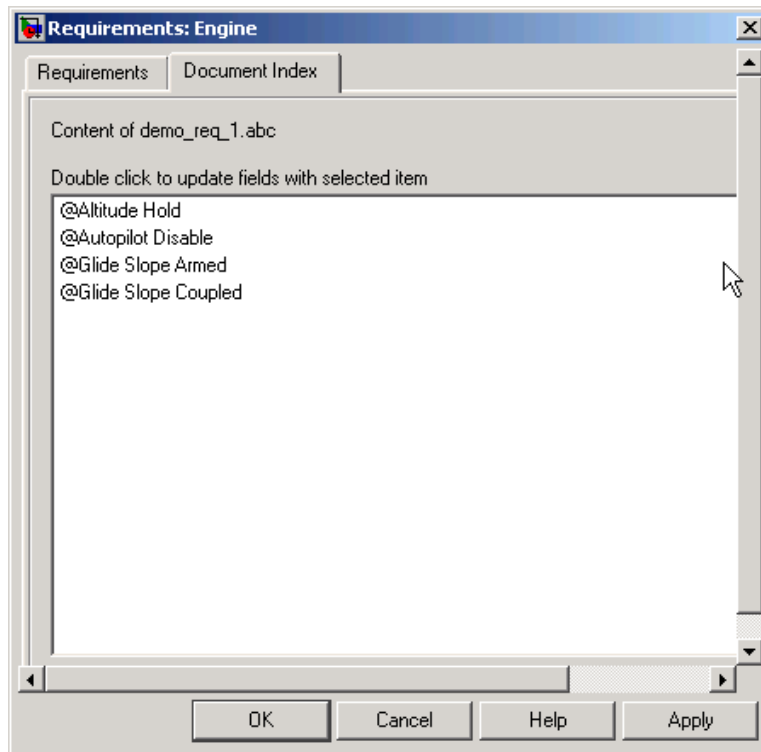
The example requirements document, `demo_req_1.abc`, defines four requirements using the string `Requirement::`. To generate the document index for an ABC file, the `ContentsFcn` function, in the `rmicustabcinterface.m` file, extracts the requirements names and inserts @ before each name.

---

**Note** To see the code for the `ContentsFcn` file, go to step 1 in Chapter 10, “Creating Custom Types of Requirements Documents”.

---

For the `demo_req_1.abc` file, in the **Requirements: Engine** dialog box, click the **Document Index** tab. The `ContentsFcn` function generates the document index automatically.





## Navigating to Simulink Objects from External Documents

The RMI includes several functions that simplify creating navigation interfaces in external documents. The external application that displays your document must support an application programming interface (API) for communicating with the MATLAB software.

### Providing Unique Object Identifiers

Whenever you create a requirement link for a Simulink or Stateflow object, the RMI creates a globally unique identifier for that object. This identifier identifies the object. The identifier does not change if you rename or move the object, or add or delete requirement links. The RMI uses the unique identifier only to resolve an object within a model. The identifier is globally unique and does not collide with identifiers in other models, unless the two models derive from the same original model. Unique object identifiers have formats such as GIDa\_cd14afcd\_7640\_4ff8\_9ca6\_14904bdf2f0f.

### Using the `rmiobjnavigate` Function

The `rmiobjnavigate` function identifies the appropriate Simulink or Stateflow object, highlights that object, and brings the appropriate editor window to the front of the screen. When you navigate to a Simulink model from an external application, invoke this function. Internally, this function creates a table of all the unique object identifiers within a model for efficient object lookup.

The first time you navigate to an item in a particular model, you might experience a slight delay while the software constructs the internal navigation table. You do not experience a long delay on subsequent navigation.

### Determining the Navigation Command

Once you create a requirement link for a Simulink or Stateflow object, at the MATLAB prompt, use the `rmi` function to find the appropriate navigation command string. The return value of the `navCmd` method is a string that navigates to the correct object when evaluated by the MATLAB software:

```
cmdString = rmi('navCmd', block);
```

Send this exact string to the MATLAB software for evaluation as part of navigating from the external application to the Simulink model.

## Using the ActiveX Navigation Control

The RMI uses software that includes a special Microsoft® ActiveX® control to enable navigation to Simulink objects from Microsoft Word and Excel® documents. You can use this same control in any other application that supports ActiveX within its documents.

The control is derived from a push button and has the Simulink icon. There are two instance properties that define how the control works. The `tooltipstring` property is the string that is displayed in the control ToolTip. The `MLEvalCmd` property is the string that you pass to the MATLAB software for evaluation when you click the control.

## Typical Code Sequence for Establishing Navigation Controls

When you create an interface to an external tool, you can automate the procedure for establishing links. This way, you do not need to manually update the dialog box fields. This type of automation occurs as part of the selection-based linking for certain built-in types, such as Microsoft Word and Excel documents.

To automate the procedure for establishing links:

- 1** Select a Simulink or Stateflow object and an item in the external document.
- 2** Invoke the link creation action either from a Simulink menu or command, or a similar mechanism in the external application.
- 3** Identify the document and current item using the scripting capability of the external tool. Pass this information to the MATLAB software. Create a requirement link on the selected object using `rmi('createempty')` and `rmi('cat')`.
- 4** Determine the MATLAB navigation command string that you must embed in the external tool, using the `navCmd` method:

```
cmdString = rmi('navCmd',obj)
```

- 5** Create a navigation item in the external document using the scripting capability of the external tool. Set the MATLAB navigation command string in the appropriate property.

For example, you can use the code for selection-based linking to the Microsoft Word, Microsoft Excel, and IBM Rational DOORS software. The files are contained in *matlabroot*\toolbox\slvnx\reqmgt\private:

```
selection_link_doors.m  
selection_link_excel.m  
selection_link_word.m
```



# Including Requirements Information with Generated Code

---

After you simulate your model and verify its performance against the requirements, you can generate code from the model for an embedded real-time application with links to the requirements.

The Real-Time Workshop® Embedded Coder™ software generates code for Embedded Real-Time (ERT) targets. If the model has any links to requirements, the software also inserts hyperlinks to any requirements links in the code comments.

For example, if a block has a requirement, the Real-Time Workshop Embedded Coder software generates code for that block. In the code comments for that block, the software inserts a hyperlink to the requirements document that contains the requirement associated with that block.

---

**Note** You must have a license for Real-Time Workshop Embedded Coder to generate code for an embedded real-time application.

---

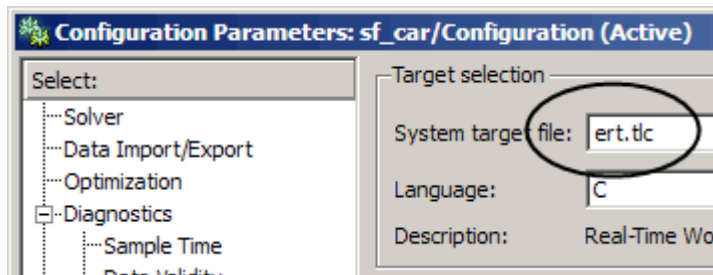
Generated code includes requirements descriptions and hyperlinks to the requirements documents in the following locations.

Model Object	Requirements Location
Model	In the main header file, <model>.h
Nonvirtual subsystems	At the call site for the subsystem
Virtual subsystems	At the call site of the closest nonvirtual parent subsystem. If a virtual subsystem does not have a nonvirtual parent, requirement descriptions appear in the main header file for the model, <model>.h.
Nonsubsystem blocks	In the generated code for the block

To specify that generated code of an ERT target include requirements:

- 1 Open the `rtwdemo_requirements` demo model.
- 2 Select **Simulation > Configuration Parameters**.
- 3 In the **Select** pane of the Configuration Parameters dialog box, select the **Real-Time Workshop** node.

The currently configured system target must be an ERT target.



- 4 Under **Real-Time Workshop**, select **Comments**.
- 5 In the **Custom comments** section on the right, select the **Requirements in block comments** check box.
- 6 Under **Real-Time Workshop**, select **Report**.
- 7 On the **Report** pane, select:

- **Create code generation report**
- **Launch report automatically**

**8** On the **Real-Time Workshop** pane, click **Build**.

**9** In the code-generation report, open `rtwdemo_requirements.c`.

**10** Scroll to the code for the Pulse Generator block, `clock`, for the hyperlink to the requirement linked to that block.

```
rtwdemo_requirements.c /* DiscretePulseGenerator: '<Root>/clock' *  
rt_zcfcn.h             * Block requirements for '<Root>/clock':  
rtwdemo_requirements.h * 1. Clock period shall be consistent with chirp tolerance  
                       */
```

**11** Click the link to open the HTML requirements document.

---

**Note** When you click a requirements link in the code, the software opens the application for the requirements document, except for a DOORS database. To view a DOORS requirement, start the DOORS software and log in before clicking the hyperlink in the code.

---





# Monitoring Signals in Your Model

---

- Chapter 12, “Using Model Verification Blocks”
- Chapter 13, “Using the Verification Manager”
- Chapter 14, “Linking Verification Blocks to Requirements Documents Using the Verification Manager”



# Using Model Verification Blocks

---

- “When to Use Model Verification Blocks” on page 12-2
- “Example: Using the Check Static Lower Bound Block to Check for Out-of-Bounds Signal” on page 12-3

## When to Use Model Verification Blocks

Blocks from the Simulink Model Verification library can monitor individual signals in your model, according to the specifications that you assign to the blocks. Use these blocks with the Verification Manager tool in the Signal Builder to construct simulation tests for your model. With the Signal Builder, you can construct simulation tests in a single location.

You set a verification block to assert when its signal leaves the limit or range that you specify. During simulation, when the signal crosses the limit, the verification block can:

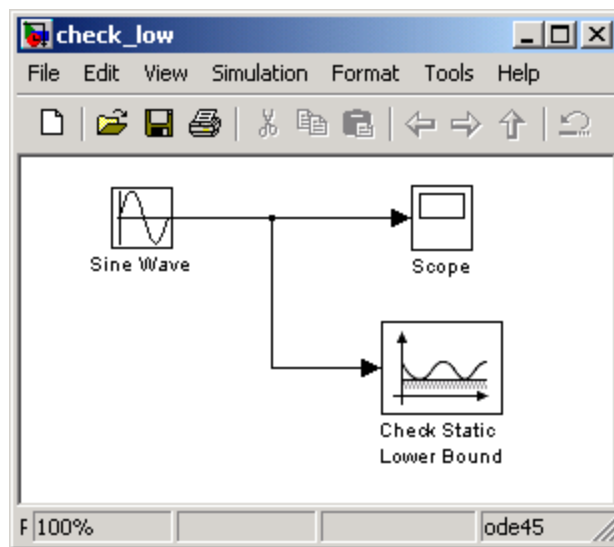
- Stop the simulation and bring immediate focus to that part of the model.
- Report the limit encounter with a logical signal output of its own. If the simulation does not encounter the limit, the signal output is true. If the simulation does encounter the limit, the signal output is false.

To see a complete description of all Model Verification blocks, see the “Model Verification” category in the Simulink Block Reference documentation.

## Example: Using the Check Static Lower Bound Block to Check for Out-of-Bounds Signal

The following example uses a Check Static Lower Bound block to stop simulation when a signal from a Sine Wave block crosses its lower bound limit.

- 1 Attach a Check Static Lower Bound block to the signal from a Sine Wave block.



- 2 Set the Simulation stop time to 2 seconds.
- 3 Double-click the Sine Wave block and set the following parameters:
  - Set the **Amplitude** to 1.
  - Set the **Frequency** to  $\pi$  radians per second.
- 4 Double-click the Check Static Lower Bound block and set the **Lower bound** parameter to -0.8.

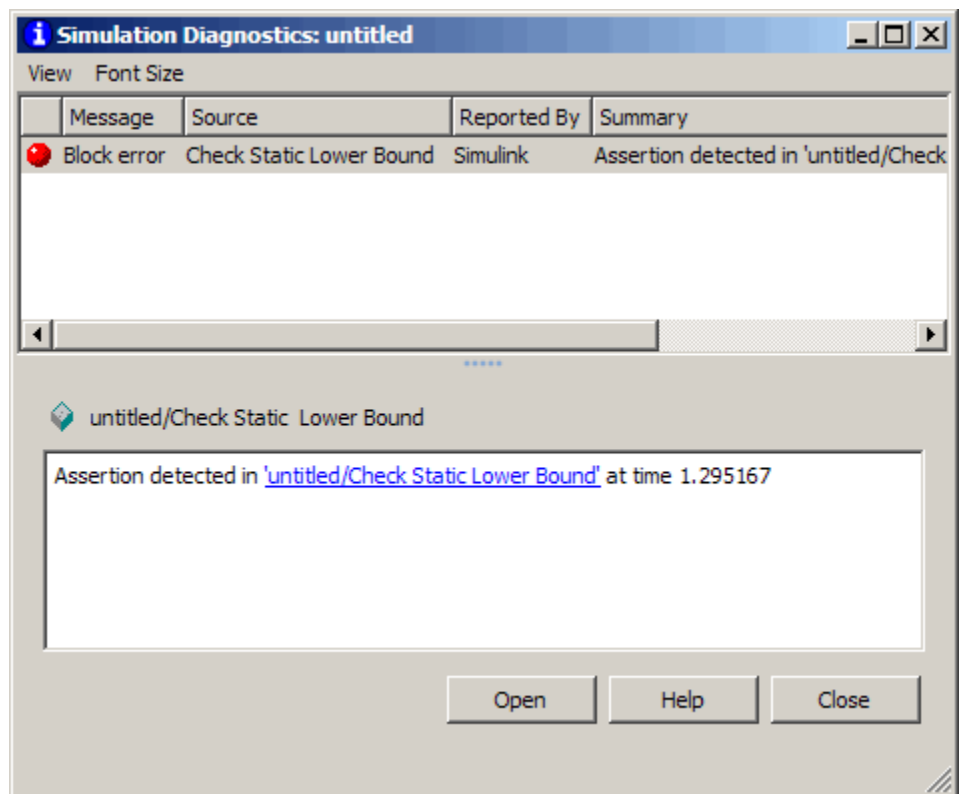
**Enable assertion** is the default. This parameter enables a verification block for an assertion. You set the Check Static Lower Bound block to

detect a signal value of  $-0.8$  or lower. If the signal value reaches that value or falls below it, the simulation stops.

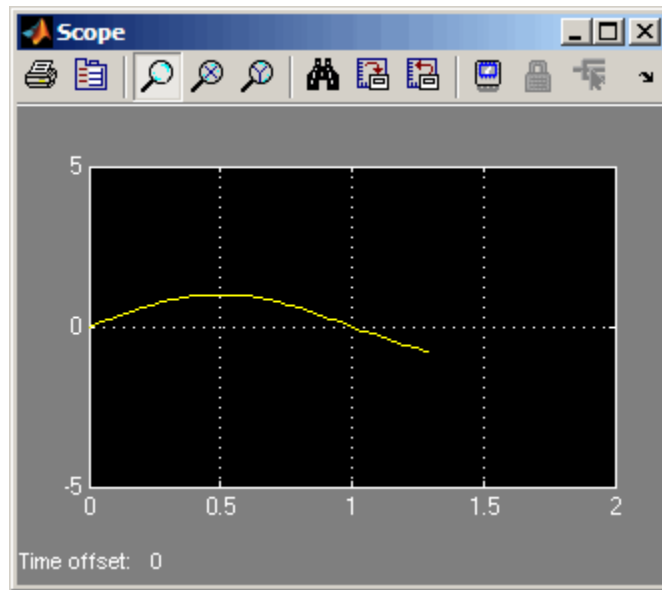
**5** Run the simulation.

The model stops simulating after 1.295 seconds, when the output is  $-0.8$ . The software highlights the Check Static Lower Bound block.

When the simulation stops, you see the following diagnostic message.

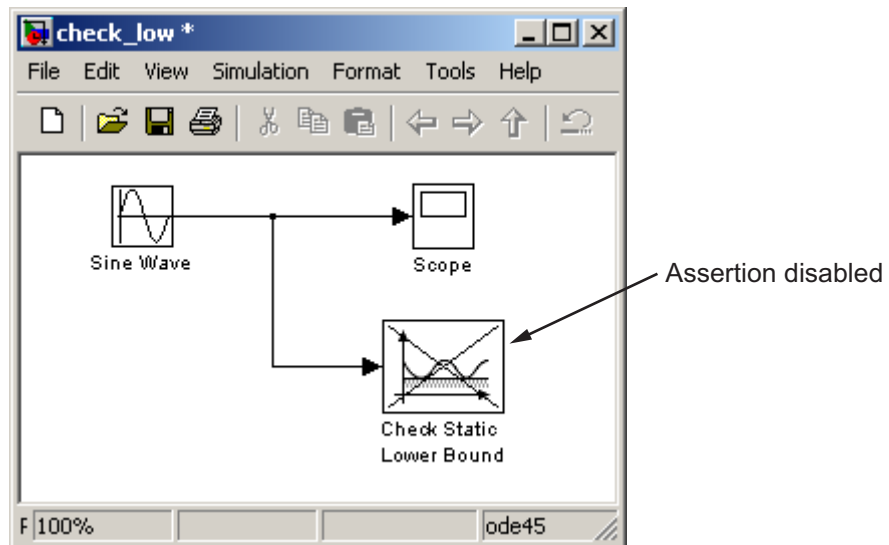


**6** To verify the signal value, double-click the Scope block.



- 7 To disable the Check Static Lower Bound block from asserting its limit, clear the **Enable assertion** check box.

The block is crossed out in the model.





# Using the Verification Manager

---

## Using the Verification Manager

In this section...
“What Is the Verification Manager?” on page 13-2
“Opening the Verification Manager” on page 13-2
“Enabling and Disabling Model Verification Blocks Using the Verification Manager” on page 13-8
“Using Enabling and Disabling Tools in the Verification Manager” on page 13-11

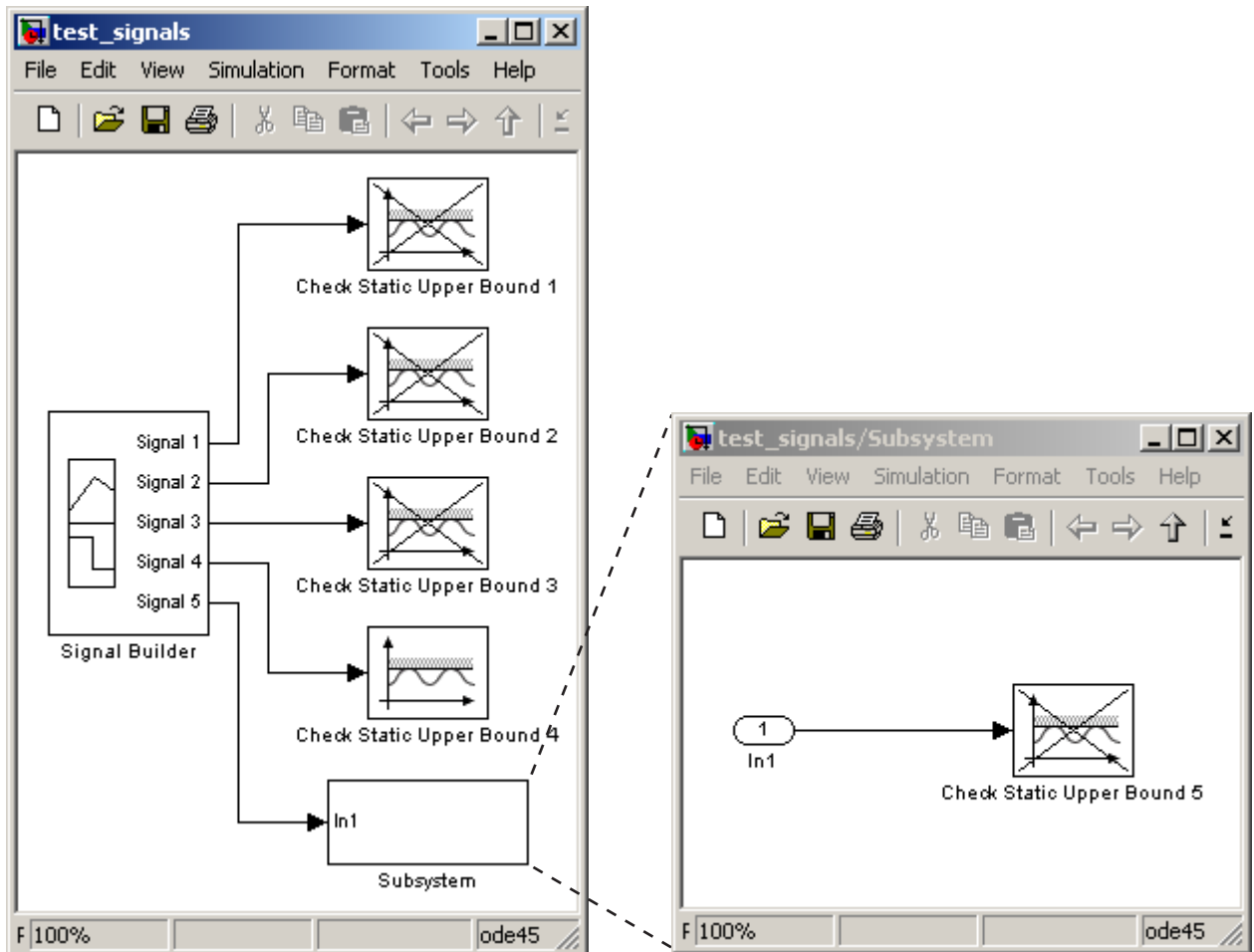
### What Is the Verification Manager?

The Verification Manager is a graphical interface in the Signal Builder dialog box. Using this tool, you can manage all the Model Verification blocks in your model from a central location .

### Opening the Verification Manager

Create a Simulink model that you can use to examine the Verification Manager.

- 1 In the Simulink software, create the following example model .



- a In the Signal Builder block, create a signal group with five signals in the group.
- b Make two copies of the signal group, so that you have three signal groups: **Group 1**, **Group 2**, **Group 3**.

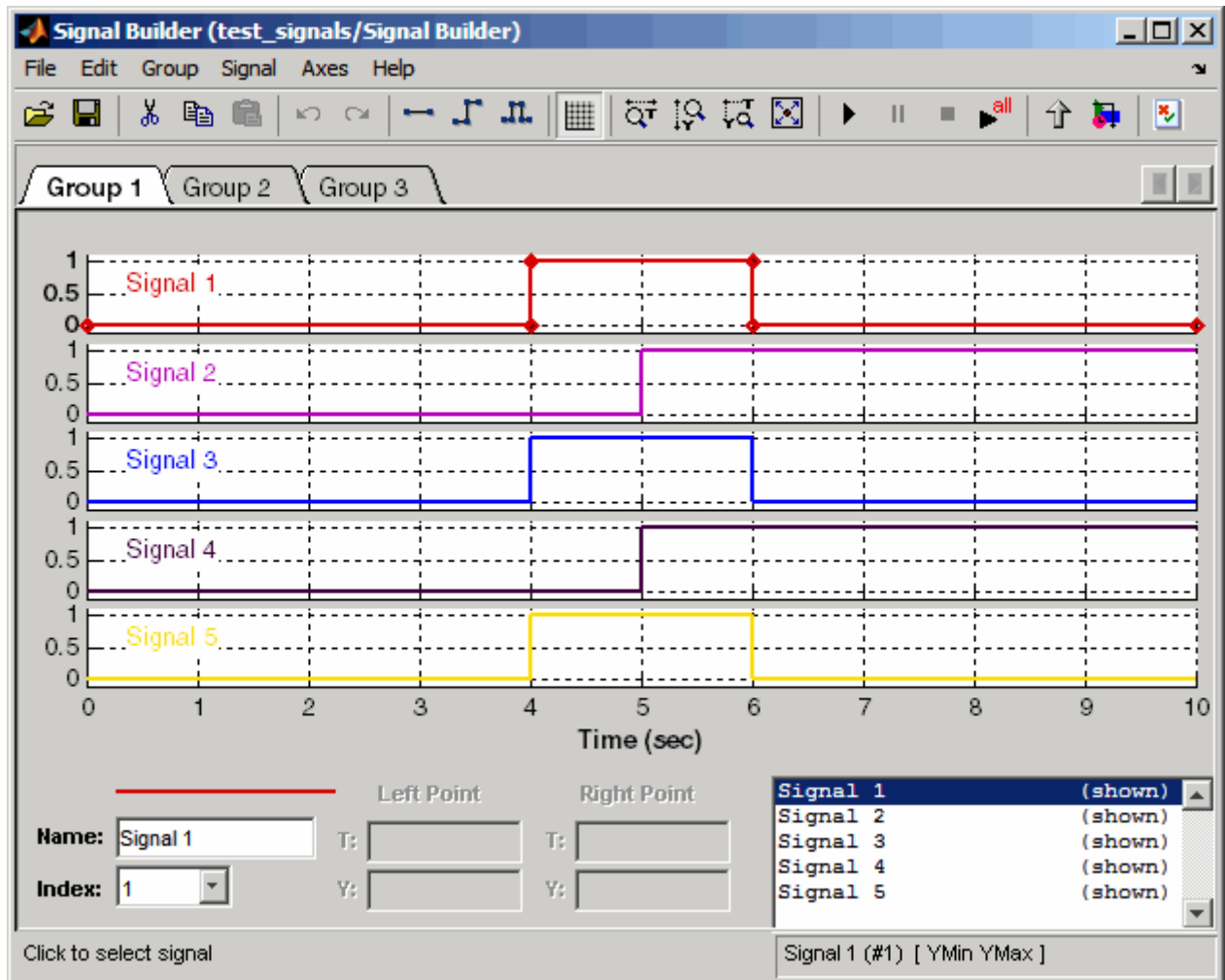
---


**Note** A Signal Builder block provides test signals for an entire model from one location. This model contains a Signal Builder block that feeds five test signals to the Model Verification blocks. The model sends the first four signals directly to Check Static Upper Bound blocks. The model sends the fifth signal to a subsystem that contains a Check Static Upper Bound block.

---

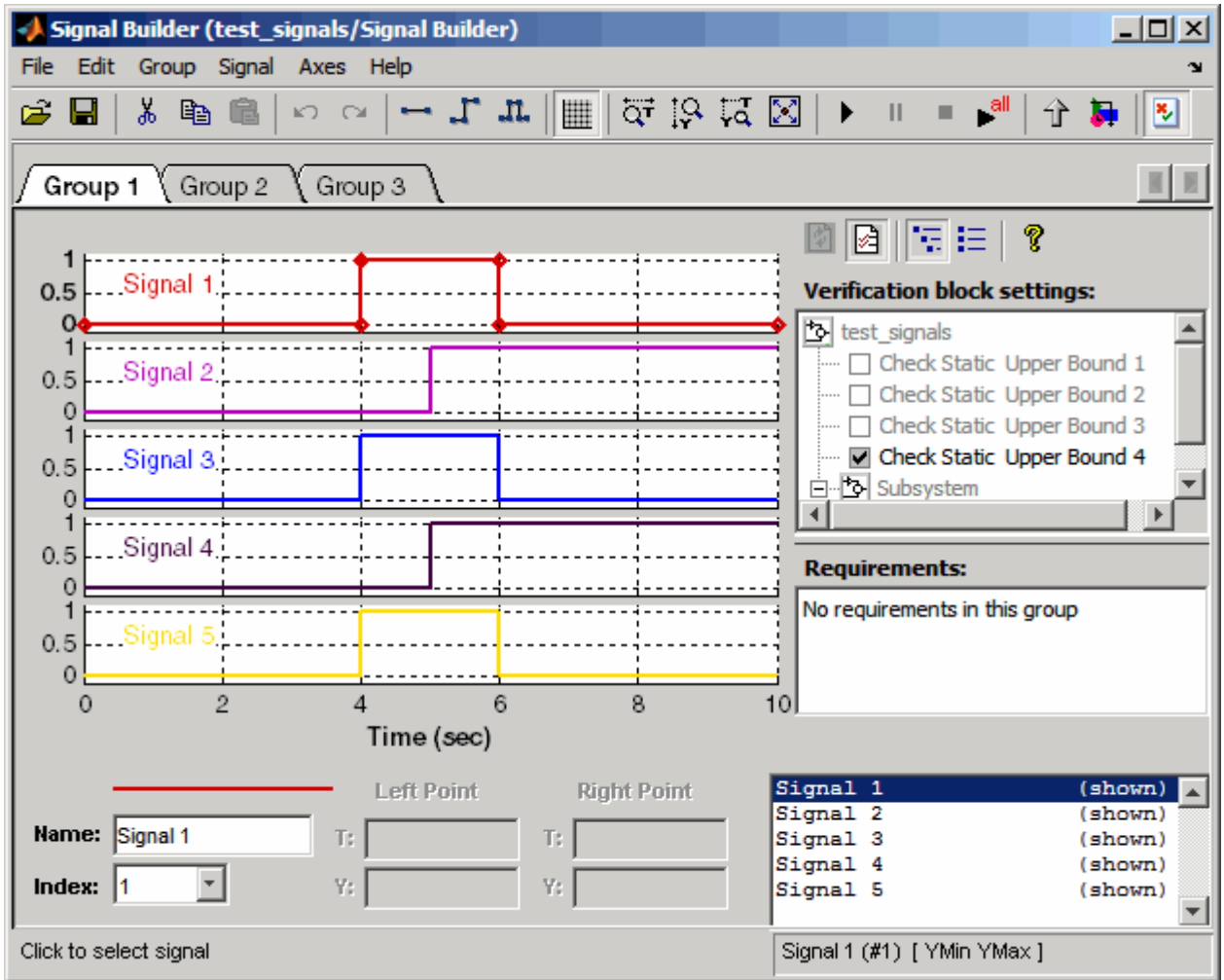
For more information on the Signal Builder block, see “Working with Signal Groups” in the Simulink documentation.

- c** To set each Check Static Upper Bound verification block to assert for an upper bound of 1, set the **Upper bound** parameter to 1.
  - d** For Check Static Upper Bound blocks 1, 2, 3, and 5, disable the assertion by clearing the **Enable assert** parameter. These blocks are crossed out in the model. To enable Check Static Upper Bound block 4, select the **Enable assert** parameter.
- 2** Save this model and name it `test_signals`.
- 3** To open the model Signal Builder dialog box, double-click the Signal Builder block. The **Group 1** signals are displayed default.



- 4 On the Signal Builder dialog box toolbar, select the Show Verification Settings tool. .


The **Verification block settings** pane and the **Requirements** pane are displayed.

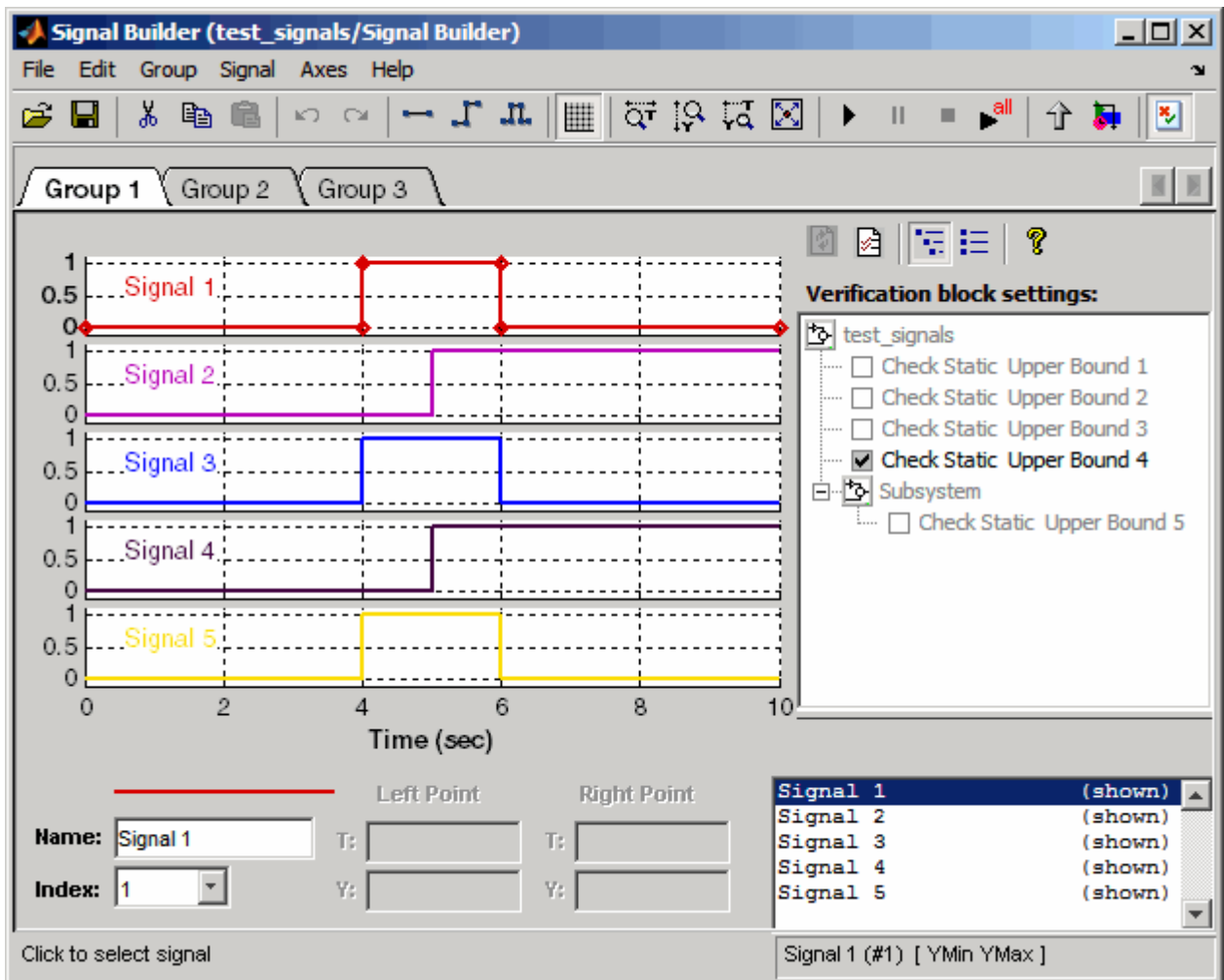


The **Verification block settings** pane lists all Model Verification blocks in the model, grouped by subsystem. In this example, the **Verification block settings** pane displays five Model Verification blocks. Four are in the top level of the model, and one is in a subsystem.

The **Requirements** pane lists the requirements document links for the current signal group. For details on adding requirement document links in

the Signal Builder dialog box, see Chapter 14, “Linking Verification Blocks to Requirements Documents Using the Verification Manager”.

- 5 In this example, delete the **Requirements** pane that is just above the **Verification block settings** pane, and select  to close the **Requirements** pane.



The screenshot shows the Signal Builder dialog box for a test signals project. The main area displays five signals (Signal 1 to Signal 5) plotted against time (0 to 10 seconds). Signal 1 is a red step function that is 0 until 4 seconds, then jumps to 1 until 6 seconds, and then returns to 0. Signal 2 is a purple step function that is 0 until 5 seconds, then jumps to 1 until 10 seconds. Signal 3 is a blue step function that is 0 until 4 seconds, then jumps to 1 until 6 seconds, and then returns to 0. Signal 4 is a purple step function that is 0 until 5 seconds, then jumps to 1 until 10 seconds. Signal 5 is a yellow step function that is 0 until 4 seconds, then jumps to 1 until 6 seconds, and then returns to 0.

The right side of the dialog box shows the **Verification block settings** pane. It contains a tree view with the following items:

- test\_signals
  - Check Static Upper Bound 1
  - Check Static Upper Bound 2
  - Check Static Upper Bound 3
  - Check Static Upper Bound 4
  - Subsystem
    - Check Static Upper Bound 5


Below the plot, there are input fields for the selected signal (Signal 1):

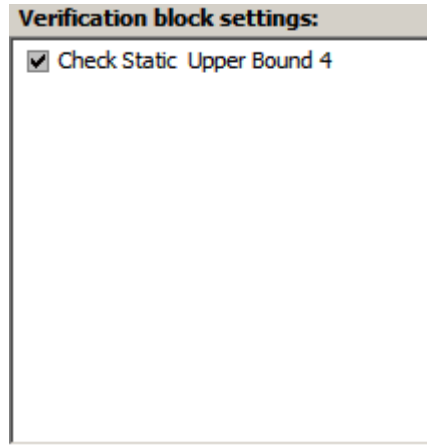
- Name:** Signal 1
- Index:** 1
- Left Point:** T: [ ] Y: [ ]
- Right Point:** T: [ ] Y: [ ]


At the bottom right, there is a list of signals and a status bar:

- Signal 1 (shown)
- Signal 2 (shown)
- Signal 3 (shown)
- Signal 4 (shown)
- Signal 5 (shown)


Signal 1 (#1) [ YMin YMax ]

- To display only the enabled Model Verification blocks for the current signal group, in the **Verification block settings** toolbar, select the List Enabled Verifications tool  .



- To redisplay all Model Verification blocks for the current group, click the Show Verification Block Hierarchy tool .

### Enabling and Disabling Model Verification Blocks Using the Verification Manager

Use the Verification Manager to enable and disable individual Model Verification blocks in signal groups. To open the Verification Manager in the Signal Builder dialog box, click .

The **Verification block settings** pane lists the Model Verification blocks in the model. Each verification block has a status node that indicates whether its assertion is enabled or disabled. Each verification block's status node also indicates whether the enabled or disabled setting applies universally or to the active group. The following table describes the different types of status nodes.

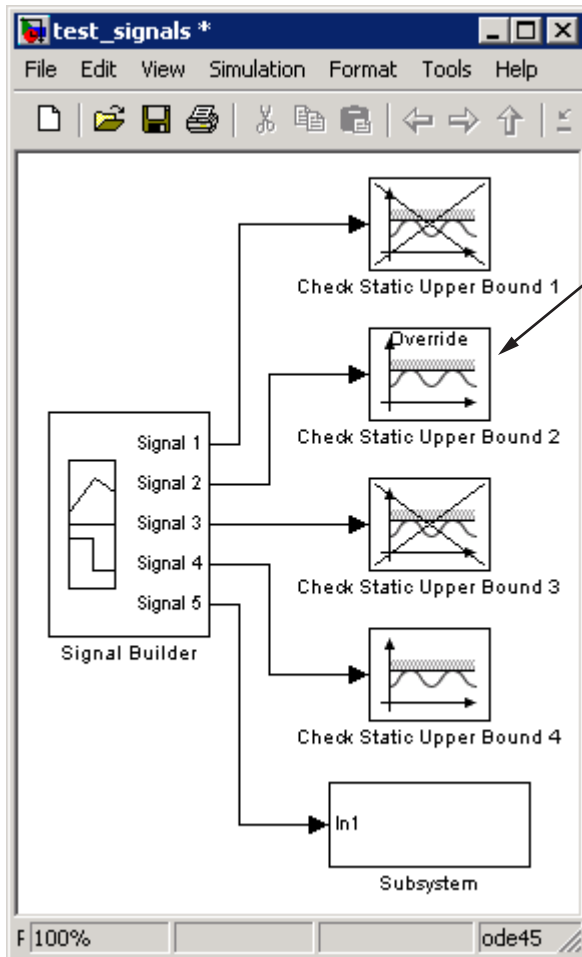


Node	Status
<input type="checkbox"/>	Verification block is disabled for this group. Click to enable for the current group.
<input checked="" type="checkbox"/>	Verification block is enabled for the current group. Click to disable for the current group.
<input checked="" type="checkbox"/>	Verification block is enabled for all test groups.

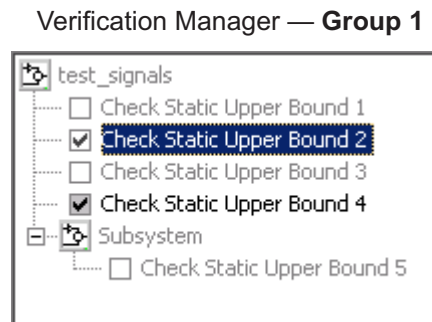
Use the Verification Manager to enable or disable model verification blocks in the `test_signals` model that you created in “Opening the Verification Manager” on page 13-2.

- 1 In the Verification Manager, click the empty check box next to the Check Static Upper Bound 2 node to enable the node for the current group (**Group 1**).

In the **Verification block settings** pane, when you enable a disabled block, you see the following change in how the block is displayed in the model.

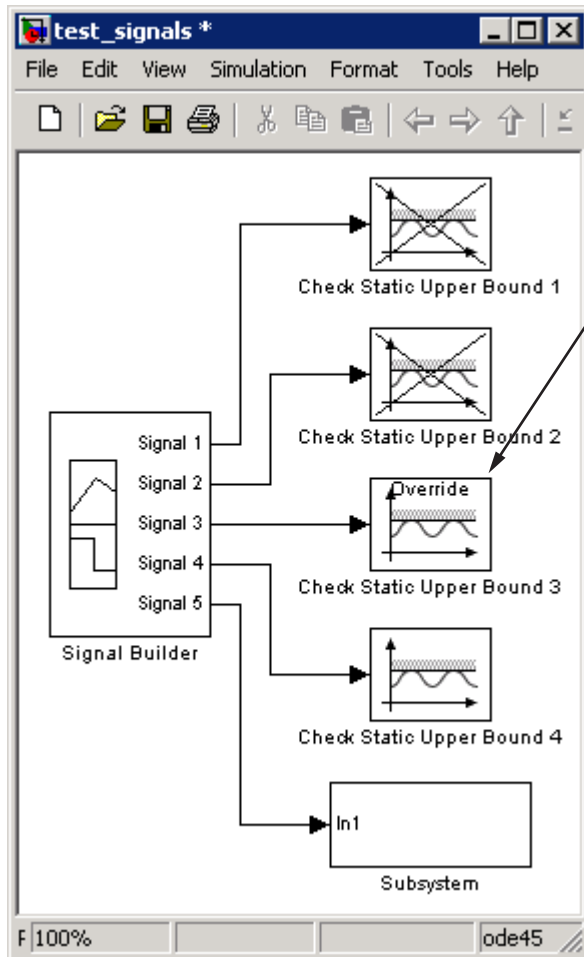


Disabled but enabled in current group (**Group 1**)

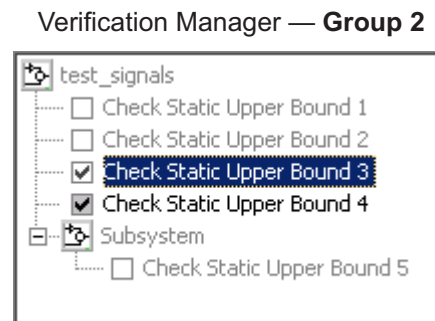


Because you enabled the Check Static Upper Bound 2 block in the current group, an **Override** label is applied to the block and it is no longer crossed out.

- 2 In the Signal Builder, click the **Group 2** tab. Select the empty check box next to the Check Static Upper Bound 3 block to enable it for the current group (**Group 2**).



Disabled but enabled in current group (**Group 2**)




The Check Static Upper Bound 3 block is no longer crossed out, indicating that the block is enabled for the current group. However, Check Static Upper Bound 2 is crossed out because it is enabled in another group.

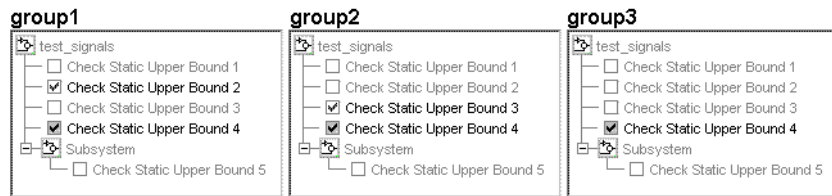
## Using Enabling and Disabling Tools in the Verification Manager

If you have a lot of verification blocks, it is tedious to enable and disable blocks individually. Using the Verification Manager, you can enable and

disable blocks from context menu options. Depending on the status of the node, you have the following options.

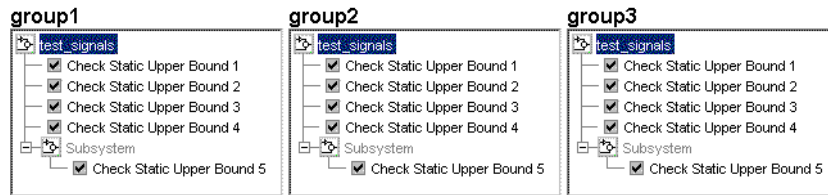
Node Status	Context Menu Options
	<ul style="list-style-type: none"> <li>• Contents enable for all groups</li> <li>• Contents enable by group</li> <li>• Contents group enable</li> <li>• Contents group disable</li> </ul>
<input checked="" type="checkbox"/>	<ul style="list-style-type: none"> <li>• Block enable by group</li> </ul>
<input type="checkbox"/>	<ul style="list-style-type: none"> <li>• Block enable for all groups</li> <li>• Block group enable</li> </ul>
<input checked="" type="checkbox"/>	<ul style="list-style-type: none"> <li>• Block enable for all groups</li> <li>• Block group disable</li> </ul>

For example, assume that you define the following groups in the Verification Manager for a model with five Model Verification blocks.



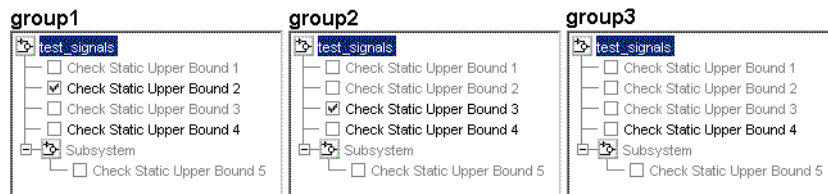
**1** Right-click the `test_signals` node and select **Contents enable for all groups**.

This option enables all verification blocks, for all test groups, in all subsystems.



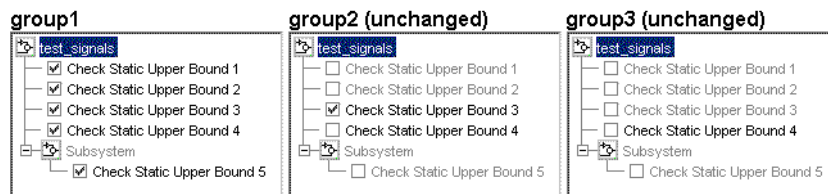
**2** Right-click `test_signals` and select **Contents enable by group**.

This option restores the individually enabled/disabled settings for each verification block in each group.



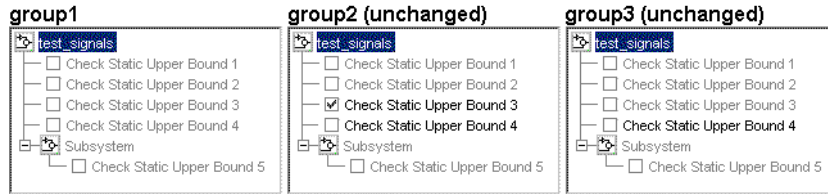
**3** Click the **Group 1** tab, right-click `test_signals`, and select **Contents group enable**.

This option individually enables all contained blocks for only **Group 1**.



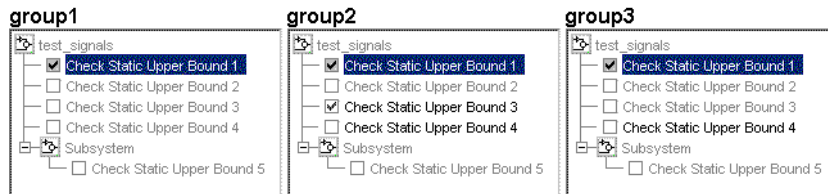
**4** Right-click `test_signals` and select **Contents group disable**.

This option individually disables all contained blocks for only **Group 1**.



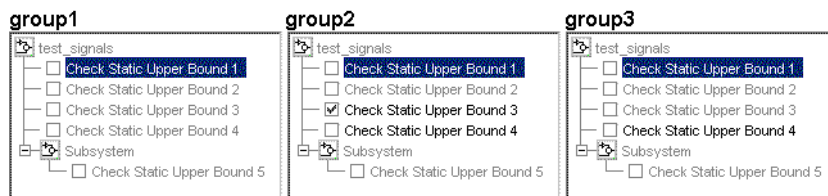
- 5 Right-click Check Static Upper Bound 1 and select **Block enable for all groups**.

This option enables the Check Static Upper Bound 1 block for all groups.



- 6 Right-click Check Static Upper Bound 1 and select **Block enable by group**.

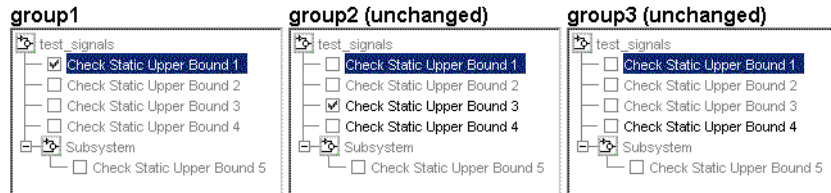
This option restores the individually enabled/disabled state to this block for all groups. The **Block enable by group** option lets you enable or disable this node individually for each group.



- 7 Right-click Check Static Upper Bound 1 and select **Block group enable**.

This option enables the Check Static Upper Bound 1 block for this group only.

Selecting **Block group disable** disables the specified block for this group only.








# Linking Verification Blocks to Requirements Documents Using the Verification Manager

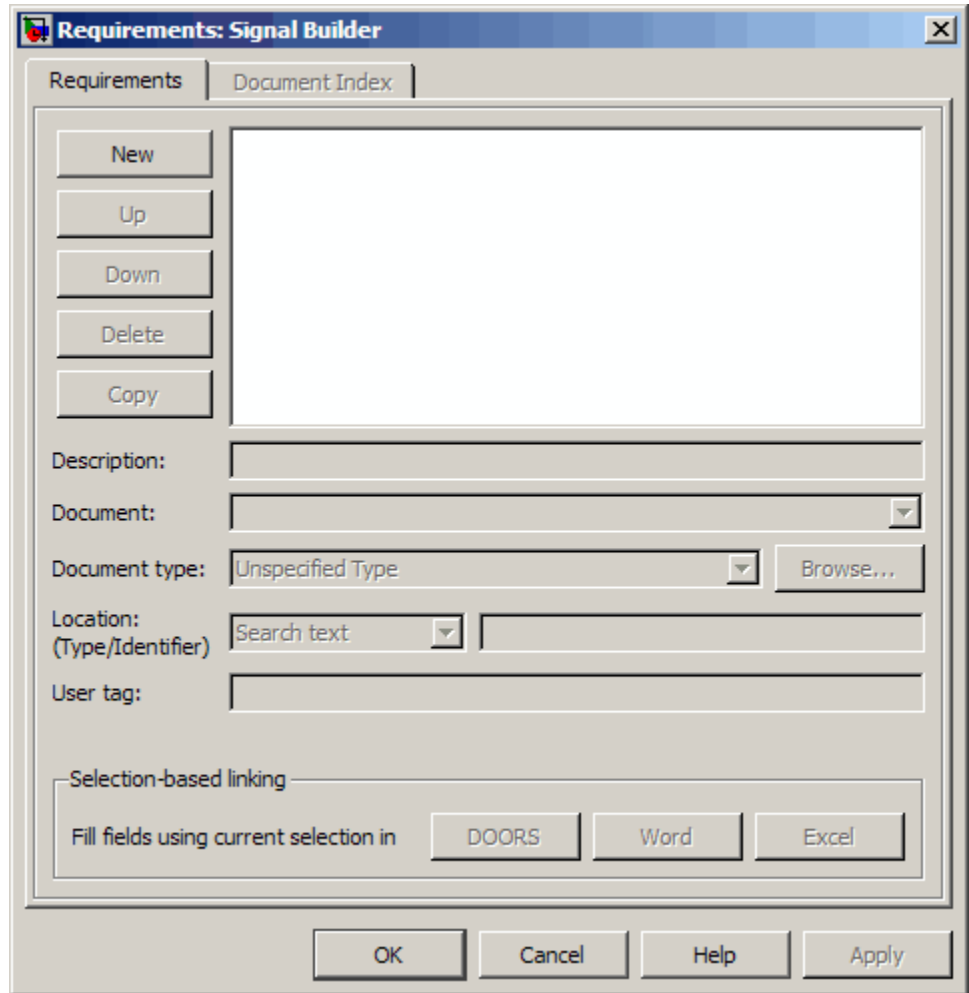
---

You can link requirements documents to individual verification blocks just as you can for any Simulink block.

You can also link requirements documents to test groups and their scheduled Model Verification blocks through the Verification Manager **Requirements** pane in the Signal Builder.

- 1 To display the **Requirements** pane, click .
- 2 In the **Requirements** pane, right-click anywhere.
- 3 From the context menu, select **Edit/Add Links**.

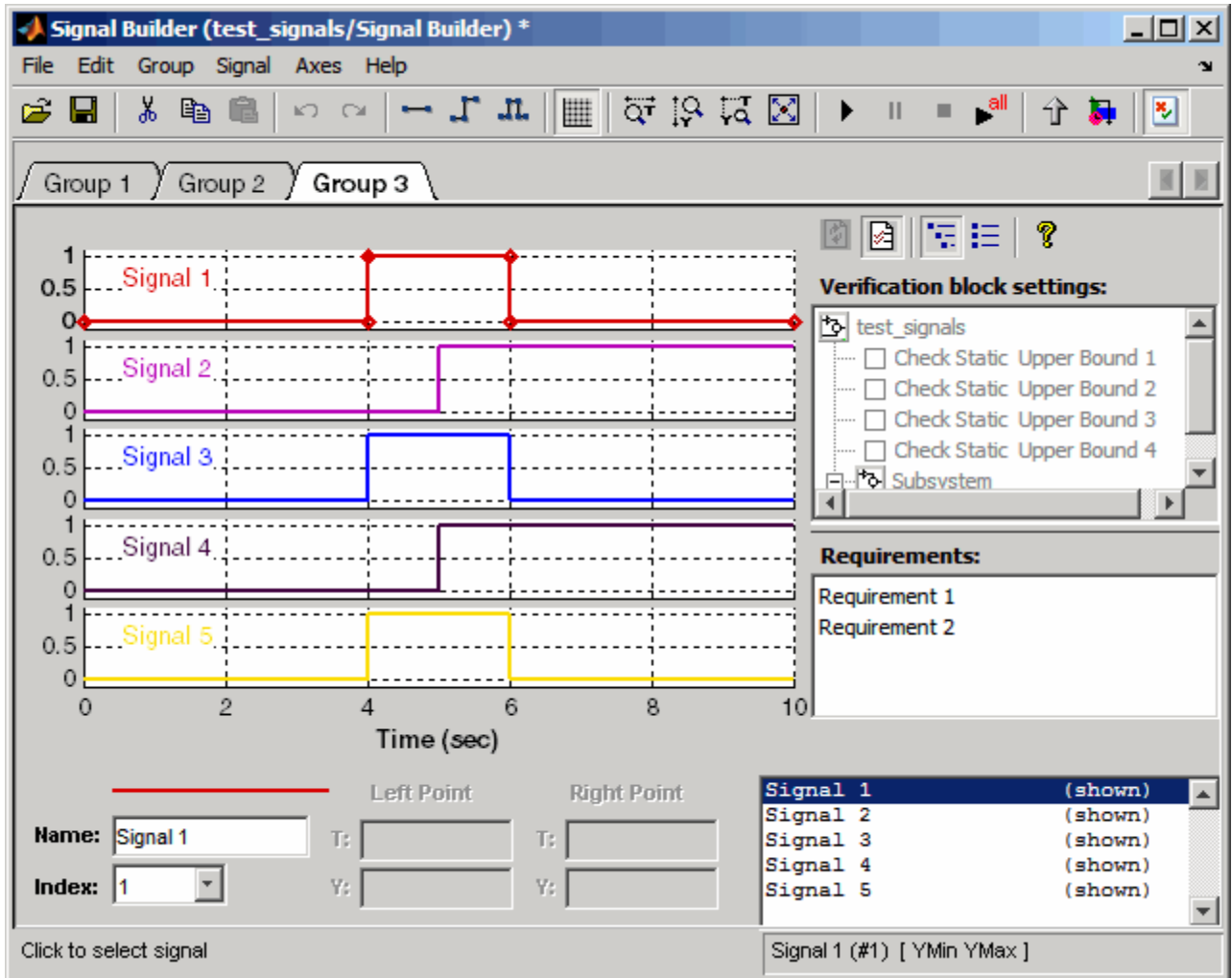
The Requirements dialog box opens.



You can also access the Requirements dialog box for a Signal Builder block by right-clicking the block in the Simulink model and selecting **Requirements > Edit/Add Links**.

- 4 Add links to requirements documents as described in Chapter 4, “Creating and Managing Requirements Links”.

The names of the linked requirements appear in the **Requirements** pane.



- 5 To view the requirements document in its native editor, right-click a requirement link and select **View**.
- 6 Optionally, to delete a requirement link, right-click the link and select **Delete**.



# Validating Your Model with Model Coverage

---

- Chapter 15, “Using Model Coverage”
- Chapter 16, “Setting the Model Coverage Options”
- Chapter 17, “Understanding Model Coverage Reports”
- Chapter 18, “Using Model Coverage Commands”



# Using Model Coverage

---

- “Introduction to Model Coverage” on page 15-2
- “Model Objects That Receive Model Coverage” on page 15-8
- “Simulink Optimizations and Model Coverage” on page 15-30
- “Analyzing Model Coverage” on page 15-32
- “Model Coverage for Embedded MATLAB Function Blocks” on page 15-37
- “Colored Simulink Diagram Coverage Display” on page 15-56

## Introduction to Model Coverage

In this section...
“What Is Model Coverage?” on page 15-2
“How Model Coverage Works” on page 15-2
“Types of Model Coverage” on page 15-3

### What Is Model Coverage?

*Model coverage* helps you validate your model tests by measuring how thoroughly the model objects are tested. Model coverage calculates how much a model test case exercises simulation pathways through a model. Model coverage is a measure of how thoroughly a test case tests a model and the percentage of pathways that a test case exercise. Model coverage helps you validate your model tests.

### How Model Coverage Works

Model coverage analyzes the execution of the following types of model objects that directly or indirectly determine simulation pathways through your model:

- Simulink blocks
- Models referenced in Model blocks
- The states and transitions of Stateflow charts

During a simulation run, the tool records the behavior of the covered objects, states, and transitions. At the end of the simulation, the tool reports the extent to which the run exercised potential simulation pathways through each covered object in the model.

The Simulink Verification and Validation software can only collect coverage for a model if its simulation mode is set to **Normal**. If the simulation mode is set to any mode other than **Normal**, the software cannot measure or report coverage for your model.



For the types of coverage that model coverage performs, see “Types of Model Coverage” on page 15-3. For an example of a model coverage report, see Chapter 17, “Understanding Model Coverage Reports”.

## Types of Model Coverage

Simulink Verification and Validation software can perform several types of coverage analysis:

- “Cyclomatic Complexity” on page 15-3
- “Decision Coverage (DC)” on page 15-4
- “Condition Coverage (CC)” on page 15-4
- “Modified Condition/Decision Coverage (MCDC)” on page 15-4
- “Lookup Table Coverage” on page 15-6
- “Signal Range Coverage” on page 15-6
- “Signal Size Coverage” on page 15-6
- “Simulink Design Verifier Coverage” on page 15-7

## Cyclomatic Complexity

Cyclomatic complexity is a measure of the structural complexity of a model. It approximates the McCabe complexity measure for code generated from the model. The McCabe complexity measure is slightly higher on the generated code due to error checks that the model coverage analysis does not consider.

To compute the cyclomatic complexity of an object (such as a block, chart, or state), model coverage uses the following formula:

$$c = \sum_{1}^{N} (o_n - 1)$$

$N$  is the number of decision points that the object represents and  $o_n$  is the number of outcomes for the  $n$ th decision point. The tool adds 1 to the complexity number for atomic subsystems and Stateflow charts.

For an example of cyclomatic complexity data in a model coverage report, see “Cyclomatic Complexity” on page 17-13.

### **Decision Coverage (DC)**

Decision coverage analyzes elements that represent decision points in a model, such as a Switch block or Stateflow states. For each item, decision coverage determines the percentage of the total number of simulation paths through the item that the simulation actually traversed.

For an example of decision coverage data in a model coverage report, see “Decisions Analyzed” on page 17-15.

### **Condition Coverage (CC)**

Condition coverage analyzes blocks that output the logical combination of their inputs (for example, the Logical Operator block) and Stateflow transitions. A test case achieves full coverage when it causes each input to each instance of a logic block in the model and each condition on a transition to be true at least once during the simulation, and false at least once during the simulation. Condition coverage analysis reports whether the test case fully covered the block for each block in the model.

When you collect coverage for a model, you may not be able to achieve 100% condition coverage. For example, if you specify to short-circuit logic blocks, you might not be able to achieve 100% condition coverage for that block. See “Treat Simulink Logic blocks as short-circuited” on page 16-13 for more information.

For an example of condition coverage data in a model coverage report, see “Conditions Analyzed” on page 17-17.

### **Modified Condition/Decision Coverage (MCDC)**

Modified condition/decision coverage analysis by the Simulink Verification and Validation software extends the decision and condition coverage capabilities. It analyzes blocks that output the logical combination of their inputs and Stateflow transitions to determine the extent to which the test case tests the independence of logical block inputs and transition conditions.

- A test case achieves full coverage for a block when a change in one input, independent of any other inputs, causes a change in the block's output.
- A test case achieves full coverage for a Stateflow transition when there is at least one time when a change in the condition triggers the transition for each condition.

Because the Simulink Verification and Validation MCDC coverage does not guarantee full decision or condition coverage, you can achieve 100% MCDC coverage *without* achieving 100% decision coverage.

Some Simulink objects support MCDC coverage, some objects support only condition coverage, and some objects support only decision coverage. The table in “Model Objects That Receive Model Coverage” on page 15-8 lists which objects receive which types of model coverage. For example, the Combinatorial Logic block can receive decision coverage and condition coverage, but not MCDC coverage.

To achieve 100% MCDC coverage for your model, as defined by the DO-178B standard, in the Coverage Settings dialog box, collect coverage for all of the following coverage metrics:

- Condition Coverage
- Decision Coverage
- MCDC Coverage

When you collect coverage for a model, you may not be able to achieve 100% MCDC coverage. For example, if you specify to short-circuit logic blocks, you may not be able to achieve 100% MCDC coverage for that block.

If you run the test cases independently and accumulate all the coverage results, you can determine if your model adheres to the modified condition and decision coverage standard. For more information about the DO-178B standard, see “DO-178B Checks” on page 27-4.

For an example of MCDC coverage data in a model coverage report, see “MCDC Analysis” on page 17-17. For an example of accumulated coverage results, see “Cumulative Coverage” on page 17-19.

### **Lookup Table Coverage**

Lookup table coverage (LUT) examines blocks, such as the Lookup Table block, that output information from inputs in a table of inputs and outputs, interpolating between or extrapolating from table entries. Lookup table coverage records the frequency that table lookups use each interpolation interval. A test case achieves full coverage when it executes each interpolation and extrapolation interval at least once. For each lookup table block in the model, the coverage report displays a colored map of the lookup table, indicating each interpolation.

For an example of lookup table coverage data in a model coverage report, see “N-Dimensional Lookup Table” on page 17-21.

---

**Note** Configure lookup table coverage only at the start of a simulation. If you tune a parameter that affects lookup table coverage at run time, the coverage settings for the affected block are not updated.

---

### **Signal Range Coverage**

Signal range coverage records the minimum and maximum signal values at each block in the model, as measured during simulation. Only blocks with output signals receive signal range coverage.

For an example of signal range coverage data in a model coverage report, see “Signal Range Analysis” on page 17-30.

### **Signal Size Coverage**

Signal size coverage records the minimum, maximum, and allocated size for all variable-size signals in a model. Only blocks with variable-size output signals are included in the report.

For an example of signal size coverage data in a model coverage report, see “Signal Size Coverage for Variable-Dimension Signals” on page 17-32.

For more information about variable-size signals, see “Working with Variable-Size Signals”.

## Simulink Design Verifier Coverage

The Simulink Verification and Validation software collects model coverage data for the following Simulink® Design Verifier™ blocks and Embedded MATLAB functions:

<b>Simulink Design Verifier blocks</b>	<b>Embedded MATLAB functions</b>
Test Condition	<code>sldv.condition</code>
Test Objective	<code>sldv.test</code>
Proof Assumption	<code>sldv.assume</code>
Proof Objective	<code>sldv.prove</code>

If you do not have a Simulink Design Verifier license, you can collect model coverage for a model that contains these blocks or functions, but you cannot analyze the model using the Simulink Design Verifier software.

By adding one or more Simulink Design Verifier blocks or functions into your model, you can:

- Check the results of a Simulink Design Verifier analysis, run generated test cases, and use the blocks to observe the results.
- Define model requirements using the Test Objective block and verify the results with model coverage data that the software collected during simulation.
- Analyze the model, create a test harness, and simulate the harness with the Test Objective block to collect model coverage data.
- Analyze the model and use the Proof Assumption block to verify any counterexamples that the Simulink Design Verifier identifies.

If you specify to collect Simulink Design Verifier coverage and your model contains a Verification Subsystem block, the software only records coverage for Simulink Design Verifier blocks in the Verification Subsystem block; it does not record coverage for any other blocks in the Verification Subsystem.

For an example of coverage data for Simulink Design Verifier blocks or functions in a model coverage report, see “Simulink® Design Verifier Coverage” on page 17-34.

## Model Objects That Receive Model Coverage

Certain Simulink objects can receive any type of model coverage. Other Simulink objects can receive only certain types of coverage, as the following table shows. “Simulink Object Coverage Criteria” on page 15-10, immediately following the table, provides detailed information about coverage for the model objects listed in the table.

For Stateflow states, events, and state temporal logic decisions, model coverage provides only decision coverage. For Stateflow transitions, model coverage provides decision, condition, and MCDC coverage. For more information, see “Understanding Model Coverage for Stateflow Charts” in the Stateflow documentation.

Model Object	Decision	Condition	MCDC	LUT	Simulink Design Verifier
Abs	•				
Combinatorial Logic	•	•			
Dead Zone	•				
Direct Lookup Table (n-D)				•	
Discrete-Time Integrator (when saturation limits are enabled or reset)	•				
Embedded MATLAB Function	•	•	•		
Enabled Subsystem	•	•	•		
Enabled and Triggered Subsystem	•	•	•		
Fcn (Boolean operators only)		•	•		
For Iterator, For Iterator Subsystem	•				
If, If Action Subsystem	•				
Interpolation Using Prelookup				•	

<b>Model Object</b>	<b>Decision</b>	<b>Condition</b>	<b>MCDC</b>	<b>LUT</b>	<b>Simulink Design Verifier</b>
Logical Operator		•	•		
Lookup Table				•	
Lookup Table (2-D)				•	
Lookup Table (n-D)				•	
MinMax	•				
Model See also Triggered models.	•	•	•	•	•
Multiport Switch	•				
Proof Assumption					•
Proof Objective					•
Rate Limiter	• (Relative to slew rates)				
Relay	•				
Simulink Design Verifier functions in Embedded MATLAB Function blocks					•
Saturation	•				
Stateflow charts	•	•	•		
Switch	•				
Switch Case, Switch Case Action Subsystem	•				
Test Condition					•
Test Objective					•
Triggered models	•	•	•		

Model Object	Decision	Condition	MCDC	LUT	Simulink Design Verifier
Triggered Subsystem	.	.	.		
While Iterator, While Iterator Subsystem	.				

## Simulink Object Coverage Criteria

### Abs

The Abs block receives decision coverage. Decision coverage is based on the input to the block being less than zero and on the data type of the input signal. The decision coverage measures:

- The number of time steps that the block input is less than zero, indicating a true decision.
- The number of time steps the block input is not less than zero, indicating a false decision.

If at least one time step is true and at least one time step is false, decision coverage is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%. The software treats each element of a vector or matrix as a separate coverage measurement.

If the input data type to the Abs block is uint8, uint16, or uint32, the Simulink Verification and Validation software reports no coverage for the block. The software sets the block output equal to the block input without making any decision. If the input data type to the Abs block is Boolean, an error occurs.

### Combinatorial Logic

The Combinatorial Logic block receives decision and condition coverage. Decision coverage is based on achieving each output row of the truth table.



The decision coverage measures the number of time steps that each output row of the truth table is set to the block output.

The condition coverage measures the number of time steps that each input is false (equal to zero) and the number of times each input is true (not equal to zero). If the Combinatorial Logic block has a single input element, the Simulink Verification and Validation software reports only decision coverage, because decision and condition coverage are equivalent.

If all truth table values are set to the block output for at least one time step, decision coverage is 100%. Otherwise, the software reports the coverage as the number of truth table values output during at least one time step, divided by the total number of truth table values. Because this block always has at least one value in the truth table as output, the minimum coverage reported is one divided by the total number of truth table values.

If all block inputs are false for at least one time step and true for at least one time step, condition coverage is 100%. Otherwise, the software reports the coverage as achieving a false value at each input for at least one time step, plus achieving a true value for at least one time step, divided by two raised to the power of the total number of inputs (i.e.,  $2^{\text{number\_of\_inputs}}$ ). The minimum coverage reported is the total number of inputs divided by two raised to the power of the total number of inputs.

## Dead Zone

The Dead Zone block receives decision coverage. The Simulink Verification and Validation software reports decision coverage for the **Start of dead zone** and **End of dead zone** parameters.

The **Start of dead zone** parameter specifies the lower limit of the dead zone. For the **Start of dead zone** parameter, decision coverage measures:

- The number of time steps that the block input is greater than or equal to the lower limit, indicating a true decision.
- The number of time steps that the block input is less than the lower limit, indicating a false decision.

The **End of dead zone** parameter specifies the upper limit of the dead zone. For the **End of dead zone**, decision coverage measures:

- The number of time steps that the block input is greater than the upper limit, indicating a true decision.
- The number of time steps that the block input is less than or equal to the upper limit, indicating a false decision.

When the upper limit is true, the software does not measure **Start of dead zone** coverage for that time step. Therefore, the total number of **Start of dead zone** decisions equals the number of time steps that the **End of dead zone** is false.

If at least one time step is true and at least one time step is false, decision coverage for each of the two individual decisions for the Dead Zone block is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%. The software treats each element of a vector or matrix as a separate coverage measurement.

### **Direct Lookup Table (n-D)**

The Direct Lookup Table (n-D) block receives lookup table coverage. For an  $n$ -D lookup table, the number of output break points is the product of all the number of break points for each table dimension.

Lookup table coverage measures:

- The number of times during simulation that each combination of dimension input values is between each of the break points.
- The number of times during simulation that each combination of dimension input values is below the lowest break point and above the highest break point for each table dimension.

The total number of coverage points for any  $n$ -D lookup table is the product of the number of break points in each table dimension plus one. In the coverage report, an increasing white-to-green color scale, with six evenly spaced data ranges starting with zero, indicates the number of time steps that the software measures each interpolation or extrapolation point.

The software determines a percentage of total coverage by measuring the total interpolation and extrapolation points that achieve a measurement of

at least one time step during simulation between a break point or beyond the end points.

### Discrete-Time Integrator

The Discrete-Time Integrator block receives decision coverage. Decision coverage is based on the **External reset** and **Limit output** parameters. If you set **External reset** to **none**, the Simulink Verification and Validation software does not report decision coverage for the reset decision. Otherwise, the decision coverage measures:

- The number of time steps that the block output is reset, indicating a true decision.
- The number of time steps that the block output is not reset, indicating a false decision.

If you do not select **Limit output**, the software does not report decision coverage for that decision. Otherwise, the software reports decision coverage for the **Lower saturation limit** and the **Upper saturation limit**.

For the **Upper saturation limit**, decision coverage measures:

- The number of time steps that the integration result is greater than or equal to the upper limit, indicating a true decision.
- The number of time steps that the integration result is less than the upper limit, indicating a false decision.

For the **Lower saturation limit**, decision coverage measures

- The number of time steps that the integration result is less than or equal to the lower limit, indicating a true decision.
- The number of time steps that the integration result is greater than the lower limit, indicating a false decision.

For a time step when the upper limit is true, the software does not measure **Lower saturation limit** coverage. Therefore, the total number of lower limit decisions equals the number of time steps that the upper limit is false.

If at least one time step is true and at least one time step is false, decision coverage for each of the three individual decisions (**Limit output**, **Lower saturation limit**, and **Upper saturation limit**) for the block is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%. The software treats each element of a vector or matrix as a separate coverage measurement.

### **Embedded MATLAB Function**

For information about the type of coverage that the Simulink Verification and Validation software reports for the Embedded MATLAB Function block, see “Types of Model Coverage in Embedded MATLAB Function Blocks” on page 15-37.

### **Enabled and Triggered Subsystem**

The Enabled and Triggered Subsystem block receives decision, condition, and MCDC coverage. Decision coverage measures:

- The number of time steps that a trigger edge occurs while the block is enabled, indicating a true decision.
- The number of time steps that a trigger edge does not occur while the block is enabled, or the block is disabled, indicating a false decision.

If at least one time step is true and at least one time step is false, decision coverage is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%.

The software measures condition coverage for the enable input and for the trigger input separately:

- For the enable input, condition coverage measures the number of time steps the enable input is true and the number of time steps the enable input is false.
- For the trigger input, condition coverage measures the number of time steps the trigger edge occurs, indicating true, and the number of time steps the trigger edge does not occur, indicating false.

The software reports condition coverage based on the total number of possible conditions and how many conditions are true for at least one time step and how many are false for at least one time step. The software treats each element of a vector as a separate condition coverage measurement.

The software measures MCDC coverage for the enable input and for the trigger input in combination. Because the enable input of the subsystem is an AND of these two inputs, MCDC coverage is 100% if all of the following occur:

- During at least one time step, both inputs are true.
- During at least one time step, the enable input is true and the trigger edge is false.
- During one time step, the enable input is false and the trigger edge is true.

The software treats each vector element as a separate MCDC coverage measurement. It measures each trigger edge element against each enable input element. However, if the number of elements in both the trigger and enable inputs exceeds 12, the software does not report MCDC coverage.

### **Enabled Subsystem**

The Enabled Subsystem block receives decision, condition, and MCDC coverage.

Decision coverage measures:

- The number of time steps that the block is enabled, indicating a true decision.
- The number of time steps that the block is disabled, indicating a false decision.

If at least one time step is true and at least one time step is false, decision coverage is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%.

The Simulink Verification and Validation software measures condition coverage for the enable input only if the enable input is a vector. For the enable input, condition coverage measures the number of time steps each element of the enable input is true and the number of time steps each element

of the enable input is false. The software reports condition coverage based on the total number of possible conditions and how many are true for at least one time step and how many are false for at least one time step.

The software measures MCDC coverage for the enable input only if the enable input is a vector. Because the enable of the subsystem is an OR of the vector inputs, MCDC coverage is 100% if, during at least one time step, each vector enable input is exclusively true and if, during at least one time step, all vector enable inputs are false. For MCDC coverage measurement, the software treats each element of the vector as a separate condition.

### **Fcn**

The Fcn block receives condition and MCDC coverage. The Simulink Verification and Validation software reports condition or MCDC coverage for Fcn blocks only if the top-level operator is Boolean (&&, | |, or !).

Condition coverage is based on input values or arithmetic expressions that are inputs to Boolean operators in the block. The condition coverage measures:

- The number of time steps that each input to a Boolean operator is true (not equal to zero).
- The number of time steps that each input to a Boolean operator is false (equal to zero).

If all Boolean operator inputs are false for at least one time step and true for at least one time step, condition coverage is 100%. Otherwise, the software reports condition coverage based on the total number of possible conditions and how many are true for at least one time step and how many are false for at least one time step.

The software measures MCDC coverage for Boolean expressions within the Fcn block. If, during at least one time step, each condition independently sets the output of the expression to true and if, during at least one time step, each condition independently sets the output of the expression to false, MCDC coverage is 100%. Otherwise, the software reports MCDC coverage based on the total number of possible conditions and how many times each condition independently sets the output to true during at least one time step and how many conditions independently set the output to false during at least one time step.

### **For Iterator, For Iterator Subsystem**

The For Iterator block and For Iterator Subsystem receive decision coverage. The Simulink Verification and Validation software measures decision coverage for the loop condition value, which is determined by one of the following:

- The iteration value being at or below the iteration limit, indicated as true.
- The iteration value being above the iteration limit, indicated as false.

The software reports the total number of times that each loop condition evaluates to true and to false. If the loop condition evaluates to true at least once and false at least once, decision coverage is 100%. If no loop conditions are true, or if no loop conditions are false, decision coverage is 50%.

### **If, If Action Subsystem**

The If block that is used to execute an If Action Subsystem receives decision coverage. The Simulink Verification and Validation software measures decision coverage for the `if` condition and all `elseif` conditions defined in the If block.

The software does not directly measure the `else` condition, because if there are not `elseif` conditions, the `else` condition is directly the complement of the `if` condition, or the `else` condition is the complement of the last `elseif` condition.

The software reports the total number of time steps that each `if` and `elseif` condition evaluates to true and to false. If the `if` or `elseif` condition evaluates to true at least once, and evaluates to false at least once, decision coverage is 100%. If no `if` or `elseif` conditions are true, or if no `if` or `elseif` conditions are false, decision coverage is 50%. If the previous `if` or `elseif` condition never evaluates as false, an `elseif` condition can have 0% decision coverage.

### **Interpolation Using Prelookup**

The Interpolation Using Prelookup block receives lookup table coverage. For an  $n$ -D lookup table, the number of output break points equals the product of all the number of break points for each table dimension. The lookup table coverage measures:

- The number of times during simulation that each combination of dimension input values is between each of the break points.
- The number of times during simulation that each combination of dimension input values is below the lowest break point and above the highest break point for each table dimension.

The total number of coverage points for any  $n$ -D lookup table is the product of the number of break points in each table dimension plus one. In the coverage report, an increasing white-to-green color scale, with six evenly spaced data ranges starting with zero, indicates the number of time steps that the software measures each interpolation or extrapolation point.

The software determines a percentage of total coverage by measuring the total interpolation and extrapolation points that achieve a measurement of at least one time step during simulation between a break point or beyond the end points.

### **Logical Operator**

The Logical Operator block receives condition and MCDC coverage. The Simulink Verification and Validation software measures condition coverage for each input to the block. The condition coverage measures:

- The number of time steps that each input is true (not equal to zero).
- The number of time steps that each input is false (equal to zero).

If all block inputs are false for at least one time step and true for at least one time step, the software condition coverage is 100%. Otherwise, the software reports the condition coverage based on the total number of possible conditions and how many are true at least one time step and how many are false at least one time step.

The software measures MCDC coverage for all inputs to the block. If, during at least one time step, each condition independently sets the output of the block to true and if, during at least one time step, each condition independently sets the output of the block to false, MCDC coverage is 100%. Otherwise, the software reports the MCDC coverage based on the total number of possible conditions and how many times each one of them



independently set the output to true for at least one time step and how many independently set the output to false for at least one time step.

### **Lookup Table**

The Lookup Table block receives lookup table coverage. For a 1-D lookup table, the number of input and output break points is equal. Lookup table coverage measures:

- The number of times during simulation that the input and output values are between each of the break points.
- The number of times during simulation that the input and output values are below the lowest break point and above the highest break point.

The total number of coverage points for any 1-D lookup table is the number of break points in the table plus one. In the coverage report, an increasing white-to-green color scale, with six evenly spaced data ranges starting with zero, indicates the number of time steps that the software measures each interpolation or extrapolation point.

The software determines a percentage of total coverage by measuring the total interpolation and extrapolation points that achieve a measurement of at least one time step during simulation between a break point or beyond the end points.

### **Lookup Table (2-D)**

The Lookup Table (2-D) block receives lookup table coverage. For a 2-D lookup table, the number of output break points equals the number of row break points multiplied by the number of column inputs. Lookup table coverage measures:

- The number of times during simulation that each combination of row input and column input values is between each of the break points.
- The number of times during simulation that each combination of row input and column input values is below the lowest break point and above the highest break point for each row and column.

The total number of coverage points for any 2-D lookup table is the number of row break points in the table plus one, multiplied by the number of column break points in the table plus one. In the coverage report, an increasing white-to-green color scale, with six evenly spaced data ranges starting with zero, indicates the number of time steps that the software measures each interpolation or extrapolation point.

### **Lookup Table (n-D)**

The Lookup Table (n-D) block receives lookup table coverage. For an  $n$ -D lookup table, the number of output break points equals the product of all the number of break points for each table dimension. Lookup table coverage measures:

- The number of times during simulation that each combination of dimension input values is between each of the break points.
- The number of times during simulation that each combination of dimension output values is below the lowest break point and above the highest break point for each table dimension.

The total number of coverage points for any n-D lookup table is the product of the number of break points in each table dimension plus one. In the coverage report, an increasing white-to-green color scale, with six evenly spaced data ranges starting with zero, indicates the number of time steps that the software measures each interpolation or extrapolation point.

The software determines a percentage of total coverage by measuring the total interpolation and extrapolation points that achieve a measurement of at least one time step during simulation between a break point or beyond the end points.

### **MinMax**

The MinMax block receives decision coverage. Decision coverage is based on passing each input to the output of the block. The decision coverage measures the number of time steps that the simulation passes each input to the block output. The number of decision points is based on the number of inputs to the block and whether they are scalar, vector, or matrix.

If all inputs are passed to the block output for at least one time step, the Simulink Verification and Validation software reports the decision coverage as 100%. Otherwise, the software reports the coverage as the number of inputs passed to the output during at least one time step, divided by the total number of inputs.

## **Model**

The Model block does not receive coverage directly; the model the block references receives coverage. If the referenced model is in Normal mode, the Simulink Verification and Validation software reports coverage for all objects within the referenced model that receive coverage. If the simulation mode is set to any value other than **Normal**, the software cannot measure or report coverage for the referenced model.

In the Coverage Settings dialog box, select which referenced models that you want to report coverage for. The software generates separate coverage reports for all referenced models that you select. The software links those coverage reports to a summary report for the parent model. Any referenced models that you do not select, even if they are in Normal mode, do not generate a coverage report. For details on how to select referenced models to report coverage, see “Creating and Running Test Cases” on page 15-32.

## **Multiport Switch**

The Multiport Switch block receives decision coverage. Decision coverage is based on passing each input, excluding the first control input, to the output of the block. The decision coverage measures the number of time steps that each input is passed to the block output. The number of decision points is based on the number of inputs to the block and whether the control input is scalar or vector.

If all inputs, excluding the first control input, are passed to the block output for at least one time step, decision coverage is 100%. Otherwise, the Simulink Verification and Validation software reports coverage as the number of inputs passed to the output during at least one time step, divided by the total number of inputs minus one.

### **Proof Assumption**

The Proof Assumption block receives Simulink Design Verifier coverage. Simulink Design Verifier coverage is based on the points and intervals defined in the block dialog box. Simulink Design Verifier coverage measures the number of time steps that each point or interval defined in the block is satisfied. The total number of objective outcomes is based on the number of points or intervals defined in the Proof Assumption block.

If all points and intervals defined in the block are satisfied for at least one time step, Simulink Design Verifier coverage is 100%. Otherwise, the Simulink Verification and Validation software reports coverage as the number of points and intervals satisfied during at least one time step, divided by the total number of points and intervals defined for the block.

### **Proof Objective**

The Proof Objective block receives Simulink Design Verifier coverage. Simulink Design Verifier coverage is based on the points and intervals defined in the block dialog box. Simulink Design Verifier coverage measures the number of time steps that each point or interval defined in the block is satisfied. The total number of objective outcomes is based on the number of points or intervals defined in the Proof Objective block.

If all points and intervals defined in the block are satisfied for at least one time step, Simulink Design Verifier coverage is 100%. Otherwise, the Simulink Verification and Validation software reports coverage as the number of points and intervals satisfied during at least one time step, divided by the total number of points and intervals defined for the block.

### **Rate Limiter**

The Rate Limiter block receives decision coverage. The Simulink Verification and Validation software reports decision coverage for the **Rising slew rate** and **Falling slew rate** parameters.

For the **Rising slew rate**, decision coverage measures:

- The number of time steps that the block input changes more than or equal to the rising rate, indicating a true decision.

- The number of time steps that the block input changes less than the rising rate, indicating a false decision.

For the **Falling slew rate**, decision coverage measures:

- The number of time steps that the block input changes less than or equal to the falling rate, indicating a true decision.
- The number of time steps that the block input changes more than the falling rate, indicating a false decision.

The software does not measure **Falling slew rate** coverage for a time step when the **Rising slew rate** is true. Therefore, the total number of **Falling slew rate** decisions equals the number of time steps that the **Rising slew rate** is false.

If at least one time step is true and at least one time step is false, decision coverage for each of the two individual decisions for the block is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%. The software treats each element of a vector or matrix as a separate coverage measurement.

## Relay

The Relay block receives decision coverage. The Simulink Verification and Validation software reports decision coverage for the **Switch on point** and the **Switch off point** parameters.

For the **Switch on point**, decision coverage measures:

- The number of consecutive time steps that the block input is greater than or equal to the **Switch on point**, indicating a true decision.
- The number of consecutive time steps that the block input is less than the **Switch on point**, indicating a false decision.

For the **Switch off point**, decision coverage measures:

- The number of consecutive time steps that the block input is less than or equal to the **Switch off point**, indicating a true decision.

- The number of consecutive time steps that the block input is greater than the **Switch off point**, indicating a false decision.

The software does not measure **Switch off point** coverage for a time step when the switch on threshold is true. Therefore, the total number of **Switch off point** decisions equals the number of time steps that the **Switch on point** is false.

If at least one time step is true and at least one time step is false, decision coverage for each of the two individual decisions for the block is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%. The software treats each element of a vector or matrix as a separate coverage measurement.

### **Saturation**

The Saturation block receives decision coverage. The Simulink Verification and Validation software reports decision coverage for the **Lower limit** and **Upper limit** parameters.

For the **Upper limit**, decision coverage measures:

- The number of time steps that the block input is greater than or equal to the upper limit, indicating a true decision.
- The number of time steps that the block input is less than the upper limit, indicating a false decision.

For the **Lower limit**, decision coverage measures:

- The number of time steps that the block input is greater than the lower limit, indicating a true decision.
- The number of time steps that the block input is less than or equal to the lower limit, indicating a false decision.

The software does not measure **Lower limit** coverage for a time step when the upper limit is true. Therefore, the total number of **Lower limit** decisions equals the number of time steps that the **Upper limit** is false.

If at least one time step is true and at least one time step is false, decision coverage for each of the two individual decisions for the Saturation block is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%. The software treats each element of a vector or matrix as a separate coverage measurement.

## Simulink Design Verifier functions in Embedded MATLAB Function blocks

The following Embedded MATLAB functions receive Simulink Design Verifier coverage:

- `sldv.condition`
- `sldv.test`
- `sldv.assume`
- `sldv.proof`

Each of these functions evaluates an expression *expr*, for example, `sldv.test(expr)`, where *expr* is any valid Boolean Embedded MATLAB expression. Simulink Design Verifier coverage measures the number of time steps that the expression *expr* evaluates to true.

If *expr* is true for at least one time step, Simulink Design Verifier coverage for that function is 100%. Otherwise, the Simulink Verification and Validation software reports coverage for that function as 0%.

## Switch

The Switch block receives decision coverage. Decision coverage is based on the control input to block. Decision coverage measures:

- The number of time steps that the control input evaluates to true.
- The number of time steps the control input evaluates to false.

The number of decision points is based on whether the control input is scalar or vector.

If the control input evaluates to true for at least one time step and evaluates to false for at least one time step, decision coverage is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%. If the control input is a vector, the Simulink Verification and Validation software reports this coverage individually for each vector element.

### **SwitchCase, SwitchCase Action Subsystem**

The SwitchCase block and SwitchCase Action Subsystem receive decision coverage. The Simulink Verification and Validation software measures decision coverage individually for each switch case defined in the block and also for the default case.

The software reports the total number of time steps that each case evaluates to true. If each case, including the default case, evaluates to true at least once, decision coverage is 100%. The software determines the decision coverage by the number of cases that evaluate true for at least one time step divided by the total number of cases.

### **Test Condition**

The Test Condition block receives Simulink Design Verifier coverage. Simulink Design Verifier coverage is based on the points and intervals defined in the block dialog box. Simulink Design Verifier coverage measures the number of time steps that each point or interval defined in the block is satisfied. The total number of objective outcomes is based on the number of points or intervals defined in the Test Condition block.

If all points and intervals defined in the block are satisfied for at least one time step, Simulink Design Verifier coverage is 100%. Otherwise, the Simulink Verification and Validation software reports coverage as the number of points and intervals satisfied during at least one time step, divided by the total number of points and intervals defined for the block.

### **Test Objective**

The Test Objective block receives Simulink Design Verifier coverage. Simulink Design Verifier coverage is based on the points and intervals defined in the block dialog box. Simulink Design Verifier coverage measures the number of time steps that each point or interval defined in the block is



satisfied. The total number of objective outcomes is based on the number of points or intervals defined in the Test Objective block.

If all points and intervals defined in the block are satisfied for at least one time step, Simulink Design Verifier coverage is 100%. Otherwise, the Simulink Verification and Validation software reports coverage as the number of points and intervals satisfied during at least one time step, divided by the total number of points and intervals defined for the block.

### **Triggered models**

A Model block can reference a model that contains edge-based trigger ports at the root level of the model. Triggered models receive decision, condition, and MCDC coverage.

Decision coverage measures:

- The number of time steps that the referenced model is triggered, indicating a true decision.
- The number of time steps that the referenced model is not triggered, indicating a false decision.

If at least one time step is true and at least one time step is false, decision coverage for the Model block that references the triggered model is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%.

Only if the trigger input is a vector, the Simulink Verification and Validation software measures condition coverage for the trigger port in the referenced model. For the trigger port, condition coverage measures:

- The number of time steps that each element of the trigger port is true.
- The number of time steps that each element of the trigger port is false.

The software reports condition coverage based on the total number of possible conditions and how many are true for at least one time step and how many are false for at least one time step.

If the trigger port is a vector, the software measures MCDC coverage for the trigger port only. Because the trigger port of the referenced model is an OR of

the vector inputs, if, during at least one time step, each vector trigger port is exclusively true and if, during at least one time step, all vector trigger port inputs are false, MCDC coverage is 100%. The software treats each element of the vector as a separate condition for MCDC coverage measurement.

### **Triggered Subsystem**

The Triggered Subsystem block receives decision, condition, and MCDC coverage.

Decision coverage measures:

- The number of time steps that the block is triggered, indicating a true decision.
- The number of time steps that the block is not triggered, indicating a false decision.

If at least one time step is true and at least one time step is false, decision coverage is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%.

The Simulink Verification and Validation software measures condition coverage for the trigger input only if the trigger input is a vector. For the trigger input, condition coverage measures:

- The number of time steps that each element of the trigger edge is true.
- The number of time steps that each element of the trigger edge is false.

The software reports condition coverage based on the total number of possible conditions and how many are true for at least one time step and how many are false for at least one time step.

If the trigger input is a vector, the software measures MCDC coverage for the trigger input only. Because the trigger edge of the subsystem is an OR of the vector inputs, if, during at least one time step, each vector trigger edge input is exclusively true and if, during at least one time step, all vector trigger edge inputs are false, MCDC coverage is 100%. The software treats each element of the vector as a separate condition for MCDC coverage measurement.

## **While Iterator, While Iterator Subsystem**

The While Iterator block and While Iterator Subsystem receive decision coverage. Decision coverage is measured for the `while` condition value, which is determined by the `while` condition being satisfied (true), or the `while` condition not being satisfied (false). Simulink Verification and Validation software reports the total number of times that each `while` condition evaluates to true and to false. If the `while` condition evaluates to true at least once, and false at least once, decision coverage for the `while` condition is 100%. If no `while` conditions are true, or if no `while` conditions are false, decision coverage is 50%.

If the iteration limit is exceeded (true) or is not exceeded (false), the software measures decision coverage independently. If the iteration limit evaluates to true at least once, and false at least once, decision coverage for the iteration limit is 100%. If no iteration limits are true, or if no iteration limits are false, decision coverage is 50%. If you set **Maximum number of iterations** to -1 (no limit), the decision coverage for the iteration limit is true for all iterations and false for zero iterations, and decision coverage is 50%.

## Simulink Optimizations and Model Coverage

In the Configuration Parameters dialog box **Optimization** pane, there are two Simulink optimization parameters that can affect your model coverage data:

In this section...
“Block reduction” on page 15-30
“Conditional input branch execution” on page 15-30

### Block reduction

To achieve faster execution during model simulation and in generated code, in the Configuration Parameters dialog box, on the **Optimization** pane, select the **Block reduction** parameter. The Simulink software collapses certain groups of blocks into a single, more efficient block, or removes them entirely.

One of the model coverage options, **Force block reduction off**, allows you to ignore the **Block reduction** parameter when collecting model coverage.

If you do not select the **Block reduction** parameter, or if you select **Force block reduction off**, the Simulink Verification and Validation software provides coverage data for every block in the model that collects coverage.

If you select the **Block reduction** parameter and do not set **Force block reduction off**, the coverage report lists the reduced blocks that would have collected coverage.

### Conditional input branch execution

To improve model execution when the model contains Switch and Multiport Switch blocks, select **Conditional input branch execution**. If you select this parameter, the simulation executes only blocks that are required to compute the control input and the data input selected by the control input.

You can apply this optimization only to certain kinds of Switch blocks, as described in the *Real-Time Workshop User's Guide*.

For example, if your model has a Switch block with output flagged as a test point, such as when a Scope block is attached, that Switch block is not executed, and the model coverage data is incomplete. If you have a model with Switch blocks and you want to ensure that the model coverage data is complete, clear **Conditional input branch execution**.

## Analyzing Model Coverage

In this section...
“Model Coverage Analysis Workflow” on page 15-32
“Creating and Running Test Cases” on page 15-32

### Model Coverage Analysis Workflow

To develop effective tests with model coverage:

- 1 Develop one or more test cases for your model. (See “Creating and Running Test Cases” on page 15-32.)
- 2 Run the test cases to verify that the model behavior is correct.
- 3 Analyze the coverage reports produced by the Simulink Verification and Validation software.
- 4 Using the information in the coverage reports, modify the test cases to increase their coverage or add new test cases to cover areas not currently covered.
- 5 Repeat the preceding steps until you are satisfied with the coverage of your test set.

---

**Note** The Simulink Verification and Validation software comes with an online demonstration of model coverage to validate model tests. To run the demo, at the MATLAB prompt, enter `simcovdemo`.

---

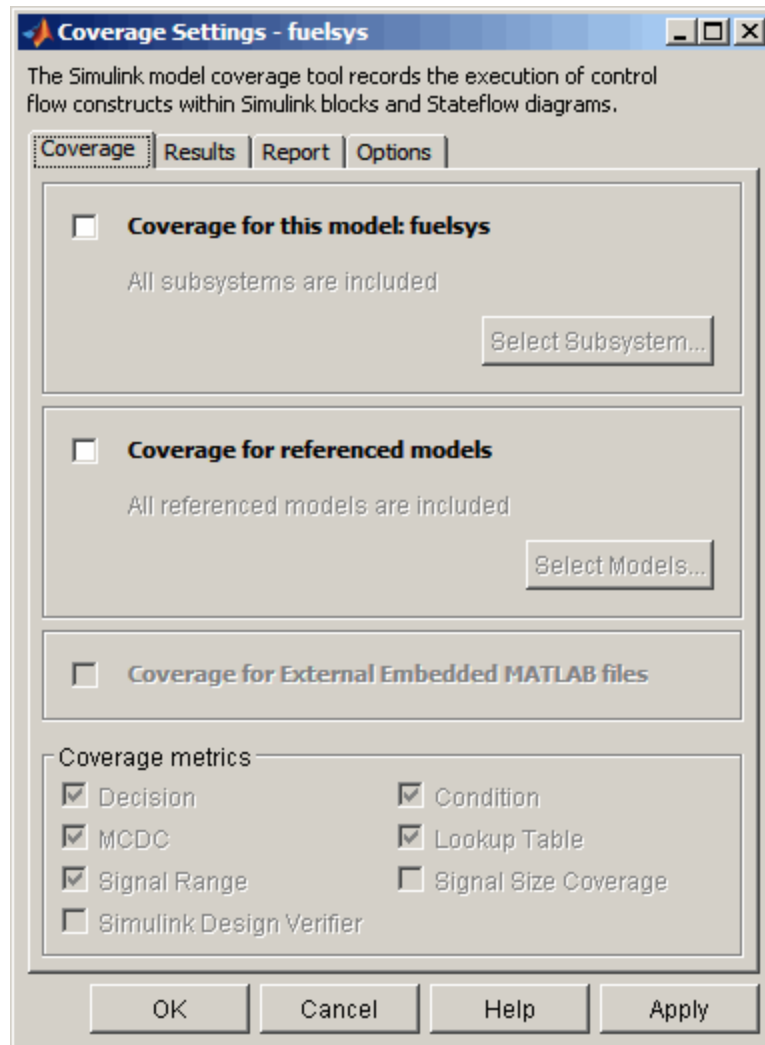
### Creating and Running Test Cases

To create and run test cases, model coverage provides two MATLAB commands, `cvtest` and `cvsim`. The `cvtest` command creates test cases that the `cvsim` command runs. (See “Running Tests with `cvsim`” on page 18-6.)

You can also run the coverage tool interactively:

- 1 Open the fuelsys model.
- 2 In the Simulink model window, select **Tools > Coverage Settings**.

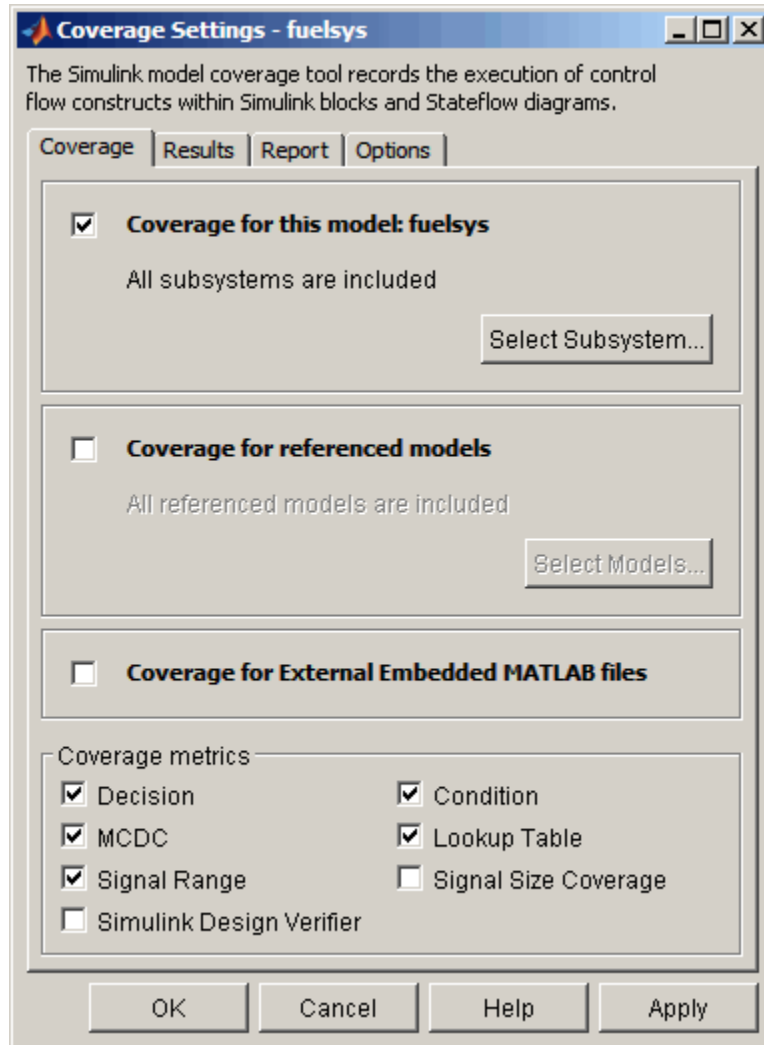
The Coverage Settings dialog box **Coverage** tab appears.



**3** Select **Coverage for this model**, which enables:

- The **Select Subsystem** button
- The **Coverage for External Embedded MATLAB files** option
- The metrics options in the **Coverage metrics** section
- Fields on the other tabs of the Coverage Settings dialog box





- 4 Under **Coverage metrics**, select the types of coverage that you want to record in the coverage report.

For a complete description of all coverage options in the Coverage Settings dialog box, see Chapter 16, “Setting the Model Coverage Options”.

- 5 Click **OK**.
- 6 In the Simulink model window, select **Start > Simulation** or on the Simulink toolbar, click the **Start** button to start simulating the model.

If you specify to report model coverage, the Simulink Verification and Validation software saves coverage data for the current run in the workspace object `covdata` and cumulative coverage data in `covCumulativeData`, by default. At the end of the simulation, this data appears in an HTML report that opens in a browser window.

---

**Note** You cannot run simulations if you select both model coverage reporting and acceleration options. The Simulink Verification and Validation software clears the model coverage reporting option if you select acceleration mode.

You cannot select both block reduction and conditional branch input optimization when you perform coverage analysis because they interfere with coverage recording.

---

## Model Coverage for Embedded MATLAB Function Blocks

### In this section...

“Types of Model Coverage in Embedded MATLAB Function Blocks” on page 15-37

“Creating a Model with Embedded MATLAB Function Block Decisions” on page 15-39

“Understanding Embedded MATLAB Function Block Model Coverage” on page 15-43

### Types of Model Coverage in Embedded MATLAB Function Blocks

- “Decision Coverage” on page 15-37
- “Condition and MCDC Coverage” on page 15-38
- “Simulink® Design Verifier Coverage” on page 15-38

This section describes the model coverage that an Embedded MATLAB Function block receives.

---

**Note** Model coverage is available only if you have a Simulink Verification and Validation software license.

---

#### Decision Coverage

During simulation, the following Embedded MATLAB Function block function statements are tested for decision coverage:

- Function header — Decision coverage is 100% if the function or subfunction is executed.
- `if` — Decision coverage is 100% if the `if` expression evaluates to true at least once, and false at least once.

- `switch` — Decision coverage is 100% if every `switch` case is taken, including the fall-through case.
- `for` — Decision coverage is 100% if the equivalent loop condition evaluates to true at least once, and false at least once.
- `while` — Decision coverage is 100% if the equivalent loop condition evaluates to true at least once, and false at least once.

### **Condition and MCDC Coverage**

During simulation, in the Embedded MATLAB Function block function, the following logical conditions are tested for condition and MCDC coverage:

- `if` statement conditions
- `while` statement conditions, if present

### **Simulink Design Verifier Coverage**

The following Embedded MATLAB functions receive Simulink Design Verifier coverage:

- `sldv.condition`
- `sldv.test`
- `sldv.assume`
- `sldv.proof`

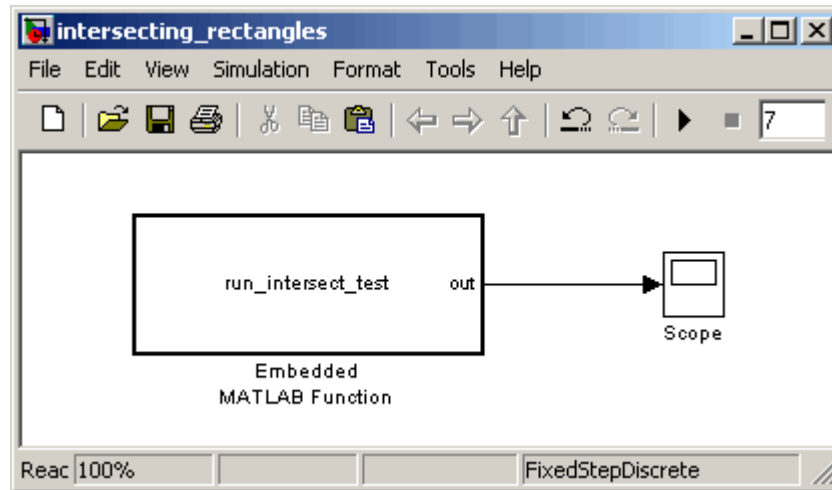
Each of these functions evaluates an expression *expr*, for example, `sldv.test(expr)`, where *expr* is any valid Boolean Embedded MATLAB expression. Simulink Design Verifier coverage measures the number of time steps that the expression *expr* evaluates to true.

If *expr* is true for at least one time step, Simulink Design Verifier coverage for that function is 100%. Otherwise, the Simulink Verification and Validation software reports coverage for that function as 0%.

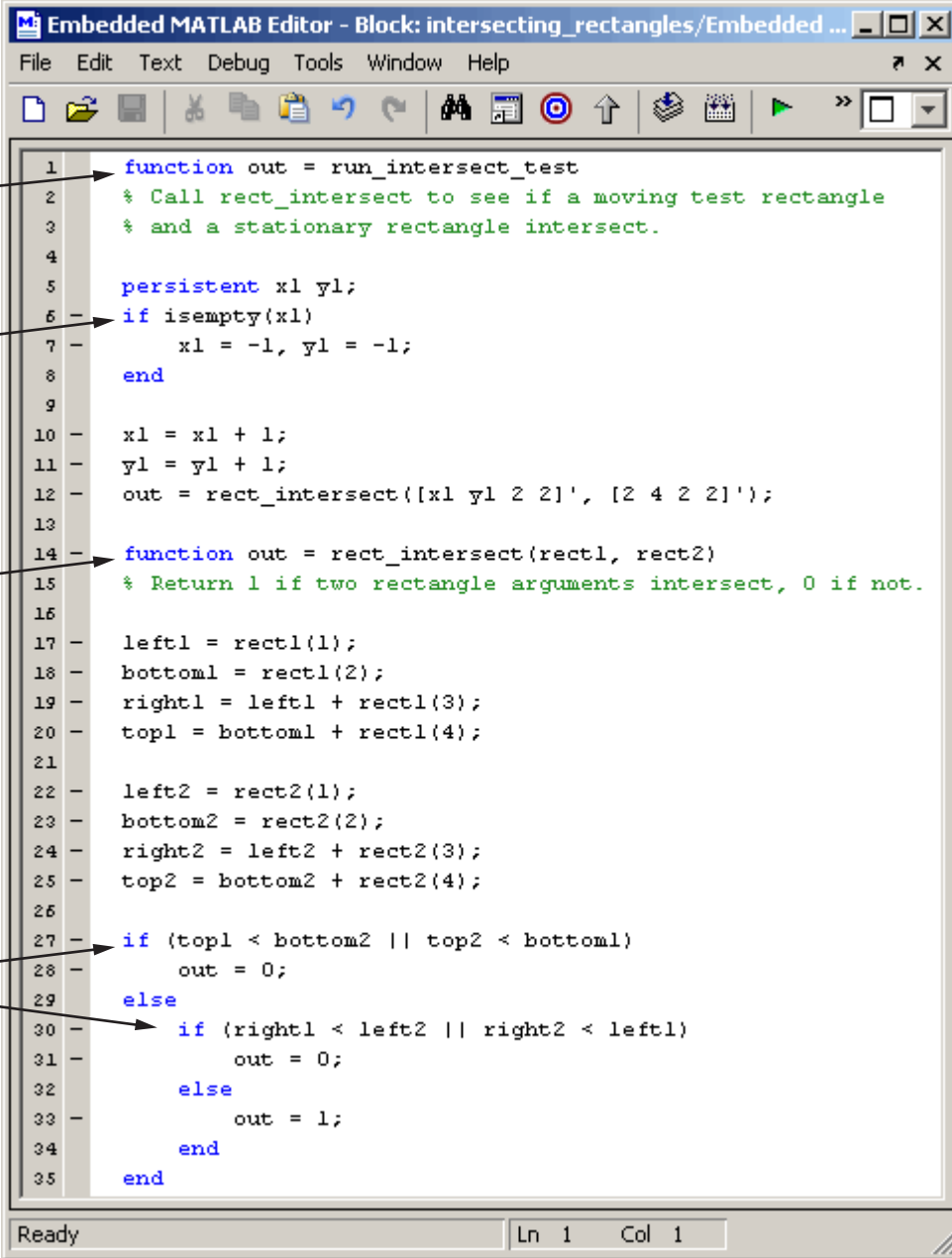
For an example of coverage data for Simulink Design Verifier functions in a coverage report, see “Simulink® Design Verifier Coverage” on page 17-34.

## Creating a Model with Embedded MATLAB Function Block Decisions

In this topic you use an example model to examine model coverage of an Embedded MATLAB Function block. The following model contains a single Embedded MATLAB Function block with output data sent to a Scope block.



Double-click the Embedded MATLAB Function block to specify its program content.



```
1 function out = run_intersect_test
2 % Call rect_intersect to see if a moving test rectangle
3 % and a stationary rectangle intersect.
4
5 persistent x1 y1;
6 if isempty(x1)
7     x1 = -1, y1 = -1;
8 end
9
10 x1 = x1 + 1;
11 y1 = y1 + 1;
12 out = rect_intersect([x1 y1 2 2]', [2 4 2 2]');
13
14 function out = rect_intersect(rect1, rect2)
15 % Return 1 if two rectangle arguments intersect, 0 if not.
16
17 left1 = rect1(1);
18 bottom1 = rect1(2);
19 right1 = left1 + rect1(3);
20 top1 = bottom1 + rect1(4);
21
22 left2 = rect2(1);
23 bottom2 = rect2(2);
24 right2 = left2 + rect2(3);
25 top2 = bottom2 + rect2(4);
26
27 if (top1 < bottom2 || top2 < bottom1)
28     out = 0;
29 else
30     if (right1 < left2 || right2 < left1)
31         out = 0;
32     else
33         out = 1;
34     end
35 end
```

Function

Decision

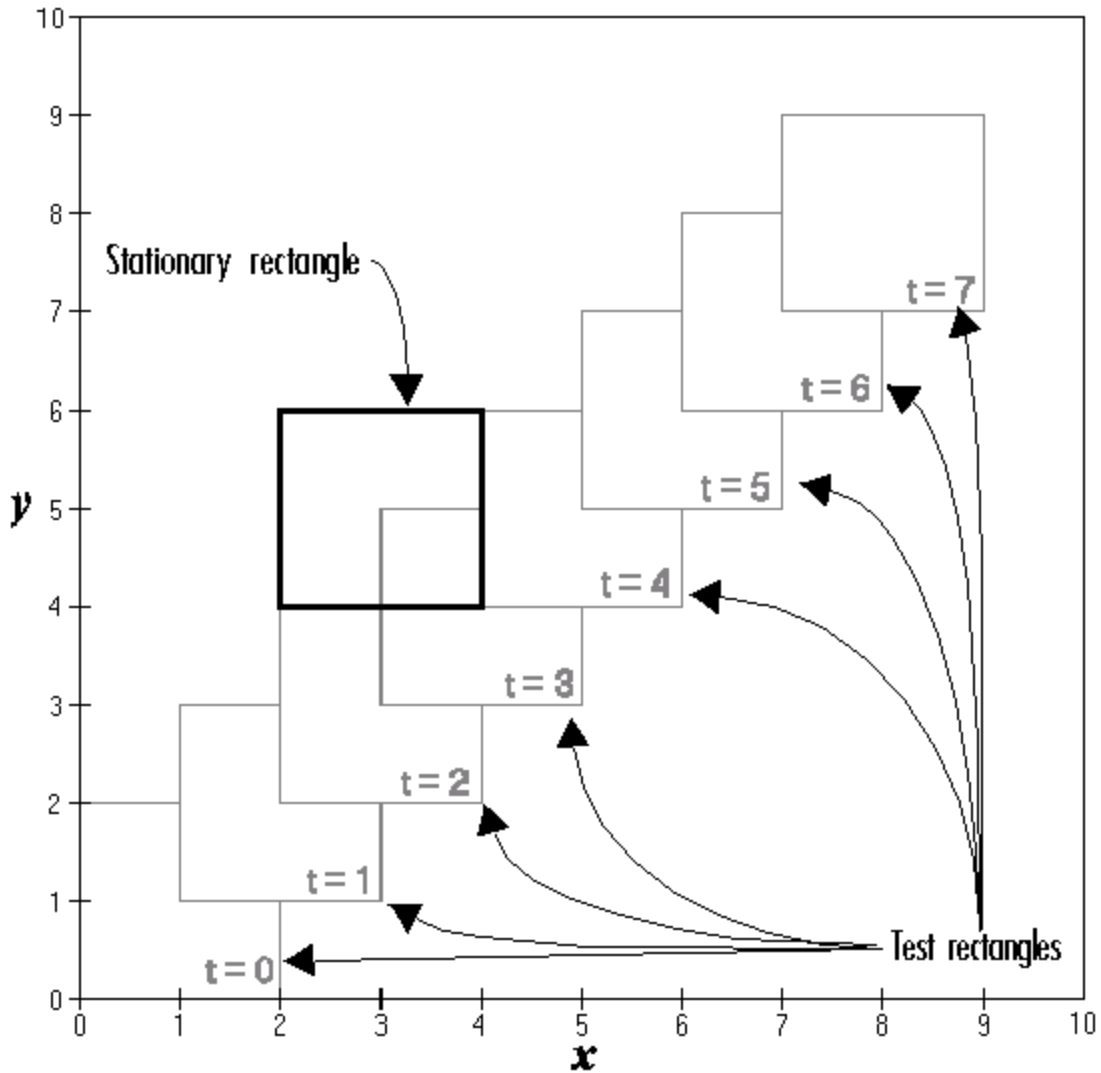
Subfunction

Decisions

Ready Ln 1 Col 1

The `run_intersect_test` Embedded MATLAB Function block contains two functions. The top-level function, `run_intersect_test`, sends the coordinates for two rectangles, one fixed and the other moving, as arguments to the subfunction `rect_intersect`, which tests for intersection between the two. The origin of the moving rectangle increases by 1 in the x and y directions with each time step.

The coordinates for the origin of the moving test rectangle are represented by persistent data `x1` and `y1`, which are both initialized to -1. For the first sample, `x1` and `y1` are both incremented to 0. From then on, the progression of rectangle arguments during simulation is as shown in the following graphic.



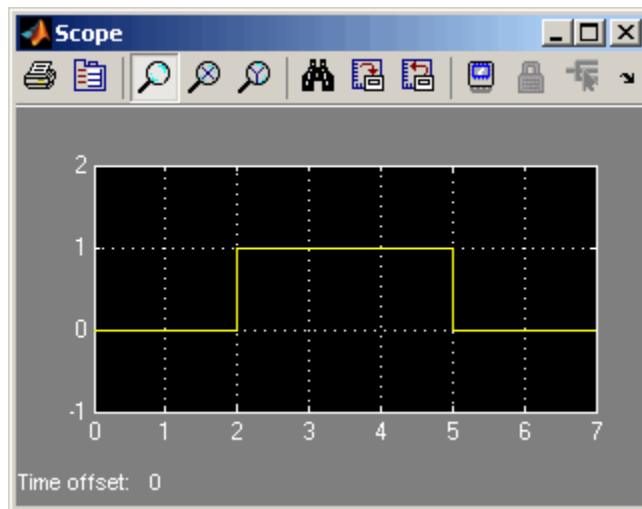
The fixed rectangle is shown in bold with a lower-left origin of (2,4) and a width and height of 2. At time  $t = 0$ , the first test rectangle has an origin of (0,0) and a width and height of 2. For each succeeding sample, the origin



of the test rectangle increments by  $(1, 1)$ . The rectangles at sample times  $t = 2, 3,$  and  $4$  intersect with the test rectangle.

The subfunction `rect_intersect` checks to see if its two rectangle arguments intersect. Each argument consists of coordinates for the lower-left corner of the rectangle (origin), and its width and height.  $x$  values for the left and right sides and  $y$  values for the top and bottom are calculated for each rectangle and compared in nested `if-else` decisions. The function returns a logical value of 1 if the rectangles intersect and 0 if they do not.

Scope output during simulation, which plots the return value against the sample time, confirms the intersecting rectangles for sample 2, 3, and 4.



## Understanding Embedded MATLAB Function Block Model Coverage

You can specify that model coverage reports generate automatically after a simulation. For instructions on how to specify a model coverage report, see Chapter 17, “Understanding Model Coverage Reports”.

After the simulation, the model coverage report appears in a browser window. After the summary for the model, the Details section of the model coverage report reports on each of the parts of the model. Model coverage for the

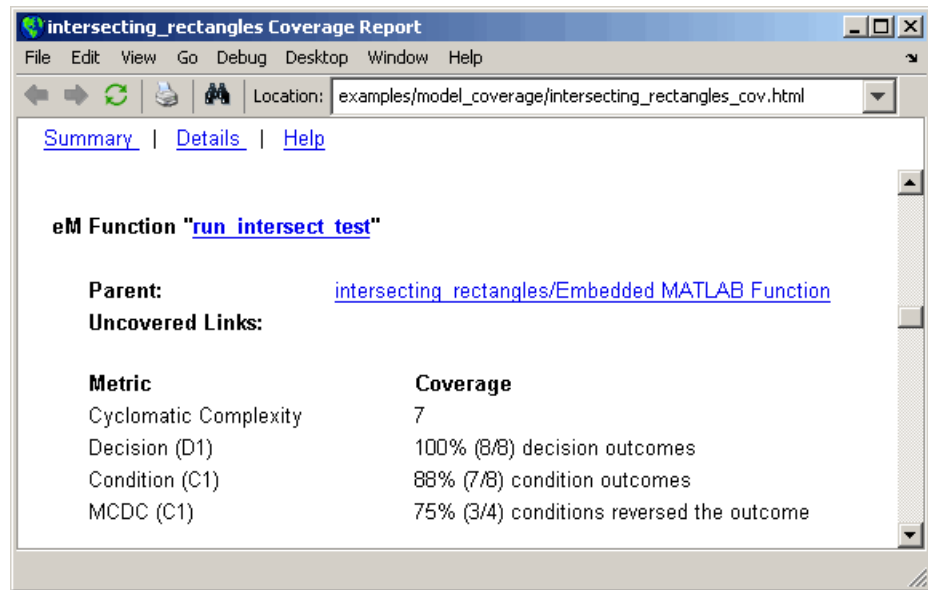
parts of the example model in “Creating a Model with Embedded MATLAB Function Block Decisions” on page 15-39, in model-block-function order, is explained in the following table.

Model:	<code>intersecting_rectangles</code>
Block:	Embedded MATLAB Function
Function:	<code>run_intersect_test</code>
Decision Lines:	<code>1: function out = rect_intersect_test</code> <code>6: if isempty(x1)</code> <code>14: function out = rect_intersect(rect1, rect2)</code> <code>27: if (top1 &lt; bottom2    top2 &lt; bottom1)</code> <code>30: if (right1 &lt; left2    right2 &lt; left1)</code>

The following sections examine the model coverage report for the example model in reverse function-block-model order. Reversing the order helps you make sense of the summary information at the top of each section.

### **Model Coverage for the Embedded MATLAB Function Block Function `run_intersect_test`**

You see model coverage for the Embedded MATLAB Function block function `run_intersect_test` under the linked name of the function. Clicking this link opens the function in the Embedded MATLAB Editor. Following the linked function name is a link to the model coverage report for the parent Embedded MATLAB Function block of `run_intersect_test`.



The top half of the report for the function summarizes its model coverage results. The coverage metrics for `run_intersect_test` include decision, condition, and MCDC coverage. You can best understand these metrics by examining the code listing for `run_intersect_test`.

```

1 function out = run_intersect_test
2 % Call rect_intersect to see if a moving test rectangle
3 % and a stationary rectangle intersect.
4
5 persistent x1 y1;
6 if isempty(x1)
7     x1 = -1, y1 = -1;
8 end
9
10 x1 = x1 + 1;
11 y1 = y1 + 1;
12 out = rect_intersect([x1 y1 2 2]', [2 4 2 2]');
13
14 function out = rect_intersect(rect1, rect2)
15 % Return 1 if two rectangle arguments intersect, 0 if not.
16
17 left1 = rect1(1);
18 bottom1 = rect1(2);
19 right1 = left1 + rect1(3);
20 top1 = bottom1 + rect1(4);
21
22 left2 = rect2(1);
23 bottom2 = rect2(2);
24 right2 = left2 + rect2(3);
25 top2 = bottom2 + rect2(4);
26
27 if (top1 < bottom2 || top2 < bottom1)
28     out = 0;
29 else
30     if (right1 < left2 || right2 < left1)
31         out = 0;
32     else
33         out = 1;
34     end
35 end

```

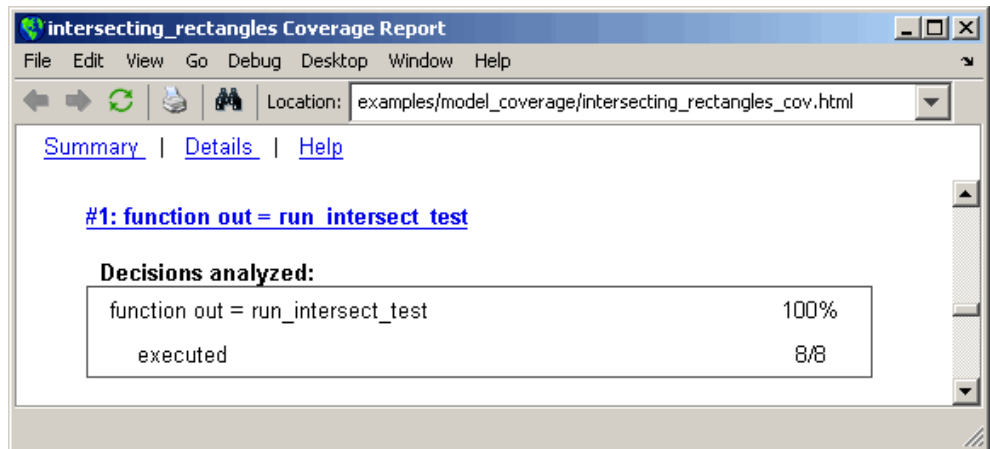
Lines with coverage elements are marked by a highlighted line number in the listing. Line 1 receives decision coverage on whether the top-level function `run_intersect_test` is executed. Line 6 receives decision coverage for its `if` statement. Line 14 receives decision coverage on whether the subfunction

`rect_intersect` is executed. Lines 27 and 30 receive decision, condition, and MCDC coverage for their `if` statements and conditions. Each of these lines is the subject of a report that follows the listing.

Notice that the condition `right1 < left2` in line 30 is highlighted in red. This means that this condition was not tested for all of its possible outcomes during simulation. Exactly which of the outcomes was not tested is in the report for the decision in line 30.

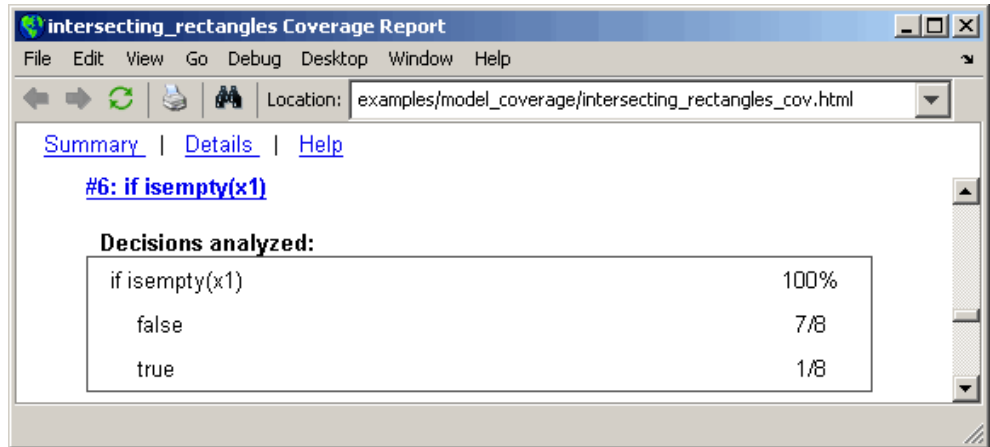
The following sections display the coverage for each `run_intersect_test` decision line. The coverage for each line is titled with the line itself, which is linked to display the function with the line highlighted.

**Coverage for Line 1.** The coverage metrics for line 1 are below the listing for the function `run_intersect_test`.



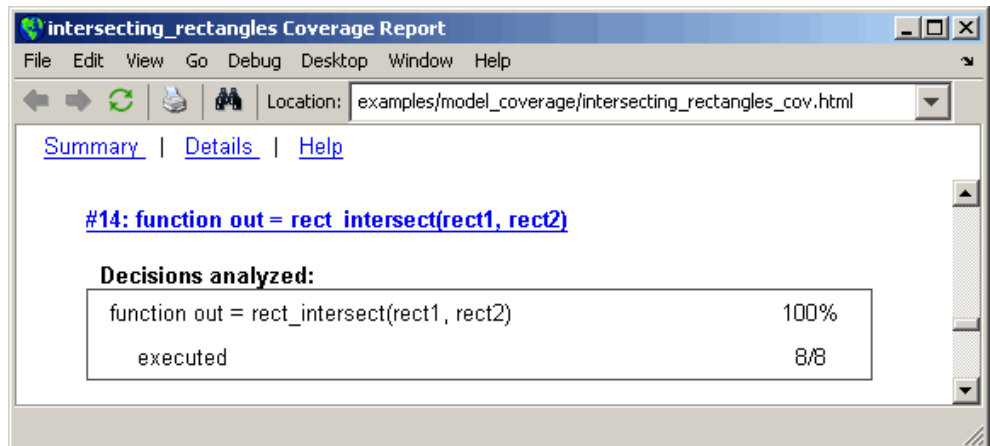
The first line of every function receives coverage analysis indicative of the decision to run the function in response to a call. Coverage for `run_intersect_test` indicates that it executed during testing.

**Coverage for Line 6.** The coverage metrics for line 6 are below the coverage metrics for line 1.



The **Decisions analyzed** table indicates that the decision in line 6, `if isempty(x1)`, executed a total of eight times. The first time it executed, the decision evaluated to true, enabling `run_intersect_test` to initialize the values of its persistent data. The remaining seven times the decision executed, it evaluated to false. Because both possible outcomes occurred, decision coverage is 100%.

**Coverage for Line 14.** The coverage metrics for line 14 are below the coverage metrics for line 6.



This table indicates that the subfunction `rect_intersect` executed during testing.

**Coverage for Line 27.** Coverage metrics for line 27 are below the coverage metrics for line 14.

The screenshot shows a web browser window titled "intersecting\_rectangles Coverage Report". The address bar shows the location: "examples/model\_coverage/intersecting\_rectangles\_cov.html". The page has three tabs: "Summary", "Details", and "Help". The main content area displays the following information:

**#27: if (top1 < bottom2 || top2 < bottom1)**

**Decisions analyzed:**

if (top1 < bottom2    top2 < bottom1)	100%
false	5/8
true	3/8

**Conditions analyzed:**

Description:	True	False
top1 < bottom2	2	6
top2 < bottom1	1	5

**MC/DC analysis (combinations in parentheses did not occur)**

Decision/Condition:	True Out	False Out
top1 < bottom2    top2 < bottom1		
top1 < bottom2	Tx	FF
top2 < bottom1	FT	FF

The **Decisions analyzed** table indicates that there are two possible outcomes for the decision in line 27: true and false. Five of the eight times it was executed, the decision evaluated to false. The remaining three times, it

evaluated to true. Because both possible outcomes occurred, decision coverage is 100%.

The **Conditions analyzed** table sheds some additional light on the decision in line 27. Because this decision consists of two conditions linked by a logical OR (|) operation, only one condition must evaluate true for the decision to be true. If the first condition evaluates to true, there is no need to evaluate the second condition. The first condition, `top1 < bottom2`, was evaluated eight times, and was true twice. This means that it was necessary to evaluate the second condition only six times. In only one case was it true, which brings the total true occurrences for the decision to three, as reported in the **Decisions analyzed** table.

MCDC coverage looks for decision reversals that occur because one condition outcome changes from T to F or from F to T. The **MC/DC analysis** table identifies all possible combinations of outcomes for the conditions that lead to a reversal in the decision. The character x is used to indicate a condition outcome that is irrelevant to the decision reversal. Decision-reversing condition outcomes that are not achieved during simulation are marked with a set of parentheses. There are no parentheses, therefore all decision-reversing outcomes occurred and MCDC coverage is complete for the decision in line 27.

**Coverage for Line 30.** Coverage metrics for line 30 are below the coverage metrics for line 27.



[Summary](#) | [Details](#) | [Help](#)

**#30: if (right1 < left2 || right2 < left1)**

**Decisions analyzed:**

if (right1 < left2    right2 < left1)	100%
false	3/5
true	2/5

**Conditions analyzed:**

Description:	True	False
right1 < left2	0	5
right2 < left1	2	3

**MC/DC analysis (combinations in parentheses did not occur)**

Decision/Condition:	True Out	False Out
right1 < left2    right2 < left1		
right1 < left2	(Tx)	FF
right2 < left1	FT	FF

The line 30 decision, `if (right1 < left2 || right2 < left1)`, is nested in the `if` statement of the line 27 decision and is evaluated only if the line 27 decision is false. Because the line 27 decision evaluated false five times, line 30 is evaluated five times, three of which are false. Because both the true and false outcomes were achieved, decision coverage for line 30 is 100%.

Because line 30, like line 27, has two conditions related by a logical OR operator (`||`), condition 2 is tested only if condition 1 is false. Because condition 1 tests false five times, condition 2 is tested five times. Of these,

condition 2 tests true two times and false three times, which accounts for the two occurrences of the true outcome for this decision.

Because the first condition of the line 30 decision does not test true, both outcomes do not occur for that condition and the condition coverage for the first condition is highlighted with a rose color. MCDC coverage is also highlighted in the same way for a decision reversal based on the true outcome for that condition.

**Coverage for run\_intersect\_test.** The metrics that summarize coverage for the entire run\_intersect\_test function are reported prior to its listing and are repeated here as shown.

Metric	Coverage
Cyclomatic Complexity	7
Decision (D1)	100% (8/8) decision outcomes
Condition (C1)	88% (7/8) condition outcomes
MCDC (C1)	75% (3/4) conditions reversed the outcome

The results summarized in the coverage metrics summary can be expressed in the following conclusions:

- There are eight decision outcomes reported for run\_intersect\_test in the line reports: one for line 1 (executed), two for line 6 (true and false), one for line 14 (executed), two for line 27 (true and false), and two for line 30

(true and false). The decision coverage for each line shows 100% decision coverage. This means that decision coverage for `run_intersect_test` is eight of eight possible outcomes, or 100%.

- There are four conditions reported for `run_intersect_test` in the line reports. Lines 27 and 30 each have two conditions, and each condition has two condition outcomes (true and false), for a total of eight condition outcomes in `run_intersect_test`. All conditions tested positive for both the true and false outcome except for the first condition of line 30 (`right1 < left2`). This means that condition coverage for `run_intersect_test` is seven of eight, or 88%.
- The MCDC coverage tables for decision lines 27 and 30 each list two cases of decision reversal for each condition, for a total of four possible reversals. Only the decision reversal for a change in the evaluation of the condition `right1 < left2` of line 30 from true to false did not occur during simulation. This means that three of four, or 75% of the possible reversal cases were tested for during simulation, for a coverage of 75%.

### **Model Coverage for the Embedded MATLAB Function Block and the Model**

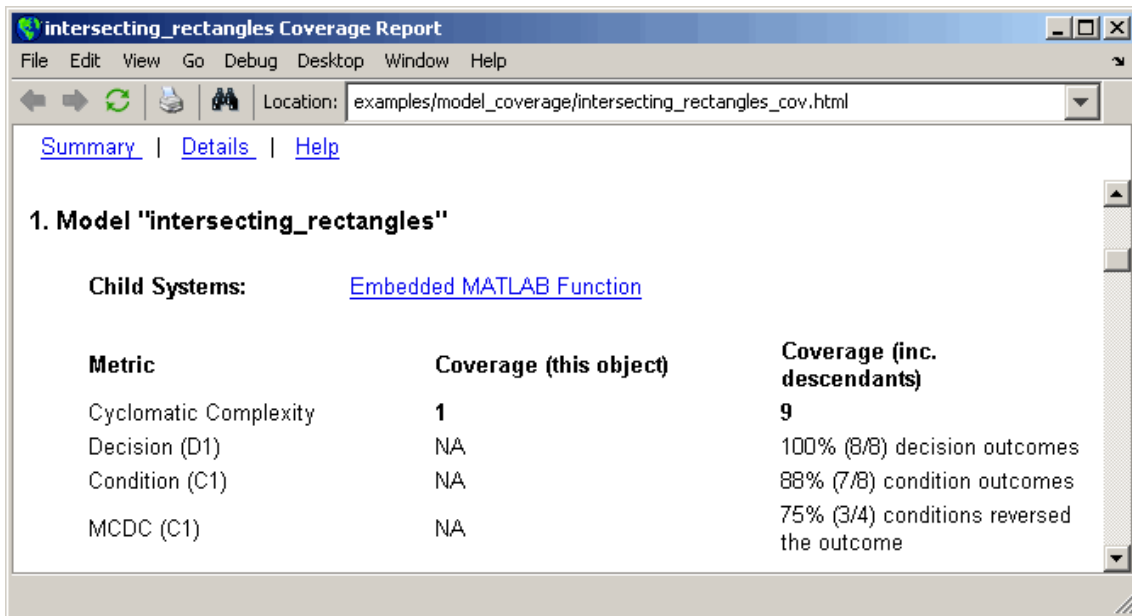
The model coverage report for the block Embedded MATLAB Function shows that it has no decisions of its own apart from its function. However, it does repeat the summary information for its function `run_intersect_test` as coverage for its descendent objects.

**2. EML Script block "Embedded MATLAB Function"**

Parent: [/intersecting\\_rectangles](#)

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	1	8
Decision (D1)	NA	100% (8/8) decision outcomes
Condition (C1)	NA	88% (7/8) condition outcomes
MCDC (C1)	NA	75% (3/4) conditions reversed the outcome

Because there are no additional coverage objects in the model apart from the Embedded MATLAB Function block, the remaining report for the model `intersecting_rectangles` also repeats the preceding coverage for descendent objects.



**intersecting\_rectangles Coverage Report**

File Edit View Go Debug Desktop Window Help

Location: examples/model\_coverage/intersecting\_rectangles\_cov.html

[Summary](#) | [Details](#) | [Help](#)

### 1. Model "intersecting\_rectangles"

**Child Systems:** [Embedded MATLAB Function](#)

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	<b>1</b>	<b>9</b>
Decision (D1)	NA	100% (8/8) decision outcomes
Condition (C1)	NA	88% (7/8) condition outcomes
MCDC (C1)	NA	75% (3/4) conditions reversed the outcome

## Colored Simulink Diagram Coverage Display

### In this section...

“How Model Coverage Highlighting Works” on page 15-56

“Enabling the Colored Diagram Display” on page 15-56

“Displaying Model Coverage with Model Coloring” on page 15-57

“Accessing Coverage Information for Colored Blocks” on page 15-59

### How Model Coverage Highlighting Works

The Simulink Verification and Validation software displays model coverage results for individual blocks directly in Simulink diagrams. If you enable model coverage, the tool:

- Highlights (colors) blocks that have received model coverage during simulation
- Provides a context-sensitive display of summary model coverage information for each block

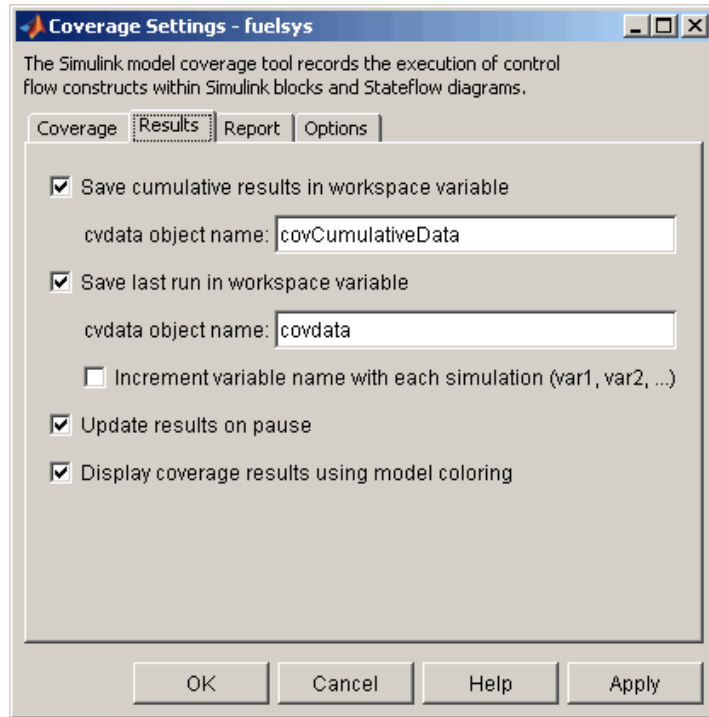
Coloring highlights structural coverage in Simulink models. When you enable coloring for model coverage results (see “Enabling the Colored Diagram Display” on page 15-56), the tool highlights blocks that received the following types of model coverage:

- “Decision Coverage (DC)” on page 15-4
- “Condition Coverage (CC)” on page 15-4
- “Modified Condition/Decision Coverage (MCDC)” on page 15-4

### Enabling the Colored Diagram Display

To enable the model coverage colored diagram display:

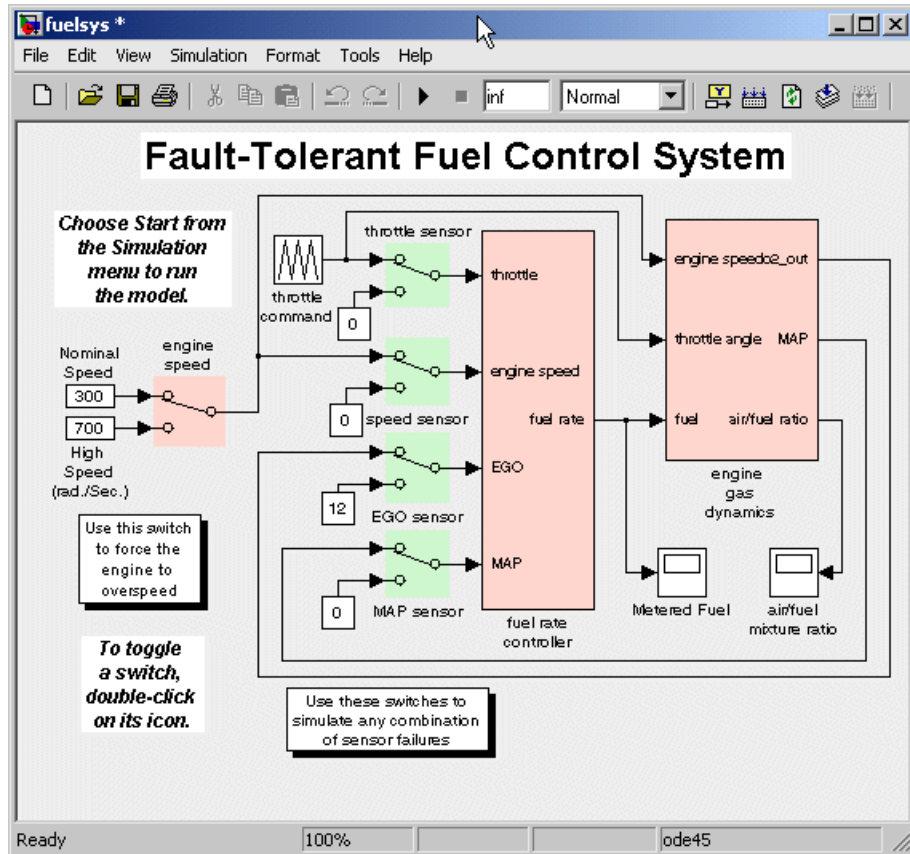
- 1** In the Simulink window, from the **Tools** menu, select **Coverage Settings**.
- 2** In the **Coverage** tab, select **Coverage for this model**.
- 3** Select the **Results** tab.



The **Display coverage results using model coloring** option is selected by default for all models. This check box becomes visible only after **Coverage for this model** is enabled. You can clear this option for the current session by clearing this check box.

## Displaying Model Coverage with Model Coloring

You enable display coverage as described in “Enabling the Colored Diagram Display” on page 15-56. After you enable this display, any time that the model generates a model coverage report, individual blocks receiving coverage are highlighted with light green or light red.



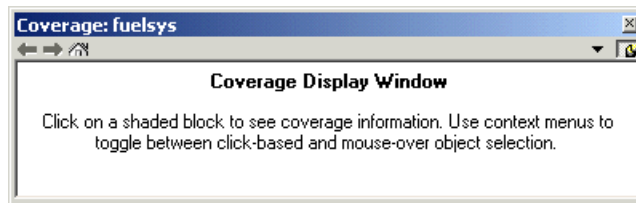
The light green Manual Switch blocks received full coverage during testing. The light red blocks (the engine speed Manual Switch block, and the fuel rate controller and engine gas dynamics subsystems) received incomplete coverage during testing. Blocks with no color highlighting (the Constant blocks, Scope blocks, and the throttle command Repeating Sequence block) received no coverage.



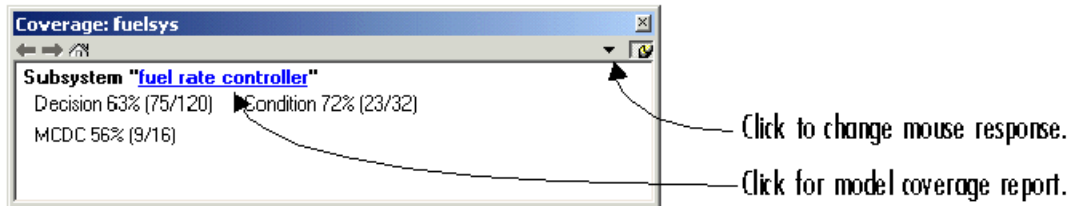
**Note** To restore the Simulink diagram to its original colors, right-click a colored block and, from the context menu, select **Coverage**. From the resulting submenu, select **Remove information**. Alternatively, to remove model coloring, from the Simulink **View** menu or the diagram context menu, select **Remove Highlighting**.

## Accessing Coverage Information for Colored Blocks

“Displaying Model Coverage with Model Coloring” on page 15-57 describes the highlighted Simulink diagram that appears after simulation when you enable display coverage with model coloring. Along with the highlighted Simulink diagram, a Coverage Display window appears.



In the Simulink model, if you click a colored block, the model summarized coverage appears in the Coverage Display window. From the preceding example, you see the following summary report when you click the fuel rate controller subsystem.



Summary coverage information appears in the Coverage Display window for the block, whose hyperlinked name is at the top of the window. Click the hyperlink to access the appropriate section of the coverage report for this block. You can also see this section of the report by right-clicking the block and selecting **Coverage > Report**.

To set the Coverage Display window to display coverage for a block in response to a hovering mouse cursor (instead of a mouse click):

- On the right side of the Coverage Display window, select the down arrow. From the resulting menu, select **Focus**. Or,
- Right-click a colored block. From the context menus, select **Coverage** followed by **Display details on mouse-over**.

---

**Tip** You can adjust the font size in the Coverage Display window. To increase the font size, press **Ctrl+**; to decrease the font size, press **Ctrl-**.

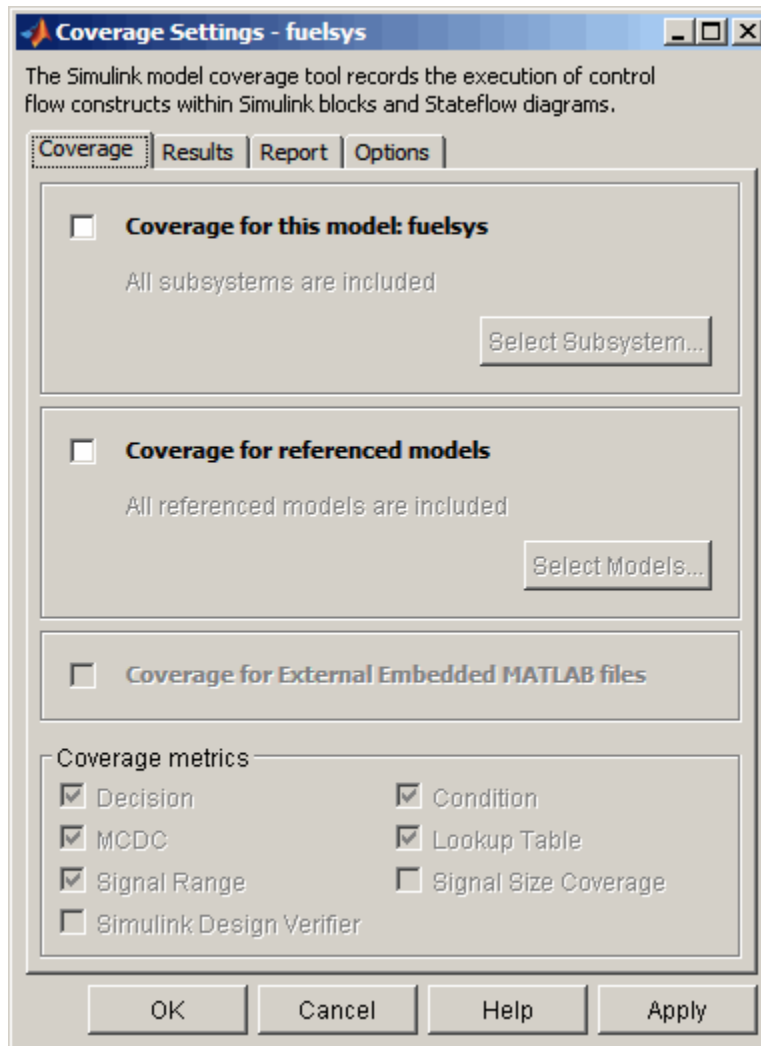
---

# Setting the Model Coverage Options

---

## Coverage Settings Dialog Box

Before starting a model coverage analysis, you must specify several model coverage recording and reporting options. In a Simulink model, select **Tools > Coverage Settings**. The Coverage Settings dialog box opens, with the **Coverage** tab displayed.



The following sections describe the settings for each tab of the Coverage Settings dialog box.

**In this section...**

“Coverage Tab” on page 16-3

“Results Tab” on page 16-6

“Report Tab” on page 16-8

“Options Tab” on page 16-12

## Coverage Tab

On the **Coverage** tab, select the model coverages calculated during simulation.

### Coverage for this model

Instructs the software to gather and report the model coverages that you specify during simulation. When you select the **Coverage for this model** option, the **Select Subsystem** button and the **Coverage metrics** section of the **Coverage** pane become available.

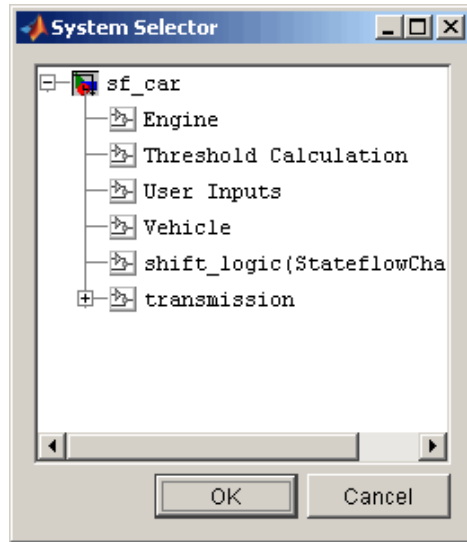
### Select Subsystem

Specifies the subsystem for which the software gathers and reports coverage data. When you select the **Coverage for this model** option, the software, by default, generates coverage data for the entire model.

To restrict coverage reporting to a particular subsystem:

- 1 On the **Coverage** tab, click **Select Subsystem**.

The System Selector dialog box appears.



- 2 In the System Selector dialog box, select the subsystem for which you want to enable coverage reporting and click **OK**.

### Coverage for referenced models

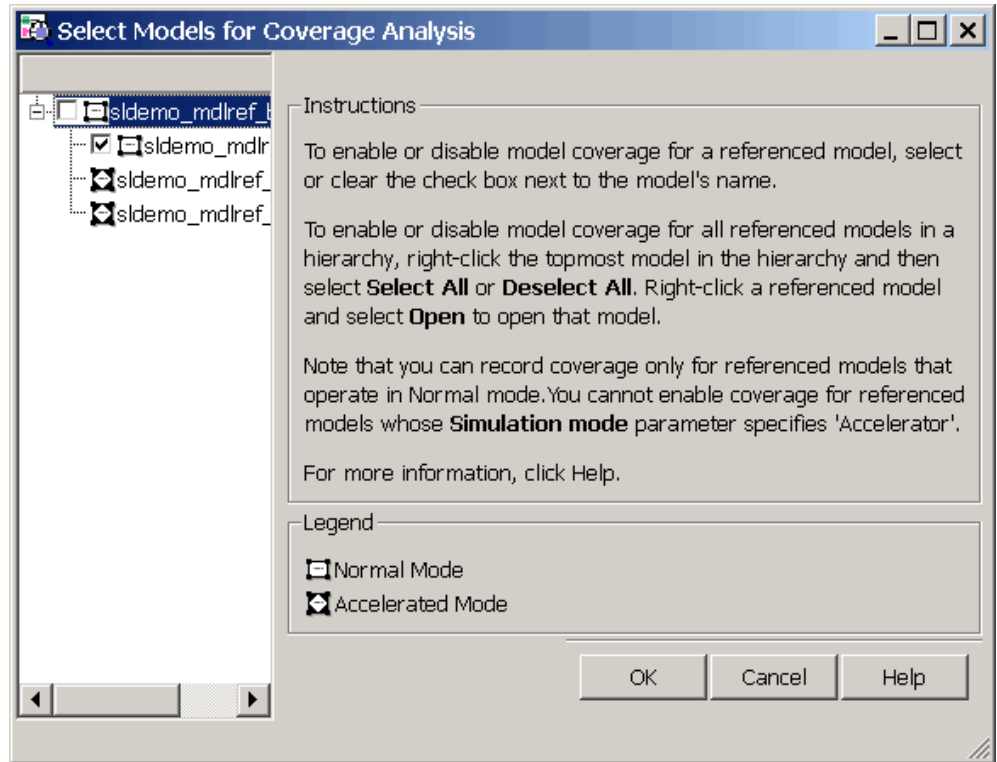
Causes the software to gather and report the model coverages that you specify for referenced models during simulation. When you select the **Coverage for referenced models** option, the **Select Models** button and the **Coverage metrics** section of the **Coverage** tab become available.

### Select Models

Specifies the referenced models for which the Simulink Verification and Validation software gathers and reports coverage data. When you select **Coverage for referenced models**, the software, by default, generates coverage data for all referenced models.

To enable coverage reporting for particular referenced models:

- 1 On the **Coverage** pane, click **Select Models**.



- 2 In the Select Models for Coverage Analysis dialog box, select the referenced models for which you want coverage reporting and then click **OK**.

---

**Note** The Simulink Verification and Validation software provides model coverage support only for referenced models that operate in Normal mode. The software cannot record coverage for Model blocks whose **Simulation mode** parameter specifies Accelerator.

---

### Coverage for External Embedded MATLAB Files

Enables coverage for any external functions that Embedded MATLAB functions call in your model. The Embedded MATLAB functions may be defined in an Embedded MATLAB Function block or in a Stateflow chart.

You must select either **Coverage for this model** or **Coverage for referenced models** to select the **Coverage for External Embedded MATLAB Files** option.

### **Coverage metrics**

Select the types of test case coverage analysis that you want the tool to perform (see “Types of Model Coverage” on page 15-3). The Simulink Verification and Validation software gathers and reports those types of coverage for the subsystem, model, and referenced models .

---

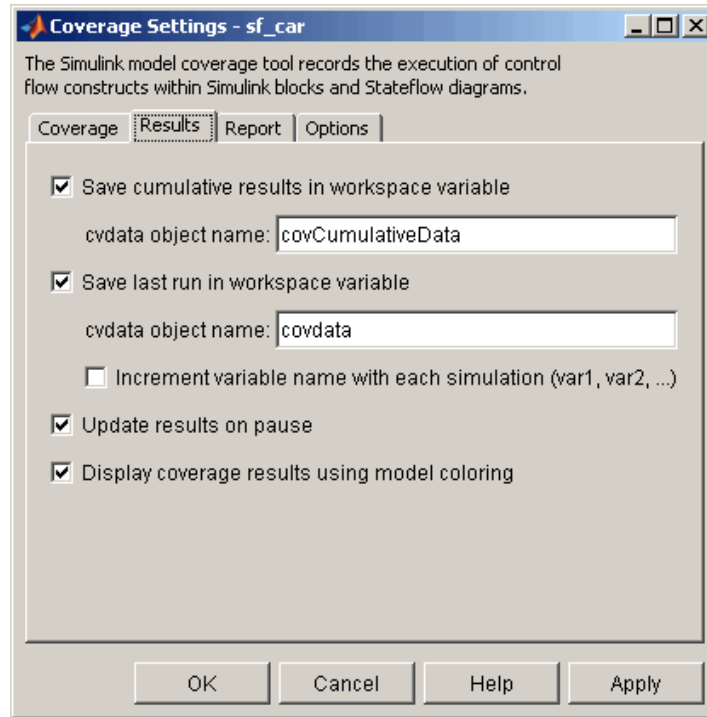
**Note** To specify different types of coverage analysis for each of the referenced models in a hierarchy, use the `cv.cvtestgroup` and `cvsimref` functions. For more information, see “Using Model Coverage Commands for Referenced Models” on page 18-12.

---

### **Results Tab**

On the **Results** pane, select the destination for model coverage results.





### Save Cumulative Results in Workspace Variable

Causes model coverage to accumulate and save the results of successive simulations in the workspace variable that you specify in the **cvdata object name** field.

### Save Last Run in Workspace Variable

Causes model coverage to save the results of the last simulation run in the workspace variable that you specify in the **cvdata object name** field below.

### Increment Variable Name with Each Simulation

Causes the Simulink Verification and Validation software to increment the name of the coverage data object variable used to save the last run with each simulation, so that the current simulation run does not overwrite the results of the previous run.

### **Update Results on Pause**

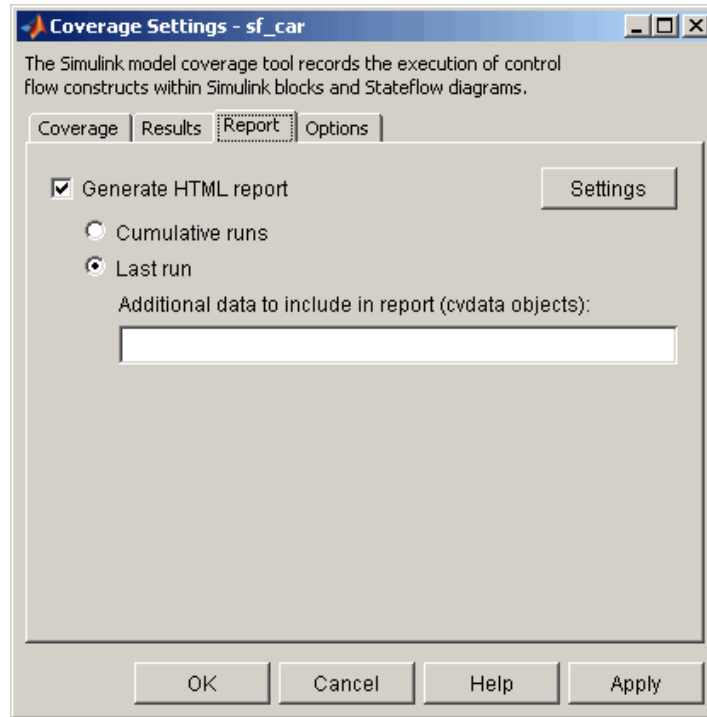
Causes the model coverage results to be recorded up to the point at which you pause the simulation for the first time. When you resume simulation and later pause or stop, the model coverage report reappears, with coverage results up to the current pause or stop time.

### **Display Coverage Results Using Model Coloring**

Causes coloring of Simulink blocks according to their level of model coverage, after simulation. Blocks highlighted in light green received full coverage during testing. Blocks highlighted in light red received incomplete coverage. See “Colored Simulink Diagram Coverage Display” on page 15-56.

### **Report Tab**

On the **Report** tab, specify whether the model coverage tool should generate an HTML report and what data the report should include.

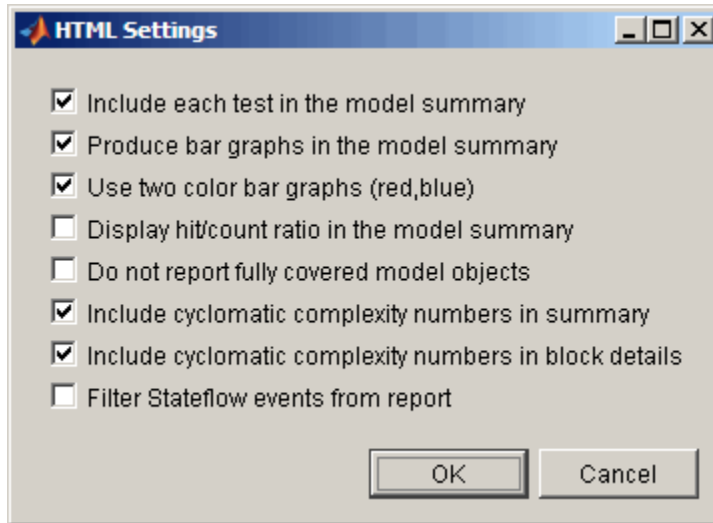


### Generate HTML Report

Causes the Simulink Verification and Validation software to create an HTML report containing the coverage data. At the end of the simulation, the report appears in the MATLAB Web browser. Click the **Settings** button to select various reporting options.

### Settings

On the **Report** tab, click **Settings** to open the HTML Settings dialog box. In the HTML Settings dialog box, choose model coverage report options.



Option	Description
<b>Include each test in the model summary</b>	The model hierarchy table at the top of the HTML report includes columns listing the coverage metrics for each test. If you do not select this option, the model summary reports only the total coverage.
<b>Produce bar graphs in the model summary</b>	Causes the model summary to include a bar graph for each coverage result for a visual representation of the coverage.
<b>Use two color bar graphs (red, blue)</b>	Red and blue bar graphs appear in the report instead of black and white.
<b>Display hit/count ratio in the model summary</b>	Reports coverage numbers as both a percentage and a ratio, for example, 67% (8/12).

Option	Description
<b>Do not report fully covered model objects</b>	The coverage report includes only model objects that the simulation does not cover fully, useful when developing tests, because it reduces the size of the generated reports.
<b>Include cyclomatic complexity numbers in summary</b>	Includes the cyclomatic complexity (see “Types of Model Coverage” on page 15-3) of the model and its top-level subsystems and charts in the report summary. A cyclomatic complexity number shown in boldface indicates that the analysis considered the subsystem itself to be an object when computing its complexity. This occurs for atomic and conditionally executed subsystems as well as for Stateflow Chart blocks.
<b>Include cyclomatic complexity numbers in block details</b>	Includes the cyclomatic complexity metric in the block details section of the report.
<b>Filter Stateflow events from report</b>	Excludes coverage data on Stateflow events.

### Cumulative Runs

Display the coverage results from successive simulations in the report. For more information, see “Save Cumulative Results in Workspace Variable” on page 16-7.

If you select the **Save cumulative results in workspace variable** check box on the **Results** tab, a coverage running total is updated with new results at the end of each simulation. However, if you change model or block settings between simulations that are incompatible with settings from previous simulations and affect the type or number of coverage points, the cumulative coverage resets.

You can make cumulative coverage results persist between MATLAB sessions by using `cvsave` to save results to a file at the end of the session and `cvload` to load the results at the beginning of the session. The `cvload` parameter `RESTORETOTAL` must be 1 in order to restore cumulative results.

When you save the coverage results to a file using `cvsave` and a model name argument, the file also contains the cumulative running total. When you load that file into the coverage tool using `cvload`, you can select whether you want to restore the running total from the file.

When you restore a running total from saved data, the saved results are reflected in the next cumulative report. If a running total already exists when you restore a saved value, the existing value is overwritten.

Whenever you report on more than one single simulation, the coverage displayed for truth tables and lookup-table maps is based on the total coverage of all the reported runs. For cumulative reports, this includes all the simulations where cumulative results are stored.

You can also calculate cumulative coverage results at the command line through the `+` operator:

```
covdata1 = cvsim(test1);
covdata2 = cvsim(test2);
cvhtml('cumulative_report', covdata + covdata2);
```

### **Last run**

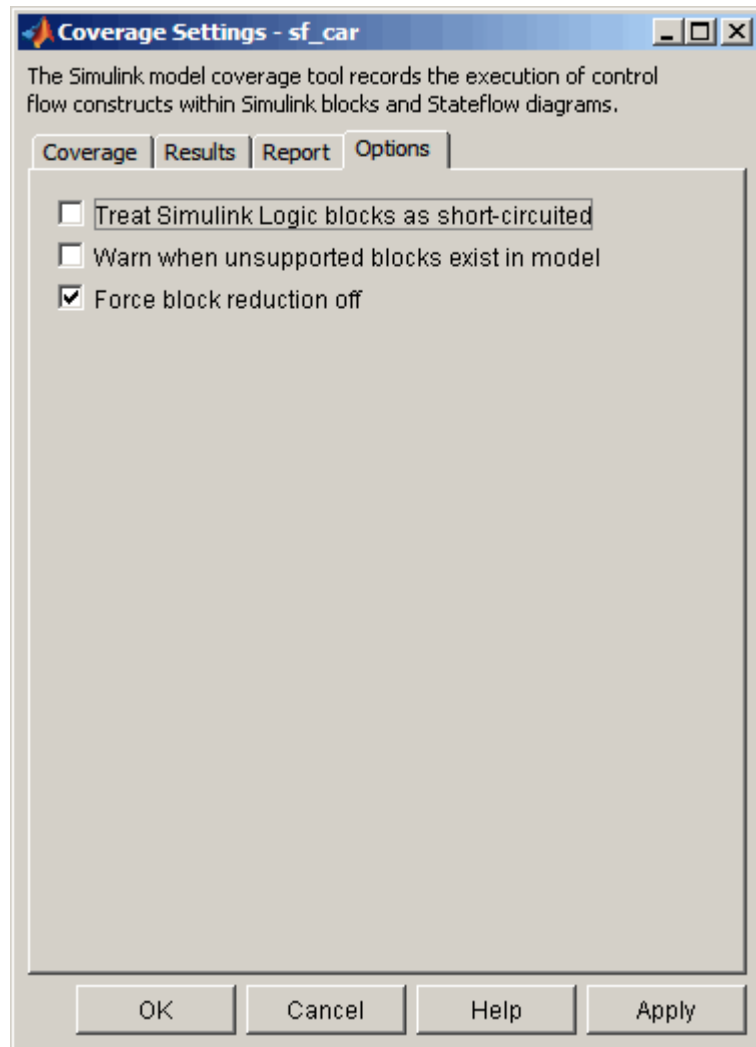
Include only the results of the most recent simulation run in the report.

### **Additional data to include in report**

Specify names of coverage data from previous runs to include in the current report along with the current coverage data. Each entry creates a new set of columns in the report.

### **Options Tab**

On the **Options** tab, select options for model coverage reports.



### **Treat Simulink Logic blocks as short-circuited**

The **Treat Simulink Logic blocks as short-circuited** option applies only to condition and MCDC coverage. If you select this option, coverage analysis treats Simulink logic blocks as if the block ignores remaining inputs when the previous inputs alone determine the block's output. For example, if the

first input to a Logical Operator block whose **Operator** parameter specifies AND is false, MCDC coverage analysis ignores the values of the other inputs when determining MCDC coverage for a test case.

If you enable this feature and logic blocks are short-circuited while collecting model coverage, you may not be able to achieve 100% coverage for that block.

Select this option to generate code from a model and where you want the MCDC coverage analysis to approximate the degree of coverage that your test cases achieve for the generated code (most high-level languages short-circuit logic expressions).

---

**Note** A test case that does not achieve full MCDC coverage for non-short-circuited logic expressions might achieve full coverage for short-circuited expressions.

---

### **Warn when unsupported blocks exist in model**

Select this option to warn you at the end of the simulation that the model contains blocks that require coverage analysis but are not currently covered by the tool.

### **Force block reduction off**

To achieve faster execution during model simulation and in generated code, enable the **Block reduction** parameter on the Configuration Parameters dialog box **Optimization** pane. The Simulink software collapses certain groups of blocks into a single, more efficient block, or removes them entirely.

One of the model coverage options, **Force block reduction off**, allows you to ignore the **Block reduction** parameter when collecting model coverage.

If you do not enable the **Block reduction** parameter, or if you select **Force block reduction off**, the Simulink Verification and Validation software provides coverage data for every block in the model that collects coverage.



If you enable the **Block reduction** parameter and do not set **Force block reduction off**, the coverage report lists the reduced blocks that would have collected coverage.

The model coverage report identifies any reduced blocks. For an example of a reduced blocks report, see “Block Reduction” on page 17-28.



# Understanding Model Coverage Reports

---

- “Types of Coverage Reports” on page 17-2
- “Model Coverage Reports” on page 17-3
- “Model Summary Reports” on page 17-38
- “Model Reference Coverage Reports” on page 17-39
- “External MATLAB File Coverage Reports” on page 17-40
- “Subsystem Coverage Reports” on page 17-45

## Types of Coverage Reports

In the Coverage Settings dialog box, on the **Report** tab, if you select the **Generate HTML report** option, the Simulink Verification and Validation software creates one or more model coverage reports after a simulation.

Report Type	Description	HTML Report File Name
“Model Coverage Reports” on page 17-3	Provides coverage information for all model elements, including the model itself.	<i>model_name_cov.html</i>
“Model Summary Reports” on page 17-38	Provides links to coverage results for all referenced models and external MATLAB files in the model hierarchy. Created when the top-level model includes Model blocks or calls one or more external files.	<i>model_name_summary_cov.html</i>
“Model Reference Coverage Reports” on page 17-39	Created for each referenced model in the model hierarchy; has the same format as the model coverage report.	<i>reference_model_name_cov.html</i>
“External MATLAB File Coverage Reports” on page 17-40	Provides detailed coverage information about any external MATLAB file that the model calls. There is one report for each external file called.	<i>MATLAB_file_name_cov.html</i>
“Subsystem Coverage Reports” on page 17-45	Model coverage report includes only coverage results for the subsystem, if you select one.	<i>model_name_cov.html</i> ; <i>model_name</i> is the name of the top-level model

## Model Coverage Reports

The Simulink Verification and Validation software always creates a model coverage report for the top-level model named *model\_name\_cov.html*. The model coverage report contains several sections:

In this section...
“Coverage Summary” on page 17-3
“Details” on page 17-5
“Cyclomatic Complexity” on page 17-13
“Decisions Analyzed” on page 17-15
“Conditions Analyzed” on page 17-17
“MCDC Analysis” on page 17-17
“Cumulative Coverage” on page 17-19
“N-Dimensional Lookup Table” on page 17-21
“Block Reduction” on page 17-28
“Signal Range Analysis” on page 17-30
“Signal Size Coverage for Variable-Dimension Signals” on page 17-32
“Simulink® Design Verifier Coverage” on page 17-34

### Coverage Summary

The coverage summary section contains basic information about the model being analyzed:

- **Model Information**
- **Simulation Optimization Options**
- **Coverage Options**

The coverage summary has two subsections:

- **Tests** — The simulation start and stop time of each test case and any setup commands that preceded the simulation. The heading for each test case includes any test case label specified using the `cvtest` command.
- **Summary** — Summaries of the subsystem results. To see detailed results for a specific subsystem, in the Summary subsection, click the subsystem name.

[Summary](#) | [Details](#) | [Signal Ranges](#) | [Help](#)

## Coverage Report for fuelsys

### Model Information

Model Version 1.111  
 Author The MathWorks Inc.  
 Last Saved Wed May 14 18:49:10 2008

### Simulation Optimization Options

Inline Parameters off  
 Block Reduction forced off  
 Conditional Branch Optimization on

### Coverage Options

Logic block short circuiting off

## Tests

### Test 1

Started Execution: 02-Jun-2008 12:44:23  
 Ended Execution: 02-Jun-2008 12:45:42

## Summary

Model Hierarchy/Complexity:	Test 1				
	D1	C1	MCDC	TBL	
1. <a href="#">fuelsys</a>	83 39%		34%	13%	1%
2. ... <a href="#">EGO sensor</a>	1 50%		NA	NA	NA
3. ... <a href="#">MAP sensor</a>	1 50%		NA	NA	NA
4. ... <a href="#">engine speed</a>	1 50%		NA	NA	NA
5. ... <a href="#">engine gas dynamics</a>	5 60%		NA	NA	NA
6. ... <a href="#">Mixing &amp; Combustion</a>	1 50%		NA	NA	NA

## Details

The Details section reports the detailed model coverage results. Each section of the detailed report summarizes the results for the metrics that test each object in the model:

- “Model Details” on page 17-6
- “Subsystem Details” on page 17-7
- “Block Details” on page 17-8
- “Chart Details” on page 17-9
- “Coverage Details for Embedded MATLAB Functions and Simulink® Design Verifier Functions” on page 17-10

You can also access a model element Details subsection as follows:

- 1 Right-click a Simulink element.
- 2 In the context menu, select **Coverage > Report**.

### **Model Details**

The Details section contains a results summary for the model as a whole, followed by a list of elements. Click the model element name to see its coverage results.

The following graphic shows the Details section for the `fuel_sys` model.



**Details:****1. Model "fuelsys"**

**Child Systems:** [EGO sensor](#), [MAP sensor](#), [engine speed](#), [engine gas dynamics](#), [fuel rate controller](#), [speed sensor](#), [throttle command](#), [throttle sensor](#)

<b>Metric</b>	<b>Coverage (this object)</b>	<b>Coverage (inc. descendants)</b>
Cyclomatic Complexity	<b>1</b>	<b>83</b>
Decision (D1)	NA	39% (53/135) decision outcomes
Condition (C1)	NA	34% (11/32) condition outcomes
MCDC (C1)	NA	13% (2/16) conditions reversed the outcome
Look-up Table	NA	1% (15/1508) interpolation/extrapolation intervals

**Subsystem Details**

Each subsystem Details section contains a summary of the test coverage results for the subsystem and a list of the subsystems it contains. The overview is followed by sections for blocks, charts, and Embedded MATLAB functions, one for each object that contains a decision point in the subsystem.

The following graphic shows the coverage results for the EGO sensor subsystem in the fuelsys model.

**2. Subsystem "EGO sensor"**

**Parent:** [/fuelsys](#)

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	0	1
Decision (D1)	NA	50% (1/2) decision outcomes

**Block Details**

The following graphic shows the coverage results for the Switch block in the EGO sensor subsystem of the fuelsys model.

**Switch block "SwitchControl"**

**Parent:** [fuelsys/EGO sensor](#)

**Uncovered Links:** 

Metric	Coverage
Cyclomatic Complexity	1
Decision (D1)	50% (1/2) decision outcomes

**Decisions analyzed:**

trigger > threshold	50%
false (output is from 3rd input port)	0/204508
true (output is from 1st input port)	204508/204508

The **Uncovered Links** element first appears in the Block Details section of the first block in the model hierarchy that does not achieve 100% coverage.

The first **Uncovered Links** element has an arrow that links to the Block Details section in the report of the *next* block that does not achieve 100% coverage.

Subsequent blocks that do not achieve 100% coverage have links to the Block Details sections in the report of the previous and next blocks that do not achieve 100% coverage.

Switch block "[SwitchControl](#)"

**Parent:** [fuelsys/MAP sensor](#)

**Uncovered Links:** 

## Chart Details

The following graphic shows the coverage results for the Stateflow chart, Chart2, in the mExternalMfile model.

**2. Subsystem "[Chart2](#)"**

**Parent:** [/mExternalMfile](#)

**Child Systems:** [Chart2](#)

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	1	2
Decision (D1)	NA	0% (0/1) decision outcomes

For more information about model coverage reports for Stateflow charts and their objects, see “Understanding Model Coverage for Stateflow Charts” in the Stateflow documentation.

### Coverage Details for Embedded MATLAB Functions and Simulink Design Verifier Functions

By default, the Simulink Verification and Validation software records coverage for all Embedded MATLAB functions in a model. Embedded MATLAB functions are in Embedded MATLAB Function blocks, Stateflow charts, or external MATLAB files.

---

**Note** For a detailed example of coverage reports for external MATLAB files, see “External MATLAB File Coverage Reports” on page 17-40.

---

To record Simulink Design Verifier coverage for `sldv.*` functions called by Embedded MATLAB functions, and any Simulink Design Verifier blocks, in the Coverage Settings dialog box, on the **Coverage** tab, select **Simulink Design Verifier**.

The following example shows coverage details for an Embedded MATLAB function, `hFcnsInExternalEML`, that calls four Simulink Design Verifier functions. In this example, the code for `hFcnsInExternalEML` resides in an external file.

This example also shows Simulink Design Verifier coverage details for the following functions:

- `sldv.assume`
- `sldv.condition`
- `sldv.prove`
- `sldv.test`

In the coverage results, code that achieves 100% coverage is green; code that achieves less than 100% coverage is red.

### Embedded MATLAB function "[hfcnsinexternalem1](#)"

Parent: [hfcnsinexternalem1](#)

Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	4
Decision (D1)	40% (2/5) decision outcomes
Test Objective	50% (1/2) objective outcomes
Proof Objective	0% (0/1) objective outcomes
Test Condition	100% (1/1) objective outcomes
Proof Assumption	0% (0/1) objective outcomes

```

1 function y = hFcnsInExternalEML(u1, u2)
2 % use all four functions.
3 %#eml
4 sldv.assume(u1 > u2);
5 sldv.condition(u1 == 0);
6 switch u1
7     case 0
8         y = u2;
9     case 1
10        y = 3;
11    case 2
12        y = 0;
13    otherwise
14        y = 0;
15        sldv.prove(u2 < u1);
16 end
17 sldv.test(y > u1); sldv.test(y == 4);
18

```

Coverage for the hFcnsInExternalEML function and the sldv.\* calls is:

- Line 1, the function declaration for `hFcnsInExternalEML` is green because the simulation executes that function at least once. `fcn` calls `hFcnsInExternalEML` 11 times during simulation.

[#1: function y = hFcnsInExternalEML\(u1, u2\)](#)

**Decisions analyzed:**

function y = hFcnsInExternalEML(u1, u2)	100%
executed	11/11

Line 4, `sldv.assume(u1 > u2)`, achieves 0% coverage because `u1 > u2` never evaluates to true.

[#4: sldv.assume\(u1 > u2\);](#)

**Proof Assumption analyzed:**

sldv.assume(u1 > u2)	0/11
----------------------	------

- Line 5, `sldv.condition(u1 == 0)`, achieves 100% coverage because `u1 == 0` evaluates to true for at least one time step.

[#5: sldv.condition\(u1 == 0\);](#)

**Test Condition analyzed:**

sldv.condition(u1 == 0)	11/11
-------------------------	-------

- Line 6, `switch u1`, achieves 25% coverage because only one of the four outcomes in the switch statement (case 0) occurs during simulation.

**#6: switch u1****Decisions analyzed:**

switch u1	25%
otherwise	0/11
case 0	11/11
case 1	0/11
case 2	0/11

- Line 17, `sldv.test(y > u1); sldv.test (y == 4)` achieves 50% coverage. The first `sldv.test` call achieves 100% coverage, but the second `sldv.test` call achieves 0% coverage.

**#17: sldv.test(y > u1); sldv.test(y == 4);****Test Objective analyzed:**

<code>sldv.test(y &gt; u1)</code>	11/11
<code>sldv.test(y == 4)</code>	0/11

For more information about coverage for Embedded MATLAB functions, see “Model Coverage for Embedded MATLAB Function Blocks” on page 15-37.

For more information about coverage for Simulink Design Verifier functions, see “Simulink Design Verifier Coverage” on page 15-7.

## Cyclomatic Complexity

You can specify that the model coverage report include cyclomatic complexity numbers in two locations in the report:

- The Summary section contains the cyclomatic complexity numbers for each object in the model hierarchy. For a subsystem or Stateflow chart, that number includes the cyclomatic complexity numbers for all their descendants.

<b>Summary</b>	
Model Hierarchy/Complexity:	
1. <a href="#">fuelsys</a>	83
2. . . . <a href="#">EGO sensor</a>	1
3. . . . <a href="#">MAP sensor</a>	1
4. . . . <a href="#">engine speed</a>	1
5. . . . <a href="#">engine gas dynamics</a>	5
6. . . . . <a href="#">Mixing &amp; Combustion</a>	1
7. . . . . <a href="#">Throttle &amp; Manifold</a>	4
8. . . . . . <a href="#">Throttle</a>	2
9. . . . <a href="#">fuel rate controller</a>	72

- The Details sections for each object list the cyclomatic complexity numbers for all individual objects.



## 7. Subsystem "Throttle & Manifold"

Parent: [fuelsys/engine gas dynamics](#)

Child Systems: [Throttle](#)

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	0	4
Decision (D1)	NA	63% (5/8) decision outcomes

### Saturate block "Limit to Positive"

Parent: [fuelsys/engine gas dynamics/Throttle & Manifold](#)

Metric	Coverage
Cyclomatic Complexity	2
Decision (D1)	50% (2/4) decision outcomes

## Decisions Analyzed

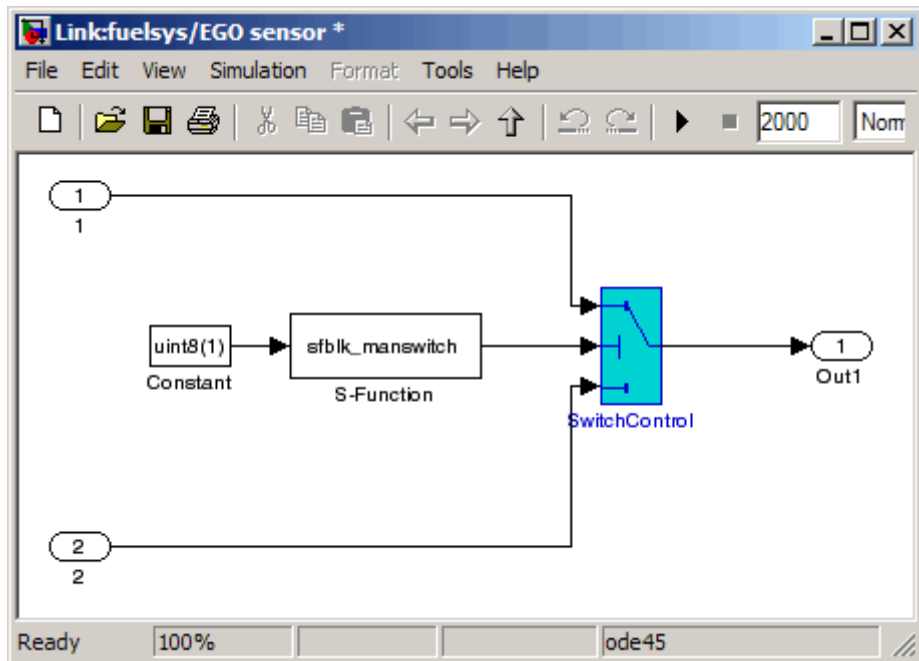
The Decisions analyzed table lists possible outcomes for a decision and the number of times that an outcome occurred in each test simulation. Outcomes that did not occur are in red highlighted table rows.

The following graphic shows the Decisions analyzed table for the Switch Control block in the EGO sensor subsystem of the fuelsys model.

**Decisions analyzed:**

trigger > threshold	50%
false (output is from 3rd input port)	0/204508
true (output is from 1st input port)	204508/204508

To display and highlight the block in question, click the block name associated with the Decisions analyzed table, as in this example from the `fuelsys` model.



The next graphic shows the Decisions analyzed table for the `lib_em2` function call in `Chart2` of the `MexternalMfile` model.

**#1: function em2 out = lib em2(em2 in)****Decisions analyzed:**

function em2_out = lib_em2(em2_in)	0%
executed	0/0

**Conditions Analyzed**

The Conditions analyzed table lists the number of occurrences of true and false conditions on each input port of the corresponding block.

**Conditions analyzed:**

Description:	True	False
input port 1	481	199520
input port 2	0	200001

**MCDC Analysis**

The MC/DC analysis table lists the MCDC input condition cases represented by the corresponding block and the extent to which the reported test cases cover the condition cases.

**MC/DC analysis (combinations in parentheses did not occur)**

Decision/Condition:	True Out	False Out
expression for output		
input port 1	FF	TF
input port 2	FF	(FT)

Each row of the MC/DC analysis table represents a condition case for a particular input to the block. A condition case for input n of a block is a combination of input values. Input n is called the *deciding input* of the

condition case. Changing the value of input *n* alone changes the value of the block's output.

The MC/DC analysis table shows a condition case expression to represent a condition case. A condition case expression is a character string where:

- The position of a character in the string corresponds to the input port number.
- The character at the position represents the value of the input. (T means true; F means false).
- A boldface character corresponds to the value of the deciding input.

For example, **FTF** represents a condition case for a three-input block where the second input is the deciding input.

The **Decision/Condition** column specifies the deciding input for an input condition case. The **#1 True Out** column specifies the deciding input value that causes the block to output a true value for a condition case. The **#1 True Out** entry uses a condition case expression, for example, **FF**, to express the values of all the inputs to the block, with the value of the deciding variable in bold.

Parentheses around the expression indicate that the specified combination of inputs did not occur during the first (or only) test case included in this report. In other words, the test case did not cover the corresponding condition case. The **#1 False Out** column specifies the deciding input value that causes the block to output a false value and whether the value actually occurred during the first (or only) test case included in the report.

If you select **Treat Simulink Logic blocks as short-circuited** in the Coverage Settings dialog box, MC/DC coverage analysis does not verify whether short-circuited inputs actually occur. The MC/DC analysis table uses an x in a condition expression (for example, TFxxx) to indicate short-circuited inputs that were not analyzed by the tool.

If you enable this feature and Logic blocks are short-circuited while collecting model coverage, you may not be able to achieve 100% coverage for that block.

**Uncovered Links.** The section for each block that did not achieve 100% coverage contains a backward and a forward arrow. Click the forward arrow to go to the next section in the report that describes a block that did not achieve 100% coverage. Click the back arrow to return to the previous section in the report that describes a block that did not achieve 100% coverage.

## Cumulative Coverage

On the **Results** tab, if you select **Save cumulative results in workspace variable** and on the **Report** tab, **Cumulative runs**, the results of each simulation are saved and recorded in the report.

In a cumulative coverage report, the results located in the right-most area in all tables reflect the running total value. The report is organized so that you can easily compare the additional coverage from the most recent run with the coverage from all prior runs in the session.

A cumulative coverage report contains information about:

- **Current Run** — The coverage results of the simulation just completed.
- **Delta** — Percentage of coverage added to the cumulative coverage achieved with the simulation just completed. If the previous simulation's cumulative coverage and the current coverage are nonzero, the delta may be 0 if the new coverage does not add to the cumulative coverage.
- **Cumulative** — The total coverage collected for the model up to, but not including, the simulation just completed.

After running three test cases for the `slvny_autopilot_test_harness` model, the Summary report shows how much additional coverage the third test case achieved and the cumulative coverage achieved for the first two test cases.

## Summary

Model Hierarchy/Complexity:	Current Run			Delta			Cumulative		
	D1	C1	MCDC	D1	C1	MCDC	D1	C1	MCDC
1. <a href="#">slvndemo_autopilot_test_harness</a>	31 38%	41%	17%	8%	6%	0%	51%	41%	17%
2. <a href="#">Logic</a>	25 34%	38%	17%	9%	8%	0%	47%	38%	17%
3. <a href="#">SF: Logic</a>	24 34%	38%	17%	9%	8%	0%	47%	38%	17%
4. <a href="#">SF: Altitude</a>	11 64%	67%	33%	21%	17%	0%	93%	67%	33%
5. <a href="#">SF: Active</a>	4 38%	NA	NA	13%	NA	NA	88%	NA	NA
6. <a href="#">SF: GS</a>	13 11%	8%	0%	0%	0%	0%	11%	8%	0%
7. <a href="#">SF: Active</a>	6 0%	NA	NA	0%	NA	NA	0%	NA	NA
8. <a href="#">SF: Coupled</a>	3 0%	NA	NA	0%	NA	NA	0%	NA	NA
9. <a href="#">Verify Outputs</a>	5 60%	50%	NA	0%	0%	NA	80%	50%	NA
10. <a href="#">Subsystem1</a>	1 0%	NA	NA	0%	NA	NA	100%	NA	NA
11. <a href="#">Capture time</a>	1 0%	NA	NA	0%	NA	NA	100%	NA	NA
12. <a href="#">Subsystem2</a>	1 100%	NA	NA	0%	NA	NA	100%	NA	NA
13. <a href="#">Capture time</a>	1 100%	NA	NA	0%	NA	NA	100%	NA	NA
14. <a href="#">Subsystem3</a>	1 0%	NA	NA	0%	NA	NA	0%	NA	NA
15. <a href="#">Capture time</a>	1 0%	NA	NA	0%	NA	NA	0%	NA	NA
16. <a href="#">Verification</a>	2 100%	50%	NA	0%	0%	NA	100%	50%	NA

The Decisions analyzed table for cumulative coverage contains three columns of data about decision outcomes that represent the current run, the delta since the last run, and the cumulative data, respectively.

### Decisions analyzed:

Transition trigger expression	100%	50%	100%
false	401/402	0/1	3399/3400
true	1/402	1/1	1/3400

The Conditions analyzed table uses column headers **#n T** and **#n F** to indicate results for individual test cases. The table uses **Tot T** and **Tot F** for the cumulative results. You can identify the true and false conditions on each input port of the corresponding block for each test case.

**Conditions analyzed:**

Description:	#1 T	#1 F	#2 T	#2 F	Tot T	Tot F
Condition 1, "in(GS.Active.Coupled)"	0	402	0	0	0	3400
Condition 2, "alt_ctrl"	401	1	0	1	3399	1
Condition 3, "wow"	0	401	0	0	0	3399

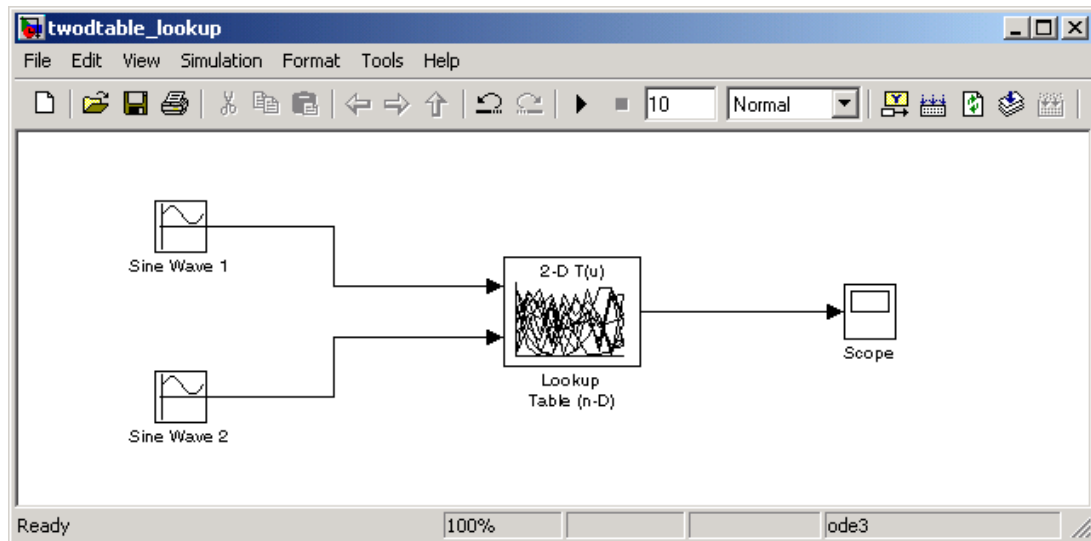
The MC/DC analysis **#n True Out** and **#n False Out** columns show the condition cases for each test case. The **Total Out T** and **Total Out F** column show the cumulative results.

**MC/DC analysis (combinations in parentheses did not occur)**

Decision/Condition:	#1 True Out	#1 False Out	#2 True Out	#2 False Out	Total Out T	Total Out F
Transition trigger expression						
Condition 1, "in(GS.Active.Coupled)"	(Txx)	FTF	(Txx)	(FTF)	(Txx)	FTF
Condition 2, "alt_ctrl"	FFx	FTF	FFx	(FTF)	FFx	FTF
Condition 3, "wow"	(FTT)	FTF	(FTT)	(FTF)	(FTT)	FTF

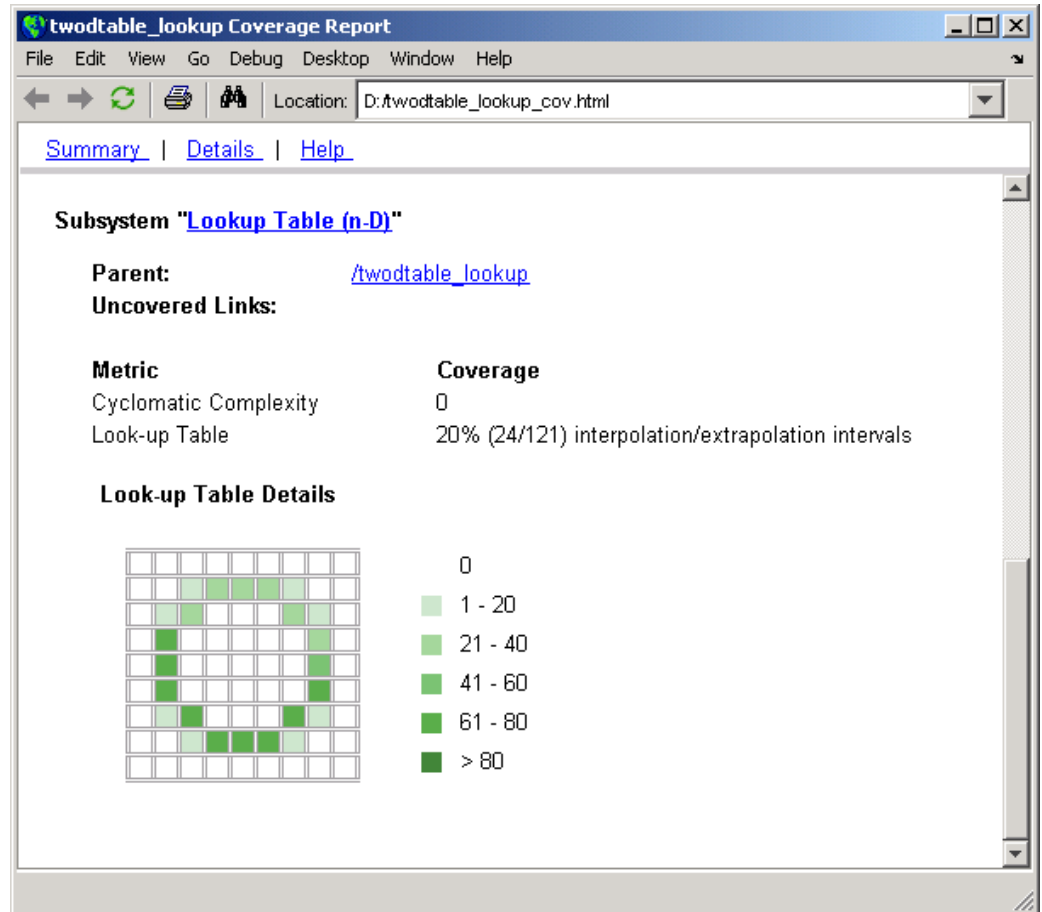
**N-Dimensional Lookup Table**

The following interactive chart summarizes the extent to which elements of a lookup table are accessed. In this example, two Sine Wave blocks generate *x* and *y* indices that access a Lookup Table (n-D) block of 10-by-10 elements filled with random values.



In this example, table indices are 1, 2,..., 10 in each direction. The Sine Wave 2 block is out of phase with the Sine Wave 1 block by  $\pi/2$  radians. This generates  $x$  and  $y$  numbers for the edge of a circle, which you see when you examine the resulting Lookup Table coverage.





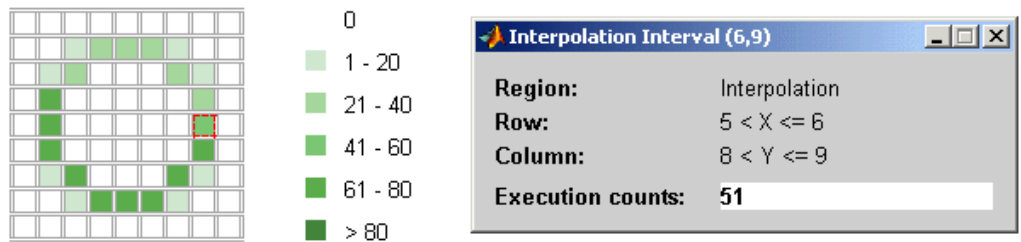
The report contains a two-dimensional table representing the elements of the lookup table. The element indices are represented by the cell border grid lines, which number 10 in each dimension. Areas where the lookup table interpolates between table values are represented by the cell areas. Areas of extrapolation left of element 1 and right of element 10 are represented by cells at the edge of the table, which have no outside border.

The number of values interpolated (or extrapolated) for each cell (*execution counts*) during testing is represented by a shade of green assigned to the

cell. Each of six levels of green shading and the range of execution counts represented are displayed on one side of the table.

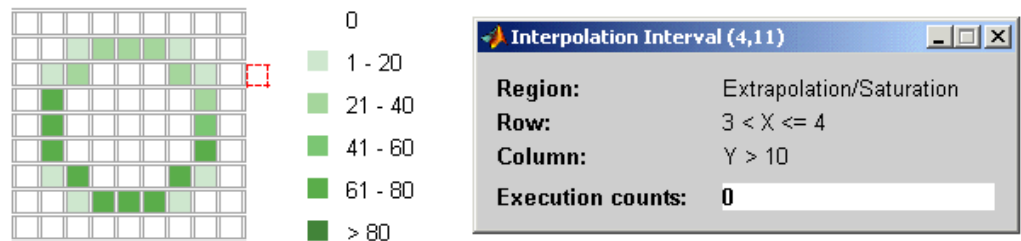
If you click an individual table cell, you see a dialog box that displays the index location of the cell and the exact number of execution counts generated for it during testing. The following example shows the contents of a color-shaded cell on the right edge of the circle.

**Lookup Table Details**



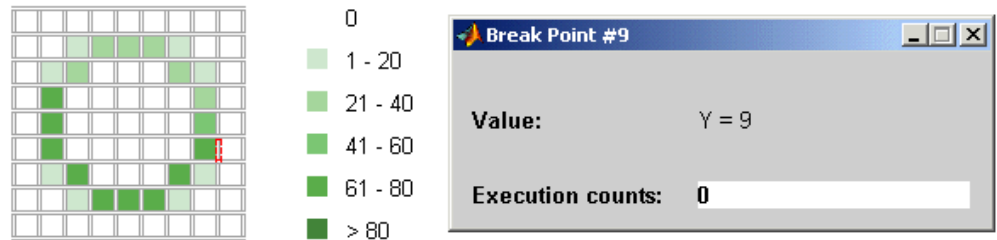
The selected cell is outlined in red. You can also click the extrapolation cells on the edge of the table.

**Lookup Table Details**

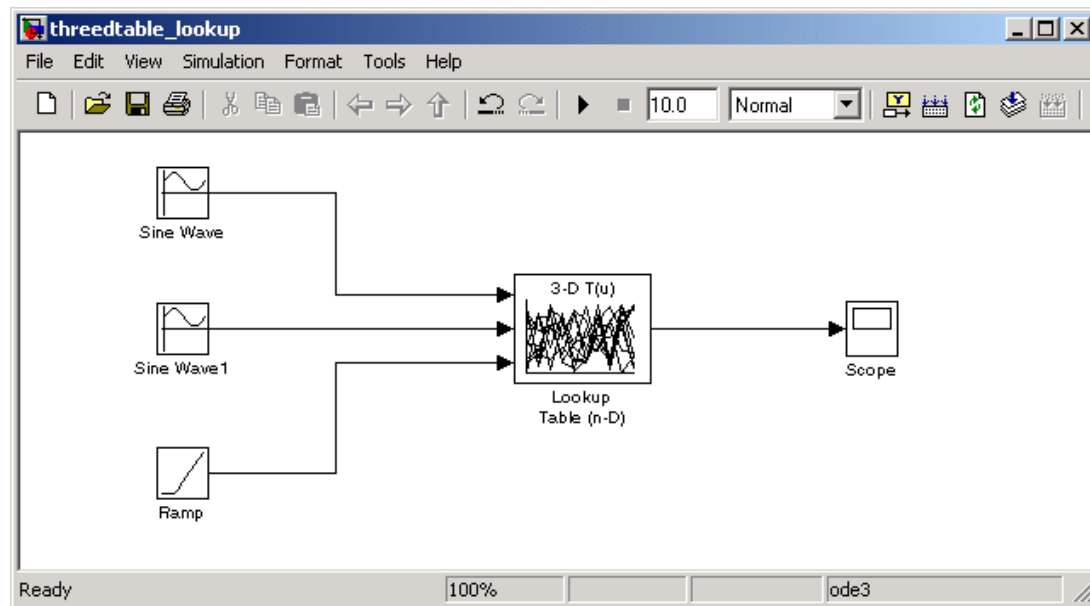


A bold grid line indicates that at least one block input equal to its exact index value occurred during the simulation. Click the border to display the exact number of hits for that index value.

### Lookup Table Details

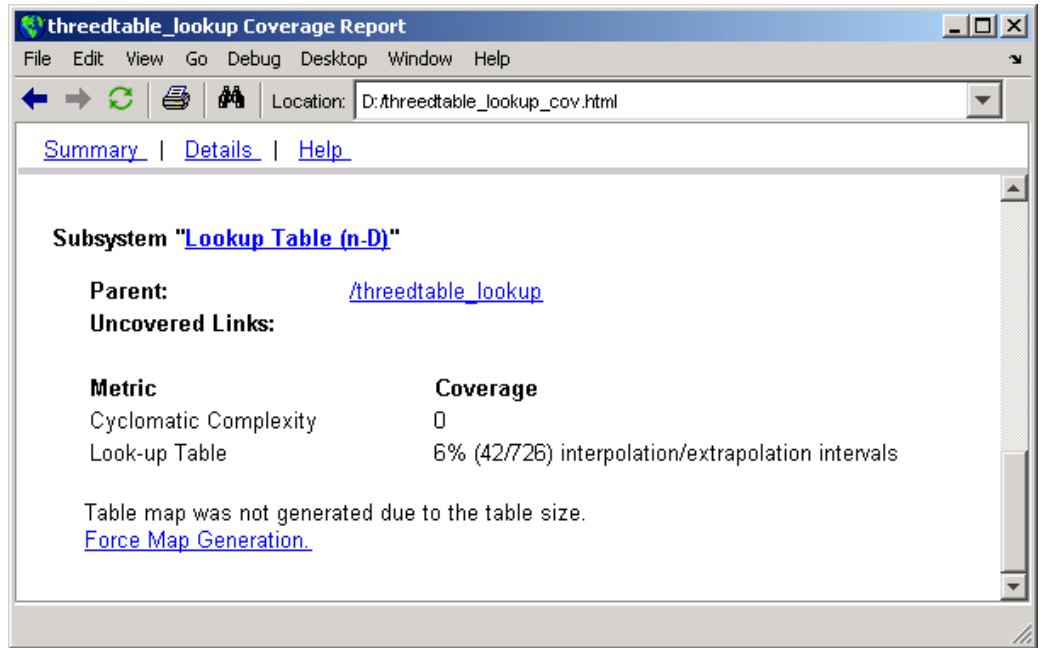


The following example model uses a Lookup Table (n-D) block of 10-by-10-by-5 elements filled with random values.

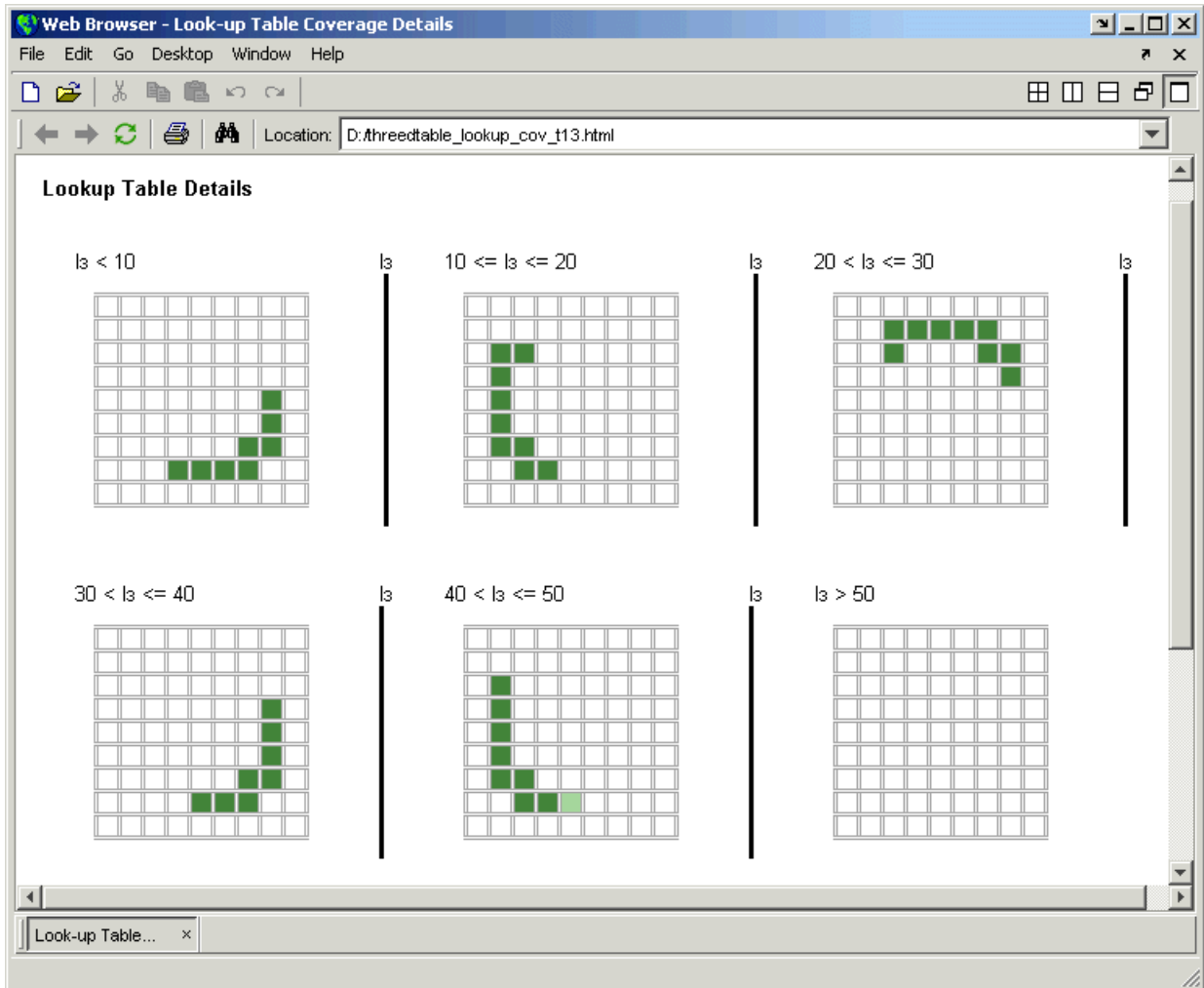


Both the  $x$  and  $y$  table axes have the indices 1, 2,..., 10. The  $z$  axis has the indices 10, 20,..., 50. Lookup table values are accessed with  $x$  and  $y$  indices that the two Sine Wave blocks generated, in the preceding example, and a  $z$  index that a Ramp block generates.

After simulation, you see the following lookup table report.



Instead of a two-dimensional table, you see the link Force Map Generation , which displays the following tables



Lookup table coverage for a three-dimensional lookup table block is reported as a set of two-dimensional tables.

The vertical bars represent the exact  $z$  index values: 10, 20, 30, 40, 50. If a vertical bar is bold, this indicates that at least one block input was equal to the exact index value it represents during the simulation. Click a bar to get a coverage report for the exact index value that bar represents.

You can report lookup table coverage for lookup tables of any dimension. Coverage for four-dimensional tables is reported as sets of three-dimensional sets, like those in the preceding example. Five-dimensional tables are reported as sets of sets of three-dimensional sets, and so on.

## Block Reduction

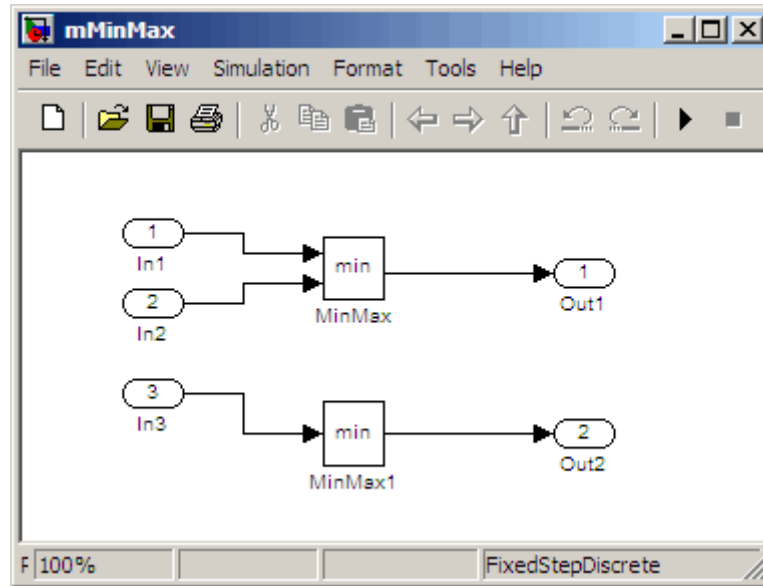
All model coverage reports indicate the status of the Simulink **Block reduction** parameter at the beginning of the report. In the following example, you set **Force block reduction off**.

Simulation Optimization Options	
Inline Parameters	off
Block Reduction	forced off
Conditional Branch Optimization	on

In the next example, you enabled the Simulink **Block reduction** parameter, and you did not set **Force block reduction off**.

Simulation Optimization Options	
Inline Parameters	off
Block Reduction	on
Conditional Branch Optimization	on


Consider the following model where the simulation does not execute the MinMax1 block because there is only one input—the constant 3.



If you set **Force block reduction off**, the report contains no coverage data for this block because the minimum input to the MinMax1 block is always 1

**MinMax block "[MinMax1](#)"**

Parent: [/mMinMax](#)

Uncovered Links: 

Metric	Coverage
Cyclomatic Complexity	1
Decision (D1)	0% (0/1) decision outcomes

**Decisions analyzed:**

Logic to determine output	0%
input 1 is the minimum	0/0

If you do not set **Force block reduction off**, the report contains no coverage data for reduced blocks.

### Reduced Blocks

Blocks eliminated from coverage analysis by block reduction model simulation setting:

... [mMinMax/MinMax1](#)

### Signal Range Analysis

If you select **Signal Range Coverage**, the software creates a Signal Range Analysis section at the bottom of the model coverage report. This section lists the maximum and minimum signal values for each output signal in the model measured during simulation.

Access the Signal Range Analysis report quickly with the **Signal Ranges** link in the nonscrolling region at the top of the model coverage report, as shown for the `fuelsys` model.

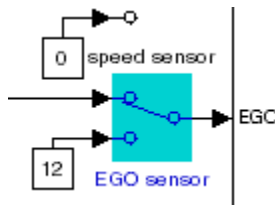


[Summary](#) | [Details](#) | [Signal Ranges](#) | [Help](#)

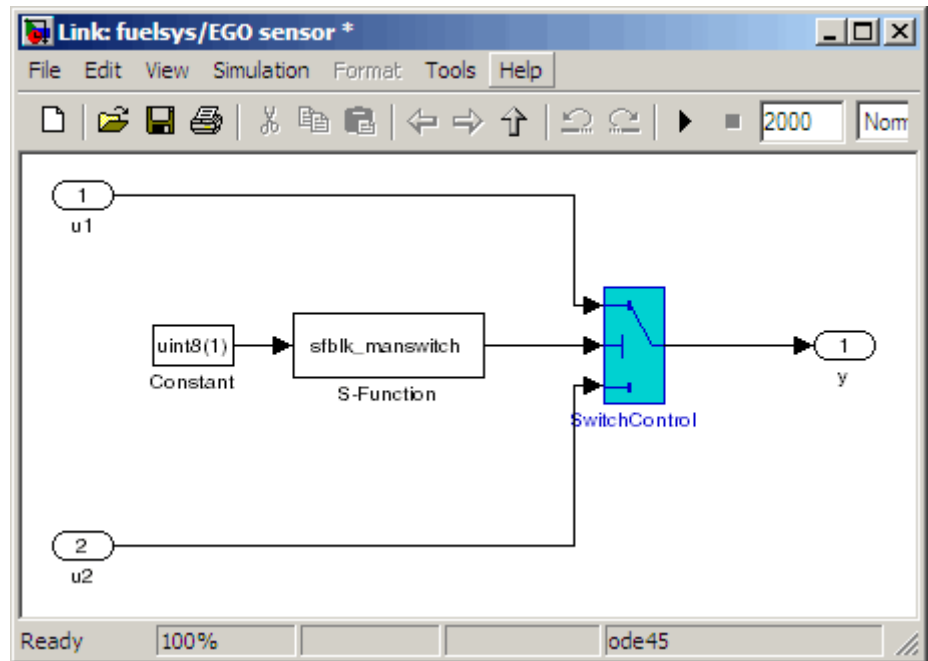
### Signal Ranges:

Hierarchy	Min	Max
fuelsys		
... <a href="#">Constant2</a>	-	-
... <a href="#">Constant3</a>	12	12
... <a href="#">Constant4</a>	0	0
... <a href="#">Constant5</a>	0	0
... <a href="#">High Speed (rad./Sec.)</a>	-	-
... <a href="#">Nominal Speed</a>	300	300
... <a href="#">EGO sensor</a>		
..... <a href="#">SwitchControl</a>	0.229199	1
... <a href="#">MAP sensor</a>		
..... <a href="#">SwitchControl</a>	0.405559	0.889674
... <a href="#">engine speed</a>		

Each block is reported in hierarchical fashion; child blocks appear directly under parent blocks. Each block name in the **Signal Ranges** report is a link. For example, select the EGO sensor link to display this block highlighted in its native diagram.



Select the SwitchControl link to display this block in its own subsystem by looking under the mask for EGO sensor.



## Signal Size Coverage for Variable-Dimension Signals

If you select **Signal Size Coverage**, the software creates a Variable Signal Widths section after the Signal Ranges data in the model coverage report. This section lists the maximum and minimum signal sizes for all output ports in the model that have variable-size signals. It also lists the memory that Simulink allocated for that signal, as measured during simulation. This list does *not* include signals whose size does not vary during simulation.

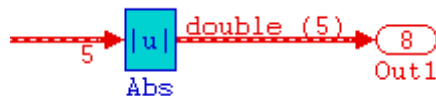
The following example shows the Variable Signal Widths section in a coverage report.

## Variable Signal Widths:

Hierarchy	Min	Max	Allocated
... <a href="#">Abs</a>	2	5	5
... <a href="#">Abs1</a>	4	4	5
... <a href="#">MinMax1</a>	2	5	5
... <a href="#">Switch</a>	2	5	5
... <a href="#">Switch1</a>	2	5	5
... <a href="#">Selector</a>	4	4	5
... <a href="#">c2ri</a>			
..... out1	4	4	5
..... out2	4	4	5
... <a href="#">Subsystem</a>			
..... <a href="#">LogicalOperator</a>	1	2	2
..... <a href="#">Switch1</a>	1	2	2
..... <a href="#">Switch2</a>	1	2	2

Each block is reported in hierarchical fashion; child blocks appear directly under parent blocks. Each block name in the Variable Signal Widths list is a link.

In this example, the Abs block signal size varied from 2 to 5, with an allocation of 5. Click the Abs link in the report. The Model Editor becomes current, with the Abs block highlighted.

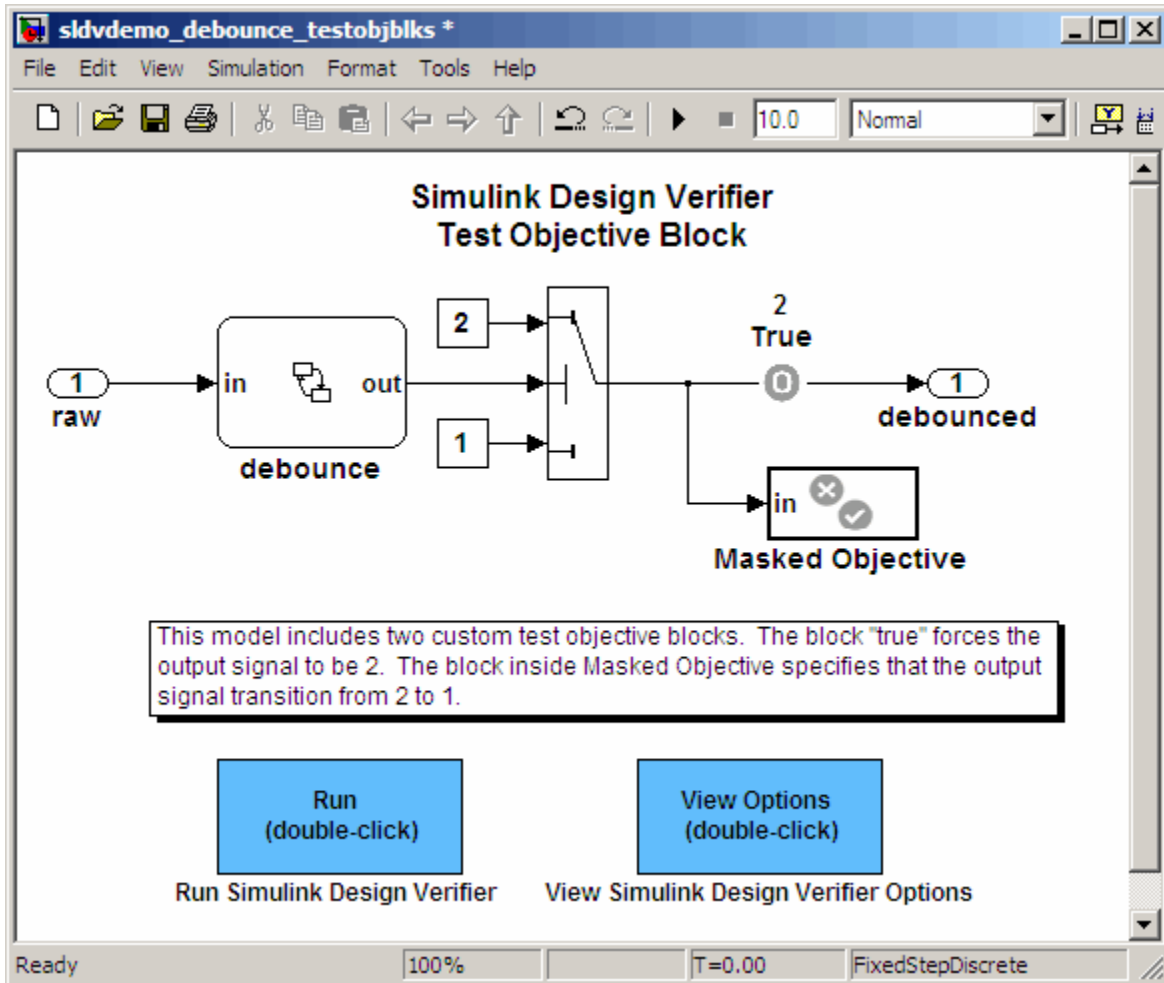


After the analysis, the variable-size signals have a wider line design. `double (5)` in this example indicates the data type and allocation for that signal.

### **Simulink Design Verifier Coverage**

If you select **Simulink Design Verifier**, the analysis collects coverage data for all Simulink Design Verifier blocks in your model.

For an example of how this works, consider the `sldvdemo_debounce_testobjblks` model.



This model contains two Test Objective blocks:

- The True block defines a property that the signal have a value of 2.
- The Edge block, in the Masked Objective subsystem, describes the property where the output of the AND block in the Masked Objective subsystem changes from 2 to 1.

The Simulink Design Verifier software analyzes this model and produces a harness model that contains test cases that achieve certain test objectives. To see if the original model achieves those objectives, simulate the harness model and collect model coverage data. The model coverage tool analyzes any decision points or values within an interval that you specify in the Test Objective block.

In this example, the coverage report shows that you achieved 100% coverage of the True block because the signal value was 2 at least once. The signal value was 2 in 6 out of 14 time steps.

**Design Verifier Test Objective block "True"**

**Parent:** [sldvdemo debounce testobjblks harness/Test Unit \(copied from sldvdemo debounce testobjblks\)](#)

<b>Metric</b>	<b>Coverage</b>
Test Objective	100% (1/1) objective outcomes

**Points/Intervals analyzed:**

Point : 2	6/14
-----------	------

The input signal to the Edge block achieved a value of True once out of 14 time steps.

**Design Verifier Test Objective block "[Edge](#)"**

**Parent:** [sldvdemo debounce testobjblks harness/Test Unit \(copied from sldvdemo debounce testobjblks\)/Masked Objective](#)

<b>Metric</b>	<b>Coverage</b>
Test Objective	100% (1/1) objective outcomes

**Points/Intervals analyzed:**

Point : T

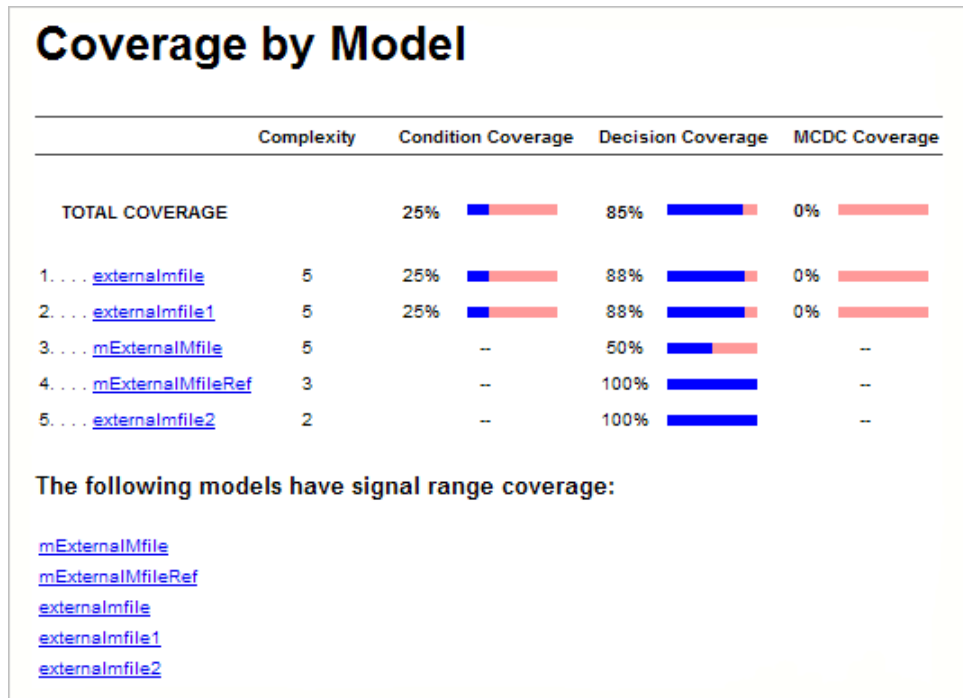
1/14

## Model Summary Reports

If the top-level model contains Model blocks or calls external files, the software creates a model summary coverage report named *model\_name\_summary\_cov.html*. The title of this report is **Coverage by Model**.

The summary report lists and provides links to coverage reports for all Model block referenced models and external files called by Embedded MATLAB code in the model. For more information, see “External MATLAB File Coverage Reports” on page 17-40.

The following graphic shows an example of a model summary report. It contains links to the model coverage report (`mExternalMfile`), a report for the Model block (`mExternalMfileRef`), and three external files called from the model (`externalmfile`, `externalmfile1`, and `externalmfile2`).





## Model Reference Coverage Reports

If your top-level model references a model in a Model block, the software creates a separate report, named *reference\_model\_name\_cov.html*, that includes coverage for the referenced model. This report has the same format as the “Model Coverage Reports” on page 17-3. Coverage results are recorded as if the referenced model was a standalone model; the report gives no indication that the model is referenced in a Model block.

## External MATLAB File Coverage Reports

If your top-level model calls any external MATLAB files, select **Coverage for External Embedded MATLAB files** on the **Coverage** tab of the Coverage Settings dialog box. The software creates a report, named *MATLAB\_file\_name\_cov.html*, for each distinct file called from the model. If there are several calls to a given file from the model, the software creates only one report for that file, but it accumulates coverage from all the calls to the file. The external MATLAB file coverage report does not include information about what parts of the model call the external file.

The first section of the external MATLAB file coverage report contains summary information about the external file, similar to the model coverage report.

# Coverage Report for externalmfile1

## Embedded MATLAB File Information

Last Saved 03-Oct-2008 18:01:02

## Simulation Optimization Options

Inline Parameters off  
 Block Reduction forced off  
 Conditional Branch Optimization on

## Coverage Options

Logic block short circuiting off

## Tests

### Test 1

Started Execution: 02-Dec-2008 17:08:01

Ended Execution: 02-Dec-2008 17:08:02

## Summary

Model Hierarchy/Complexity:

		Test 1		
		D1	C1	MCDC
1. <a href="#">externalmfile1</a>	5	88%	25%	0%

The **Details** section reports coverage for the external file and the function in that file.

**Details:**

**1. Embedded MATLAB file "[externalmfile](#)"**

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	1	5
Decision (D1)	NA	88% (7/8) decision outcomes
Condition (C1)	NA	25% (1/4) condition outcomes
MCDC (C1)	NA	0% (0/6) conditions reversed the outcome

**Embedded MATLAB function "[externalmfile](#)"**

**Parent:** [externalmfile](#)

**Uncovered Links:**

Metric	Coverage
Cyclomatic Complexity	4
Decision (D1)	88% (7/8) decision outcomes
Condition (C1)	25% (1/4) condition outcomes
MCDC (C1)	0% (0/6) conditions reversed the outcome

The **Details** section also lists the content of the file, highlighting the code lines that have decision points or function definitions.

```
1  %#eml
2  function y = externalmfile1(u)
3
4  %   Copyright 2008 The MathWorks, Inc.
5
6  if u>1 && u<5
7      a = 2;
8  else
9      a = 3;
10 end
11
12 for i=1:5
13     a = a+2;
14 end
15
16 y = a+localtest(a);
17
18 [x,y] = pol2cart(u,u);
19 [y2,y3] = cart2pol(x,y);
20
21 function y = localtest(u)
22
23 y = 0;
24 flg = true;
25 while flg
26     u = u/2;
27     y = y+1;
28     flg = u>2;
29 end
30
```

Coverage results for each of the highlighted code lines follow in the report. The following graphic shows a portion of these coverage results from the preceding code example.

**#2: function y = externalmfile1(u)****Decisions analyzed:**

function y = externalmfile1(u)	100%
executed	102/102

**#6: if u>1 && u<5****Decisions analyzed:**

if u>1 && u<5	50%
false	102/102
true	0/102

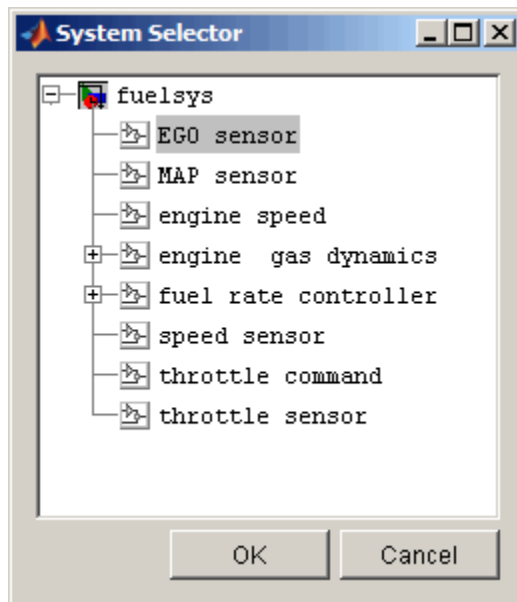
## Subsystem Coverage Reports

In the Coverage Settings dialog box, when you select **Coverage for this model**, you can click **Select Subsystem** to request coverage for only the selected subsystem in the model. The software creates a model coverage report for the top-level model, but includes coverage results only for the subsystem.

However, if the top-level model calls any external files and you select **Coverage for External Embedded MATLAB files** in the Coverage Settings dialog box, the results include coverage for all external files called from:

- The subsystem for which you are recording coverage
- The top-level model that includes the subsystem

For example, in the `fuelsys` model, you click **Select Subsystem**, and select coverage for the EGO sensor subsystem.




The report is similar to the model coverage report, except that it includes only results for the EGO sensor subsystem and its contents.

## Coverage Report for fuelsys

•  
•  
•

### Summary

Model Hierarchy/Complexity: Test 1  
D1

1. [EGO sensor](#)      2 50% 

### Details:

#### 1. Subsystem "[EGO sensor](#)"

Parent: [/fuelsys](#)

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	1	2
Decision (D1)	NA	50% (1/2) decision outcomes

#### Test 1

Started Execution: 02-Dec-2008 18:05:02  
Ended Execution: 02-Dec-2008 18:05:52



# Using Model Coverage Commands

---

- “About Model Coverage Commands” on page 18-2
- “Creating Tests with cvtest” on page 18-3
- “Running Tests with cvsim” on page 18-6
- “Creating HTML Reports with cvhtml” on page 18-8
- “Saving Test Runs to a File with cvsave” on page 18-9
- “Loading Stored Coverage Test Results with cvload” on page 18-10
- “Coverage Script Example” on page 18-11
- “Using Model Coverage Commands for Referenced Models” on page 18-12

## About Model Coverage Commands

Using model coverage commands lets you automate the entire model coverage process with MATLAB scripts. You can use model coverage commands to set up model coverage tests, execute them in simulation, and store and report the results. For a list of the model coverage commands that the Simulink Verification and Validation software provides, see Chapter 23, “Function Reference”.

The following sections describe a workflow for using model coverage commands to create, run, store, and report model coverage tests.

## Creating Tests with cvtest

The `cvtest` command creates a test specification object. Once you create the object, you simulate it with the `cvsim` command.

The call to `cvtest` has the following default syntax:

```
cvto = cvtest(root)
```

`root` is the name of, or a handle to, a Simulink model or a subsystem of a model. `cvto` is a handle to the resulting test specification object. Only the specified model or subsystem and its descendants are subject to model coverage.

To create a test object with a specified label (used for reporting results):

```
cvto = cvtest(root, label)
```

To create a test with a setup command:

```
cvto = cvtest(root, label, setupcmd)
```

You execute the setup command in the base MATLAB workspace, just prior to running the instrumented simulation. Use this command for loading data prior to a test.

The returned `cvtest` object, `cvto`, has the following structure.

Field	Description
<code>id</code>	Read-only internal data-dictionary ID
<code>modelcov</code>	Read-only internal data-dictionary ID
<code>rootPath</code>	Name of the system or subsystem for analysis
<code>label</code>	String for reporting results
<code>setupCmd</code>	Command executed prior to simulation

<b>Field</b>	<b>Description</b>
<code>settings.condition</code>	Set to 1 for condition coverage
<code>settings.decision</code>	Set to 1 for decision coverage
<code>settings.designverifier</code>	Set to 1 for coverage for Simulink Design Verifier blocks.
<code>settings.mcdc</code>	Set to 1 for MCDC coverage
<code>settings.sigrange</code>	Set to 1 for signal range coverage
<code>settings.sigsize</code>	Set to 1 for signal size coverage.
<code>settings.tableExec</code>	Set to 1 for lookup table coverage
<code>modelRefSettings.enable</code>	String specifying one of the following values: <ul style="list-style-type: none"> <li>• <code>Off</code> — Disables coverage for all referenced models</li> <li>• <code>all</code> — Enables coverage for all referenced models</li> <li>• <code>filtered</code> — Enables coverage for only referenced models not listed in the <code>excludedModels</code> subfield</li> </ul>
<code>modelRefSettings.excludeTopModel</code>	Set to 1 for excluding coverage for the top model
<code>modelRefSettings.excludedModels</code>	String specifying a comma-separated list of referenced models for which coverage is disabled when <code>modelRefSettings.enable</code> specifies <code>filtered</code>
<code>emlSettings.enableExternal</code>	Set to 1 to enable coverage for external program files called by Embedded MATLAB functions in your model
<code>options.forceBlockReduction</code>	Set to 1 to override the Simulink <b>Block reduction</b> parameter if it is enabled.



## Running Tests with `cvsim`

Use the `cvsim` command to simulate a test specification object.

---

**Note** You do not have to enable model coverage reporting (see “Creating and Running Test Cases” on page 15-32) to use the `cvsim` command.

---

The call to `cvsim` has the following default syntax:

```
cvdo = cvsim(cvto)
```

This command executes the `cvtest` object `cvto` by starting a simulation run for the corresponding model. The results are returned in the `cvdata` object `cvdo`. When recording coverage for multiple models in a hierarchy, `cvsim` returns its results in a `cv.cvdatagroup` object.

You can also control the simulation in a `cvsim` command by using parameters for the Simulink `sim` command:

- The following command returns the simulation time vector `t`, matrix of state values `x`, and matrix of output values `y`.

```
[cvdo,t,x,y] = cvsim(cvto)
```

- The following command overrides default simulation values with new values.

```
[cvdo,t,x,y] = cvsim(cvto, timespan, options)
```

For descriptions of the parameters `t`, `x`, `y`, `timespan`, and `options` in the previous examples, see documentation for the Simulink command.

You can execute multiple test objects with the `cvsim` command. The following command executes a set of coverage test objects, `cvto1`, `cvto2`, ... and returns the results in a set of `cvdata` objects, `cvdo1`, `cvdo2`, ....

```
[cvdo1, cvdo2, ...] = cvsim(cvto1, cvto2, ...)
```

You can also use the `cvsim` command to create and execute a `cvtest` object in one command:

```
[cvdo,t,x,y] = cvsim(cvto, label, setupcmd)
```

## Creating HTML Reports with `cvhtml`

Once you run a test in simulation with `cvsim`, results are saved to `cv.cvdatagroup` or `cvdata` objects in the base MATLAB workspace. Use the `cvhtml` command to create an HTML report of these objects.

The following command creates an HTML report of the coverage results in the `cvdata` object `cvdo`. The results are written to the file `file` in the current MATLAB folder.

```
cvhtml(file, cvdo)
```

The following command creates a combined report of several `cvdata` objects:

```
cvhtml(file, cvdo1, cvdo2, ...)
```

The results from each object are displayed in a separate column of the HTML report. Each `cvdata` object must correspond to the same root model or subsystem, or the function produces errors.

You can specify the detail level of the report with the value of `detail`, an integer between 0 and 3:

```
cvhtml(file, cvdo1, cvdo2, ..., detail)
```

Higher numbers for `detail` indicate greater detail. The default value is 2.



## Saving Test Runs to a File with `cvsave`

Once you run a test with `cvsim`, save its coverage tests and results to a file with the function `cvsave`:

```
cvsave(filename, model)
```

Save all the tests and results related to `model` in the text file `filename.cvt`:

```
cvsave(filename, cvto1, cvto2, ...)
```

Save the tests in the text file `filename.cvt`. Information about the referenced models is also saved.

You can save specified `cvdata` objects to file. The following example saves the tests, test results, and referenced models' structure in `cvdata` objects to the text file `filename.cvt`:

```
cvsave(filename, cvdo1, cvdo2, ...)
```

## Loading Stored Coverage Test Results with `cvload`

The `cvload` command loads into memory the coverage tests and results stored in a file by the `cvsave` command. The following example loads the tests and data stored in the text file `filename.cvt`:

```
[cvtos, cvdos] = cvload(filename)
```

The `cvtest` objects that are successfully loaded are returned in `cvtos`, a cell array of `cvtest` objects. The `cvdata` objects that are successfully loaded are returned in `cvdos`, a cell array of `cvdata` objects. `cvdos` has the same size as `cvtos`, but can contain empty elements if a particular test has no results.

In the following example, if `restorettotal` is 1, the cumulative results from prior runs are restored:

```
[cvtos, cvdos] = cvload(filename, restorettotal)
```

If `restorettotal` is unspecified or 0, the model's cumulative results are cleared.

### **cvload Special Considerations**

When using the `cvload` command, be aware of the following considerations:

- When a model with the same name exists in the coverage database, only the compatible results are loaded from the file. They reference the existing model to prevent duplication.
- When the Simulink models referenced in the file are open but do not exist in the coverage database, the coverage tool resolves the links to the models that are already open.
- When you are loading several files that reference the same model, only the results that are consistent with the earlier files are loaded.

## Coverage Script Example

The following example is a portion of `simcovdemo2.m`, located in the coverage root folder. This example demonstrates common model coverage commands.

```
mdl = 'slvndemo_ratelim_harness';

testObj1 = cvtest([mdl, '/Adjustable Rate Limiter']);
testObj1.label = 'Gain within slew limits';
testObj1.setupCmd = 'load(''within_lim.mat'');';
testObj1.settings.mcdc = 1;

testObj2 = cvtest([mdl, '/Adjustable Rate Limiter']);
testObj2.label='Rising gain that temporarily exceeds slew limit';
testObj2.setupCmd = 'load(''rising_gain.mat'');';
testObj2.settings.mcdc = 1;

[dataObj1,T,X,Y] = cvsim(testObj1,[0 2]);
[dataObj2,T,X,Y] = cvsim(testObj2,[0 2]);

cvhtml('ratelim_report',dataObj1,dataObj2);
cumulative = dataObj1+dataObj2;
cvsave('ratelim_testdata',cumulative);
```

In this example, you create two `cvtest` objects, `testObj1` and `testObj2`, and simulate them according to their specifications. Each `cvtest` object uses the `setupCmd` property to load a data file before simulation. Decision coverage is enabled by default. MCDC coverage is enabled as well. After simulation, you use `cvhtml` to display the coverage results for two tests and the cumulative coverage. Lastly, you compute cumulative coverage with the `+` operator and save the results. For another detailed example of how to use the model coverage commands, at the MATLAB command prompt, enter `simcovdemo`.

## Using Model Coverage Commands for Referenced Models

In this section...
“Introduction” on page 18-12
“Creating a Test Group with <code>cv.cvtestgroup</code> ” on page 18-15
“Running Tests with <code>cvsimref</code> ” on page 18-15
“Extracting Results from <code>cv.cvdatagroup</code> ” on page 18-16

### Introduction

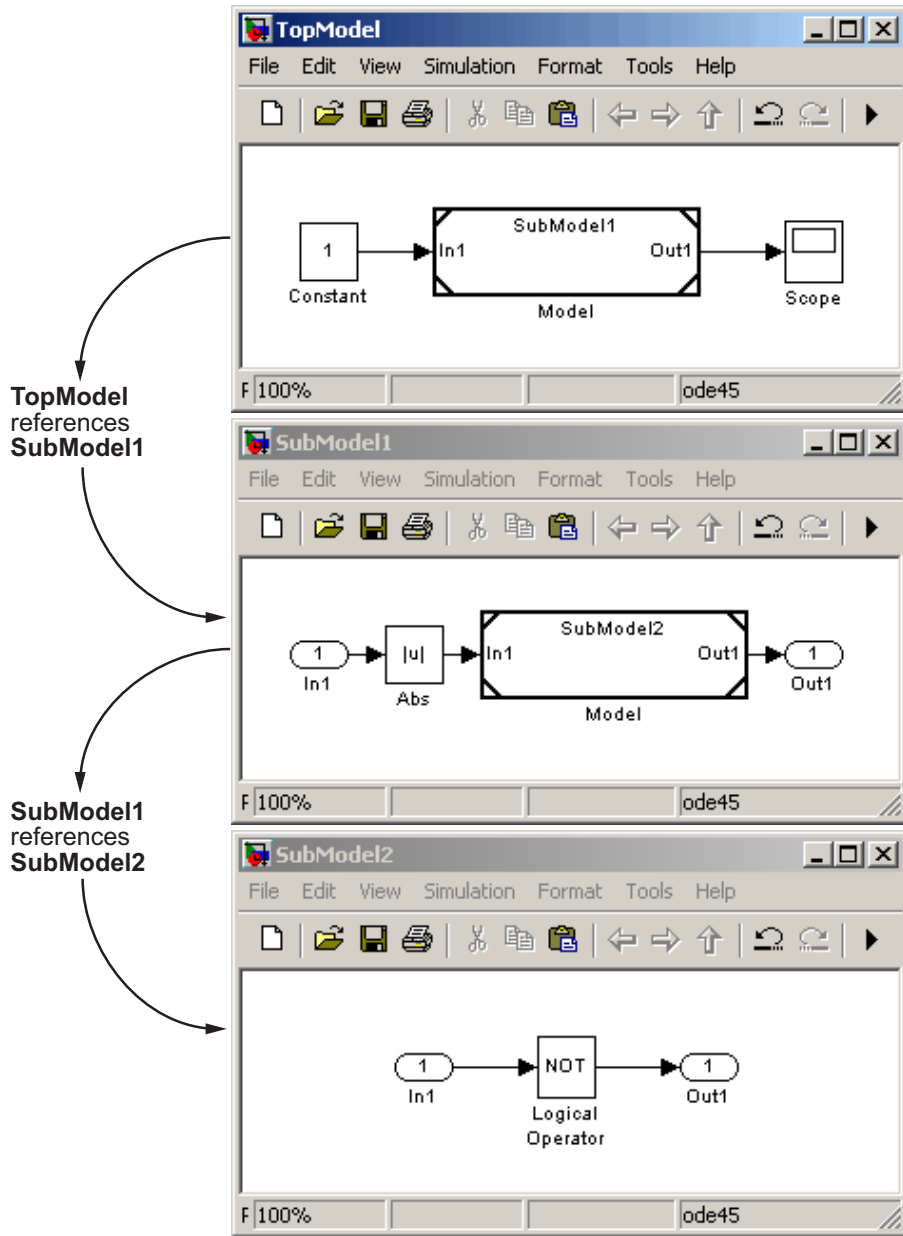
Using Simulink software, you can include one model in another with Model blocks. Each Model block represents a reference to another model, called a *referenced model* or *submodel*. A referenced model can contain Model blocks that reference other models. You can construct a hierarchy of referenced models, in which the topmost model is called the *top model*. For more information, see “Referencing a Model” in the *Simulink User’s Guide*.

Model coverage supports referenced models that operate in Normal mode. You can record coverage only for those Model blocks whose **Simulation mode** parameter specifies **Normal**. You can use model coverage commands to record coverage for referenced models (see Chapter 18, “Using Model Coverage Commands”). However, if you want to record different types of coverage for models in a hierarchy, you must use the `cvsimref` function. The following steps describe a basic workflow for using this function to obtain model coverage results for Model blocks.

Step	Description	See...
1	Use <code>cv.cvtestgroup</code> to group together test specification objects that correspond to each model in a hierarchy.	“Creating a Test Group with <code>cv.cvtestgroup</code> ” on page 18-15
2	Use <code>cvsimref</code> to simulate the top model in a hierarchy and record coverage results for its referenced models.	“Running Tests with <code>cvsimref</code> ” on page 18-15

<b>Step</b>	<b>Description</b>	<b>See...</b>
3	Use <code>cv.cvdatagroup</code> to extract the coverage data objects that correspond to each model in a hierarchy.	“Extracting Results from <code>cv.cvdatagroup</code> ” on page 18-16

The next sections illustrate how to complete each of these steps using the following model hierarchy.



## Creating a Test Group with `cv.cvtestgroup`

The `cvtest` command creates a test specification object for a Simulink model (see “Creating Tests with `cvtest`” on page 18-3). If your model references other models, you might use a different test specification object for each model in the hierarchy. In this case, the `cv.cvtestgroup` object allows you to group together multiple test specification objects. After you create a group of test specification objects, you simulate it using the `cvsimref` function.

For example, suppose that you create a different test specification object for each of the models in your hierarchy:

```
cvto1 = cvtest('TopModel1')
cvto2 = cvtest('SubModel11')
cvto3 = cvtest('SubModel12')
```

The following command creates a test group object named `cvtg`, which contains all the `cvtest` objects associated with your model hierarchy:

```
cvtg = cv.cvtestgroup(cvto1, cvto2, cvto3)
```

A `cv.cvtestgroup` object provides methods, such as `add` and `get`, so that you can customize the contents of the `cv.cvtestgroup` object to meet your needs. For more information, see the documentation for the `cv.cvtestgroup` function.

## Running Tests with `cvsimref`

Once you create a test group object, you simulate it with the `cvsimref` function.

---

**Note** You must use the `cvsimref` function to record coverage for referenced models in a hierarchy.

---

The call to `cvsimref` has the following default syntax:

```
cvdg = cvsimref(topModelName, cvtg)
```

This command executes the test group object `cvtg` by simulating the top model in the corresponding model hierarchy, `topModelName`. It returns the coverage results in a `cv.cvdatabroup` object named `cvdg`.

You can use parameters from the Simulink `sim` function in a `cvsimref` command to control the simulation:

- To return the simulation time vector `t`, matrix of state values `x`, and matrix of output values `y`:

```
[cvdg,t,x,y] = cvsimref(topModelName, cvtg)
```

- To override default simulation values with new values:

```
[cvdg,t,x,y] = cvsimref(topModelName, cvtg, timespan, options)
```

For descriptions of the parameters `t`, `x`, `y`, `timespan`, and `options`, see the documentation for the `sim` function in the *Simulink Reference*.

## Extracting Results from `cv.cvdatabroup`

Once you simulate a test group with `cvsimref`, the function returns results that reside in a `cv.cvdatabroup` object. The data group object contains multiple `cvdata` objects, each of which corresponds to coverage results for a particular model in the hierarchy.

A `cv.cvdatabroup` object provides methods, such as `allNames` and `get`, so that you can extract individual `cvdata` objects. For example, enter the following command to obtain a cell array that lists all model names associated with the data group `cvdg`:

```
modelName = cvdg.allNames
```

To extract the `cvdata` objects that correspond to the particular models, enter:

```
cvdo1 = cvdg.get('TopModel')  
cvdo2 = cvdg.get('SubModel1')  
cvdo3 = cvdg.get('SubModel2')
```

After you extract the individual `cvdata` objects, you can use other model coverage commands to use the coverage data of a particular model. For example, you can use the `cvhtml` function to create and display an HTML



report of the coverage results (see “Creating HTML Reports with cvhtml” on page 18-8).



# Customizing the Model Advisor

---

- Chapter 19, “Overview of the Model Advisor”
- Chapter 20, “Authoring Custom Checks”
- Chapter 21, “Creating Custom Configurations by Organizing Checks and Folders”
- Chapter 22, “Deploying Custom Configurations”



# Overview of the Model Advisor

---

- “Why Use and Customize the Model Advisor?” on page 19-2
- “Customizing and Using the Model Advisor Workflow” on page 19-4
- “Before Customizing the Model Advisor” on page 19-5

## Why Use and Customize the Model Advisor?

In this section...
“About the Model Advisor” on page 19-2
“Customizing the Model Advisor” on page 19-2

### About the Model Advisor

The Model Advisor is a GUI that provides a way for you to check a Simulink model or subsystem for consistent modeling guidelines, using MathWorks checks. Using the checks, you can easily apply these guidelines across projects and development teams. For more information, see “Consulting the Model Advisor” in the Simulink documentation.

The Model Advisor includes MathWorks checks, which help you define and implement consistent design guidelines. Running the checks reviews your model for conditions and configuration settings that cause inaccurate or inefficient simulation and code generation of the system that the model represents. The Model Advisor displays different MathWorks checks depending on which products you have installed. For more information on individual checks, see:

- “Simulink Checks”
- “Real-Time Workshop® Checks”
- “Simulink® Verification and Validation Checks” on page 27-2

### Customizing the Model Advisor

The Simulink Verification and Validation product allows you to extend the capabilities of the Model Advisor. Using Model Advisor APIs and the Model Advisor Configuration Editor, you can:

- Customize the behavior of the Model Advisor by defining your own custom checks, and writing your own callback functions.
- Organize checks and folders to create custom Model Advisor configurations.

- Create multiple custom configurations that you use for different projects or modeling guidelines, and switch between these configurations in the Model Advisor.
- Deploy the custom configurations to your users.

For more information, see “Customizing and Using the Model Advisor Workflow” on page 19-4.

## Customizing and Using the Model Advisor Workflow

To customize and use the Model Advisor, perform the following high-level tasks:

- 1** Review the information in “Before Customizing the Model Advisor” on page 19-5.
- 2** Optionally, author custom checks in a customization file. For detailed information, see Chapter 20, “Authoring Custom Checks”.
- 3** Organize checks into new and existing folders to create custom configurations. To organize the Model Advisor, use the Model Advisor Configuration Editor or create a customization file. For detailed information, see Chapter 21, “Creating Custom Configurations by Organizing Checks and Folders”.
- 4** Optionally, deploy custom configurations. For detailed information, see Chapter 22, “Deploying Custom Configurations”.
- 5** Verify that models comply with modeling guidelines using the Model Advisor. For detailed information, see “Consulting the Model Advisor”.



## Before Customizing the Model Advisor

Before customizing the Model Advisor:

- If you want to create custom checks, know how to create a MATLAB script. For more information, see “Scripts” in the MATLAB documentation.
- If you want to create custom checks, understand how to access model constructs that you want to check. For example, know how to find block and model parameters. For more information on using utilities for creating check callbacks, see “Common Utilities for Authoring Checks” on page 20-23.
- Identify which MathWorks checks you want to include in your custom Model Advisor configuration.

When you are ready to create a custom configuration, follow the “Customizing and Using the Model Advisor Workflow” on page 19-4. Each section provides you with detailed examples of how to create custom checks and configurations in the Model Advisor.



# Authoring Custom Checks

---

- “Authoring Checks Workflow” on page 20-2
- “Customization File Overview” on page 20-3
- “Register Checks and Process Callbacks” on page 20-6
- “Defining Custom Checks” on page 20-11
- “Creating Callback Functions and Results” on page 20-22

## Authoring Checks Workflow

- 1** On your MATLAB path, create a *customization file* called `sl_customization.m`. In this file, create a `sl_customization()` function to register the custom checks that you create and optional process callbacks with the Model Advisor. For detailed information, see “Register Checks and Process Callbacks” on page 20-6.
- 2** Define custom checks and where they appear in the Model Advisor. For detailed information, see “Defining Custom Checks” on page 20-11.
- 3** Specify what actions you want the Model Advisor to take for the custom checks by creating a check callback function for each custom check. For detailed information, see “Creating Callback Functions and Results” on page 20-22.
- 4** Optionally, specify what automatic fix operations the Model Advisor performs by creating an action callback function. For detailed information, see “Action Callback Function” on page 20-37.
- 5** Optionally, specify startup and post-execution actions by creating a process callback function. For detailed information, see “Defining Startup and Post-Execution Actions Using Process Callback Functions” on page 20-8.

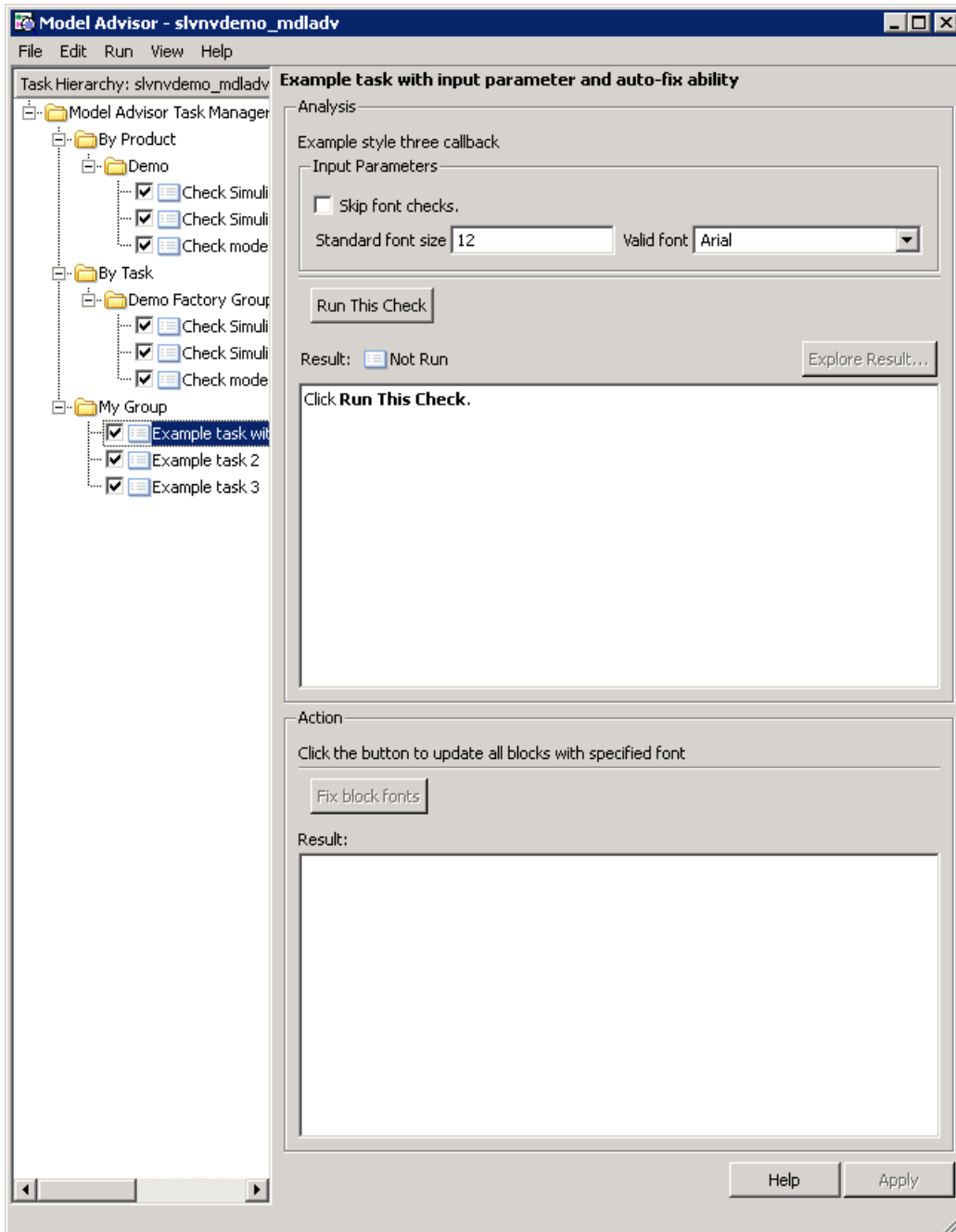
## Customization File Overview

A *customization file* is a MATLAB file that you create and name `sl_customization.m`. The `sl_customization.m` file contains a set of functions for registering and defining custom checks, tasks, and groups. To set up the `sl_customization.m` file, follow the guidelines in this table.

Function	Description	When Required
<code>sl_customization()</code>	Registers custom checks, tasks, folders, and callbacks with the Simulink customization manager at startup (see “Register Checks and Process Callbacks” on page 20-6).	Required for all customizations to the Model Advisor.
One or more check definitions	Defines all custom checks (see “Defining Custom Checks” on page 20-11).	Required for custom checks and to add custom checks to the <b>By Product</b> folder.
Check callback functions	Defines the actions of the custom checks (see “Creating Callback Functions and Results” on page 20-22).	Required for custom checks. You must write one callback function for each custom check.
One or more calls to check input parameters	Specifies input parameters to custom checks (see “Defining Check Input Parameters” on page 20-16).	Optional.
One or more calls to check list views	Specifies calls to the Model Advisor Result Explorer for custom checks (see “Defining Model Advisor Result Explorer Views” on page 20-18).	Optional.

<b>Function</b>	<b>Description</b>	<b>When Required</b>
One or more calls to check actions	Specifies actions the software performs for custom checks (see “Defining Check Actions” on page 20-19 and “Action Callback Function” on page 20-37).	Optional.
One process callback function	Specifies actions to be performed at startup and post-execution time (see “Defining Startup and Post-Execution Actions Using Process Callback Functions” on page 20-8).	Optional.

The following is an example of a custom configuration of the Model Advisor that has custom checks defined in custom folders. The selected check includes input parameters, list view parameters, and actions.



## Register Checks and Process Callbacks

### In this section...

“Create `sl_customization` Function” on page 20-6

“Registering Checks and Process Callbacks” on page 20-6

“Defining Startup and Post-Execution Actions Using Process Callback Functions” on page 20-8

### Create `sl_customization` Function

To add checks to the Model Advisor, on your MATLAB path, in the `sl_customization.m` file, create the `sl_customization()` function.

---

#### Tip

- You can have more than one `sl_customization.m` file on your MATLAB path.
- Do not place an `sl_customization.m` file that customizes checks and folders in the Model Advisor in your root MATLAB folder or any of its subfolders, except for the `matlabroot/work` folder. Otherwise, the Model Advisor ignores the customizations that the file specifies.

---

The `sl_customization` function accepts one argument, a customization manager object, as in this example:

```
function sl_customization(cm)
```

The customization manager object includes methods for registering custom checks and process callbacks. Use these methods to register customizations specific to your application, as described in the following sections.

### Registering Checks and Process Callbacks

To register custom checks and process callbacks, the customization manager includes the following methods:



- `addModelAdvisorCheckFcn` (*@checkDefinitionFcn*)

Registers the checks that you define in *checkDefinitionFcn* to the **By Product** folder of the Model Advisor.

The *checkDefinitionFcn* argument is a handle to the function that defines all custom checks that you want to add to the Model Advisor as instances of the `ModelAdvisor.Check` class (see “Defining Custom Checks” on page 20-11).

- `addModelAdvisorProcessFcn` (*@modelAdvisorProcessFcn*)

Registers the process callback function for the Model Advisor checks (see “Defining Startup and Post-Execution Actions Using Process Callback Functions” on page 20-8).

---

**Caution** The Model Advisor registers only one process callback function. If you have more than one `sl_customization.m` file on your MATLAB path, the Model Advisor registers the process callback function from the `sl_customization.m` file that has the highest priority.

---

---

**Note** The `@` sign defines a function handle that MATLAB calls. For more information, see “At — @” in the MATLAB documentation.

---

## Model Advisor Code Example: Registering Custom Checks and Process Callbacks

The following code example registers custom checks and a process callback function:

```
function sl_customization(cm)

% register custom checks
cm.addModelAdvisorCheckFcn(@defineModelAdvisorChecks);

% register custom process callback
cm.addModelAdvisorProcessFcn(@ModelAdvisorProcessFunction);
```

---

**Note** If you add custom tasks and folders within the `sl_customization.m` file, include methods for registering the tasks and folders in the `sl_customization` function. For more information, see “Registering Tasks and Folders” on page 21-14.

---

## Defining Startup and Post-Execution Actions Using Process Callback Functions

The *process callback function* is an optional function that you use to configure the Model Advisor and process check results at run time. The process callback function specifies actions that the software performs at different stages of Model Advisor execution:

- **configure stage:** The Model Advisor executes `configure` actions at startup, after all checks and tasks have been initialized. At this stage, you can customize how the Model Advisor constructs lists of checks and tasks by modifying `Visible`, `Enable`, and `Value` properties. For example, you can remove, rename, and selectively display checks and tasks.
- **process\_results stage:** The Model Advisor executes `process_results` actions after checks complete execution. You can specify actions to examine and report on the results returned by check callback functions.

If you create a process callback function, you must register it, as described in “Register Checks and Process Callbacks” on page 20-6. The following sections provide mode information about defining your own process callback functions.

### Process Callback Function Arguments

The process callback function takes the following arguments.

Argument	I/O Type	Data Type	Description
stage	Input	Enumeration	Specifies the stages at which process callback actions are executed. Use this argument in a switch statement to specify actions for the stages <code>configure</code> and <code>process_results</code> .
system	Input	Path	Model or subsystem that the Model Advisor analyzes.
checkCellArray	Input/Output	Cell array	As input, the array of checks constructed in the check definition function. As output, the array of checks modified by actions in the <code>configure</code> stage.
taskCellArray	Input/Output	Cell array	As input, the array of tasks constructed in the task definition function. As output, the array of tasks modified by actions in the <code>configure</code> stage.

### Model Advisor Code Example: Process Callback Function

The following code is an example of a process callback function that specifies actions in the `configure` stage, to make only custom checks visible. In the `process_results` stage, this function displays information at the MATLAB command line for checks that do not pass.

```
% Process Callback Function
% Defines actions to execute at startup and post-execution
function [checkCellArray taskCellArray] = ...
    ModelAdvisorProcessFunction(stage, system, checkCellArray, taskCellArray)
switch stage
    % Specify the appearance of the Model Advisor window at startup
```

```
case 'configure'
    for i=1:length(checkCellArray)
        % Hide all checks that do not belong to custom folder
        if isempty(strfind(checkCellArray{i}.ID, 'mathworks.example'))
            checkCellArray{i}.Visible = false;
            checkCellArray{i}.Value = false;
        end
    end
end
% Specify actions to perform after the Model Advisor completes execution
case 'process_results'
    for i=1:length(checkCellArray)
        % Print message if check does not pass
        if checkCellArray{i}.Selected && (strcmp(checkCellArray{i}.Title, ...
            'Check Simulink window screen color'))
            mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system);
            % Verify whether the check was run and if it failed
            if mdladvObj.verifyCheckRan(checkCellArray{i}.ID)
                if ~mdladvObj.getCheckResultStatus(checkCellArray{i}.ID)
                    % Display text in MATLAB Command Window
                    disp(['Example message from Model Advisor Process'...
                        ' callback.']);
                end
            end
        end
    end
end
end
end
```

## Defining Custom Checks

### In this section...

“About Custom Checks” on page 20-11  
 “Contents of Check Definitions” on page 20-11  
 “Displaying and Enabling Checks” on page 20-13  
 “Defining Where Custom Checks Appear” on page 20-14  
 “Model Advisor Code Example: Check Definition Function” on page 20-15  
 “Defining Check Input Parameters” on page 20-16  
 “Defining Model Advisor Result Explorer Views” on page 20-18  
 “Defining Check Actions” on page 20-19

### About Custom Checks

You can create a custom check to use in the Model Advisor. Creating custom checks provides you with the ability to specify which conditions and configuration settings the Model Advisor reviews.

You define custom checks in one or more functions that specify the properties of each instance of the `ModelAdvisor.Check` class. Define one instance of this class for each custom check that you want to add to the Model Advisor, and register the custom check as described in “Register Checks and Process Callbacks” on page 20-6.

---

**Tip** You can add a check to multiple folders by creating a *task*. For more information, see “Adding a Check to Custom or Multiple Folders Using Tasks” on page 21-16.

---

The following sections describe how to define custom checks.

### Contents of Check Definitions

When you define a Model Advisor check, it contains the information listed in the following table.

<b>Contents</b>	<b>Description</b>
Check ID (required)	Uniquely identifies the check. The Model Advisor uses this id to access the check.
Handle to check callback function (required)	Function that specifies the contents of a check.
Check name (recommended)	Creates a name for the check that the Model Advisor displays.
Check properties (optional)	<p>Creates a user interface with the check. When adding checks as tasks, the Model Advisor uses the task properties instead of the check properties, except for <code>Visible</code> and <code>LicenseName</code>. For more information, see <code>ModelAdvisor.Check</code> and <code>ModelAdvisor.Task</code>.</p> <hr/> <p><b>Tip</b> When you add checks to the Model Advisor as tasks, specify only the required properties of a check, because the task definition includes the additional properties. For example, you define the description of the check in the task definition using the <code>ModelAdvisor.Task.Description</code> property instead of the <code>ModelAdvisor.Check.TitleTips</code> property.</p> <hr/>
Input Parameters (optional)	Adds input parameters that request input from the user. The Model Advisor uses the input to perform the check.

Contents	Description
Action (optional)	Adds automatic fixing action.
<b>Explore Result</b> button (optional)	Adds the <b>Explore Result</b> button that the user clicks to open the Model Advisor Result Explorer.

## Displaying and Enabling Checks

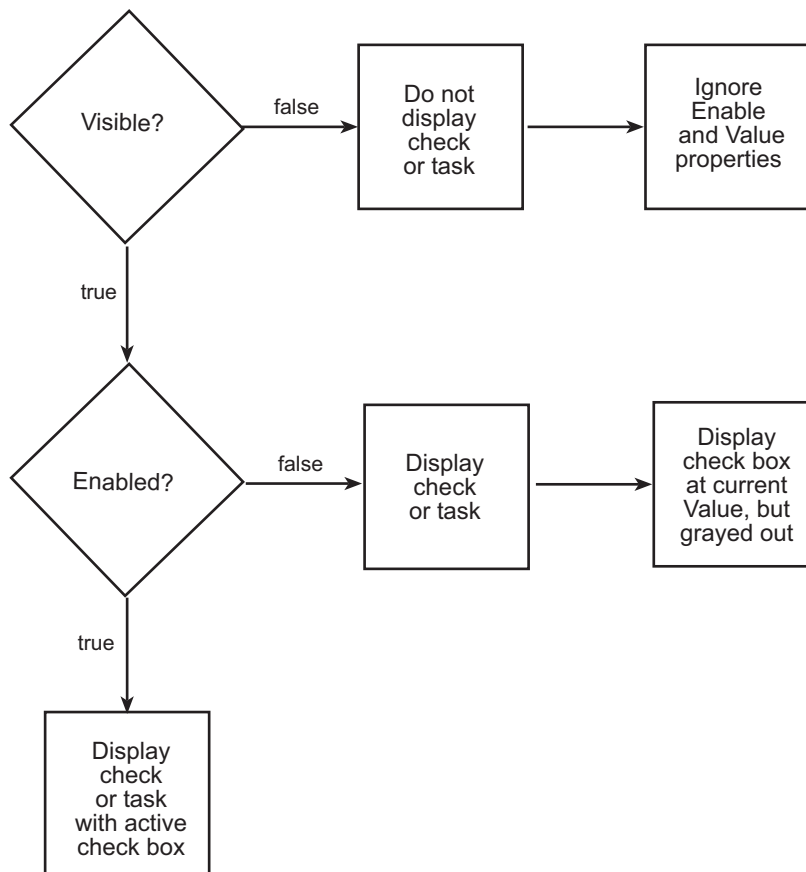
You can create a check and specify how it appears in the Model Advisor. You can define when to display a check, or whether a user can select or clear a check using the `Visible`, `Enable`, and `Value` properties of the `ModelAdvisor.Check` class.

---

**Note** When adding checks to the Model Advisor as tasks, specify these properties in the `ModelAdvisor.Task` class. If you specify the properties in both `ModelAdvisor.Check` and `ModelAdvisor.Task`, the `ModelAdvisor.Task` properties take precedence, except for the `Visible` and `LicenseName` properties. For more information, see `ModelAdvisor.Task`.

---

Modify the behavior of the `Visible`, `Enable`, and `Value` properties in a process callback function (see “Defining Startup and Post-Execution Actions Using Process Callback Functions” on page 20-8). The following chart illustrates how these properties interact.



## Defining Where Custom Checks Appear

Specify where the Model Advisor places custom checks using the following guidelines:

- To place a check in a new folder in the **Model Advisor** root, use the `ModelAdvisor.Group` class. See “Defining Custom Tasks” on page 21-15.
- To place a check in a new folder in the **By Task** folder, use the `ModelAdvisor.FactoryGroup` class. See “Defining Custom Tasks” on page 21-15.



- To place a check in the **By Product** folder, use the `ModelAdvisor.Root.publish` method.

## Model Advisor Code Example: Check Definition Function

The following is an example of a function that defines the custom checks associated with the callback functions described in “Creating Callback Functions and Results” on page 20-22. The check definition function returns a cell array of custom checks to be added to the Model Advisor.

The check definitions in the example use are used the tasks described in “Defining Custom Tasks” on page 21-15.

```
% Defines custom Model Advisor checks
function defineModelAdvisorChecks

% Sample check 1: Informational check
rec = ModelAdvisor.Check('mathworks.example.configManagement');
rec.Title = 'Informational check for model configuration management';
setCallbackFcn(rec, @modelVersionChecksumCallbackUsingFT,'None','StyleOne');
rec.CallbackContext = 'PostCompile';
mdladvRoot = ModelAdvisor.Root;
mdladvRoot.register(rec);

% Sample check 2: Basic Check with Pass/Fail Status
rec = ModelAdvisor.Check('mathworks.example.unconnectedObjects');
rec.Title = 'Check for unconnected objects';
setCallbackFcn(rec, @unconnectedObjectsCallbackUsingFT,'None','StyleOne');
mdladvRoot = ModelAdvisor.Root;
mdladvRoot.register(rec);

% Sample Check 3: Check with Subchecks and Actions
rec = ModelAdvisor.Check('mathworks.example.optimizationSettings');
rec.Title = 'Check safety-related optimization settings';
setCallbackFcn(rec, @OptimizationSettingCallback,'None','StyleOne');
% Define an automatic fix action for this check
modifyAction = ModelAdvisor.Action;
setCallbackFcn(modifyAction, @modifyOptimizationSetting);
modifyAction.Name = 'Modify Settings';
```

```

modifyAction.Description = ['Modify model configuration optimization' ...
                            ' settings that can impact safety.'];

modifyAction.Enable = true;
setAction(rec, modifyAction);
mdladvRoot = ModelAdvisor.Root;
mdladvRoot.register(rec);

```

## Defining Check Input Parameters

With input parameters, the check author can request input from the user for a Model Advisor check. Define input parameters using the `ModelAdvisor.InputParameter` class inside a custom check function (see “Defining Custom Checks” on page 20-11). You must define one instance of this class for each input parameter that you want to add to a Model Advisor check.

---

**Note** You do not have to create input parameters for every custom check.

---

## Specifying Input Parameter Layout

Specify the layout of input parameters in an input parameter definition. To place input parameters, use the following methods.

Method	Description
<code>ModelAdvisor.Check</code> <code>setInputParametersLayoutGrid</code>	Specifies the size of the input parameter grid.
<code>ModelAdvisor.InputParameter</code> <code>setRowSpan</code>	Specifies the number of rows the parameter occupies in the Input Parameter layout grid.
<code>ModelAdvisor.InputParameter</code> <code>setColSpan</code>	Specifies the number of columns the parameter occupies in the Input Parameter layout grid.

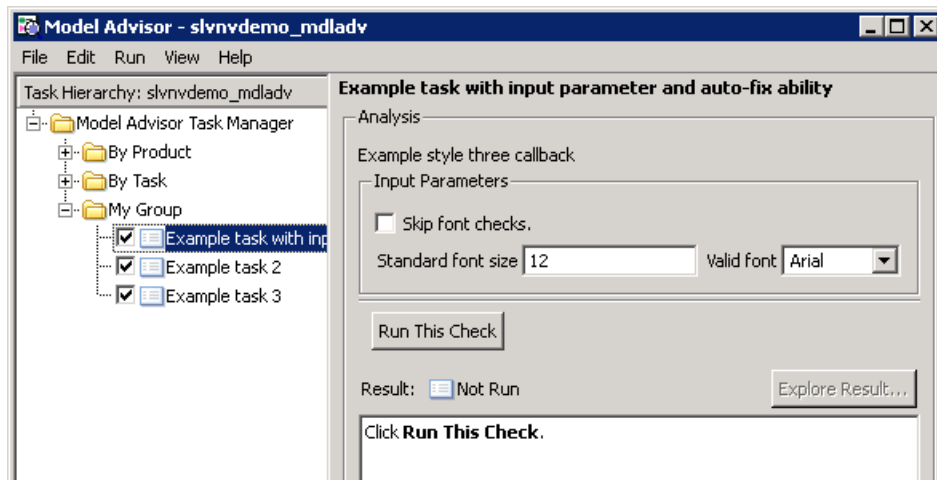
For information on using these methods, see the `ModelAdvisor.Check` and `ModelAdvisor.InputParameter` class documentation.

## Model Advisor Code Example: Input Parameter Definition

The following is an example of defining input parameters that you add to a custom check. You must include input parameter definitions inside a custom check definition (see “Model Advisor Code Example: Check Definition Function” on page 20-15). The following code, when included in a custom check definition, creates three input parameters.

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.setInputParametersLayoutGrid([3 2]);
% define input parameters
inputParam1 = ModelAdvisor.InputParameter;
inputParam1.Name = 'Skip font checks.';
inputParam1.Type = 'Bool';
inputParam1.Value = false;
inputParam1.Description = 'sample tooltip';
inputParam1.setRowSpan([1 1]);
inputParam1.setColSpan([1 1]);
inputParam2 = ModelAdvisor.InputParameter;
inputParam2.Name = 'Standard font size';
inputParam2.Value='12';
inputParam2.Type='String';
inputParam2.Description='sample tooltip';
inputParam2.setRowSpan([2 2]);
inputParam2.setColSpan([1 1]);
inputParam3 = ModelAdvisor.InputParameter;
inputParam3.Name='Valid font';
inputParam3.Type='Combobox';
inputParam3.Description='sample tooltip';
inputParam3.Entries={'Arial', 'Arial Black'};
inputParam3.setRowSpan([2 2]);
inputParam3.setColSpan([2 2]);
rec.setInputParameters({inputParam1,inputParam2,inputParam3});
```

The Model Advisor displays these input parameters in the right pane, in an **Input Parameters** box.



## Defining Model Advisor Result Explorer Views

A *list view* provides a way for users to fix check warnings and failures using the Model Advisor Result Explorer. Creating a list view allows you to :

- Add the **Explore Result** button to the custom check in the Model Advisor window.
- Provide the information to populate the Model Advisor Result Explorer.

For information on using the Model Advisor Results Explorer, see “Batch-Fixing Warnings or Failures” in the Simulink documentation.

Define list views using the `ModelAdvisor.ListViewParameter` class inside a custom check function (see “Defining Custom Checks” on page 20-11). You must define one instance of this class for each list view that you want to add to a Model Advisor Result Explorer window.

---

**Note** You do not have to create list views for every custom check.

---

## Model Advisor Code Example: List View Definition

The following is an example of defining list views. You must make the **Explore Result** button visible using the `ModelAdvisor.Check.ListViewVisible` property inside a custom check function, and include list view definitions inside a check callback function (see “Detailed Check Callback Function” on page 20-31).

The following code, when included in a check definition function, adds the **Explore Result** button to the check in the Model Advisor.

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
% add 'Explore Result' button
rec.ListViewVisible = true;
```

The following code, when included in a check callback function, provides the information to populate the Model Advisor Result Explorer.

```
mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system);
mdladvObj.setCheckResultStatus(true);

% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
myLVParam.Data = get_param(searchResult,'object');
myLVParam.Attributes = {'FontName'}; % name is default property
mdladvObj.setListViewParameters({myLVParam});
```

## Defining Check Actions

An *action* provides a way for you to specify an action that the Model Advisor performs to fix a Model Advisor check. When you define an action, the Model Advisor window includes an **Action** box below the **Analysis** box.

You define actions using the `ModelAdvisor.Action` class inside a custom check function (see “Defining Custom Checks” on page 20-11). You must define:

- One instance of this class for each action that you want to take.
- One action callback function for each action (see “Action Callback Function” on page 20-37).

---

**Note**

- Each check can contain only one action.
  - You do not have to create actions for every custom check.
- 

**Model Advisor Code Example: Action Definition**

The following is an example of defining actions within a custom check. You must include action definitions inside a check definition function (see “Model Advisor Code Example: Check Definition Function” on page 20-15).

The following code, when included in a check definition function, provides the information to populate the **Action** box in the Model Advisor.

```
rec = ModelAdvisor.Check('mathworks.example.optimizationSettings');
% Define an automatic fix action for this check
modifyAction = ModelAdvisor.Action;
modifyAction.setCallbackFcn(@modifyOptimizationSetting);
modifyAction.Name = 'Modify Settings';
modifyAction.Description = ['Modify model configuration optimization' ...
                           ' settings that can impact safety'];
modifyAction.Enable = true;
rec.setAction(modifyAction);
```

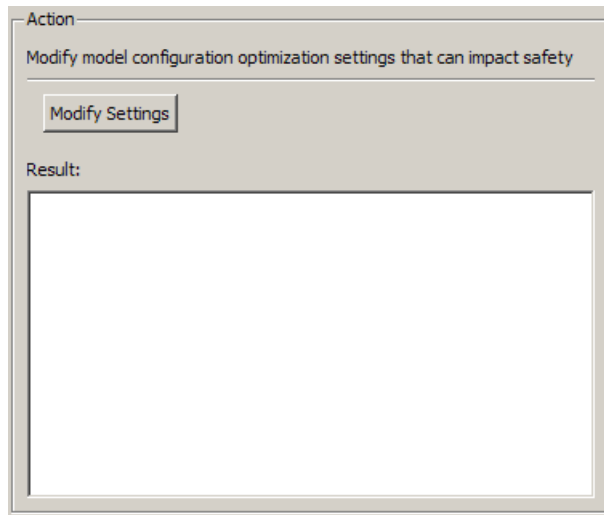
The Model Advisor, in the right pane, displays an **Action** box.

Action

Modify model configuration optimization settings that can impact safety

Modify Settings

Result:



## Creating Callback Functions and Results

### In this section...

- “About Callback Functions” on page 20-22
- “Common Utilities for Authoring Checks” on page 20-23
- “Simple Check Callback Function” on page 20-23
- “Detailed Check Callback Function” on page 20-31
- “Check Callback Function with Hyperlinked Results” on page 20-33
- “Action Callback Function” on page 20-37
- “Formatting Model Advisor Results” on page 20-38

### About Callback Functions

A *callback function* specifies the actions that the Model Advisor performs on a model or subsystem, based on the check or action that the user runs. You must create a callback function for each custom check and action so that the Model Advisor can execute the function when a user runs the check. There are several types of callback functions:

- “Simple Check Callback Function” on page 20-23
- “Detailed Check Callback Function” on page 20-31
- “Check Callback Function with Hyperlinked Results” on page 20-33
- “Action Callback Function” on page 20-37

All types of callback functions provide one or more return arguments for displaying the results after executing the check or action. In some cases, return arguments are strings or cell arrays of strings that support embedded HTML tags for text formatting. The MathWorks™ recommends that you use the Model Advisor Result Template API to format check results, as described in “Formatting Model Advisor Results” on page 20-38. Limit HTML tags to be compatible with alternate output formats.



## Common Utilities for Authoring Checks

When you create a check, there are common Simulink utilities that you can use to make the check perform different actions. Following is a list of utilities and when to use them. In the Utility column, click the link for more information about the utility.

Utility	Used for...
<a href="#">find_system</a>	Getting handle or path to: <ul style="list-style-type: none"> <li>• Blocks</li> <li>• Lines</li> <li>• Annotations</li> </ul> When getting the object, you can: <ul style="list-style-type: none"> <li>• Specify a search depth</li> <li>• Search under masks and libraries</li> </ul>
<a href="#">get_param / set_param</a>	Getting and setting system and block parameter values.
<a href="#">inspect</a>	Getting object properties. First you must get a handle to the object.
<a href="#">evalin</a>	Working in the base workspace.
<a href="#">Stateflow API</a>	Programmatic access to Stateflow objects.

## Simple Check Callback Function

Use a simple check callback function with results formatted using the Result Template API to indicate whether the model passed or failed the check, or to recommend correcting an issue. The keyword for this callback function is `StyleOne`. The check definition requires this keyword (see “Defining Custom Checks” on page 20-11).

The check callback function takes the following arguments.

Argument	I/O Type	Description
system	Input	Path to the model or subsystem analyzed by the Model Advisor.
result	Output	MATLAB string that supports Model Advisor Formatting API calls or embedded HTML tags for text formatting.

### Model Advisor Code Example: Informational Check Callback Function

The following code is an example of a callback function for a custom *informational* check that finds and displays the model configuration and checksum information. The informational check uses the Result Template API to format the check result.

An *informational* check includes the following items in the results:

- A description of what the check is reviewing.
- References to standards, if applicable.

An informational check does not include the following items in the results:

- The check status. The Model Advisor displays the overall check status, but the status is not in the result.
- A description of the status.
- The recommended action to take when the check does not pass.
- Subcheck results.
- A line below the results.

```
% Sample Check 1 Callback Function: Informational Check
% Find and display model configuration and checksum information
% Informational checks do not have a passed or warning status in the results

function resultDescription = modelVersionChecksumCallbackUsingFT(system)
resultDescription = [];
system = getfullname(system);
```

```

model = bdroot(system);

% Format results in a list using Model Advisor Result Template API
ft = ModelAdvisor.FormatTemplate('ListTemplate');
% Add See Also section for references to standards
docLinkSfunction{1} = {'IEC 61508-3, Table A.8 (5)' ...
                    ' 'Software configuration management' ' '}};
setRefLink(ft,docLinkSfunction);

% Description of check in results
desc = 'Display model configuration and checksum information.';
% If running the Model Advisor on a subsystem, add note to description
if strcmp(system, model) == false
    desc = strcat(desc, ['<br/>NOTE: The Model Advisor is reviewing a ' ...
                        ' sub-system, but these results are based on root-level settings.']);
end
setCheckText(ft, desc);

mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system);
% If err, use these values
mdlver = 'Error - could not retrieve Version';
mdlauthor = 'Error - could not retrieve Author';
mdldate = 'Error - could not retrieve Date';
mdlsum = 'Error - could not retrieve CheckSum';

% Get model configuration and checksum information
try
    mdlver = get_param(model,'ModelVersion');
    mdlauthor = get_param(model,'LastModifiedBy');
    mdldate = get_param(model,'LastModifiedDate');
    mdlsum = Simulink.BlockDiagram.getChecksum(model);
    mdlsum = [num2str(mdlsum(1)) ' ' num2str(mdlsum(2)) ' ' ...
              num2str(mdlsum(3)) ' ' num2str(mdlsum(4))];
    mdladvObj.setCheckResultStatus(true); % init to true
catch err
    mdladvObj.setCheckResultStatus(false);
    setSubResultStatusText(ft,err.message);
    resultDescription{end+1} = ft;
    return
end

```

```
% Display the results
lbStr = '<br/>';
resultStr = ['Model Version: ' mdlver lbStr 'Author: ' mdlauthor lbStr ...
            'Date: ' mdldate lbStr 'Model Checksum: ' mdlsum];
setSubResultStatusText(ft,resultStr);

% Informational checks do not have subresults, suppress line
setSubBar(ft,false);
resultDescription{end+1} = ft;
```

### **Model Advisor Code Example: Basic Check with Pass/Fail Status**

Here is an example of a callback function for a custom *basic* check that finds and reports unconnected lines, input ports, and output ports.

A *basic* check includes the following items in the results:

- A description of what the check is reviewing.
- References to standards, if applicable.
- The status of the check.
- A description of the status.
- Results for the check.
- The recommended actions to take when the check does not pass.

A basic check does not include the following items in the results:

- Subcheck results.
- A line below the results.

```
% Sample Check 2 Callback Function: Basic Check with Pass/Fail Status
% Find and report unconnected lines, input ports, and output ports
function ResultDescription = unconnectedObjectsCallbackUsingFT(system)
mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system);
% Initialize variables
mdladvObj.setCheckResultStatus(false);
```

```

ResultDescription = {};
ResultStatus = false; % Default check status is 'Warning'
system = getfullname(system);
isSubsystem = ~strcmp(bdroot(system), system);

% Format results in a list using Model Advisor Result Template API
% Create a list template object
ft = ModelAdvisor.FormatTemplate('ListTemplate');

% Description of check in results
if isSubsystem
    checkDescStr = ['Identify unconnected lines, input ports, and ' ...
                   'output ports in the subsystem.'];
else
    checkDescStr = ['Identify unconnected lines, input ports, and ' ...
                   'output ports in the model.'];
end
setCheckText(ft,checkDescStr);

% Add See Also section with references to applicable standards
checkStdRef = 'IEC 61508-3, Table A.3 (3) 'Language subset' ';
docLinkSfunction{1} = {checkStdRef};
setRefLink(ft,docLinkSfunction);

% Basic checks do not have subresults, suppress line
setSubBar(ft,false);

% Check for unconnected lines, inputs, and outputs
sysHandle = get_param(system, 'Handle');
uLines = find_system(sysHandle, ...
    'Findall', 'on', ...
    'LookUnderMasks', 'on', ...
    'Type', 'line', ...
    'Connected', 'off');
uPorts = find_system(sysHandle, ...
    'Findall', 'on', ...
    'LookUnderMasks', 'on', ...
    'Type', 'port', ...
    'Line', -1);

```

```

% Use parents of port objects for the correct highlight behavior
if ~isempty(uPorts)
    for i=1:length(uPorts)
        uPorts(i) = get_param(get_param(uPorts(i), 'Parent'), 'Handle');
    end
end

% Create cell array of unconnected object handles
modelObj = {};
searchResult = union(uLines, uPorts);
for i = 1:length(searchResult)
    modelObj{i} = searchResult(i);
end

% No unconnected objects in model
% Set result status to 'Pass' and display text describing the status
if isempty(modelObj)
    setSubResultStatus(ft,'Pass');
    if isSubsystem
        setSubResultStatusText(ft,['There are no unconnected lines, ' ...
            'input ports, and output ports in this subsystem.']);
    else
        setSubResultStatusText(ft,['There are no unconnected lines, ' ...
            'input ports, and output ports in this model.']);
    end
    ResultStatus = true;
% Unconnected objects in model
% Set result status to 'Warning' and display text describing the status
else
    setSubResultStatus(ft,'Warn');
    if ~isSubsystem
        setSubResultStatusText(ft,['The following lines, input ports, ' ...
            'or output ports are not properly connected in the system: ' system]);
    else
        setSubResultStatusText(ft,['The following lines, input ports, or ' ...
            'output ports are not properly connected in the subsystem: ' system]);
    end
    % Specify recommended action to fix the warning
    setRecAction(ft,'Connect the specified blocks.');
```

% Create a list of handles to problem objects

```

        setListObj(ft,modelObj);
        ResultStatus = false;
    end
    % Pass the list template object to the Model Advisor
    ResultDescription{end+1} = ft;
    % Set overall check status
    mdladvObj.setCheckResultStatus(ResultStatus);

```

## Model Advisor Code Example: Check With Subchecks and Actions

Here is an example of a callback function for a custom check that finds and reports optimization settings. The check consists of two subchecks. The first reviews the **Block reduction** optimization setting, and the second reviews the **Conditional input branch execution** optimization setting.

*A check with subchecks* includes the following items in the results:

- A description of what the overall check is reviewing.
- A title for the subcheck.
- A description of what the subcheck is reviewing.
- References to standards, if applicable.
- The status of the subcheck.
- A description of the status.
- Results for the subcheck.
- Recommended actions to take when the subcheck does not pass.
- A line between the subcheck results.

```

% Sample Check 3 Callback Function: Check with Subchecks and Actions
% Find and report optimization settings
function ResultDescription = OptimizationSettingCallback(system)
% Initialize variables
system =getfullname(system);
mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system);
mdladvObj.setCheckResultStatus(false); % Default check status is 'Warning'
ResultDescription = {};

```

```

system = bdroot(system);

% Format results in a list using Model Advisor Result Template API
% Create a list template object for first subcheck
ft1 = ModelAdvisor.FormatTemplate('ListTemplate');

% Description of check in results
setCheckText(ft1,['Check model configuration for optimization settings that...
    'can impact safety.']);

% Title and description of first subcheck
setSubTitle(ft1,'Verify Block reduction setting');
setInformation(ft1,'Check whether the ''Block reduction'' check box is cleared. ');
% Add See Also section with references to applicable standards
docLinks{1} = [['Reference DO-178B Section 6.3.4e - Source code ' ...
    'is traceable to low-level requirements']];
% Review 'Block reduction' optimization
setRefLink(ft1,docLinks);
if strcmp(get_param(system,'BlockReduction'),'off')
    % 'Block reduction' is cleared
    % Set subresult status to 'Pass' and display text describing the status
    setSubResultStatus(ft1,'Pass');
    setSubResultStatusText(ft1,'The ''Block reduction'' check box is cleared. ');
    ResultStatus = true;
else
    % 'Block reduction' is selected
    % Set subresult status to 'Warning' and display text describing the status
    setSubResultStatus(ft1,'Warn');
    setSubResultStatusText(ft1,'The ''Block reduction'' check box is selected. ');
    setRecAction(ft1,['Clear the ''Optimization > Block reduction'' ...
        ' check box in the Configuration Parameters dialog box.']);
    ResultStatus = false;
end

ResultDescription{end+1} = ft1;

% Title and description of second subcheck
ft2 = ModelAdvisor.FormatTemplate('ListTemplate');
setSubTitle(ft2,'Verify Conditional input branch execution setting');
setInformation(ft2,['Check whether the ''Conditional input branch execution''...

```



```

        ' check box is cleared.'])
% Add See Also section and references to applicable standards
docLinks{1} = {'Reference DO-178B Section 6.4.4.2 - Test coverage ' ...
    'of software structure is achieved'}];
setRefLink(ft2,docLinks);

% Last subcheck, suppress line
setSubBar(ft2,false);

% Check status of the 'Conditional input branch execution' check box
if strcmp(get_param(system,'ConditionallyExecuteInputs'),'off')
    % The 'Conditional input branch execution' check box is cleared
    % Set subresult status to 'Pass' and display text describing the status
    setSubResultStatus(ft2,'Pass');
    setSubResultStatusText(ft2,['The ''Conditional input branch execution'' ...
        'check box is cleared.']);
else
    % 'Conditional input branch execution' is selected
    % Set subresult status to 'Warning' and display text describing the status
    setSubResultStatus(ft2,'Warn');
    setSubResultStatusText(ft2,['The ''Conditional input branch execution''...
        'check box is selected.']);
    setRecAction(ft2,['Clear the ''Optimization > Conditional input branch ' ...
        'execution'' check box in the Configuration Parameters dialog box.']);
    ResultStatus = false;
end

ResultDescription{end+1} = ft2; % Pass list template object to Model Advisor
mdladvObj.setCheckResultStatus(ResultStatus); % Set overall check status
% Enable Modify Settings button when check fails
mdladvObj.setActionEnable(~ResultStatus);

```

## Detailed Check Callback Function

Use the detailed check callback function to return and organize results as strings in a layered, hierarchical fashion. The function provides two output arguments so you can associate text descriptions with one or more paragraphs of detailed information. The keyword for the detailed callback function is `StyleTwo`. The check definition requires this keyword (see “Defining Custom Checks” on page 20-11).

The detailed callback function takes the following arguments.

Argument	I/O Type	Description
system	Input	Path to the model or system analyzed by the Model Advisor.
ResultDescription	Output	Cell array of MATLAB strings that supports Model Advisor Formatting API calls or embedded HTML tags for text formatting. The Model Advisor concatenates the ResultDescription string with the corresponding array of ResultDetails strings.
ResultDetails	Output	Cell array of cell arrays, each of which contains one or more strings.

---

**Note** The ResultDetails cell array must be the same length as the ResultDescription cell array.

---

Here is an example of a detailed check callback function that checks optimization settings for simulation and code generation.

```
function [ResultDescription, ResultDetails] = SampleStyleTwoCallback(system)
ResultDescription = {};
ResultDetails = {};

model = bdroot(system);
mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system); % get object
mdladvObj.setCheckResultStatus(true); % init result status to pass

% Check Simulation optimization setting
ResultDescription{end+1} = ModelAdvisor.Paragraph(['Check Simulation '...
'optimization settings:']);
if strcmp(get_param(model, 'BlockReduction'), 'off');
    ResultDetails{end+1} = {ModelAdvisor.Text(['It is recommended to '...
'turn on Block reduction optimization option.', {'italic'}])};
end
end
```

```

        mdladvObj.setCheckResultStatus(false); % set to fail
        mdladvObj.setActionEnable(true);
    else
        ResultDetails{end+1}    = {ModelAdvisor.Text('Passed',{'pass'})};
    end

    % Check code generation optimization setting
    ResultDescription{end+1} = ModelAdvisor.Paragraph(['Check code generation '...
        'optimization settings:']);
    ResultDetails{end+1} = {};
    if strcmp(get_param(model,'LocalBlockOutputs'),'off');
        ResultDetails{end}{end+1}    = ModelAdvisor.Text(['It is recommended to'...
            ' turn on Enable local block outputs option.'',{'italic'}]);
        ResultDetails{end}{end+1}    = ModelAdvisor.LineBreak;
        mdladvObj.setCheckResultStatus(false); % set to fail
        mdladvObj.setActionEnable(true);
    end
    if strcmp(get_param(model,'BufferReuse'),'off');
        ResultDetails{end}{end+1}    = ModelAdvisor.Text(['It is recommended to'...
            ' turn on Reuse block outputs option.'',{'italic'}]);
        mdladvObj.setCheckResultStatus(false); % set to fail
        mdladvObj.setActionEnable(true);
    end
    if isempty(ResultDetails{end})
        ResultDetails{end}{end+1}    = ModelAdvisor.Text('Passed',{'pass'});
    end
end

```

## Check Callback Function with Hyperlinked Results

This callback function automatically displays hyperlinks for every object returned by the check so that you can easily locate problem areas in your model or subsystem. The keyword for this type of callback function is `StyleThree`. The check definition requires this keyword (see “Defining Custom Checks” on page 20-11).

This callback function takes the following arguments.

Argument	I/O Type	Description
system	Input	Path to the model or system analyzed by the Model Advisor.
ResultDescription	Output	Cell array of MATLAB strings that supports the Model Advisor Formatting API calls or embedded HTML tags for text formatting.
ResultDetails	Output	Cell array of cell arrays, each of which contains one or more Simulink objects such as blocks, ports, lines, and Stateflow charts. The objects must be in the form of a handle or Simulink path.

---

**Note** The `ResultDetails` cell array must be the same length as the `ResultDescription` cell array.

---

The Model Advisor automatically concatenates each string from `ResultDescription` with the corresponding array of objects from `ResultDetails`. The Model Advisor displays the contents of `ResultDetails` as a set of hyperlinks, one for each object returned in the cell arrays. When you click a hyperlink, the Model Advisor displays the target object highlighted in your Simulink model.

The following is an example of a check callback function with hyperlinked results. This example checks a model for consistent use of font type and font size in its blocks. It also contains input parameters, actions, and a call to the Model Advisor Result Explorer, which are described in later sections.

```
function [ResultDescription, ResultDetails] = SampleStyleThreeCallback(system)
    ResultDescription = {};
    ResultDetails = {};

    mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system);
    mdladvObj.setCheckResultStatus(true);
    needEnableAction = false;
```

```

% get input parameters
inputParams = mdladvObj.getInputParameters;
skipFontCheck = inputParams{1}.Value;
regularFontSize = inputParams{2}.Value;
regularFontName = inputParams{3}.Value;
if skipFontCheck
    ResultDescription{end+1} = ModelAdvisor.Paragraph('Skipped. ');
    ResultDetails{end+1} = {};
    return
end
regularFontSize = str2double(regularFontSize);
if regularFontSize<1 || regularFontSize>=99
    mdladvObj.setCheckResultStatus(false);
    ResultDescription{end+1} = ModelAdvisor.Paragraph(['Invalid font size. '...
        'Please enter a value between 1 and 99']);
    ResultDetails{end+1} = {};
end

% find all blocks inside current system
allBlks = find_system(system);

% block diagram doesn't have font property
% get blocks inside current system that have font property
allBlks = setdiff(allBlks, {system});

% find regular font name blocks
regularBlks = find_system(allBlks, 'FontName', regularFontName);

% look for different font blocks in the system
searchResult = setdiff(allBlks, regularBlks);
if ~isempty(searchResult)
    ResultDescription{end+1} = ModelAdvisor.Paragraph(['It is recommended to '...
        'use same font for blocks to ensure uniform appearance of model. '...
        'The following blocks use a font other than ' regularFontName ': ']);
    ResultDetails{end+1} = searchResult;
    mdladvObj.setCheckResultStatus(false);
    myLVParam = ModelAdvisor.ListViewParameter;
    myLVParam.Name = 'Invalid font blocks'; % pull down filter name
    myLVParam.Data = get_param(searchResult, 'object');
    myLVParam.Attributes = {'FontName'}; % name is default property

```

```

        mdladvObj.setListViewParameters({myLVParam});
        needEnableAction = true;
    else
        ResultDescription{end+1} = ModelAdvisor.Paragraph(['All block font names '...
            'are identical.']);
        ResultDetails{end+1}      = {};
    end

% find regular font size blocks
regularBlks = find_system(allBlks, 'FontSize', regularFontSize);
% look for different font size blocks in the system
searchResult = setdiff(allBlks, regularBlks);
if ~isempty(searchResult)
    ResultDescription{end+1} = ModelAdvisor.Paragraph(['It is recommended to '...
        'use same font size for blocks to ensure uniform appearance of model. '...
        'The following blocks use a font size other than ' ...
        num2str(regularFontSize) ': ']);
    ResultDetails{end+1}      = searchResult;
    mdladvObj.setCheckResultStatus(false);
    myLVParam = ModelAdvisor.ListViewParameter;
    myLVParam.Name = 'Invalid font size blocks'; % pull down filter name
    myLVParam.Data = get_param(searchResult, 'object');
    myLVParam.Attributes = {'FontSize'}; % name is default property
    mdladvObj.setListViewParameters...
        ({mdladvObj.getListViewParameters{:}, myLVParam});
    needEnableAction = true;
else
    ResultDescription{end+1} = ModelAdvisor.Paragraph(['All block font sizes '...
        'are identical.']);
    ResultDetails{end+1}      = {};
end

mdladvObj.setActionEnable(needEnableAction);
mdladvObj.setCheckErrorSeverity(1);

```

In the Model Advisor, if you run **Example task with input parameter and auto-fix ability** for the `slvnvdemo_mdladv` model, you can view the hyperlinked results. Clicking the first hyperlink, `slvnvdemo_mdladv/Input`, displays the Simulink model with the Input block highlighted.

## Action Callback Function

An *action callback function* specifies the actions that the Model Advisor performs on a model or subsystem when the user clicks the action button. You must create one callback function for the action that you want to take.

The action callback function takes the following arguments.

Argument	I/O Type	Description
taskobj	Input	The ModelAdvisor.Task object for the check that includes an action definition.
result	Output	MATLAB string that supports Model Advisor Formatting API calls or embedded HTML tags for text formatting.

### Model Advisor Code Example: Action Callback Function

The following is an example of an action callback function that fixes the optimization settings that the Model Advisor reviews as defined in “Model Advisor Code Example: Check With Subchecks and Actions” on page 20-29.

```
% Sample Check 3 Action Callback Function: Check with Subresults and Actions
% Fix optimization settings
function result = modifyOptimizationSetting(taskobj)
% Initialize variables
result = ModelAdvisor.Paragraph();
mdladvObj = taskobj.MAObj;
system = bdroot(mdladvObj.System);

% 'Block reduction' is selected
% Clear the check box and display text describing the change
if ~strcmp(get_param(system,'BlockReduction'),'off')
    set_param(system,'BlockReduction','off');
    result.addItem(ModelAdvisor.Text( ...
        'Cleared the ''Block reduction'' check box.', {'Pass'}));
    result.addItem(ModelAdvisor.LineBreak);
end

% 'Conditional input branch execution' is selected
% Clear the check box and display text describing the change
if ~strcmp(get_param(system,'ConditionallyExecuteInputs'),'off')
```

```
set_param(system,'ConditionallyExecuteInputs','off');
result.addItem(ModelAdvisor.Text( ...
    'Cleared the ''Conditional input branch execution'' check box.', ...
    {'Pass'}));
end
```

For an example of an action callback function that updates all of the blocks in the model with the font specified in the Input Parameter defined in “Model Advisor Code Example: Input Parameter Definition” on page 20-17, review the customization source code in `slvndemo_mdladv`.

## Formatting Model Advisor Results

- “Overview of Displaying Results” on page 20-38
- “Formatting Model Advisor Results” on page 20-39
- “Formatting Text” on page 20-39
- “Formatting Lists” on page 20-40
- “Formatting Tables” on page 20-40
- “Formatting Paragraphs” on page 20-41
- “Model Advisor Code Example: Formatted Output” on page 20-41

## Overview of Displaying Results

You can make all of the analysis results of your custom checks appear similar to each other with minimal scripting using the Model Advisor `ModelAdvisor.FormatTemplate` class, as described in `ModelAdvisor.FormatTemplate`. For examples of callback functions using the `ModelAdvisor.FormatTemplate` class, see “Simple Check Callback Function” on page 20-23.

If this format template does not meet your needs, or if you want to format action results, use the Model Advisor Formatting API, as described in the following sections.



## Formatting Model Advisor Results

Use the Model Advisor Formatting API to produce formatted outputs in the Model Advisor. The following constructors of the `ModelAdvisor` class provide the ability to format the output. For more information on each constructor and associated methods, in the Constructor column, click the link.

Constructor	Description
<code>ModelAdvisor.Text</code>	Formats element text.
<code>ModelAdvisor.Paragraph</code>	Combines elements into paragraphs.
<code>ModelAdvisor.List</code>	Creates a list of elements.
<code>ModelAdvisor.LineBreak</code>	Adds a line break between elements.
<code>ModelAdvisor.Table</code>	Creates a table.
<code>ModelAdvisor.Image</code>	Adds an image to the output.

## Formatting Text

Text is the simplest form of output. You can format text in many different ways. The default text formatting is:

- Empty
- Default color (black)
- Unformatted (not bold, italicized, underlined, linked, subscripted, or superscripted)

To change text formatting, use the `ModelAdvisor.Text` constructor. When you want one type of formatting for all text, use this syntax:

```
ModelAdvisor.Text(content, {attributes})
```

When you want multiple types of formatting, you must build the text.

```
t1 = ModelAdvisor.Text('It is ');
t2 = ModelAdvisor.Text('recommended', {'italic'});
t3 = ModelAdvisor.Text(' to use same font for ');
t4 = ModelAdvisor.Text('blocks', {'bold'});
t5 = ModelAdvisor.Text(' to ensure uniform appearance of model.');
```

```
result = [t1, t2, t3, t4, t5];
```

Add ASCII and Extended ASCII characters using the `MATLAB char` command. For more information, see the `ModelAdvisor.Text` class page.

## Formatting Lists

You can create two types of lists: numbered and bulleted. The default list formatting is bulleted. Use the `ModelAdvisor.List` constructor to create and format lists (see `ModelAdvisor.List`). You can create lists with indented subsections, formatted as either numbered or bulleted.

```
subList = ModelAdvisor.List();
subList.setType('numbered')
subList.addItem(ModelAdvisor.Text('Sub entry 1', {'pass','bold'}));
subList.addItem(ModelAdvisor.Text('Sub entry 2', {'pass','bold'}));

topList = ModelAdvisor.List();
topList.addItem([ModelAdvisor.Text('Entry level 1',{'keyword','bold'}), subList]);
topList.addItem([ModelAdvisor.Text('Entry level 2',{'keyword','bold'}), subList]);
```

## Formatting Tables

The default table formatting is:

- Default color (black)
- Left justified
- Bold title, row, and column headings

Change table formatting using the `ModelAdvisor.Table` constructor. The following example code creates a subtable within a table.

```
table1 = ModelAdvisor.Table(1,1);
table2 = ModelAdvisor.Table(2,3);
table2.setHeading('Table 2');
table2.setHeadingAlign('center');
table2.setColHeading(1, 'Header 1');
table2.setColHeading(2, 'Header 2');
table2.setColHeading(3, 'Header 3');
```

```
table1.setHeading('Table 1');
table1.setEntry(1,1,table2);
```

Table 1														
<table border="1"> <thead> <tr> <th colspan="3">Table 2</th> </tr> <tr> <th>Header 1</th> <th>Header 2</th> <th>Header 3</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> </tr> </tbody> </table>			Table 2			Header 1	Header 2	Header 3						
Table 2														
Header 1	Header 2	Header 3												

## Formatting Paragraphs

You must handle paragraphs explicitly because most markup languages do not support line breaks. The default paragraph formatting is:

- Empty
- Default color (black)
- Unformatted, (not bold, italicized, underlined, linked, subscripted, or superscripted)
- Aligned left

If you want to change paragraph formatting, use the `ModelAdvisor.Paragraph` class.

## Model Advisor Code Example: Formatted Output

The following is the example from “Simple Check Callback Function” on page 20-23, reformatted using the Model Advisor Formatting API.

```
function result = SampleStyleOneCallback(system)
mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system);
if strcmp(get_param(bdroot(system), 'ScreenColor'),'white')
    result = ModelAdvisor.Text('Passed',{ 'pass' });
    mdladvObj.setCheckResultStatus(true);
else
    msg1 = ModelAdvisor.Text(...
        ['It is recommended to select a Simulink window screen color'...
```

```
        ' of white to ensure a readable and printable model. Click ');  
msg2 = ModelAdvisor.Text('here');  
msg2.setHyperlink('matlab: set_param(bdroot, 'ScreenColor', 'white')');  
msg3 = ModelAdvisor.Text(' to change screen color to white.');
```

```
result = [msg1, msg2, msg3];  
mdladvObj.setCheckResultStatus(false);  
end
```

# Creating Custom Configurations by Organizing Checks and Folders

---

- “Overview of Creating Custom Configurations” on page 21-2
- “Organizing Checks and Folders Using the Model Advisor Configuration Editor” on page 21-4
- “Organizing Checks and Folders Within a Customization File” on page 21-12
- “Verifying and Using Custom Configurations” on page 21-22

## Overview of Creating Custom Configurations

In this section...
“About Creating Custom Configurations” on page 21-2
“Creating Custom Configurations Workflow” on page 21-2
“Using the Model Advisor Configuration Editor Versus Customization File” on page 21-3

### About Creating Custom Configurations

The Simulink Verification and Validation product allows you to extend the capabilities of the Model Advisor. Using Model Advisor APIs and the Model Advisor Configuration Editor, you can:

- Customize the behavior of the Model Advisor by defining your own custom checks, and writing your own callback functions.
- Organize checks and folders to create custom Model Advisor configurations.
- Create multiple custom configurations that you use for different projects or modeling guidelines, and switch between these configurations in the Model Advisor.

### Creating Custom Configurations Workflow

When you create custom configurations, you:

- 1** Optionally author custom checks, as described in Chapter 20, “Authoring Custom Checks”.
- 2** Identify which MathWorks checks you want to include in your custom Model Advisor configuration.
- 3** Organize checks and folders to create custom configurations. You can create custom configurations either using the Model Advisor Configuration Editor (see “Organizing Checks and Folders Using the Model Advisor Configuration Editor” on page 21-4), or within a customization file (see “Organizing Checks and Folders Within a Customization File” on page 21-12).

- 4 Verify the custom configuration, as described in “Verifying and Using Custom Configurations” on page 21-22.

## **Using the Model Advisor Configuration Editor Versus Customization File**

The Model Advisor Configuration Editor is a GUI that expedites creating and deploying custom configurations. While you can organize Model Advisor configurations in a customization file, The MathWorks recommends that you create custom configurations using the Model Advisor Configuration Editor. For more details, see “Organizing Checks and Folders Using the Model Advisor Configuration Editor” on page 21-4.

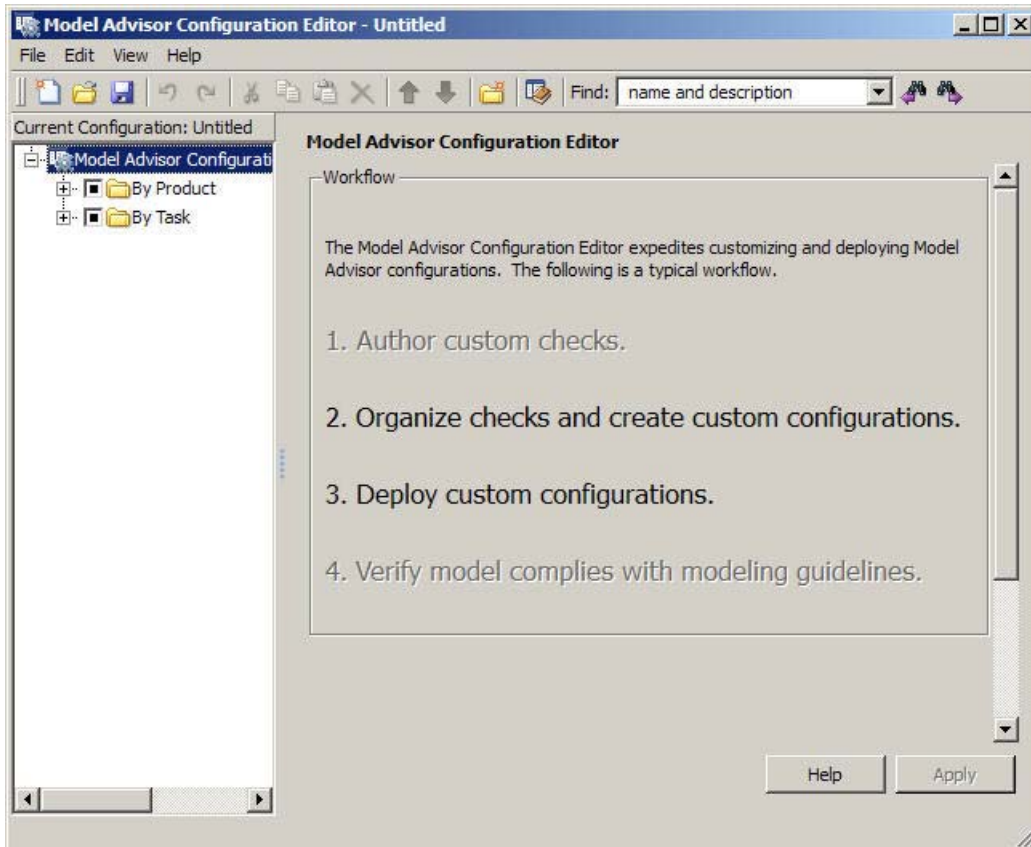
## **Organizing Checks and Folders Using the Model Advisor Configuration Editor**

<b>In this section...</b>
“Overview of the Model Advisor Configuration Editor” on page 21-4
“Starting the Model Advisor Configuration Editor” on page 21-9
“How To Organize Checks and Folders Using the Model Advisor Configuration Editor” on page 21-10

### **Overview of the Model Advisor Configuration Editor**

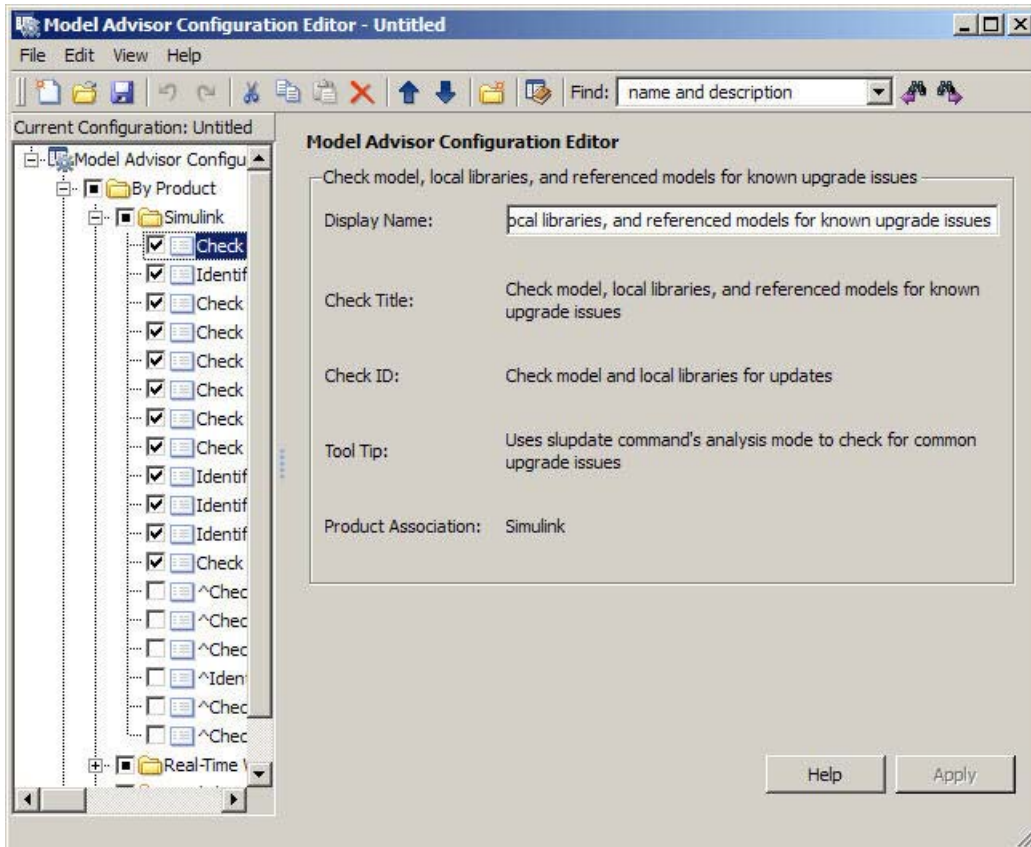
When you start the Model Advisor Configuration Editor, two windows open; the Model Advisor Configuration Editor and the Model Advisor Check Browser. The Configuration Editor window consists of two panes: the Model Advisor Configuration Editor hierarchy and the Workflow. The Model Advisor Configuration Editor hierarchy lists the checks and folders in the current configuration. The Workflow on the right shows the common workflow you use to create a custom configuration.





### Model Advisor Configuration Editor

When you select a folder or check in the Model Advisor Configuration Editor hierarchy, the Workflow pane changes to display information about the check or folder. You can change the display name of the check or folder in this pane.

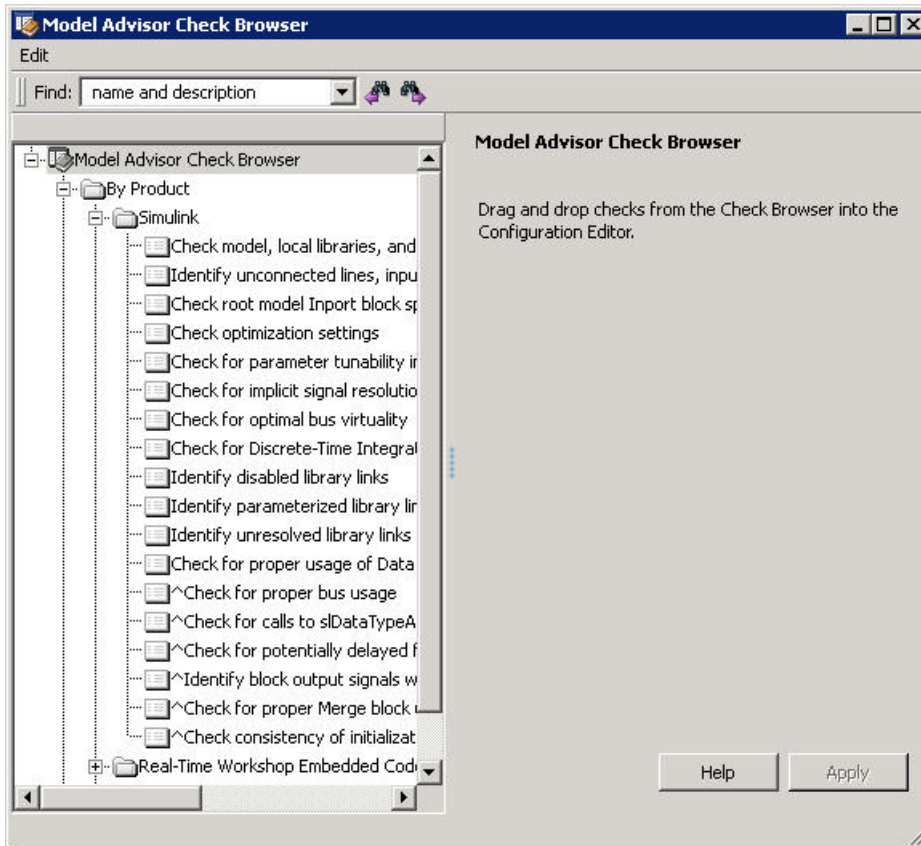


The Model Advisor Check Browser window includes a read-only list of available checks. If you delete a check in the Model Advisor Configuration Editor, you can retrieve a copy of it from the Model Advisor Check Browser.

---

**Tip** If you use a process callback function in a `sl_customization` file to hide checks and folders, the Model Advisor Configuration Editor and Model Advisor Check Browser do not display the hidden checks and folders. For a complete list of checks and folders, remove process callback functions and update the Simulink environment (see “Updating the Environment to Include Your `sl_customization` File” on page 21-22).

---



### Model Advisor Check Browser

Using the Model Advisor Configuration Editor, you can perform the following actions.

To...	Select...
Create new configurations	<b>File &gt; New</b>
Find checks and folders in the Model Advisor Check Browser	<b>View &gt; Check Browser</b>

To...	Select...
Add checks and folders to the configuration	<b>Edit &gt; Copy</b> <b>Edit &gt; Paste</b> <b>Edit &gt; New folder</b> The check or folder and drag and drop
Remove checks and folders from the configuration	<b>Edit &gt; Delete</b> <b>Edit &gt; Cut</b>
Reorder checks and folders	<b>Edit &gt; Move up</b> <b>Edit &gt; Move down</b> The check or folder and drag and drop
Rename checks and folders	The check or folder and edit <b>Display Name</b> in right pane.
<hr/> <b>Note</b> The MathWorks folder display names are restricted. When you rename a folder, you cannot use the restricted display names.	
Allow or gray out the check box control for checks and folders  <hr/> <b>Tip</b> This capability is equivalent to enabling checks, described in “Displaying and Enabling Checks” on page 20-13.	<b>Edit &gt; Enable</b> <b>Edit &gt; Disable</b>
Save the configuration as a MAT file for use and distribution	<b>File &gt; Save</b> <b>File &gt; Save As</b>
Set the configuration so it opens by default in the Model Advisor	<b>File &gt; Set Current Configuration as Default</b>
Restore the MathWorks default configuration	<b>File &gt; Restore Default Configuration</b>
Load and edit saved configurations	<b>File &gt; Open</b>

## Starting the Model Advisor Configuration Editor

### Note

- Before starting the Model Advisor Configuration Editor, ensure that the current folder is writable. If the folder is not writable, you see an error message when you start the Model Advisor Configuration Editor.
- The Model Advisor Configuration Editor uses the Simulink project (slprj) folder (for details about storing reports and other relevant information, see “Model Reference Simulation Targets”) in the current folder. If this folder does not exist in the current folder, the Model Advisor Configuration Editor creates it.

- 1 To include custom checks in the new Model Advisor configuration, update the Simulink environment to include your `sl_customization.m` file. For more information, see “Updating the Environment to Include Your `sl_customization` File” on page 21-22.
- 2 Start the Model Advisor Configuration Editor.

To start the Model Advisor Configuration Editor...	Do this:
Programmatically	At the MATLAB command line, enter <code>Simulink.ModelAdvisor.openConfigUI</code> . For more information, see the <code>Simulink.ModelAdvisor</code> function reference page.
From the Model Advisor	<ol style="list-style-type: none"> <li>1 Start the Model Advisor.</li> <li>2 Select <b>File &gt; Open Configuration Editor</b>.</li> </ol>

The Model Advisor Configuration Editor and Model Advisor Check Browser windows open.

- 3 Optionally, to edit an existing configuration in the Model Advisor Configuration Editor window:
  - a Select **File > Open**.
  - b In the Open dialog box, navigate to the configuration file that you want to edit.
  - c Click **Open**.

### How To Organize Checks and Folders Using the Model Advisor Configuration Editor

The following tutorial steps you through creating a custom configuration.

- 1 Open the Model Advisor Configuration Editor at the MATLAB command line by entering `Simulink.ModelAdvisor.openConfigUI` . For more options, see “Starting the Model Advisor Configuration Editor” on page 21-9.
- 2 In the Model Advisor Configuration Editor, in the left pane, delete the **By Product** and **By Task** folders, to start with a blank configuration.
- 3 Select the root node which is labeled Model Advisor Configuration Editor.
- 4 In the toolbar, click the **New Folder** button to create a folder.
- 5 In the left pane, select the new folder.
- 6 In the right pane, edit **Display Name** to rename the folder. For the purposes of this tutorial, rename the folder to **Review Optimizations**.
- 7 In the Model Advisor Check Browser window, in the **Find** field, enter optimization to find **Simulink > Check optimization settings**.
- 8 Drag and drop **Check optimization settings** into **Review Optimizations**.
- 9 In the Model Advisor Check Browser window, find **Simulink Verification and Validation > DO-178B Checks > Check safety-related optimization settings**.

- 10** Drag and drop **Check safety-related optimization settings** into **Review Optimizations**.
- 11** In the Model Advisor Configuration Editor window, expand **Review Optimizations**.
- 12** Rename **Check optimization settings** to **Check Simulink optimization settings**.
- 13** Select **File > Save As** to save the configuration.
- 14** Name the configuration `optimization_configuration.mat`.
- 15** Close the Model Advisor Configuration Editor window.

## Organizing Checks and Folders Within a Customization File

### In this section...

“Customization File Overview” on page 21-12

“Register Tasks and Folders” on page 21-13

“Defining Custom Tasks” on page 21-15

“Defining Custom Folders” on page 21-18

“Demo and Code Example” on page 21-20

---

**Note** While you can organize checks and folders within a customization file, The MathWorks recommends that you use the Model Advisor Configuration Editor. For more information, see “Using the Model Advisor Configuration Editor Versus Customization File” on page 21-3.

---

### Customization File Overview

The `sl_customization.m` file contains a set of functions for registering and defining custom checks, tasks, and groups. To set up the `sl_customization.m` file, follow the guidelines in this table.

Function	Description	Required or Optional
<code>sl_customization()</code>	Registers custom checks, tasks, folders, and callbacks with the Simulink customization manager at startup (see “Register Checks and Process Callbacks” on page 20-6).	Required for all customizations to the Model Advisor.
One or more check definitions	Defines all custom checks (see “Defining Custom Checks” on page 20-11).	Required for custom checks and to add custom checks to the <b>By Product</b> folder.



Function	Description	Required or Optional
One or more task definitions	Defines all custom tasks (see “Defining Custom Tasks” on page 21-15).	Required to add custom checks to the Model Advisor, except when adding the checks to the <b>By Product</b> folder. Write one task for each check that you add to the Model Advisor.
One or more groups	Defines all custom groups (see “Defining Custom Tasks” on page 21-15).	Required to add custom tasks to new folders in the Model Advisor, except when adding a new subfolder to the <b>By Product</b> folder. Write one group definition for each new folder.
One process callback function	Specifies actions that Simulink performs at startup and post-execution time (see “Defining Startup and Post-Execution Actions Using Process Callback Functions” on page 20-8).	Optional.

## Register Tasks and Folders

- “Create `sl_customization` Function” on page 21-13
- “Registering Tasks and Folders” on page 21-14

### Create `sl_customization` Function

To add tasks and folders to the Model Advisor, create the `sl_customization.m` file on your MATLAB path. Then create the `sl_customization()` function in the `sl_customization.m` file on your MATLAB path.

---

### Tip

- You can have more than one `sl_customization.m` file on your MATLAB path.
  - Do not place an `sl_customization.m` file that customizes the Model Advisor in your root MATLAB folder or any of its subfolders, except for the `matlabroot/work` folder. Otherwise, the Model Advisor ignores the customizations that the file specifies.
- 

The `sl_customization` function accepts one argument, a customization manager object, as in this example:

```
function sl_customization(cm)
```

The customization manager object includes methods for registering custom checks, tasks, folders, and process callbacks. Use these methods to register customizations specific to your application, as described in the sections that follow.

### Registering Tasks and Folders

The customization manager provides the following methods for registering custom tasks and folders:

- `addModelAdvisorTaskFcn (@factorygroupDefinitionFcn)`

Registers the tasks that you define in *factorygroupDefinitionFcn* to the **By Task** folder of the Model Advisor.

The *factorygroupDefinitionFcn* argument is a handle to the function that defines the checks to add to the Model Advisor as instances of the `ModelAdvisor.FactoryGroup` class (see “Defining Custom Tasks” on page 21-15).

- `addModelAdvisorTaskAdvisorFcn (@taskDefinitionFcn)`

Registers the tasks and folders that you define in *taskDefinitionFcn* to the folder in the Model Advisor that you specify using the `ModelAdvisor.Root.publish` method or the `ModelAdvisor.Group` class.

The *taskDefinitionFcn* argument is a handle to the function that defines all custom tasks and folders. Simulink adds the checks and folders to the Model Advisor as instances of the `ModelAdvisor.Task` or `ModelAdvisor.Group` classes (see “Defining Custom Tasks” on page 21-15).

---

**Note** The @ sign defines a function handle that MATLAB calls. For more information, see “At — @” in the MATLAB documentation.

---

### **Model Advisor Code Example: Registering Custom Tasks and Folders.**

The following code example registers custom tasks and folders:

```
function sl_customization(cm)

% register custom factory group
cm.addModelAdvisorTaskFcn(@defineModelAdvisorTasks);

% register custom tasks.
cm.addModelAdvisorTaskAdvisorFcn(@defineTaskAdvisor);
```

---

**Note** If you add custom checks and process callbacks within the `sl_customization.m` file, include methods for registering the checks and process callbacks in the `sl_customization` function. For more information, see “Register Checks and Process Callbacks” on page 20-6.

---

## **Defining Custom Tasks**

- “Adding a Check to Custom or Multiple Folders Using Tasks” on page 21-16
- “Creating Custom Tasks Using MathWorks Checks” on page 21-16
- “Displaying and Enabling Tasks” on page 21-17
- “Defining Where Tasks Appear” on page 21-17
- “Model Advisor Code Example: Task Definition Function” on page 21-17

### Adding a Check to Custom or Multiple Folders Using Tasks

You can use custom tasks for adding checks to the Model Advisor, either in multiple folders or in a single, custom folder. You define custom tasks in one or more functions that specify the properties of each instance of the `ModelAdvisor.Task` class. Define one instance of this class for each custom task that you want to add to the Model Advisor. Then register the custom task, as described in “Register Tasks and Folders” on page 21-13. The following sections describe how to define custom tasks.

To add a check to multiple folders or a single, custom folder:

- 1 Create a check using the `ModelAdvisor.Check` class, as described in “Defining Custom Checks” on page 20-11.
- 2 Register a task wrapper for the check, as described in “Register Tasks and Folders” on page 21-13.
- 3 If you want to add the check to folders that are not already present, register and create the folders using the `ModelAdvisor.Group` class.
- 4 Add a check to the task using the `ModelAdvisor.Task.setCheck` method.
- 5 Add the task to each folder using the `ModelAdvisor.Group.addTask` method and the task ID.

### Creating Custom Tasks Using MathWorks Checks

You can add MathWorks checks to your custom folders by defining the checks as custom tasks. When you add the checks as custom tasks, you identify checks by the check ID.

To find MathWorks check IDs:

- 1 In the Model Advisor, select **View > Source Tab**.
- 2 Navigate to the folder that contains the MathWorks check.
- 3 In the right pane, click **Source**. The Model Advisor displays the **Title**, **TitleID**, and **Source** information for each check in the folder.
- 4 Select and copy the **TitleID** of the check that you want to add as a task.

## Displaying and Enabling Tasks

The `Visible`, `Enable`, and `Value` properties interact the same way for tasks as they do for checks (see “Displaying and Enabling Checks” on page 20-13).

## Defining Where Tasks Appear

You can specify where the Model Advisor places tasks within the Model Advisor using the following guidelines:

- To place a task in a new folder in the **Model Advisor Task Manager**, use the `ModelAdvisor.Group` class. See “Defining Custom Folders” on page 21-18.
- To place a task in a new folder in the **By Task** folder, use the `ModelAdvisor.FactoryGroup` class. See “Defining Custom Folders” on page 21-18.

## Model Advisor Code Example: Task Definition Function

The following is an example of a task definition function. This function defines three tasks. The tasks are derived from the checks defined in “Model Advisor Code Example: Check Definition Function” on page 20-15.

For an example of placing these tasks into a custom group, see “Model Advisor Code Example: Group Definition” on page 21-19.

```
% Defines Model Advisor tasks and a custom folder
% Add checks to a custom folder using task definitions
function defineTaskAdvisor
mdladvRoot = ModelAdvisor.Root;

% Define task that uses Sample Check 1: Informational check
MAT1 = ModelAdvisor.Task('mathworks.example.task.configManagement');
MAT1.DisplayName = 'Informational check for model configuration management';
MAT1.Description = 'Display model configuration and checksum information.';
setCheck(MAT1, 'mathworks.example.configManagement');
mdladvRoot.register(MAT1);

% Define task that uses Sample Check 2: Basic Check with Pass/Fail Status
MAT2 = ModelAdvisor.Task('mathworks.example.task.unconnectedObjects');
MAT2.DisplayName = 'Check for unconnected objects';
```

```
setCheck(MAT2, 'mathworks.example.unconnectedObjects');
MAT2.Description = ['Identify unconnected lines, input ports, and output ' ...
                  'ports in the model or subsystem.'];

mdladvRoot.register(MAT2);

% Define task that uses Sample Check 3: Check with Subresults and Actions
MAT3 = ModelAdvisor.Task('mathworks.example.task.optimizationSettings');
MAT3.DisplayName = 'Check safety-related optimization settings';
MAT3.Description = ['Check model configuration for optimization ' ...
                  'settings that can impact safety.'];
MAT3.setCheck('mathworks.example.optimizationSettings');
mdladvRoot.register(MAT3);

% Custom folder definition
MAG = ModelAdvisor.Group('mathworks.example.ExampleGroup');
MAG.DisplayName = 'My Group';
% Add tasks to My Group folder
addTask(MAG, MAT1);
addTask(MAG, MAT2);
addTask(MAG, MAT3);
% Add My Group folder to the Model Advisor under 'Model Advisor' (root)
mdladvRoot.publish(MAG);
```

### Defining Custom Folders

- “About Custom Folders” on page 21-18
- “Adding Custom Folders” on page 21-19
- “Defining Where Custom Folders Appear” on page 21-19
- “Model Advisor Code Example: Group Definition” on page 21-19

### About Custom Folders

Use folders to group checks in the Model Advisor by functionality or usage. You define custom folders in:

- A factory group definition function that specifies the properties of each instance of the `ModelAdvisor.FactoryGroup` class.

- A task definition function that specifies the properties of each instance of the `ModelAdvisor.Group` class. For more information about task definition functions, see “Adding a Check to Custom or Multiple Folders Using Tasks” on page 21-16.

Define one instance of the group classes for each folder that you want to add to the Model Advisor. Then register the custom folder, as described in “Register Tasks and Folders” on page 21-13. The following sections describe how to define custom groups.

## Adding Custom Folders

To add a custom folder:

- 1 Create the folder using the `ModelAdvisor.Group` or `ModelAdvisor.FactoryGroup` classes.
- 2 Add the folder to the Model Advisor, as described in “Defining Custom Folders” on page 21-18.

## Defining Where Custom Folders Appear

You can specify the location of custom folders within the Model Advisor using the following guidelines:

- To define a new folder in the **Model Advisor Task Manager**, use the `ModelAdvisor.Group` class.
- To define a new folder in the **By Task** folder, use the `ModelAdvisor.FactoryGroup` class.

---

**Note** To define a new folder in the **By Product** folder, use the `ModelAdvisor.Root.publish` method within a custom check. For more information, see “Defining Where Custom Checks Appear” on page 20-14.

---

## Model Advisor Code Example: Group Definition

The following is an example of a group definition. The definition places the tasks defined in “Model Advisor Code Example: Task Definition Function” on

page 21-17 inside a folder called **My Group** under the **Model Advisor** root. The task definition function includes this group definition.

```
% Custom folder definition
MAG = ModelAdvisor.Group('mathworks.example.ExampleGroup');
MAG.DisplayName='My Group';
% Add tasks to My Group folder
MAG.addTask(MAT1);
MAG.addTask(MAT2);
MAG.addTask(MAT3);
% Add My Group folder to the Model Advisor under 'Model Advisor' (root)
mdladvRoot.publish(MAG);
```

The following is an example of a factory group definition function. The definition places the checks defined in “Model Advisor Code Example: Check Definition Function” on page 20-15 into a folder called **Demo Factory Group** inside of the **By Task** folder.

```
function defineModelAdvisorTasks
mdladvRoot = ModelAdvisor.Root;

% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
rec.DisplayName='Demo Factory Group';
rec.Description='Demo Factory Group';
rec.addCheck('mathworks.example.configManagement');
rec.addCheck('mathworks.example.unconnectedObjects');
rec.addCheck('mathworks.example.optimizationSettings');
mdladvRoot.publish(rec); % publish inside By Task
```

### Demo and Code Example

The Simulink Verification and Validation software provides a demo that shows how to customize the Model Advisor by adding:

- Custom checks
- Check input parameters
- Check actions
- Check list views to call the Model Advisor Result Explorer



- Custom tasks to include the custom checks in the Model Advisor
- Custom folders for grouping the checks
- A process callback function

The demo also provides the source code of the `sl_customization.m` file that executes the customizations.

To run the demo:

- 1** At the MATLAB command line, type `slvndemo_md1adv`.
- 2** Follow the instructions in the model.

# Verifying and Using Custom Configurations

In this section...
“Updating the Environment to Include Your sl_customization File” on page 21-22
“Verifying Custom Configurations” on page 21-22

## Updating the Environment to Include Your sl\_customization File

When you start Simulink, it reads customization (`sl_customization.m`) files. If you change the contents of your customization file, update your environment by performing these tasks:

- 1 If you previously started the Model Advisor:
  - a Close the model from which you started the Model Advisor
  - b Clear the data associated with the previous Model Advisor session by removing the `slprj` folder from your working folder.

- 2 At the MATLAB command line, enter:

```
sl_refresh_customizations
```

- 3 Open your model.
- 4 Start the Model Advisor.

## Verifying Custom Configurations

To verify a custom configuration:

- 1 If you created custom checks, or created the custom configuration using the `sl_customization` method, update the Simulink environment. For more information, see “Updating the Environment to Include Your `sl_customization` File” on page 21-22.
- 2 Open a model.
- 3 From the model window, start the Model Advisor.

- 4** Select **File > Load Configuration**. If you see a warning that the Model Advisor report corresponds to a different configuration, click **Load** to continue.
- 5** In the Open dialog box, navigate to and select your custom configuration. For example, if you created the custom configuration in “How To Organize Checks and Folders Using the Model Advisor Configuration Editor” on page 21-10, select `optimization_configuration.mat`.
- 6** When the Model Advisor reopens, verify that the new configuration contains the appropriate folders and checks. For example, the **Review Optimizations** folder and the **Check Simulink optimization settings** and **Check safety-related optimization settings** checks.
- 7** Optionally, run the checks.



# Deploying Custom Configurations

---

- “Overview of Deploying Custom Configurations” on page 22-2
- “How to Deploy Custom Configurations” on page 22-3
- “Loading and Setting the Default Configuration” on page 22-4

## Overview of Deploying Custom Configurations

In this section...
“About Deploying Custom Configurations” on page 22-2
“Deploying Custom Configurations Workflow” on page 22-2

### About Deploying Custom Configurations

When you create a custom configuration, often you *deploy* the custom configuration to your development group. Deploying the custom configuration allows your development group to review models using the same checks.

After you create a custom configuration, you can use it in the Model Advisor, or deploy the configuration to your users. You can deploy custom configurations whether you created the configuration using the Model Advisor Configuration Editor or within the customization file.

### Deploying Custom Configurations Workflow

When you deploy custom configurations, you:

- 1 Optionally author custom checks, as described in Chapter 20, “Authoring Custom Checks”.
- 2 Organize checks and folders to create custom configurations, as described in Chapter 21, “Creating Custom Configurations by Organizing Checks and Folders”.
- 3 Deploy the custom configuration to your users, as described in “How to Deploy Custom Configurations” on page 22-3.

## How to Deploy Custom Configurations

To deploy a custom configuration:

- 1 Determine which files to distribute. You might need to distribute more than one file.

<b>If You...</b>	<b>Using the...</b>	<b>Distribute...</b>
Created custom checks	Customization file	<ul style="list-style-type: none"> <li>• <code>sl_customization.m</code></li> <li>• Files containing check and action callback functions (if separate)</li> </ul>
Organized checks and folders	Model Advisor Configuration Editor	Configuration MAT file
	Customization file	<code>sl_customization.m</code>

- 2 Distribute the files and tell the user to include these files on the MATLAB path.
- 3 Instruct the user to load the custom configuration. For more details, see “Loading and Setting the Default Configuration” on page 22-4.

## Loading and Setting the Default Configuration

When you use the Model Advisor, you can load any configuration. Once you load a configuration, you can set it to be the configuration that the Model Advisor uses every time you open the Model Advisor.

- 1** Open the Model Advisor.
- 2** Select **File > Load Configuration**.
- 3** In the Open dialog box, navigate to and select the configuration file that you want to edit.
- 4** Click **Open**.

Simulink reloads the Model Advisor using the new configuration.

- 5** Optionally, set the current configuration as the default when the Model Advisor opens by selecting **File > Set Current Configuration as Default**.

---

**Tip** You can restore the MathWorks default configuration by selecting **File > Restore Default Configuration**.

---



# Function Reference

---

Requirements Management  
Interface (p. 23-2)

Model Coverage (p. 23-3)

Model Advisor Customization API  
(p. 23-5)

Model Advisor Result Template API  
(p. 23-7)

Model Advisor Formatting API  
(p. 23-8)

Access Requirements Management  
Interface

Configure and execute model  
coverage tests; store and report test  
results

Customize the Model Advisor tree;  
create new checks and folders

Template for formatting Model  
Advisor results

Format Model Advisor outputs

## Requirements Management Interface

rmi	Interact programmatically with Requirements Management Interface
rmidocrename	Update model requirements document paths and file names
rmitag	Manage user tags for requirements links

## Model Coverage

<code>add (cv.cvtestgroup)</code>	Add <code>cvtest</code> objects
<code>allNames (cv.cvdatagroup)</code>	Get names of all models associated with <code>cvdata</code> objects in <code>cv.cvdatagroup</code>
<code>allNames (cv.cvtestgroup)</code>	Get names of all models associated with <code>cvtest</code> objects in <code>cvtestgroup</code>
<code>conditioninfo</code>	Collect condition coverage information for model object
<code>cv.cvdatagroup</code>	Create collection of <code>cvdata</code> objects for model reference hierarchy
<code>cv.cvtestgroup</code>	Create collection of <code>cvtest</code> objects for model reference hierarchy
<code>cvexit</code>	Exit model coverage environment
<code>cvhtml</code>	Produce HTML report from model coverage objects
<code>cvload</code>	Load coverage tests and stored results into memory
<code>cvmodelview</code>	Display model coverage results with model coloring
<code>cvsave</code>	Save coverage tests and results to file
<code>cvsim</code>	Simulate and return model coverage results for test objects
<code>cvsimref</code>	Simulate and return model coverage results for referenced models
<code>cvtest</code>	Create model coverage test specification object
<code>decisioninfo</code>	Display decision coverage information for model object
<code>get (cv.cvdatagroup)</code>	Get <code>cvdata</code> object

<code>get (cv.cvtestgroup)</code>	Get <code>cvtest</code> objects
<code>getAll (cv.cvdatagroup)</code>	Get all <code>cvdata</code> objects
<code>getCoverageInfo</code>	Coverage information for Simulink Design Verifier blocks
<code>mcdeinfo</code>	Collect modified condition/decision coverage information for model object
<code>sigrangeinfo</code>	Collect signal range coverage information for model object
<code>tableinfo</code>	Display lookup table coverage information for model object

## Model Advisor Customization API

addCheck (ModelAdvisor.FactoryGroup)	Add check to folder
addGroup (ModelAdvisor.Group)	Add subfolder to folder
addTask (ModelAdvisor.Group)	Add task to folder
getID (ModelAdvisor.Check)	Return check identifier
ModelAdvisor.Action	Add actions to custom checks
ModelAdvisor.Check	Create custom checks
ModelAdvisor.FactoryGroup	Define subfolder in <b>By Task</b> folder
ModelAdvisor.Group	Define custom folder
ModelAdvisor.InputParameter	Add input parameters to custom checks
ModelAdvisor.ListViewParameter	Add list view parameters to custom checks
ModelAdvisor.Root	Identify root node
ModelAdvisor.Task	Define custom tasks
publish (ModelAdvisor.Root)	Publish object in Model Advisor root
register (ModelAdvisor.Root)	Register object in Model Advisor root
setAction (ModelAdvisor.Check)	Specify action for check
setCallbackFcn (ModelAdvisor.Action)	Specify action callback function
setCallbackFcn (ModelAdvisor.Check)	Specify callback function for check
setCheck (ModelAdvisor.Task)	Specify check used in task
setColSpan (ModelAdvisor.InputParameter)	Specify number of columns for input parameter
setInputParameters (ModelAdvisor.Check)	Specify input parameters for check

setInputParametersLayoutGrid  
(ModelAdvisor.Check)

Specify layout grid for input parameters

setRowSpan  
(ModelAdvisor.InputParameter)

Specify rows for input parameter

## Model Advisor Result Template API

<code>addRow</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add row to table
<code>ModelAdvisor.FormatTemplate</code>	Construct template object for formatting Model Advisor analysis results
<code>setCheckText</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add description of check to result
<code>setColTitles</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add column titles to table
<code>setInformation</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add description of subcheck to result
<code>setListObj</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add list of hyperlinks to model objects
<code>setRecAction</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add Recommended Action section and text
<code>setRefLink</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add See Also section and links
<code>setSubBar</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add line between subcheck results
<code>setSubResultStatus</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add status to check or subcheck result
<code>setSubResultStatusText</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add text below status in result
<code>setSubTitle</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add title for subcheck in result
<code>setTableInfo</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add data to table
<code>setTableTitle</code> ( <code>ModelAdvisor.FormatTemplate</code> )	Add title to table

## Model Advisor Formatting API

<code>addItem (ModelAdvisor.List)</code>	Add item to list
<code>addItem (ModelAdvisor.Paragraph)</code>	Add item to paragraph
<code>getEntry (ModelAdvisor.Table)</code>	Get table cell contents
<code>ModelAdvisor.Image</code>	Include image in Model Advisor output
<code>ModelAdvisor.LineBreak</code>	Insert line break
<code>ModelAdvisor.List</code>	Create list class
<code>ModelAdvisor.Paragraph</code>	Create and format paragraph
<code>ModelAdvisor.Table</code>	Create table
<code>ModelAdvisor.Text</code>	Create Model Advisor text output
<code>setAlign (ModelAdvisor.Paragraph)</code>	Specify paragraph alignment
<code>setBold (ModelAdvisor.Text)</code>	Specify bold text
<code>setColHeading (ModelAdvisor.Table)</code>	Specify table column title
<code>setColHeadingAlign (ModelAdvisor.Table)</code>	Specify column title alignment
<code>setColor (ModelAdvisor.Text)</code>	Specify text color
<code>setColWidth (ModelAdvisor.Table)</code>	Specify column widths
<code>setEntry (ModelAdvisor.Table)</code>	Add cell to table
<code>setEntryAlign (ModelAdvisor.Table)</code>	Specify table cell alignment
<code>setHeading (ModelAdvisor.Table)</code>	Specify table title
<code>setHeadingAlign (ModelAdvisor.Table)</code>	Specify table title alignment
<code>setHyperlink (ModelAdvisor.Image)</code>	Specify hyperlink location
<code>setHyperlink (ModelAdvisor.Text)</code>	Specify hyperlinked text
<code>setImageSource (ModelAdvisor.Image)</code>	Specify image location
<code>setItalic (ModelAdvisor.Text)</code>	Italicize text



<code>setRetainSpaceReturn</code> (ModelAdvisor.Text)	Retain spacing and returns in text
<code>setRowHeading</code> (ModelAdvisor.Table)	Specify table row title
<code>setRowHeadingAlign</code> (ModelAdvisor.Table)	Specify table row title alignment
<code>setSubscript</code> (ModelAdvisor.Text)	Specify subscripted text
<code>setSuperscript</code> (ModelAdvisor.Text)	Specify superscripted text
<code>setType</code> (ModelAdvisor.List)	Specify list type
<code>setUnderlined</code> (ModelAdvisor.Text)	Underline text



# Class Reference

---

- “Model Coverage” on page 24-2
- “Model Advisor Customization API” on page 24-3
- “Model Advisor Result Template API” on page 24-4
- “Model Advisor Formatting API” on page 24-5

## **Model Coverage**

cv.cvdatagroup

Collection of cvdata objects

cv.cvtestgroup

Collection of cvtest objects

## Model Advisor Customization API

ModelAdvisor.Action	Add actions to custom checks
ModelAdvisor.Check	Create custom checks
ModelAdvisor.FactoryGroup	Define subfolder in <b>By Task</b> folder
ModelAdvisor.Group	Define custom folder
ModelAdvisor.InputParameter	Add input parameters to custom checks
ModelAdvisor.ListViewParameter	Add list view parameters to custom checks
ModelAdvisor.Root	Identify root node
ModelAdvisor.Task	Define custom tasks

## **Model Advisor Result Template API**

ModelAdvisor.FormatTemplate

Template for formatting Model  
Advisor analysis results

## Model Advisor Formatting API

ModelAdvisor.Image	Include image in Model Advisor output
ModelAdvisor.LineBreak	Insert line break
ModelAdvisor.List	Create list class
ModelAdvisor.Paragraph	Create and format paragraph
ModelAdvisor.Table	Create table
ModelAdvisor.Text	Create Model Advisor text output





# Alphabetical List

---

# cv.cvtestgroup.add

---

**Purpose** Add cvtest objects

**Syntax** `add(cvtg, cvto1, cvto2, ...)`

**Description** `add(cvtg, cvto1, cvto2, ...)` adds the cvtest objects specified by the strings `cvto1`, `cvto2`, etc. to `cvtg`, which is an instantiation of the `cv.cvtestgroup` class.

**Example** Create two cvtest objects and add them to a newly created `cv.cvtestgroup` object:

```
cvto1 = cvtest;  
cvto2 = cvtest;  
cvtg = cv.cvtestgroup;  
add(cvtg, cvto1, cvto2);
```

# ModelAdvisor.FactoryGroup.addCheck

---

**Purpose** Add check to folder

**Syntax** addCheck(fg\_obj, check\_ID)

**Description** addCheck(fg\_obj, check\_ID) adds checks, identified by check\_ID, to the folder specified by fg\_obj, which is an instantiation of the ModelAdvisor.FactoryGroup class.

**Examples** Add three checks to rec:

```
% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
.
.
.
addCheck(rec, 'com.mathworks.sample.Check1');
addCheck(rec, 'com.mathworks.sample.Check2');
addCheck(rec, 'com.mathworks.sample.Check3');
```

# ModelAdvisor.Group.addGroup

---

**Purpose** Add subfolder to folder

**Syntax** `addGroup(group_obj, child_obj)`

**Description** `addGroup(group_obj, child_obj)` adds a new subfolder, identified by `child_obj`, to the folder specified by `group_obj`, which is an instantiation of the `ModelAdvisor.Group` class.

**Examples** Add three checks to rec:

```
group_obj = ModelAdvisor.Group('com.mathworks.sample.group');  
.  
.  
.  
addGroup(group_obj, 'com.mathworks.sample.subgroup1');  
addGroup(group_obj, 'com.mathworks.sample.subgroup2');  
addGroup(group_obj, 'com.mathworks.sample.subgroup3');
```

**Purpose** Add item to list

**Syntax** `addItem(element)`

**Description** `addItem(element)` adds items to the list created by the `ModelAdvisor.List` constructor.

**Input Arguments** *element* Specifies an element to be added to a list in one of the following:

- Element
- Cell array of elements. When you add a cell array to a list, they form different rows in the list.
- String

**Example**

```
subList = ModelAdvisor.List();
setType(subList, 'numbered')
addItem(subList, ModelAdvisor.Text('Sub entry 1', {'pass','bold'}));
addItem(subList, ModelAdvisor.Text('Sub entry 2', {'pass','bold'}));
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Paragraph.addItem

---

**Purpose** Add item to paragraph

**Syntax** addItem(text, element)

**Description** addItem(text, element) adds an element to text. element is one of the following:

- String
- Element
- Cell array of elements

**Example** Add two lines of text:

```
result = ModelAdvisor.Paragraph;  
addItem(result, [resultText1 ModelAdvisor.LineBreak resultText2]);
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.FormatTemplate.addRow

---

**Purpose** Add row to table

**Syntax** `addRow(ft_obj, {item1, item2, ..., itemn})`

**Description** `addRow(ft_obj, {item1, item2, ..., itemn})` is an optional method that adds a row to the end of a table in the result. `ft_obj` is a handle to the template object previously created. `{item1, item2, ..., itemn}` is a cell array of strings and objects to add to the table. The order of the items in the array determines which column the item is in. If you do not add data to the table, the Model Advisor does not display the table in the result.

---

**Note** Before adding rows to a table, you must specify column titles using the `setColTitle` method.

---

**Examples** Find all of the blocks in the model and create a table of the blocks:

```
% Create FormatTemplate object, specify table format
ft = ModelAdvisor.FormatTemplate('TableTemplate');

% Add information to the table
setTableTitle(ft, {'Blocks in Model'});
setColTitles(ft, {'Index', 'Block Name'});
% Find all the blocks in the system and add them to a table.
allBlocks = find_system(system);
for inx = 2 : length(allBlocks)
    % Add information to the table
    addRow(ft, {inx-1,allBlocks(inx)});
end
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

# ModelAdvisor.Group.addTask

---

**Purpose** Add task to folder

**Syntax** `addTask(group_obj, task_obj)`

**Description** `addTask(group_obj, task_obj)` adds a task, specified by `task_obj`, to the folder `group_obj.group_obj` is an instantiation of the `ModelAdvisor.Group` class.

**Example** Add three tasks to MAG.

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');  
addTask(MAG, MAT1);  
addTask(MAG, MAT2);  
addTask(MAG, MAT3);
```



- Purpose** Get names of all models associated with cvdata objects in cv.cvdatagroup
- Syntax** `models = allNames(cvdg)`
- Description** `models = allNames(cvdg)` returns a cell array of strings identifying all model names associated with the cvdata objects in cvdg, an instantiation of the cv.cvdatagroup class.
- Examples** Add three cvdata objects to cvdg and return a cell array of model names:
- ```
a = cvdata;  
b = cvdata;  
c = cvdata;  
cvdg = cv.cvdatagroup;  
add (cvdg, a, b, c);  
model_names = allNames(cvdg)
```

# cv.cvtestgroup.allNames

---

**Purpose** Get names of all models associated with `cvtest` objects in `cvtestgroup`

**Syntax** `models = allNames(cvtg)`

**Description** `models = allNames(cvtg)` returns a cell array of strings identifying all model names associated with the `cvtest` objects in `cvtg`, an instantiation of the `cv.cvtestgroup` class.

**Examples** Add three `cvtest` objects to `cvtg` and return a cell array of model names:

```
d = cvtest;
e = cvtest;
f = cvtes;
cvtg = cv.cvtestgroup;
add (cvtg, d, e, f);
model_names = allNames(cvtg)
```

## Purpose

Collect condition coverage information for model object

## Syntax

```
coverage = conditioninfo(cvdo, object)
coverage = conditioninfo(cvdo, object, ignore_descendants)
[coverage, description] = conditioninfo(cvdo, object)
```

## Description

`coverage = conditioninfo(cvdo, object)` returns condition coverage results from the cvdata object `cvdo` for the model component specified by `object`.

`coverage = conditioninfo(cvdo, object, ignore_descendants)` returns condition coverage results for `object`, depending on the value of `ignore_descendants`.

`[coverage, description] = conditioninfo(cvdo, object)` returns condition coverage results and textual descriptions of each condition in `object`.

## Input Arguments

`cvdo`

cvdata object

`ignore_descendants`

Logical value that specifies whether to ignore the coverage of descendant objects

1 to ignore coverage of descendant objects

0 (default) to collect coverage of descendant objects

`object`

An object in the Simulink model or Stateflow diagram that receives decision coverage. Valid values for `object` are as follows:

`BlockPath`

Full path to a Simulink model or block

`BlockHandle`

Handle to a Simulink model or block

# conditioninfo

---

|                                  |                                                                                                         |
|----------------------------------|---------------------------------------------------------------------------------------------------------|
| <code>s1Obj</code>               | Handle to a Simulink API object                                                                         |
| <code>sfID</code>                | Stateflow ID                                                                                            |
| <code>sfObj</code>               | Handle to a Stateflow API object                                                                        |
| <code>{BlockPath, sfID}</code>   | Cell array with the path to a Stateflow chart and the ID of an object contained in that chart           |
| <code>{BlockPath, sfObj}</code>  | Cell array with the path to a Stateflow chart and a Stateflow object API handle contained in that chart |
| <code>[BlockHandle, sfID]</code> | Array with a Stateflow block handle and the ID of an object contained in that chart                     |

## Output Arguments

`coverage`

The value of `coverage` is a two-element vector of form `[covered_outcomes total_outcomes]`. `coverage` is empty if `cvdo` does not contain condition coverage results for object. The two elements are:

|                               |                                                   |
|-------------------------------|---------------------------------------------------|
| <code>covered_outcomes</code> | Number of condition outcomes satisfied for object |
| <code>total_outcomes</code>   | Total number of condition outcomes for object     |

`description`

A structure array with the following fields:

|           |                                                                     |
|-----------|---------------------------------------------------------------------|
| text      | String describing a condition or the block port to which it applies |
| trueCnts  | Number of times the condition was true in a simulation              |
| falseCnts | Number of times the condition was false in a simulation             |

## Examples

The following example opens the `slvndemo_cv_small_controller` demo model, creates the test specification object `testObj`, enables condition coverage for `testObj`, and executes `testObj`. Then retrieve the condition coverage results for the Logic block (in the Gain subsystem) and determine its percentage of condition outcomes covered:

```
mdl = 'slvndemo_cv_small_controller';
open_system(mdl)
testObj = cvtest(mdl)
testObj.settings.condition = 1;
data = cvsim(testObj)
blk_handle = get_param([mdl, '/Gain/Logic'], 'Handle');
cov = conditioninfo(data, blk_handle)
percent_cov = 100 * cov(1) / cov(2)
```

## Alternatives

To collect condition coverage for a model using the GUI:

- 1 Open the model for which you want condition coverage.
- 2 In the Model Editor, select **Tools > Coverage Settings**.
- 3 On the **Coverage** tab, under **Coverage Metrics**, select **Condition Coverage**.
- 4 View the **Results** and **Report** tab to specify the type of output you need.
- 5 Click **OK**.

# conditioninfo

---

6 Simulate the model.

## See Also

[decisioninfo](#) | [mcdcinfo](#) | [sigrangeinfo](#) | [tableinfo](#)

## How To

- “Condition Coverage (CC)” on page 15-4

|                       |                                                                                                                                                                                                                                  |                |                                                                          |     |                   |        |                        |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|--------------------------------------------------------------------------|-----|-------------------|--------|------------------------|
| <b>Purpose</b>        | Collection of cvdata objects                                                                                                                                                                                                     |                |                                                                          |     |                   |        |                        |
| <b>Description</b>    | Instances of this class contain a collection of cvdata objects. For more information, see “Extracting Results from cv.cvdatagroup” on page 18-16.                                                                                |                |                                                                          |     |                   |        |                        |
| <b>Construction</b>   | <table><tr><td>cv.cvdatagroup</td><td>Create collection of cvdata objects for model reference hierarchy</td></tr></table>                                                                                                        | cv.cvdatagroup | Create collection of cvdata objects for model reference hierarchy        |     |                   |        |                        |
| cv.cvdatagroup        | Create collection of cvdata objects for model reference hierarchy                                                                                                                                                                |                |                                                                          |     |                   |        |                        |
| <b>Methods</b>        | <table><tr><td>allNames</td><td>Get names of all models associated with cvdata objects in cv.cvdatagroup</td></tr><tr><td>get</td><td>Get cvdata object</td></tr><tr><td>getAll</td><td>Get all cvdata objects</td></tr></table> | allNames       | Get names of all models associated with cvdata objects in cv.cvdatagroup | get | Get cvdata object | getAll | Get all cvdata objects |
| allNames              | Get names of all models associated with cvdata objects in cv.cvdatagroup                                                                                                                                                         |                |                                                                          |     |                   |        |                        |
| get                   | Get cvdata object                                                                                                                                                                                                                |                |                                                                          |     |                   |        |                        |
| getAll                | Get all cvdata objects                                                                                                                                                                                                           |                |                                                                          |     |                   |        |                        |
| <b>Properties</b>     | <table><tr><td>name</td><td>cv.cvdatagroup object name</td></tr></table>                                                                                                                                                         | name           | cv.cvdatagroup object name                                               |     |                   |        |                        |
| name                  | cv.cvdatagroup object name                                                                                                                                                                                                       |                |                                                                          |     |                   |        |                        |
| <b>Copy Semantics</b> | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.                                                                                               |                |                                                                          |     |                   |        |                        |

# cv.cvdatagroup

---

**Purpose** Create collection of cvdata objects for model reference hierarchy

**Syntax** `cvdg = cv.cvdatagroup(cvdo1, cvdo2, ...)`

**Description** `cvdg = cv.cvdatagroup(cvdo1, cvdo2, ...)` creates an instantiation of the `cv.cvdatagroup` class (`cvdg`) that contains the `cvdata` objects `cvdo1`, `cvdo2`, etc. A `cvdata` object contains results of the simulation runs.

**Examples** Create an instantiation of the `cv.cvdatagroup` class and add two `cvdata` objects to it:

```
a = cvdata;  
b = cvdata;  
cvdg = cv.cvdatagroup(a, b);
```



|                       |                                                                                                                                                                                                                               |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Collection of <code>cvtest</code> objects                                                                                                                                                                                     |
| <b>Description</b>    | Instances of this class contain a collection of <code>cvtest</code> objects. For more information, see “Creating a Test Group with <code>cv.cvtestgroup</code> ” on page 18-15.                                               |
| <b>Construction</b>   | <code>cv.cvtestgroup</code> Create collection of <code>cvtest</code> objects for model reference hierarchy                                                                                                                    |
| <b>Methods</b>        | <code>add</code> Add <code>cvtest</code> objects<br><code>allNames</code> Get names of all models associated with <code>cvtest</code> objects in <code>cvtestgroup</code><br><code>get</code> Get <code>cvtest</code> objects |
| <b>Properties</b>     | <code>name</code> <code>cv.cvtestgroup</code> object name                                                                                                                                                                     |
| <b>Copy Semantics</b> | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.                                                                                            |

## cv.cvtestgroup

---

**Purpose** Create collection of `cvtest` objects for model reference hierarchy

**Syntax** `cvtg = cv.cvtestgroup(cvto1, cvto2, ...)`

**Description** `cvtg = cv.cvtestgroup(cvto1, cvto2, ...)` creates an instantiation of the `cv.cvtestgroup` class (`cvtg`) that contains the `cvtest` objects `cvto1`, `cvto2`, etc. A `cvtest` object is a test specification object for a Simulink model.

**Examples** Create an instantiation of the `cv.cvtestgroup` class and add two `cvtest` objects to it:

```
a = cvtest;  
b = cvtest;  
cvtg = cv.cvtestgroup(a, b);
```

**See Also** `cvtest`

|                    |                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Exit model coverage environment                                                                                                                                                                       |
| <b>Syntax</b>      | <code>cvexit</code>                                                                                                                                                                                   |
| <b>Description</b> | <code>cvexit</code> exits the model coverage environment. Issuing this command closes the Coverage Display window and removes coloring from a block diagram that displays its model coverage results. |

# cvhtml

---

**Purpose** Produce HTML report from model coverage objects

**Syntax**

```
cvhtml(file, cvdo)
cvhtml(file, cvdo1, cvdo2, ...)
cvhtml(file, cvdo1, cvdo2, ..., options)
cvhtml(file, cvdo1, cvdo2, ..., options, detail)
```

**Description** `cvhtml(file, cvdo)` creates an HTML report of the coverage results in the `cvdata` or `cv.cvdatagroup` object `cvdo` when you run model coverage in simulation. `cvhtml` saves the coverage results in `file`. The model must be open when you use `cvhtml` to generate its coverage report.

`cvhtml(file, cvdo1, cvdo2, ...)` creates a combined report of several `cvdata` objects. The results from each object appear in a separate column of the HTML report. Each `cvdata` object must correspond to the same root model or subsystem. Otherwise, the function fails.

`cvhtml(file, cvdo1, cvdo2, ..., options)` creates a combined report of several `cvdata` objects using the report options specified by `options`.

`cvhtml(file, cvdo1, cvdo2, ..., options, detail)` creates a combined report of several `cvdata` objects and specifies the detail level of the report with the value of `detail`.

## Input Arguments

`cvdo`

A `cv.cvdatagroup` object

`detail`

Specifies the level of detail in the report. Set `detail` to an integer from 0 to 3. Greater numbers for `detail` indicate greater detail.

**Default:** 2

`file`

String specifying the HTML file in the MATLAB current folder where cvhtml stores the results

**Default:** []

### options

Specify the report options that you specify in options:

- To enable an option, set it to 1 (e.g., '-hTR=1').
- To disable an option, set it to 0 (e.g., '-bRG=0').
- To specify multiple report options, list individual options in a single options string separated by commas or spaces (e.g., '-hTR=1 -bRG=0 -scm=0').

The following table lists all the options:

| <b>Option</b> | <b>Description</b>                                     | <b>Default</b> |
|---------------|--------------------------------------------------------|----------------|
| -aTS          | Include each test in the model summary                 | on             |
| -bRG          | Produce bar graphs in the model summary                | on             |
| -bTC          | Use two color bar graphs (red, blue)                   | off            |
| -hTR          | Display hit/count ratio in the model summary           | off            |
| -nFC          | Do not report fully covered model objects              | off            |
| -scm          | Include cyclomatic complexity numbers in summary       | on             |
| -bcm          | Include cyclomatic complexity numbers in block details | on             |

## Examples

Make sure you have write access to the default MATLAB folder. Create a cumulative coverage report for the `slvndemo_cv_small_controller` mode and save it as `ratelim_coverage.html`:

```
model = 'slvndemo_cv_small_controller';  
open_system(model);  
cvt = cvtest(model);  
cvd = cvsim(cvt);  
outfile = 'ratelim_coverage.html';  
cvhtml(outfile, cvd);
```

## Alternatives

To create an HTML model coverage report:

- 1** Open the model for which you want model coverage.
- 2** In the Model Editor, select **Tools > Coverage Settings**.
- 3** On the **Report** tab of the Coverage Settings dialog box, select **Generate HTML report**.
- 4** Click **OK**.

## See Also

`cv.cvdatagroup` | `cvmodelview` | `cvsim`

## How To

- “Creating HTML Reports with `cvhtml`” on page 18-8

**Purpose** Load coverage tests and stored results into memory

**Syntax** `[cvtos, cvdos] = cvload(filename)`  
`[cvtos, cvdos] = cvload(filename, restoretotal)`

**Description** `[cvtos, cvdos] = cvload(filename)` loads the tests and data stored in the text file `filename.cvt`. `cvtos` is a cell array of `cvtest` objects that are successfully loaded. `cvdos` is a cell array of `cvdata` objects that are successfully loaded. `cvdos` has the same size as `cvtos`, but if a particular test has no results, `cvdos` can contain empty elements.

`[cvtos, cvdos] = cvload(filename, restoretotal)` restores or clears the cumulative results from prior runs, depending on the value of `restoretotal`. If `restoretotal` is 1, `cvload` restores the cumulative results from prior runs. If `restoretotal` is unspecified or 0, `cvload` clears the model's cumulative results.

The following are special considerations for using the `cvload` command:

- If a model with the same name exists in the coverage database, the software loads only the compatible results that reference the existing model to prevent duplication.
- If the Simulink models referenced from the file are open but do not exist in the coverage database, the coverage tool resolves the links to the existing models.
- When you are loading several files that reference the same model, the software loads only the results that are consistent with the earlier files.

**Examples** Store coverage results in `cvtest` and `cvdata` objects:

```
[test_objects, data_objects] = cvload(test_results, 1);
```

**See Also** `cvsave`

**How To** • “Loading Stored Coverage Test Results with `cvload`” on page 18-10

# cvmodelview

---

**Purpose** Display model coverage results with model coloring

**Syntax** `cvmodelview(cvdo)`

**Description** `cvmodelview(cvdo)` displays coverage results from the `cvdata` object `cvdo` by coloring the objects in the model that have model coverage results.

**Examples** Open the `slvndemo_cv_small_controller` demo model, create the test specification object `testObj`, and execute `testObj` to collect model coverage. Run `cvmodelview` to color the model objects for which you collect model coverage information:

```
mdl = 'slvndemo_cv_small_controller';
open_system(mdl)
testObj = cvtest(mdl)
data = cvsim(testObj)
cvmodelview(data)
```

**Alternatives** To display model coverage results by coloring objects:

- 1** Open the model.
- 2** Select **Tools > Coverage Settings**.
- 3** On the **Coverage** tab, select **Coverage for this model**.
- 4** On the **Results** tab, select **Display coverage results using model coloring**.
- 5** Click **OK** to close the Coverage Settings dialog box.
- 6** Simulate the model.

**See Also** `cvhtml` | `cvsim`

**How To** • “Enabling the Colored Diagram Display” on page 15-56



- “Displaying Model Coverage with Model Coloring” on page 15-57

**Purpose** Save coverage tests and results to file

**Syntax**

```
cvsave(filename, model)
cvsave(filename, cvto1, cvto2, ...)
cvsave(filename, cvdo1, cvdo2, ...)
```

**Description** `cvsave(filename, model)` saves all the tests (cvtest objects) and results (cvdata objects) related to `model` in the text file `filename.cvt`. `model` is a handle to or name of a Simulink model.

`cvsave(filename, cvto1, cvto2, ...)` saves multiple cvtest objects in the text file `filename.cvt`. `cvsave` also saves information about any referenced models.

`cvsave(filename, cvdo1, cvdo2, ...)` saves the tests and test results for multiple cvdata objects to the text file `filename.cvt`. `cvsave` also saves information about any referenced models.

**Examples** Save coverage results for the `slvndemo_cv_small_controller` model in `ratelim_testdata.cvt`:

```
model = 'slvndemo_cv_small_controller';
open_system(model);
cvt = cvtest(model);
cvd = cvsim(cvt);
cvsave('ratelim_testdata', model);
```

**Alternatives** To save cumulative coverage results:

- 1** In the Model Editor, select **Tools > Coverage Settings**.
- 2** On the **Results** tab:
  - a** Select **Save cumulative results in workspace variable**.
  - b** Select **Save last run in workspace variable**.
- 3** Click OK to close the Coverage Settings dialog box.

**4** Simulate the model.

**See Also**

`cvload`

**How To**

- “Saving Test Runs to a File with `cvsave`” on page 18-9

**Purpose** Simulate and return model coverage results for test objects

**Syntax**

```
cvdo = cvsim(cvto)
[cvdo,t,x,y] = cvsim(cvto)
[cvdo,t,x,y] = cvsim(cvto, timespan, options)
[cvdo,t,x,y] = cvsim(cvto, label, setupcmd)
[cvdo1, cvdo2, ...] = cvsim(cvto1, cvto2, ...)
```

**Description**

`cvdo = cvsim(cvto)` simulates the `cvtest` object `cvto` by starting a simulation run for the corresponding model. The software returns the results in the `cvdata` object `cvdo`. However, when recording coverage for multiple models in a hierarchy, `cvsim` returns its results in a `cv.cvdatagroup` object.

`[cvdo,t,x,y] = cvsim(cvto)` returns the time vector `t`, matrix of state values `x`, and matrix of output values `y` from the simulation.

`[cvdo,t,x,y] = cvsim(cvto, timespan, options)` returns the time vector `t`, matrix of state values `x`, and matrix of output values `y` from the simulation, and overrides default simulation values with the values for `timespan` and `options`.

`[cvdo,t,x,y] = cvsim(cvto, label, setupcmd)` creates the `cvtest` object `cvto` and simulates it in one command. The arguments `label` and `setupcmd` are passed directly to the `cvtest` function, which creates the `cvtest` object `cvto`.

`[cvdo1, cvdo2, ...] = cvsim(cvto1, cvto2, ...)` executes the `cvtest` objects `cvto1`, `cvto2`, ... and returns the results in the set of `cvdata` objects `cvdo1`, `cvdo2`, ....

You do not have to enable model coverage reporting for the model to use the `cvsim` command.

**Input Arguments**

`cvto`  
cvtest object

`label`

Label for test object (passed to cvtest)

**Default:** []

options

Optional simulation parameters specified as a structure created by the `simset` command.

setupcmd

Setup command used to create test object (passed to cvtest)

**Default:** []

timespan

Simulation start and stop time:

`tFinal` To specify the stop time. The start time is 0.

`[tStart tFinal]` To specify the start and stop times.

`[tStart OutputTimes tFinal]` To specify the start and stop times and time points to be returned in `t`.

Generally, `t` includes more time points. `OutputTimes` is equivalent to specifying **Configuration Parameters > Data Import/Export > Output options > Produce specified output only**.

## Output Arguments

|                   |                                                                                                                                                                                                                   |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>cvdo</code> | cvdata object                                                                                                                                                                                                     |
| <code>t</code>    | The simulation's time vector                                                                                                                                                                                      |
| <code>x</code>    | The simulation's state matrix consisting of continuous states followed by discrete states                                                                                                                         |
| <code>y</code>    | The simulation's output matrix. Each column contains the output of a root-level Outport block, in port number order. If any Outport block has a vector input, its output takes the appropriate number of columns. |

## Examples

Simulate the `slvndemo_cv_small_controller` model, get the test data, and simulate the model with that test data. `cvsim` returns the time vector, matrix of state values, and matrix of output values:

```
model = 'slvndemo_cv_small_controller';  
open_system(model);  
testObj = cvtest(model); %Get test data  
[data, T, X, Y] = cvsim(testObj); %Get coverage data
```

## See Also

`cv.cvdatagroup` | `cvtest` | `simset`

## How To

- “Creating and Running Test Cases” on page 15-32

**Purpose**

Simulate and return model coverage results for referenced models

**Syntax**

```
cvdg = cvsimref(topModelName)
cvdg = cvsimref(topModelName, cvtg)
[cvdg,t,x,y] = cvsimref(topModelName, cvtg)
[cvdg,t,x,y] = cvsimref(topModelName, cvtg, timespan,
    options)
[cvdg1, cvdg2, ...] = cvsimref(topModelName, cvtg1, cvtg2,
    ...)
```

**Description**

`cvdg = cvsimref(topModelName)` simulates the top model and all referenced models in the hierarchy, collects model coverage data, and returns the results in the `cv.cvdagroup` object `cvdg`. You do not have to enable model coverage reporting for any of the models in a model hierarchy to use the `cvsimref` command.

`cvdg = cvsimref(topModelName, cvtg)` simulates `topModelName` and collects model coverage data by executing the `cv.cvtestgroup` object `cvtg`. `cvtg` contains `cvtest` specifications for the top-level model and all the referenced models in the hierarchy. `cvsimref` returns the model coverage results in `cvdg`.

`[cvdg,t,x,y] = cvsimref(topModelName, cvtg)` returns the time vector `t`, matrix of state values `x`, and matrix of output values `y` from the simulation.

`[cvdg,t,x,y] = cvsimref(topModelName, cvtg, timespan, options)` overrides default simulation values with the values in `timespan` and `options`.

`[cvdg1, cvdg2, ...] = cvsimref(topModelName, cvtg1, cvtg2, ...)` executes multiple `cv.cvtestgroup` objects and returns the results in a set of `cv.cvdagroup` objects.

**Input Arguments**

`cvtg`

`cv.cvtestgroup` object that contains test specifications for the referenced models in the hierarchy

## options

Optional simulation parameters specified as a structure created by the `simset` command.

## timespan

Simulation start and stop time:

|                                          |                                                                                                                                                                                                                                                                                                                                   |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>tFinal</code>                      | Specify the stop time. The start time is 0.                                                                                                                                                                                                                                                                                       |
| <code>[tStart tFinal]</code>             | Specify the start and stop times.                                                                                                                                                                                                                                                                                                 |
| <code>[tStart OutputTimes tFinal]</code> | Specify that <code>cvsimref</code> return the start and stop times and time points in <code>t</code> . Generally, <code>t</code> includes more time points. <code>OutputTimes</code> is equivalent to specifying <b>Configuration Parameters &gt; Data Import/Export &gt; Output options &gt; Produce specified output only</b> . |

## topModelName

Name of the top-level model in the hierarchy

## Output Arguments

### cvdg

`cv.cvdatalog` object

### t

The simulation time vector

### x

The simulation state matrix consisting of continuous states followed by discrete states



y

The simulation output matrix. Each column contains the output of a root-level Outport block, in port number order. If any Outport block has a vector input, its output takes the appropriate number of columns.

## Examples

Open and simulate the `slvndemo_ratelim_harness` model and its two subsystems:

```
topModel = 'slvndemo_cv_mutual_exclusion';
load_system(topModel);
% Make sure coverage is off for this run for the entire tree
set_param(topModel, 'RecordCoverage', 'off');
set_param(topModel, 'CovModelRefEnable', 'Off');
[T1, X1, Y1] = sim(topModel);           % Normal data
[allData, T2, X2, Y2] = cvsimref(topModel); % cvsimref data
```

## See Also

`cv.cvdatagroup` | `cv.cvtestgroup` | `cvsim` | `cvtest` | `simset`

## How To

- “Creating and Running Test Cases” on page 15-32
- “Using Model Coverage Commands for Referenced Models” on page 18-12

# cvtest

---

**Purpose** Create model coverage test specification object

**Syntax**

```
cvto = cvtest(root)
cvto = cvtest(root, label)
cvto = cvtest(root, label, setupcmd)
```

**Description** `cvto = cvtest(root)` creates a test specification object with the handle `cvto`. Simulate `cvto` with the `cvsim` command.

`cvto = cvtest(root, label)` creates a test object with the label `label`, which is used for reporting results.

`cvto = cvtest(root, label, setupcmd)` creates a test object with the setup command `setupcmd`.

## Input Arguments

`label`

Label for test object

`root`

The name of, or a handle to, a Simulink model or a subsystem. Only the specified model or subsystem and its descendants are subject to model coverage testing.

`setupcmd`

Setup command for creating test object. The setup command is executed in the base MATLAB workspace just prior to running the simulation. This command is useful for loading data prior to a test.

## Output Arguments

`cvto`

A test specification object with the following structure:

| Field                 | Description           |
|-----------------------|-----------------------|
| <code>id</code>       | Read-only internal ID |
| <code>modelcov</code> | Read-only internal ID |

| Field                            | Description                                                                                                                                                                                                                                                                                            |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rootPath                         | Name of system or subsystem for analysis                                                                                                                                                                                                                                                               |
| label                            | String used when reporting results                                                                                                                                                                                                                                                                     |
| setupCmd                         | Command executed in base workspace prior to simulation                                                                                                                                                                                                                                                 |
| settings.condition               | Set to 1 for condition coverage.                                                                                                                                                                                                                                                                       |
| settings.decision                | Set to 1 for decision coverage.                                                                                                                                                                                                                                                                        |
| settings.designverifier          | Set to 1 for coverage for Simulink Design Verifier blocks.                                                                                                                                                                                                                                             |
| settings.mcdc                    | Set to 1 for MC/DC coverage.                                                                                                                                                                                                                                                                           |
| settings.sigrange                | Set to 1 for signal range coverage.                                                                                                                                                                                                                                                                    |
| settings.sigsize                 | Set to 1 for signal size coverage.                                                                                                                                                                                                                                                                     |
| settings.tableExec               | Set to 1 for lookup table coverage.                                                                                                                                                                                                                                                                    |
| modelRefSettings.enable          | <ul style="list-style-type: none"> <li>• 'off' — Disable coverage for all referenced models.</li> <li>• 'all' or on — Enable coverage for all referenced models.</li> <li>• 'filtered' — Enable coverage only for referenced models not listed in the <code>excludedModels</code> subfield.</li> </ul> |
| modelRefSettings.excludeTopModel | Set to 1 to exclude coverage for the top model.                                                                                                                                                                                                                                                        |
| modelRefSettings.excludedModels  | String specifying a comma-separated list of referenced models for which coverage is disabled.                                                                                                                                                                                                          |

| Field                           | Description                                                                                               |
|---------------------------------|-----------------------------------------------------------------------------------------------------------|
| emlSettings.<br>enableExternal  | Set to 1 to enable coverage for external program files called by Embedded MATLAB functions in your model. |
| options.<br>forceBlockReduction | Set to 1 to override the Simulink <b>Block reduction</b> parameter if it is enabled.                      |

## Examples

Create a `cvtest` object of the Adjustable Rate Limiter block in the demo model `slvndemo_ratelim_harness` and display its contents:

```
open_system('slvndemo_ratelim_harness');  
testObj1 = cvtest(['slvndemo_ratelim_harness', ...  
    '/Adjustable Rate Limiter']);  
testObj1.label = 'Gain within slew limits';  
testObj1.setupCmd = 'load(''within_lim.mat'');';  
testObj1.settings.mcdc = 1;  
testObj1
```

## See Also

`cv.cvtestgroup`

## How To

- “Creating Tests with `cvtest`” on page 18-3
- “Creating a Test Group with `cv.cvtestgroup`” on page 18-15

**Purpose** Display decision coverage information for model object

**Syntax**

```
coverage = decisioninfo(cvdo, object)
coverage = decisioninfo(cvdo, object, ignore_descendants)
[coverage, description] = decisioninfo(cvdo, object)
```

**Description** `coverage = decisioninfo(cvdo, object)` returns decision coverage results from the cvdata object `cvdo` for the model component specified by `object`.

`coverage = decisioninfo(cvdo, object, ignore_descendants)` returns decision coverage results for `object`, depending on the value of `ignore_descendants`.

`[coverage, description] = decisioninfo(cvdo, object)` returns decision coverage results and text descriptions of decision points associated with `object`.

## Input Arguments

`cvdo`  
cvdata object

`ignore_descendants`  
Specifies to ignore the coverage of descendant objects if `ignore_descendants` is set to 1.

`object`  
The `object` argument specifies an object in the model or Stateflow chart that received decision coverage. Valid values for `object` include the following:

| Object Specification | Description                     |
|----------------------|---------------------------------|
| BlockPath            | Full path to a model or block   |
| BlockHandle          | Handle to a model or block      |
| s1obj                | Handle to a Simulink API object |

## Object Specification

## Description

|                                  |                                                                                                         |
|----------------------------------|---------------------------------------------------------------------------------------------------------|
| <code>sfID</code>                | Stateflow ID                                                                                            |
| <code>sfObj</code>               | Handle to a Stateflow API object                                                                        |
| <code>{BlockPath, sfID}</code>   | Cell array with the path to a Stateflow chart and the ID of an object contained in that chart           |
| <code>{BlockPath, sfObj}</code>  | Cell array with the path to a Stateflow chart and a Stateflow object API handle contained in that chart |
| <code>[BlockHandle, sfID]</code> | Array with a Stateflow chart handle and the ID of an object contained in that chart                     |

## Output Arguments

`coverage`

The value of `coverage` is a two-element vector of the form `[covered_outcomes total_outcomes]`. `coverage` is empty if `cvdo` does not contain decision coverage results for `object`. The two elements are:

|                               |                                                               |
|-------------------------------|---------------------------------------------------------------|
| <code>covered_outcomes</code> | Number of decision outcomes satisfied for <code>object</code> |
| <code>total_outcomes</code>   | Number of decision outcomes for <code>object</code>           |

`description`

`description` is a structure array containing the following fields:

|                                              |                                                               |
|----------------------------------------------|---------------------------------------------------------------|
| <code>decision.text</code>                   | String describing a decision point, e.g., 'U > LL'            |
| <code>decision.outcome.text</code>           | String describing a decision outcome, i.e., 'true' or 'false' |
| <code>decision.outcome.executionCount</code> | Number of times a decision outcome occurred in a simulation   |

## Examples

Open the `slvndemo_cv_small_controller` model and create the test specification object `testObj`. Enable decision coverage for `testObj` and execute `testObj` using `cvsim`. Use `decisioninfo` to retrieve the decision coverage results for the Saturation block and determine the percentage of decision outcomes covered:

```
mdl = 'slvndemo_cv_small_controller';
open_system(mdl)
testObj = cvtest(mdl)
testObj.settings.decision = 1;
data = cvsim(testObj)
blk_handle = get_param([mdl, '/Saturation'], 'Handle');
cov = decisioninfo(data, blk_handle)
percent_cov = 100 * cov(1) / cov(2)
```

## Alternatives

To collect and display decision coverage results:

- 1 Open the model.
- 2 In the Model Editor, select **Tools > Coverage Settings**.
- 3 On the **Coverage** tab, under **Coverage Metrics**, select **Decision Coverage**.
- 4 Click **OK** to close the Coverage Settings dialog box.
- 5 Simulate the model and review the results.

# decisioninfo

---

## **See Also**

[conditioninfo](#) | [cvsim](#) | [mcdcinfo](#) | [sigrangeinfo](#) | [tableinfo](#)

## **How To**

- “Condition Coverage (CC)” on page 15-4



**Purpose** Get cvdata object

**Syntax** `get(cvdg, model_name)`

**Description** `get(cvdg, model_name)` returns the cvdata object in the `cv.cvdatagroup` object `cvdg` that corresponds to the model specified in `model_name`.

**Example** Get a cvdata object from the specified Simulink model:

```
get(cvdg, 'slvndemo_cv_small_controller');
```

## cv.cvtestgroup.get

---

|                    |                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get cvtest objects                                                                                                                                                                            |
| <b>Syntax</b>      | <code>get(cvtg, model_name)</code>                                                                                                                                                            |
| <b>Description</b> | <code>get(cvtg, model_name)</code> returns the cvtest object in the <code>cv.cvtestgroup</code> object <code>cvtg</code> that corresponds to the model specified in <code>model_name</code> . |
| <b>Example</b>     | Get a cvtest object from the specified Simulink model:<br><pre>get(cvtg, 'slvndemo_cv_small_controller');</pre>                                                                               |
| <b>See Also</b>    | <code>cvsimref</code> , <code>cvtest</code>                                                                                                                                                   |

|                    |                                                                                                                         |
|--------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get all cvdata objects                                                                                                  |
| <b>Syntax</b>      | <code>getAll(cvdo)</code>                                                                                               |
| <b>Description</b> | <code>getAll(cvdo)</code> returns all cvdata objects in the <code>cv.cvdatagroup</code> object <code>cvdo</code> .      |
| <b>Example</b>     | Return all cvdata object from the specified Simulink model:<br><pre>getAll(cvdg, 'slvndemo_cv_small_controller');</pre> |

# getCoverageInfo

---

**Purpose** Coverage information for Simulink Design Verifier blocks

**Syntax**

```
[coverage, description] = getCoverageInfo(cvdo, object)
[coverage, description] = getCoverageInfo(cvdo, object,
    metric)
[coverage, description] = getCoverageInfo(cvdo, object,
    metric, ignore_descendants)
```

**Description**

[coverage, description] = getCoverageInfo(cvdo, object) collects Simulink Design Verifier coverage for *object*, based on coverage results in *cvdo*. *object* can be a handle to any block, subsystem, or Stateflow chart. `getCoverageData` returns coverage data only for Simulink Design Verifier library blocks in *object*'s hierarchy.

[coverage, description] = getCoverageInfo(cvdo, object, *metric*) returns coverage data for the block type specified in *metric*. If *object* does not match the block type, `getCoverageInfo` does not return any data.

[coverage, description] = getCoverageInfo(cvdo, object, *metric*, *ignore\_descendants*) returns coverage data about *object*, omitting coverage data for its descendant objects if *ignore\_descendants* equals 1.

## Input Arguments

*cvdo*

cvdata object

*object*

In the model or Stateflow chart, object that received Simulink Design Verifier coverage. The following are valid values for *object*.

|             |                                 |
|-------------|---------------------------------|
| BlockPath   | Full path to a model or block   |
| BlockHandle | Handle to a model or block      |
| s1obj       | Handle to a Simulink API object |

|                                  |                                                                                                |
|----------------------------------|------------------------------------------------------------------------------------------------|
| <code>sfID</code>                | Stateflow ID from a singly instantiated Stateflow chart                                        |
| <code>sfObj</code>               | Handle to a Stateflow API object from a singly instantiated Stateflow chart                    |
| <code>{BlockPath, sfID}</code>   | Cell array with the path to a Stateflow chart and the ID of an object in that chart            |
| <code>{BlockPath, sfObj}</code>  | Cell array with the path to a Stateflow chart and a handle to a Stateflow object in that chart |
| <code>[BlockHandle, sfID]</code> | Array with a Stateflow chart handle and the ID of an object in that chart                      |

## *metric*

`cvmetric.Sldv` enumeration object with values that correspond to Simulink Design Verifier library blocks.

|                         |                        |
|-------------------------|------------------------|
| <code>test</code>       | Test Objective block   |
| <code>proof</code>      | Proof Objective block  |
| <code>condition</code>  | Test Condition block   |
| <code>assumption</code> | Proof Assumption block |

## *ignore\_descendants*

Boolean value that specifies to ignore the coverage of descendant objects if set to 1.

## Output Arguments

### *coverage*

Two-element vector of the form `[covered_outcomes total_outcomes]`.

# getCoverageInfo

---

|                         |                                                       |
|-------------------------|-------------------------------------------------------|
| <i>covered_outcomes</i> | Number of test objectives satisfied for <i>object</i> |
| <i>total_outcomes</i>   | Total number of test objectives for <i>object</i>     |

*coverage* is empty if *cvdo* does not contain decision coverage results for *object*.

## *description*

Structure array containing descriptions of each test objective, and descriptions and execution counts for each outcome within *object*.

## Examples

Collect and display coverage data for the Test Objective block named True in the `sldvdemo_debounce_testobjblks` model:

```
mdl = 'sldvdemo_debounce_testobjblks';
open_system(mdl)
testObj = cvtest(mdl)
testObj.settings.designverifier = 1;
data = cvsim(testObj)
blk_handle = get_param([mdl, '/True'], 'Handle');
getCoverageInfo(data, blk_handle)
```

## Alternatives

To collect and display coverage results for Simulink Design Verifier library blocks using the Coverage Settings dialog box:

- 1 Open the model.
- 2 In the Model Editor, select **Tools > Coverage Settings**.
- 3 On the **Coverage** tab, under **Coverage Metrics**, select **Simulink Design Verifier**.
- 4 Click **OK**.
- 5 Simulate the model and review the results.

**See Also**

[conditioninfo](#) | [cvsim](#) | [decisioninfo](#) | [mcdcinfo](#) | [sigrangeinfo](#) | [tableinfo](#)

**How To**

- “Simulink Design Verifier Coverage” on page 15-7

# ModelAdvisor.Table.getEntry

---

**Purpose** Get table cell contents

**Syntax** `content = getEntry(table, row, column)`

**Description** `content = getEntry(table, row, column)` gets the contents of the specified cell.

**Input Arguments**

|                     |                                                            |
|---------------------|------------------------------------------------------------|
| <code>table</code>  | Instantiation of the <code>ModelAdvisor.Table</code> class |
| <code>row</code>    | An integer specifying the row                              |
| <code>column</code> | An integer specifying the column                           |

**Output Arguments**

|                      |                                                                             |
|----------------------|-----------------------------------------------------------------------------|
| <code>content</code> | An element object or object array specifying the content of the table entry |
|----------------------|-----------------------------------------------------------------------------|

**Example** Get the content of the table cell in the third column, third row:

```
table1 = ModelAdvisor.Table(4, 4);  
.  
.  
.  
content = getEntry(table1, 3, 3);
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks



**Purpose**

Return check identifier

**Syntax**

```
id = getID(check_obj)
```

**Description**

`id = getID(check_obj)` returns the ID of the check `check_obj`. `id` is a unique string that identifies the check.

You create this unique identifier when you create the check. This unique identifier is the equivalent of the `ModelAdvisor.Check ID` property.

**See Also**

“Defining Custom Checks” on page 20-11 — Describes how to create custom actions

Customizing the Model Advisor on page 1 — Describes how to create custom checks

# mcdcinfo

---

## Purpose

Collect modified condition/decision coverage information for model object

## Syntax

```
coverage = mcdcinfo(cvdo, object)
coverage = mcdcinfo(cvdo, object, ignore_descendants)
[coverage, description] = mcdcinfo(cvdo, object)
```

## Description

`coverage = mcdcinfo(cvdo, object)` returns modified condition/decision coverage (MC/DC) results from the `cvdata` object `cvdo` for the model component specified by `object`.

`coverage = mcdcinfo(cvdo, object, ignore_descendants)` returns MC/DC results for `object`, depending on the value of `ignore_descendants`.

`[coverage, description] = mcdcinfo(cvdo, object)` returns MC/DC results and text descriptions of each condition/decision in `object`.

## Input Arguments

`cvdo`

cvdata object

`ignore_descendants`

Logical value specifying whether to ignore the coverage of descendant objects

1 — Ignore coverage of descendant objects

0 — Collect coverage for descendant objects

`object`

The `object` argument specifies an object in the Simulink model or Stateflow diagram that receives decision coverage. Valid values for `object` include the following:

| Object Specification | Description                                                                                             |
|----------------------|---------------------------------------------------------------------------------------------------------|
| BlockPath            | Full path to a model or block                                                                           |
| BlockHandle          | Handle to a model or block                                                                              |
| s1Obj                | Handle to a Simulink API object                                                                         |
| sfID                 | Stateflow ID                                                                                            |
| sfObj                | Handle to a Stateflow API object                                                                        |
| {BlockPath, sfID}    | Cell array with the path to a Stateflow chart and the ID of an object contained in that chart           |
| {BlockPath, sfObj}   | Cell array with the path to a Stateflow chart and a Stateflow object API handle contained in that chart |
| [BlockHandle, sfID]  | Array with a Stateflow chart handle and the ID of an object contained in that chart                     |

## Output Arguments

coverage

Two-element vector of the form [covered\_outcomes total\_outcomes]. `coverage` is empty if `cvdo` does not contain modified condition/decision coverage results for `object`. The two elements are:

|                  |                                                                         |
|------------------|-------------------------------------------------------------------------|
| covered_outcomes | Number of condition/decision outcomes satisfied for <code>object</code> |
| total_outcomes   | Total number of condition/decision outcomes for <code>object</code>     |

description

A structure array containing the following fields:

|                                  |                                                                                                          |
|----------------------------------|----------------------------------------------------------------------------------------------------------|
| <code>text</code>                | String denoting whether the condition/decision is associated with a block output or Stateflow transition |
| <code>condition.text</code>      | String describing a condition/decision or the block port to which it applies                             |
| <code>condition.achieved</code>  | Logical array indicating whether a condition case has been fully covered                                 |
| <code>condition.trueRslt</code>  | String representing a condition case expression that produces a true result                              |
| <code>condition.falseRslt</code> | String representing a condition case expression that produces a false result                             |

## Examples

Collect MC/DC coverage for the `slvndemo_cv_small_controller` model and determine the percentage of MC/DC coverage collected for the Logic block in the Gain subsystem:

```
mdl = 'slvndemo_cv_small_controller';
open_system(mdl)
testObj = cvtest(mdl)           %Create test specification object
testObj.settings.mcdc = 1;      %Enable MC/DC coverage
data = cvsim(testObj)          %Simulate model
blk_handle = get_param([mdl, '/Gain/Logic'], 'Handle');
cov = mcdcinfo(data, blk_handle) %Retrieve MC/DC results for Logic block
percent_cov = 100 * cov(1) / cov(2) %Percentage of MC/DC outcomes covered
```

## Alternatives

To collect MC/DC coverage for a model:

- 1 Open the model.
- 2 In the Model Editor, select **Tools > Coverage Settings**.

**3** On the **Coverage** tab, under **Coverage Metrics**, select **MCDC Coverage**.

**4** On the **Results** and **Report** tabs, select the desired options.

**5** Click **OK** to close the Coverage Settings dialog box.

**6** Simulate the model and review the MC/DC coverage in the report.

## **See Also**

[conditioninfo](#) | [cvsim](#) | [decisioninfo](#) | [sigrangeinfo](#) | [tableinfo](#)

## **How To**

- “Modified Condition/Decision Coverage (MCDC)” on page 15-4
- “MCDC Analysis” on page 17-17

# ModelAdvisor.Action class

---

|                       |                                                                                                                                                                                                       |                                  |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------|
| <b>Purpose</b>        | Add actions to custom checks                                                                                                                                                                          |                                  |
| <b>Description</b>    | Instances of this class define actions you take when the Model Advisor checks do not pass. Users access actions by clicking the <b>Action</b> button that you define in the Model Advisor window.     |                                  |
| <b>Construction</b>   | ModelAdvisor.Action                                                                                                                                                                                   | Add actions to custom checks     |
| <b>Methods</b>        | setCallbackFcn                                                                                                                                                                                        | Specify action callback function |
| <b>Properties</b>     | Description                                                                                                                                                                                           | Message in <b>Action</b> box     |
|                       | Name                                                                                                                                                                                                  | Action button label              |
| <b>Copy Semantics</b> | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.                                                                    |                                  |
| <b>Example</b>        | <pre>% define action (fix) operation myAction = ModelAdvisor.Action; myAction.Name='Fix block fonts'; myAction.Description=...     'Click the button to update all blocks with specified font';</pre> |                                  |
| <b>See Also</b>       | Customizing the Model Advisor on page 1 — Describes how to create custom checks                                                                                                                       |                                  |

**Purpose**

Add actions to custom checks

**Syntax**

```
action_obj = ModelAdvisor.Action
```

**Description**

`action_obj = ModelAdvisor.Action` creates a handle to an action object.

---

**Note**

- Include an action definition in a check definition.
  - Each check can contain only one action.
- 

**Example**

```
% define action (fix) operation  
myAction = ModelAdvisor.Action;
```

**See Also**

Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Check class

---

**Purpose** Create custom checks

**Description** The `ModelAdvisor.Check` class creates a Model Advisor check object. All checks must have an associated `ModelAdvisor.Task` object to be displayed in the Model Advisor tree.

You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution** is displayed in the **By Product > Simulink** folder and in the **By Task > Model Referencing** folder in the Model Advisor tree.

When you use checks in task definitions, the following rules apply:

- If you define the properties of the check in the check definition and the task definition, the task definition takes precedence. The Model Advisor displays the information contained in the task definition. For example, if you define the name of the check in the task definition using the `ModelAdvisor.Task.DisplayName` property and in the check definition using the `ModelAdvisor.Check.Title` property, the Model Advisor displays the information provided in `ModelAdvisor.Task.DisplayName`.
- If you define the properties of the check in the check definition but not the task definition, the task uses the properties from the check. For example, if you define the name of the check in the check definition using the `ModelAdvisor.Check.Title` property, and you register the check using a task definition, the Model Advisor displays the information provided in `ModelAdvisor.Check.Title`.
- If you define the properties of the check in the task definition but not the check definition, the Model Advisor displays the information correctly as long as you register the task with the Model Advisor instead of the check. For example, if you define the name of the check in the task definition using the `ModelAdvisor.Task.DisplayName` property instead of the `ModelAdvisor.Check.Title` property, and you register the check



using a task definition, the Model Advisor displays the information provided in `ModelAdvisor.Task.DisplayName`.

## Construction

|                                 |                      |
|---------------------------------|----------------------|
| <code>ModelAdvisor.Check</code> | Create custom checks |
|---------------------------------|----------------------|

## Methods

|                                           |                                          |
|-------------------------------------------|------------------------------------------|
| <code>getID</code>                        | Return check identifier                  |
| <code>setAction</code>                    | Specify action for check                 |
| <code>setCallbackFcn</code>               | Specify callback function for check      |
| <code>setInputParameters</code>           | Specify input parameters for check       |
| <code>setInputParametersLayoutGrid</code> | Specify layout grid for input parameters |

## Properties

|                              |                                                         |
|------------------------------|---------------------------------------------------------|
| <code>CallbackContext</code> | Model or subsystem context                              |
| <code>CallbackHandle</code>  | Callback function handle for check                      |
| <code>CallbackStyle</code>   | Callback function type                                  |
| <code>Enable</code>          | Indicate whether user can enable or disable check       |
| <code>ID</code>              | Identifier for check                                    |
| <code>LicenseName</code>     | Product license names required to display and run check |
| <code>ListViewVisible</code> | Status of button                                        |
| <code>Result</code>          | Results cell array                                      |
| <code>Title</code>           | Name of check                                           |

# ModelAdvisor.Check class

---

|           |                                   |
|-----------|-----------------------------------|
| TitleTips | Description of check              |
| Value     | Status of check                   |
| Visible   | Indicate to display or hide check |

## Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

## Examples

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
```

## See Also

Customizing the Model Advisor on page 1 — Describes how to create custom checks

**Purpose** Create custom checks

**Syntax** `check_obj = ModelAdvisor.Check(check_ID)`

**Description** `check_obj = ModelAdvisor.Check(check_ID)` creates a check object, `check_obj`, and assigns it a unique identifier, `check_ID`. `check_ID` must remain constant. To display checks in the Model Advisor tree, all checks must have an associated `ModelAdvisor.Task` or `ModelAdvisor.Root` object.

---

**Note** You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution appears** in the **By Product > Simulink folder** and in the **By Task > Model Referencing** folder in the Model Advisor tree.

---

**Example** `rec = ModelAdvisor.Check('com.mathworks.sample.Check1');`

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.FactoryGroup class

---

|                       |                                                                                                                                    |                                           |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| <b>Purpose</b>        | Define subfolder in <b>By Task</b> folder                                                                                          |                                           |
| <b>Description</b>    | The ModelAdvisor.FactoryGroup class defines a new subfolder to add to the <b>By Task</b> folder.                                   |                                           |
| <b>Construction</b>   | ModelAdvisor.FactoryGroup                                                                                                          | Define subfolder in <b>By Task</b> folder |
| <b>Methods</b>        | addCheck                                                                                                                           | Add check to folder                       |
| <b>Properties</b>     | Description                                                                                                                        | Description of folder                     |
|                       | DisplayName                                                                                                                        | Name of folder                            |
|                       | ID                                                                                                                                 | Identifier for folder                     |
|                       | MAObj                                                                                                                              | Model Advisor object                      |
| <b>Copy Semantics</b> | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation. |                                           |
| <b>Example</b>        | <pre>% --- sample factory group rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');</pre>                        |                                           |
| <b>See Also</b>       | Customizing the Model Advisor on page 1 — Describes how to create custom checks                                                    |                                           |

**Purpose** Define subfolder in **By Task** folder

**Syntax** `fg_obj = ModelAdvisor.FactoryGroup(fg_ID)`

**Description** `fg_obj = ModelAdvisor.FactoryGroup(fg_ID)` creates a handle to a factory group object, `fg_obj`, and assigns it a unique identifier, `fg_ID`. `fg_ID` must remain constant.

**Example**

```
% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.FormatTemplate class

---

**Purpose**            Template for formatting Model Advisor analysis results

**Description**        Use the `ModelAdvisor.FormatTemplate` class to format the result of a check in the analysis result pane of the Model Advisor for a uniform look and feel among the checks you create. There are two formats for the analysis result:

- Table
- List

**Construction**        `ModelAdvisor.FormatTemplate`        Construct template object for formatting Model Advisor analysis results

|                |                                     |                                         |
|----------------|-------------------------------------|-----------------------------------------|
| <b>Methods</b> | <code>addRow</code>                 | Add row to table                        |
|                | <code>setCheckText</code>           | Add description of check to result      |
|                | <code>setColTitles</code>           | Add column titles to table              |
|                | <code>setInformation</code>         | Add description of subcheck to result   |
|                | <code>setListObj</code>             | Add list of hyperlinks to model objects |
|                | <code>setRecAction</code>           | Add Recommended Action section and text |
|                | <code>setRefLink</code>             | Add See Also section and links          |
|                | <code>setSubBar</code>              | Add line between subcheck results       |
|                | <code>setSubResultStatus</code>     | Add status to check or subcheck result  |
|                | <code>setSubResultStatusText</code> | Add text below status in result         |

# ModelAdvisor.FormatTemplate class

---

|               |                                  |
|---------------|----------------------------------|
| setSubTitle   | Add title for subcheck in result |
| setTableInfo  | Add data to table                |
| setTableTitle | Add title to table               |

## Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

## Examples

The following code creates two template objects, `ft1` and `ft2`, and uses them to format the result of running the check in a table and a list. The result identifies the blocks in the model. The graphics following the code display the output as it appears in the Model Advisor when the check passes and fails.

```
% Sample Check With Subchecks Callback Function
function ResultDescription = SampleStyleOneCallback(system)
mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system); % get object

%Initialize variables
ResultDescription={};
ResultStatus = false; % Default check status is 'Warning'
mdladvObj.setCheckResultStatus(ResultStatus);

% Create FormatTemplate object for first subcheck, specify table format
ft1 = ModelAdvisor.FormatTemplate('TableTemplate');

% Add information describing the overall check
setCheckText(ft1, ['Find and report all blocks in the model. '...
    '(setCheckText method - Description of what the check reviews)']);

% Add information describing the subcheck
setSubTitle(ft1, 'Table of Blocks (setSubTitle method - Title of the subcheck)');
setInformation(ft1, ['Find and report all blocks in a table. '...
    '(setInformation method - Description of what the subcheck reviews)']);

% Add See Also section for references to standards
```

# ModelAdvisor.FormatTemplate class

---

```
setRefLink(ft1, {'Standard 1 reference (setRefLink method)',
               {'Standard 2 reference (setRefLink method)'});

% Add information to the table
setTableTitle(ft1, {'Blocks in the Model (setTableTitle method)'});
setColTitles(ft1, {'Index (setColTitles method)',
                  'Block Name (setColTitles method)'});

% Perform the check actions
allBlocks = find_system(system);
if length(find_system(system)) == 1
    % Add status for subcheck
    setSubResultStatus(ft1, 'Warn');
    setSubResultStatusText(ft1, ['The model does not contain blocks. '...
                               '(setSubResultStatusText method - Description of result status)']);
    setRecAction(ft1, {'Add blocks to the model. '...
                     '(setRecAction method - Description of how to fix the problem)'});
    ResultStatus = false;
else
    % Add status for subcheck
    setSubResultStatus(ft1, 'Pass');
    setSubResultStatusText(ft1, ['The model contains blocks. '...
                               '(setSubResultStatusText method - Description of result status)']);
    for inx = 2 : length(allBlocks)
        % Add information to the table
        addRow(ft1, {inx-1,allBlocks(inx)});
    end
    ResultStatus = true;
end

% Pass table template object for subcheck to Model Advisor
ResultDescription{end+1} = ft1;

% Create FormatTemplate object for second subcheck, specify list format
ft2 = ModelAdvisor.FormatTemplate('ListTemplate');

% Add information describing the subcheck
```



# ModelAdvisor.FormatTemplate class

---

```
setSubTitle(ft2, 'List of Blocks (setSubTitle method - Title of the subcheck)');
setInformation(ft2, ['Find and report all blocks in a list. '...
    '(setInformation method - Description of what the subcheck reviews)']);

% Add See Also section for references to standards
setRefLink(ft2, {'Standard 1 reference (setRefLink method)'},
    {'Standard 2 reference (setRefLink method)'});


% Last subcheck, suppress line
setSubBar(ft2, false);

% Perform the subcheck actions
if length(find_system(system)) == 1
    % Add status for subcheck
    setSubResultStatus(ft2, 'Warn');
    setSubResultStatusText(ft2, ['The model does not contain blocks. '...
        '(setSubResultStatusText method - Description of result status)']);
    setRecAction(ft2, {'Add blocks to the model. '...
        '(setRecAction method - Description of how to fix the problem)'});
    ResultStatus = false;
else
    % Add status for subcheck
    setSubResultStatus(ft2, 'Pass');
    setSubResultStatusText(ft2, ['The model contains blocks. '...
        '(setSubResultStatusText method - Description of result status)']);
    % Add information to the list
    setListObj(ft2, allBlocks);
end

% Pass list template object for the subcheck to Model Advisor
ResultDescription{end+1} = ft2;
% Set overall check status
mdladvObj.setCheckResultStatus(ResultStatus);
```

# ModelAdvisor.FormatTemplate class

The following graphic displays the output as it appears in the Model Advisor when the check passes.

Result:  Passed

Find and report all blocks in the model. (setCheckText method - Description of what the check reviews)

**Table of Blocks (setSubTitle method - Title of the subcheck)**  
Find and report all blocks in a table. (setInformation method - Description of what the subcheck reviews)

**See Also**

- Standard 1 reference (setRefLink method)
- Standard 2 reference (setRefLink method)

**Passed**  
The model contains blocks. (setSubResultStatusText method - Description of result status)

Blocks in the Model (setTableTitle method)

| Index (setColTitles method) | Block Name (setColTitles method)               |
|-----------------------------|------------------------------------------------|
| 1                           | <a href="#">format template test/Constant</a>  |
| 2                           | <a href="#">format template test/Constant1</a> |
| 3                           | <a href="#">format template test/Gain</a>      |
| 4                           | <a href="#">format template test/Product</a>   |
| 5                           | <a href="#">format template test/Out1</a>      |

---

**List of Blocks (setSubTitle method - Title of the subcheck)**  
Find and report all blocks in a list. (setInformation method - Description of what the subcheck reviews)

**See Also**


- Standard 1 reference (setRefLink method)
- Standard 2 reference (setRefLink method)

**Passed**  
The model contains blocks. (setSubResultStatusText method - Description of result status)

- [format template test](#)
- [format template test/Constant](#)
- [format template test/Constant1](#)
- [format template test/Gain](#)
- [format template test/Product](#)
- [format template test/Out1](#)

# ModelAdvisor.FormatTemplate class

The following graphic displays the output as it appears in the Model Advisor when the check fails.

Result:  Warning

Find and report all blocks in the model. (setCheckText method - Description of what the check reviews)

**Table of Blocks (setSubTitle method - Title of the subcheck)**  
Find and report all blocks in a table. (setInformation method - Description of what the subcheck reviews)

**See Also**

- Standard 1 reference (setRefLink method)
- Standard 2 reference (setRefLink method)

**Warning**  
The model does not contain blocks. (setSubResultStatusText method - Description of result status)

**Recommended Action**  
Add blocks to the model.  
(setRecAction method - Description of how to fix the problem)

---

**List of Blocks (setSubTitle method - Title of the subcheck)**  
Find and report all blocks in a list. (setInformation method - Description of what the subcheck reviews)

**See Also**

- Standard 1 reference (setRefLink method)
- Standard 2 reference (setRefLink method)

**Warning**  
The model does not contain blocks. (setSubResultStatusText method - Description of result status)

**Recommended Action**  
Add blocks to the model.  
(setRecAction method - Description of how to fix the problem)

## Alternatives

Use the Model Advisor Formatting API to format check analysis results, however The MathWorks recommends that you use the

# ModelAdvisor.FormatTemplate class

---

ModelAdvisor.FormatTemplate class for a uniform look and feel among the checks you create.

## See Also

Customizing the Model Advisor on page 1 — Describes how to create custom checks

“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

**Purpose** Construct template object for formatting Model Advisor analysis results

**Syntax** `obj = ModelAdvisor.FormatTemplate('type')`

**Description** `obj = ModelAdvisor.FormatTemplate('type')` creates a handle, *obj*, to an object of the `ModelAdvisor.FormatTemplate` class. *type* is a string identifying the format type of the template, either list or table. Valid values are `ListTemplate` and `TableTemplate`.

You must return the result object to the Model Advisor to display the formatted result in the analysis result pane.

---

**Note** Use the `ModelAdvisor.FormatTemplate` class in check callbacks.

---

**Examples** Create a template object, `ft`, and use it to create a list template:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

# ModelAdvisor.Group class

---

|                       |                                                                                                                                                                                                                                                                                       |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Define custom folder                                                                                                                                                                                                                                                                  |
| <b>Description</b>    | The ModelAdvisor.Group class defines a folder that is displayed in the Model Advisor tree. Use folders to consolidate checks by functionality or usage.                                                                                                                               |
| <b>Construction</b>   | ModelAdvisor.Group                      Define custom folder                                                                                                                                                                                                                          |
| <b>Methods</b>        | addGroup                                      Add subfolder to folder<br>addTask                                        Add task to folder                                                                                                                                            |
| <b>Properties</b>     | Description                                    Description of folder<br>DisplayName                                  Name of folder<br>ID                                                Identifier for folder<br>MAObj                                          Model Advisor object |
| <b>Copy Semantics</b> | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.                                                                                                                                                    |
| <b>See Also</b>       | Customizing the Model Advisor on page 1 — Describes how to create custom checks                                                                                                                                                                                                       |

**Purpose**

Define custom folder

**Syntax**

```
group_obj = ModelAdvisor.Group(group_ID)
```

**Description**

`group_obj = ModelAdvisor.Group(group_ID)` creates a handle to a group object, `group_obj`, and assigns it a unique identifier, `group_ID`. `group_ID` must remain constant.

**Examples**

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');
```

**See Also**

Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Image class

---

|                       |                                                                                                                                                                                     |                                       |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|
| <b>Purpose</b>        | Include image in Model Advisor output                                                                                                                                               |                                       |
| <b>Description</b>    | The <code>ModelAdvisor.Image</code> class adds an image to the Model Advisor output.                                                                                                |                                       |
| <b>Construction</b>   | <code>ModelAdvisor.Image</code>                                                                                                                                                     | Include image in Model Advisor output |
| <b>Methods</b>        | <code>setHyperlink</code>                                                                                                                                                           | Specify hyperlink location            |
|                       | <code>setImageSource</code>                                                                                                                                                         | Specify image location                |
| <b>Copy Semantics</b> | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.                                                  |                                       |
| <b>See Also</b>       | Customizing the Model Advisor on page 1 — Describes how to create custom checks<br>“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results |                                       |



**Purpose** Include image in Model Advisor output

**Syntax** `object = ModelAdvisor.Image`

**Description** `object = ModelAdvisor.Image` creates a handle to an image object, object, that the Model Advisor displays in the output. The Model Advisor supports many image formats, including, but not limited to, JPEG, BMP, and GIF.

**Examples** `image_obj = ModelAdvisor.Image;`

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

# ModelAdvisor.InputParameter class

---

|                       |                                                                                                                                                                                               |                                               |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| <b>Purpose</b>        | Add input parameters to custom checks                                                                                                                                                         |                                               |
| <b>Description</b>    | Instances of the <code>ModelAdvisor.InputParameter</code> class specify the input parameters a custom check uses in analyzing the model. Access input parameters in the Model Advisor window. |                                               |
| <b>Construction</b>   | <code>ModelAdvisor.InputParameter</code>                                                                                                                                                      | Add input parameters to custom checks         |
| <b>Methods</b>        | <code>setColSpan</code>                                                                                                                                                                       | Specify number of columns for input parameter |
|                       | <code>setRowSpan</code>                                                                                                                                                                       | Specify rows for input parameter              |
| <b>Properties</b>     | Description                                                                                                                                                                                   | Description of input parameter                |
|                       | Entries                                                                                                                                                                                       | Drop-down list entries                        |
|                       | Name                                                                                                                                                                                          | Input parameter name                          |
|                       | Type                                                                                                                                                                                          | Input parameter type                          |
|                       | Value                                                                                                                                                                                         | Value of input parameter                      |
| <b>Copy Semantics</b> | Handle. To learn how this affects your use of the class, see <a href="#">Copying Objects in the MATLAB Programming Fundamentals documentation</a> .                                           |                                               |
| <b>See Also</b>       | <a href="#">Customizing the Model Advisor on page 1</a> — Describes how to create custom checks                                                                                               |                                               |

**Purpose** Add input parameters to custom checks

**Syntax** `input_param = ModelAdvisor.InputParameter`

**Description** `input_param = ModelAdvisor.InputParameter` creates a handle to an input parameter object, `input_param`.

---

**Note** You must include input parameter definitions in a check definition.

---

## Example

---

**Note** The following example is a fragment of code from the `sl_customization.m` file for the demo model, `slvndemo_mdadv`. The example does not execute as shown without the additional content found in the `sl_customization.m` file.

---

# ModelAdvisor.InputParameter

---

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.setInputParametersLayoutGrid([3 2]);
% define input parameters
inputParam1 = ModelAdvisor.InputParameter;
inputParam1.Name = 'Skip font checks.';
inputParam1.Type = 'Bool';
inputParam1.Value = false;
inputParam1.Description = 'sample tooltip';
inputParam1.setRowSpan([1 1]);
inputParam1.setColSpan([1 1]);
inputParam2 = ModelAdvisor.InputParameter;
inputParam2.Name = 'Standard font size';
inputParam2.Value='12';
inputParam2.Type='String';
inputParam2.Description='sample tooltip';
inputParam2.setRowSpan([2 2]);
inputParam2.setColSpan([1 1]);
inputParam3 = ModelAdvisor.InputParameter;
inputParam3.Name='Valid font';
inputParam3.Type='Combobox';
inputParam3.Description='sample tooltip';
inputParam3.Entries={'Arial', 'Arial Black'};
inputParam3.setRowSpan([2 2]);
inputParam3.setColSpan([2 2]);
rec.setInputParameters({inputParam1,inputParam2,inputParam3});
```

## See Also

Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.LineBreak class

---

|                       |                                                                                                                                                                                     |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Insert line break                                                                                                                                                                   |
| <b>Description</b>    | Use instances of the <code>ModelAdvisor.LineBreak</code> class to insert line breaks in the Model Advisor outputs.                                                                  |
| <b>Construction</b>   | <code>ModelAdvisor.LineBreak</code> Insert line break                                                                                                                               |
| <b>Copy Semantics</b> | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.                                                  |
| <b>See Also</b>       | Customizing the Model Advisor on page 1 — Describes how to create custom checks<br>“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results |

# ModelAdvisor.LineBreak

---

**Purpose** Insert line break

**Syntax** `ModelAdvisor.LineBreak`

**Description** `ModelAdvisor.LineBreak` inserts a line break into the Model Advisor output.

**Example** Add a line break between two lines of text:

```
result = ModelAdvisor.Paragraph;  
addItem(result, [resultText1 ModelAdvisor.LineBreak resultText2]);
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

|                       |                                                                                                                                                                                     |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Create list class                                                                                                                                                                   |
| <b>Description</b>    | Use instances of the <code>ModelAdvisor.List</code> class to create list-formatted outputs.                                                                                         |
| <b>Construction</b>   | <code>ModelAdvisor.List</code> Create list class                                                                                                                                    |
| <b>Methods</b>        | <code>addItem</code> Add item to list<br><code>setType</code> Specify list type                                                                                                     |
| <b>Copy Semantics</b> | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.                                                  |
| <b>See Also</b>       | Customizing the Model Advisor on page 1 — Describes how to create custom checks<br>“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results |

# ModelAdvisor.List

---

**Purpose** Create list class

**Syntax** `list = ModelAdvisor.List`

**Description** `list = ModelAdvisor.List` creates a list object, `list`.

**Example**

```
subList = ModelAdvisor.List();
setType(subList, 'numbered')
addItem(subList, ModelAdvisor.Text('Sub entry 1', {'pass', 'bold'}));
addItem(subList, ModelAdvisor.Text('Sub entry 2', {'pass', 'bold'}));
```

**See Also**

Customizing the Model Advisor on page 1 — Describes how to create custom checks

“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results



# ModelAdvisor.ListViewParameter class

---

|                       |                                                                                                                                                                                                                                      |            |                                                        |      |                                          |      |                      |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|--------------------------------------------------------|------|------------------------------------------|------|----------------------|
| <b>Purpose</b>        | Add list view parameters to custom checks                                                                                                                                                                                            |            |                                                        |      |                                          |      |                      |
| <b>Description</b>    | The Model Advisor uses list view parameters to populate the Model Advisor Result Explorer. Access the information in list views by clicking <b>Explore Result</b> in the Model Advisor window.                                       |            |                                                        |      |                                          |      |                      |
| <b>Construction</b>   | ModelAdvisor.ListViewParameter Add list view parameters to custom checks                                                                                                                                                             |            |                                                        |      |                                          |      |                      |
| <b>Properties</b>     | <table><tr><td>Attributes</td><td>Attributes to display in Model Advisor Report Explorer</td></tr><tr><td>Data</td><td>Objects in Model Advisor Result Explorer</td></tr><tr><td>Name</td><td>Drop-down list entry</td></tr></table> | Attributes | Attributes to display in Model Advisor Report Explorer | Data | Objects in Model Advisor Result Explorer | Name | Drop-down list entry |
| Attributes            | Attributes to display in Model Advisor Report Explorer                                                                                                                                                                               |            |                                                        |      |                                          |      |                      |
| Data                  | Objects in Model Advisor Result Explorer                                                                                                                                                                                             |            |                                                        |      |                                          |      |                      |
| Name                  | Drop-down list entry                                                                                                                                                                                                                 |            |                                                        |      |                                          |      |                      |
| <b>Copy Semantics</b> | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.                                                                                                   |            |                                                        |      |                                          |      |                      |

## Example

---

**Note** The following example is a fragment of code from the `sl_customization.m` file for the demo model, `slvndemo_mdadv`. The example does not execute as shown without the additional content found in the `sl_customization.m` file.

---

```
mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system);
mdladvObj.setCheckResultStatus(true);

% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
myLVParam.Data = get_param(searchResult,'object');
```

# ModelAdvisor.ListViewParameter class

---

```
myLVParam.Attributes = {'FontName'}; % name is default property
mdladvObj.setListViewParameters({myLVParam});
```

## See Also

Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.ListViewParameter

---

**Purpose** Add list view parameters to custom checks

**Syntax** `lv_param = ModelAdvisor.ListViewParameter`

**Description** `lv_param = ModelAdvisor.ListViewParameter` defines a list view, `lv_param`.

---

**Note** Include list view parameter definitions in a check definition.

---

**See Also**

- “Defining Model Advisor Result Explorer Views” on page 20-18 — Describes how to create check list views
- Customizing the Model Advisor on page 1 — Describes how to create custom checks
- “Batch-Fixing Warnings or Failures” — Describes how to use list views in the Model Advisor
- “Demo and Code Example” on page 21-20 — Describes how to run a demo that shows how to customize the Model Advisor
- “getListViewParameters” — Describes how to get list view parameters of a check
- “setListViewParameters” — Describes how to set list view parameters of a check

# ModelAdvisor.Paragraph class

---

|                       |                                                                                                                                                                                     |                             |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|
| <b>Purpose</b>        | Create and format paragraph                                                                                                                                                         |                             |
| <b>Description</b>    | The ModelAdvisor.Paragraph class creates and formats a paragraph object.                                                                                                            |                             |
| <b>Construction</b>   | ModelAdvisor.Paragraph                                                                                                                                                              | Create and format paragraph |
| <b>Methods</b>        | addItem                                                                                                                                                                             | Add item to paragraph       |
|                       | setAlign                                                                                                                                                                            | Specify paragraph alignment |
| <b>Copy Semantics</b> | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.                                                  |                             |
| <b>Example</b>        | <pre>% Check Simulation optimization setting ResultDescription{end+1} = ModelAdvisor.Paragraph(['Check Simulation '... 'optimization settings:']);</pre>                            |                             |
| <b>See Also</b>       | Customizing the Model Advisor on page 1 — Describes how to create custom checks<br>“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results |                             |

**Purpose** Create and format paragraph

**Syntax** `para_obj = ModelAdvisor.Paragraph`

**Description** `para_obj = ModelAdvisor.Paragraph` defines a paragraph object `para_obj`.

**Example**

```
% Check Simulation optimization setting
ResultDescription{end+1} = ModelAdvisor.Paragraph(['Check Simulation '...
'optimization settings:']);
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Root class

---

|                       |                                                                                                                                    |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Identify root node                                                                                                                 |
| <b>Description</b>    | The <code>ModelAdvisor.Root</code> class returns the root object.                                                                  |
| <b>Construction</b>   | <code>ModelAdvisor.Root</code> Identify root node                                                                                  |
| <b>Methods</b>        | <code>publish</code> Publish object in Model Advisor root<br><code>register</code> Register object in Model Advisor root           |
| <b>Copy Semantics</b> | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation. |
| <b>See Also</b>       | Customizing the Model Advisor on page 1 — Describes how to create custom checks                                                    |

|                    |                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Identify root node                                                                                     |
| <b>Syntax</b>      | <code>root_obj = ModelAdvisor.Root</code>                                                              |
| <b>Description</b> | <code>root_obj = ModelAdvisor.Root</code> creates a handle to the root object, <code>root_obj</code> . |
| <b>Example</b>     | <code>mdladvRoot = ModelAdvisor.Root;</code>                                                           |
| <b>See Also</b>    | Customizing the Model Advisor on page 1 — Describes how to create custom checks                        |

# ModelAdvisor.Table class

---

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|-------------------------|----------------------------|----------------------------|---------------------------------|--------------------------------|--------------------------|-----------------------|-----------------------|-------------------|----------------------------|------------------------------|-------------------------|---------------------|------------------------------|-------------------------------|----------------------------|-------------------------|---------------------------------|-----------------------------------|
| <b>Purpose</b>                  | Create table                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <b>Description</b>              | Instances of the <code>ModelAdvisor.Table</code> class create and format a table. Specify the number of rows and columns in a table, excluding the table title and table heading row.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <b>Construction</b>             | <code>ModelAdvisor.Table</code> Create table                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <b>Methods</b>                  | <table><tr><td><code>getEntry</code></td><td>Get table cell contents</td></tr><tr><td><code>setColHeading</code></td><td>Specify table column title</td></tr><tr><td><code>setColHeadingAlign</code></td><td>Specify column title alignment</td></tr><tr><td><code>setColWidth</code></td><td>Specify column widths</td></tr><tr><td><code>setEntry</code></td><td>Add cell to table</td></tr><tr><td><code>setEntryAlign</code></td><td>Specify table cell alignment</td></tr><tr><td><code>setHeading</code></td><td>Specify table title</td></tr><tr><td><code>setHeadingAlign</code></td><td>Specify table title alignment</td></tr><tr><td><code>setRowHeading</code></td><td>Specify table row title</td></tr><tr><td><code>setRowHeadingAlign</code></td><td>Specify table row title alignment</td></tr></table> | <code>getEntry</code> | Get table cell contents | <code>setColHeading</code> | Specify table column title | <code>setColHeadingAlign</code> | Specify column title alignment | <code>setColWidth</code> | Specify column widths | <code>setEntry</code> | Add cell to table | <code>setEntryAlign</code> | Specify table cell alignment | <code>setHeading</code> | Specify table title | <code>setHeadingAlign</code> | Specify table title alignment | <code>setRowHeading</code> | Specify table row title | <code>setRowHeadingAlign</code> | Specify table row title alignment |
| <code>getEntry</code>           | Get table cell contents                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <code>setColHeading</code>      | Specify table column title                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <code>setColHeadingAlign</code> | Specify column title alignment                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <code>setColWidth</code>        | Specify column widths                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <code>setEntry</code>           | Add cell to table                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <code>setEntryAlign</code>      | Specify table cell alignment                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <code>setHeading</code>         | Specify table title                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <code>setHeadingAlign</code>    | Specify table title alignment                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <code>setRowHeading</code>      | Specify table row title                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <code>setRowHeadingAlign</code> | Specify table row title alignment                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <b>Copy Semantics</b>           | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |
| <b>See Also</b>                 | Customizing the Model Advisor on page 1 — Describes how to create custom checks<br>“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                       |                         |                            |                            |                                 |                                |                          |                       |                       |                   |                            |                              |                         |                     |                              |                               |                            |                         |                                 |                                   |



**Purpose** Create table

**Syntax** `table = ModelAdvisor.Table(row, column)`

**Description** `table = ModelAdvisor.Table(row, column)` creates a table object (`table`). The Model Advisor displays the table object containing the specified number of rows (`row`) and columns (`column`).

**Examples** In the following example, you create two table objects, `table1` and `table2`. The Model Advisor displays `table1` in the results as a table with 1 row and 1 column. The Model Advisor display `table2` in the results as a table with 2 rows and 3 columns.

```
table1 = ModelAdvisor.Table(1,1);  
table2 = ModelAdvisor.Table(2,3);
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Task class

---

**Purpose** Define custom tasks

**Description** The `ModelAdvisor.Task` class is a wrapper for a check so that you can access the check with the Model Advisor.

You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution** is displayed in the **By Product > Simulink** folder and in the **By Task > Model Referencing** folder in the Model Advisor tree.

When adding checks as tasks, the Model Advisor uses the task properties instead of the check properties, except for `Visible` and `LicenseName`.

**Construction** `ModelAdvisor.Task` Define custom tasks

**Methods** `setCheck` Specify check used in task

**Properties**

|                          |                                                        |
|--------------------------|--------------------------------------------------------|
| <code>Description</code> | Description of task                                    |
| <code>DisplayName</code> | Name of task                                           |
| <code>Enable</code>      | Indicate if user can enable and disable task           |
| <code>ID</code>          | Identifier for task                                    |
| <code>LicenseName</code> | Product license names required to display and run task |
| <code>MAObj</code>       | Model Advisor object                                   |
| <code>Value</code>       | Status of task                                         |
| <code>Visible</code>     | Indicate to display or hide task                       |

## Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');  
MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');
```

## See Also

Chapter 20, “Authoring Custom Checks” — Describes how to create custom checks

# ModelAdvisor.Task

---

**Purpose** Define custom tasks

**Syntax** `task_obj = ModelAdvisor.Task(task_ID)`

**Description** `task_obj = ModelAdvisor.Task(task_ID)` creates a task object, `task_obj`, with a unique identifier, `task_ID`. `task_ID` must remain constant. If you do not specify `task_ID`, the Model Advisor assigns a random `task_ID` to the task object.

You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution** appears in the **By Product > Simulink folder** and in the **By Task > Model Referencing** folder in the Model Advisor tree.

When adding checks as tasks, the Model Advisor uses the task properties instead of the check properties, except for `Visible` and `LicenseName`.

**Examples** In the following example, you create three task objects, `MAT1`, `MAT2`, and `MAT3`.

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');  
MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

|                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|-------------------|-----------------------|--------------------|---------------------------|--------------------------|------------------------|----------------|-----------------------------------|------------------------------------|---------------------------|--------------------------|-----------------------------|----------------------------|----------------------------|----------------|
| <b>Purpose</b>                    | Create Model Advisor text output                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <b>Description</b>                | Instances of <code>ModelAdvisor.Text</code> class create formatted text for the Model Advisor output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <b>Construction</b>               | <code>ModelAdvisor.Text</code> Create Model Advisor text output                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <b>Methods</b>                    | <table><tr><td><code>setBold</code></td><td>Specify bold text</td></tr><tr><td><code>setColor</code></td><td>Specify text color</td></tr><tr><td><code>setHyperlink</code></td><td>Specify hyperlinked text</td></tr><tr><td><code>setItalic</code></td><td>Italicize text</td></tr><tr><td><code>setRetainSpaceReturn</code></td><td>Retain spacing and returns in text</td></tr><tr><td><code>setSubscript</code></td><td>Specify subscripted text</td></tr><tr><td><code>setSuperscript</code></td><td>Specify superscripted text</td></tr><tr><td><code>setUnderlined</code></td><td>Underline text</td></tr></table> | <code>setBold</code> | Specify bold text | <code>setColor</code> | Specify text color | <code>setHyperlink</code> | Specify hyperlinked text | <code>setItalic</code> | Italicize text | <code>setRetainSpaceReturn</code> | Retain spacing and returns in text | <code>setSubscript</code> | Specify subscripted text | <code>setSuperscript</code> | Specify superscripted text | <code>setUnderlined</code> | Underline text |
| <code>setBold</code>              | Specify bold text                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <code>setColor</code>             | Specify text color                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <code>setHyperlink</code>         | Specify hyperlinked text                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <code>setItalic</code>            | Italicize text                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <code>setRetainSpaceReturn</code> | Retain spacing and returns in text                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <code>setSubscript</code>         | Specify subscripted text                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <code>setSuperscript</code>       | Specify superscripted text                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <code>setUnderlined</code>        | Underline text                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <b>Copy Semantics</b>             | Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <b>Examples</b>                   | <pre>t1 = ModelAdvisor.Text('This is some text');</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |
| <b>See Also</b>                   | Customizing the Model Advisor on page 1 — Describes how to create custom checks<br>“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results                                                                                                                                                                                                                                                                                                                                                                                                                                       |                      |                   |                       |                    |                           |                          |                        |                |                                   |                                    |                           |                          |                             |                            |                            |                |

# ModelAdvisor.Text

---

**Purpose** Create Model Advisor text output

**Syntax** `text = ModelAdvisor.Text(content, attribute)`

**Description** `text = ModelAdvisor.Text(content, attribute)` creates a text object for the Model Advisor output.

**Input Arguments**

|                      |                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------|
| <code>content</code> | Optional string specifying the content of the text object. If <code>content</code> is empty, empty text is output. |
|----------------------|--------------------------------------------------------------------------------------------------------------------|

|                               |                                                                                                                                                                                                    |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code><i>attribute</i></code> | Optional string specifying the formatting of the content. If no <code>attribute</code> is specified, the output text has default coloring with no formatting. Possible formatting options include: |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

- `normal` (default) — Text is default color and style.
- `bold` — Text is bold.
- `italic` — Text is italicized.
- `underlined` — Text is underlined.
- `pass` — Text is green.
- `warn` — Text is yellow.
- `fail` — Text is red.
- `keyword` — Text is blue.
- `subscript` — Text is subscripted.
- `superscript` — Text is superscripted.
- `retainspacereturn` — Text retains spacing and returns.

## Output Arguments

text                      The text object you create

## Example

```
text = ModelAdvisor.Text('Sub entry 1', {'pass', 'bold'})
```

## See Also

Customizing the Model Advisor on page 1 — Describes how to create custom checks

“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

# ModelAdvisor.Root.publish

---

**Purpose** Publish object in Model Advisor root

**Syntax** `publish(root_obj, check_obj, location)`  
`publish(root_obj, group_obj)`  
`publish(root_obj, fg_obj)`

**Description** `publish(root_obj, check_obj, location)` specifies where the Model Advisor places the check in the Model Advisor tree. `location` is either one of the subfolders in the **By Product** folder, or the name of a new subfolder to put in the **By Product** folder. Use a pipe-delimited string to indicate multiple subfolders. For example, to add a check to the **Simulink Verification and Validation > Modeling Standards** folder, use the following string: 'Simulink Verification and Validation|Modeling Standards'.

`publish(root_obj, group_obj)` specifies the `ModelAdvisor.Group` object to publish as a folder in the **Model Advisor Task Manager** folder.

`publish(root_obj, fg_obj)` specifies the `ModelAdvisor.FactoryGroup` object to publish as a subfolder in the **By Task** folder.

**Example**

```
% publish check into By Product > Demo group.  
mdladvRoot.publish(rec, 'Demo');
```

- See Also**
- “Defining Where Custom Checks Appear” on page 20-14 in the Simulink® Verification and Validation™ User’s Guide on page 1
  - “Defining Where Tasks Appear” on page 21-17 in the Simulink® Verification and Validation™ User’s Guide on page 1
  - “Defining Where Custom Folders Appear” on page 21-19 in the Simulink® Verification and Validation™ User’s Guide on page 1



**Purpose** Register object in Model Advisor root

**Syntax** register(MAobj, obj)

**Description** register(MAobj, obj) registers the object, *obj*, in the root object MAobj.

In the Model Advisor memory, the register method registers the following types of objects:

- ModelAdvisor.Check
- ModelAdvisor.Task
- ModelAdvisor.Group
- ModelAdvisor.FactoryGroup

The register method places objects in the Model Advisor memory that you use in other functions. The register method does not place objects in the Model Advisor tree.

## Example

```
mdladvRoot = ModelAdvisor.Root;

MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');
MAT1.DisplayName='Example task with input parameter and auto-fix ability';
MAT1.setCheck('com.mathworks.sample.Check1');
mdladvRoot.register(MAT1);

MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');
MAT2.DisplayName='Example task 2';
MAT2.setCheck('com.mathworks.sample.Check2');
mdladvRoot.register(MAT2);

MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');
MAT3.DisplayName='Example task 3';
MAT3.setCheck('com.mathworks.sample.Check3');
mdladvRoot.register(MAT3)
```

## Purpose

Interact programmatically with Requirements Management Interface

## Syntax

```
rmi setup
reqlinks = rmi('createempty')
reqlinks = rmi('get', object)
reqlinks = rmi('get', object, group)
rmi('report', object)
rmi('set', object, reqlinks)
rmi('set', object, reqlinks, group)
rmi('cat', object, reqlinks)
cnt = rmi('count', object)
rmi('clearall', object)
rmi('clearAll', object, 'deep')
rmi register linktypename
rmi unregister linktypename
rmi linktypelist
cmdstr = rmi('navcmd', object)
[cmdstr, titlestr] = rmi('navcmd', object)
guidstr = rmi('guidget', object)
object = rmi('guidlookup', model, guidstr)
rmi('highlightModel', object)
rmi('unhighlightModel', object)
rmi('view', object, index)
dialog = rmi('edit', object)
rmi('copyObj', object)
```

## Description

`rmi setup` configures RMI for use with your computer and installs the interface for use with the Telelogic® DOORS® software, if needed.

`reqlinks = rmi('createempty')` creates an empty instance of the requirement links data structure.

`reqlinks = rmi('get', object)` returns the requirement links data structure for `object`. `object` is the name or handle of a Simulink or Stateflow object with which requirements can be associated.

`reqlinks = rmi('get', object, group)` returns the requirement links data structure for the Signal Builder group specified by the index

---

group. In this case, `object` is the name or handle of a Signal Builder block whose signal groups are associated with requirements.

`rmi('report', object)` creates an HTML report that describes the requirements in `object`.

`rmi('set', object, reqlinks)` sets the requirement links data structure `reqlinks` to `object`.

`rmi('set', object, reqlinks, group)` sets the requirement links data structure `reqlinks` to the Signal Builder group specified by the index `group`. In this case, `object` is the name or handle of a Signal Builder block whose signal groups you want to associate with requirements.

`rmi('cat', object, reqlinks)` appends the requirement links data structure `reqlinks` to the end of the existing structure associated with `object`. If no structure exists, RMI sets `reqlinks` to `object`.

`cnt = rmi('count', object)` returns the number of requirement links associated with `object`.

`rmi('clearall', object)` removes the requirement links data structure associated with `object`, deleting its requirements.

`rmi('clearAll', object, 'deep')` deletes all requirements links in the model containing `object`.

`rmi register linktypename` registers the custom link type specified by the function `linktypename`.

`rmi unregister linktypename` removes the custom link type specified by the function `linktypename`.

`rmi linktypelist` displays a list of the currently registered link types. The list indicates whether each link type is built-in or custom, and provides the path to the function used for its registration.

`cmdstr = rmi('navcmd', object)` returns the MATLAB command string used to navigate to `object`.

`[cmdstr, titlestr] = rmi('navcmd', object)` returns the MATLAB command string `cmdstr` and the title string `titlestr` that provides descriptive text for `object`.

`guidstr = rmi('gidget', object)` returns the globally unique identifier for `object`. A globally unique identifier is created for `object` if it lacks one.

`object = rmi('guidlookup', model, guidstr)` returns the object name in `model` that has the globally unique identifier `guidstr`.

`rmi('highlightModel', object)` highlights all of the objects in the parent model of `object` that have requirement links.

`rmi('unhighlightModel', object)` removes highlighting of objects in the parent model of `object` that have requirement links.

`rmi('view', object, index)` accesses the requirement numbered `index` in the requirements document associated with `object`. `index` is an integer that represents the  $n$ th requirement linked to `object`.

`dialog = rmi('edit', object)` displays the Requirements dialog box for `object` and returns the handle of the dialog box.

`rmi('copyObj', object)` resets the globally unique identifier for `object`, preserving its requirement links.

## Input Arguments

`group`

Signal Builder group index

`guidstr`

Globally unique model identifier

`index`

Integer that represents the  $n$ th requirement linked to `object`

`object`

Name or handle of a Simulink or Stateflow object with which requirements can be associated.

## reqlinks

Requirement links are represented using a MATLAB structure array with the following fields:

`doc` String identifying requirements document

`id` String defining location in requirements document. The first character specifies the identifier type:

| <b>First Character</b> | <b>Identifier</b>                                                                  | <b>Example</b>   |
|------------------------|------------------------------------------------------------------------------------|------------------|
| ?                      | Search text, the first occurrence of which is located in requirements document     | '?Requirement 1' |
| @                      | Named item, such as bookmark in a Microsoft Word file or an anchor in an HTML file | '@my_req'        |
| #                      | Page or item number                                                                | '#21'            |
| >                      | Line number                                                                        | '>3156'          |
| \$                     | Worksheet range in a spreadsheet                                                   | '\$A2:C5'        |

`linked` Boolean value specifying whether the requirement link is accessible for report generation and highlighting:

1 (default). Highlight model object and include requirement link in reports.

0

`description` String describing the requirement  
`keywords` Optional string supplementing description  
`reqsys` String identifying the link type registration name; 'other' for built-in link types

## Output Arguments

`cmdstr`  
MATLAB command string

`cnt`  
Number of requirement links associated with object

`dialog`  
Handle for object

`guidstr`  
Globally unique model identifier

`object`  
Name or handle of a Simulink or Stateflow object with which requirements can be associated.

`reqlinks`  
Requirement links are represented using a MATLAB structure array. See “Input Arguments” on page 25-100 for details.

`titlestr`  
Descriptive text for object

## Examples

Get a requirement associated with a block in the `slvndemo_fuelsys_htmreq` model, change its description, and save the requirement back to that block:

```
slvndemo_fuelsys_htmreq;  
blk_with_req = ['slvndemo_fuelsys_htmreq/fuel rate' 10 'controller/...  
Airflow calculation'];
```

```
reqts = rmi('get', blk_with_req);
reqts.description = 'Mass airflow estimation';
rmi('set', blk_with_req, reqts);
rmi('get', blk_with_req);
```

---

Add a new requirement to the block in the previous example:

```
new_req = rmi('createempty');
new_req.doc = 'fuelsys_requirements2.htm';
new_req.description = 'A new requirement';
rmi('cat',blk_with_req, new_req);
```

---

Create an HTML requirements report for the `slvndemo_fuelsys_htmreq` model:

```
rmi('report', 'slvndemo_fuelsys_htmreq');
```

## How To

- Chapter 4, “Creating and Managing Requirements Links”
- Chapter 10, “Creating Custom Types of Requirements Documents”

# rmidocrename

---

**Purpose** Update model requirements document paths and file names

**Syntax** `rmidocrename(model_handle, old_path, new_path)`  
`rmidocrename(model_name, old_path, new_path)`

**Description** `rmidocrename(model_handle, old_path, new_path)` collectively updates the links from a Simulink model to requirements files whose names or locations have changed. `model_handle` is a handle to the model that contains links to the files that you have moved or renamed. `old_path` is a string that contains the existing full or partial file or path name. `new_path` is a string with the new full or partial file or path name.

`rmidocrename(model_name, old_path, new_path)` updates the links to requirements files associated with `model_name`. You can pass `rmidocrename` a model handle or a model file name.

When using the `rmidocrename` function, make sure to enter specific strings for the old document name fragments so that you do not inadvertently modify other links.

**Examples** For the current Simulink model, update all links to requirements files that contain the string 'project\_0220', replacing them with 'project\_0221':

```
rmidocrename(gcs, '00000220', '00000221')  
Processed 6 objects with requirements, 5 out of 13 links were modified.
```

**Alternatives** To update the requirements links one at a time, for each model object that has a link:

- 1 For each object with requirements, open the Requirements dialog box by right-clicking and selecting **Requirements > Edit/Add Links**.
- 2 Edit the **Document** field for each requirement that points to a moved or renamed document.
- 3 Click **Apply** to save the changes.



**See Also**      rmi

# rmitag

---

## Purpose

Manage user tags for requirements links

## Syntax

```
rmitag(model, 'add', tag)
rmitag(model, 'add', tag, doc_pattern)
rmitag(model, 'delete', tag)
rmitag(model, 'delete', tag, doc_pattern)
rmitag(model, 'replace', tag, new_tag)
rmitag(model, 'replace', tag, new_tag, doc_pattern)
rmitag(model, 'clear', tag)
rmitag(model, 'clear', tag, doc_pattern)
```

## Description

`rmitag(model, 'add', tag)` adds a string *tag* as a user tag for all requirement links in *model*.

`rmitag(model, 'add', tag, doc_pattern)` adds *tag* as a user tag for all links in *model*, where the full or partial document name matches the regular expression *doc\_pattern*.

`rmitag(model, 'delete', tag)` removes the user tag, *tag*, from all requirements links in *model*.

`rmitag(model, 'delete', tag, doc_pattern)` removes the user tag, *tag*, from all requirements links in *model*, where the full or partial document name matches *doc\_pattern*.

`rmitag(model, 'replace', tag, new_tag)` replaces *tag* with *new\_tag* for all requirements links in *model*.

`rmitag(model, 'replace', tag, new_tag, doc_pattern)` replaces *tag* with *new\_tag* for links in *model*, where the full or partial document name matches the regular expression *doc\_pattern*.

`rmitag(model, 'clear', tag)` deletes all requirement links that have the user tag, *tag*.

`rmitag(model, 'clear', tag, doc_pattern)` deletes all requirement links that have the user tag, *tag*, and link to the full or partial document name specified in *doc\_pattern*.

## Input Arguments

|                    |                                                                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>model</i>       | Simulink model name or handle                                                                                                                    |
| <i>tag</i>         | String                                                                                                                                           |
| <i>doc_pattern</i> | Regular expression to match in the linked requirements document name                                                                             |
| <i>new_tag</i>     | String that indicates the name of a user tag for a requirements link. Use this argument when replacing an existing user tag with a new user tag. |

## Examples

Open the `slvnvdemo_fuelsys_officereq` demo model; add the user tag `tmptag` to all objects with requirements links:

```
open_system('slvnvdemo_fuelsys_officereq');
rmitag(gcs, 'add', 'tmptag');
```

---

Remove the user tag `test` from all requirements links:

```
open_system('slvnvdemo_fuelsys_officereq');
rmitag(gcs, 'delete', 'test');
```

---

Delete all requirements links that have the user tag `design`:

```
open_system('slvnvdemo_fuelsys_officereq');
rmitag(gcs, 'clear', 'design');
```

---

Change all instances of the user tag `tmptag` to `safety requirement`, where the document filename extension is `.docx`:

# rmitag

---

```
open_system('slvnvdemo_fuelsys_officereq');  
rmitag(gcs, 'replace', 'tmptag', 'safety requirements'. '\.docx');
```

## See Also

rmi | rmidocrename

## How To

- “Filtering Requirements with User Tags” on page 5-21

**Purpose** Specify action for check

**Syntax** `setAction(check_obj, action_obj)`

**Description** `setAction(check_obj, action_obj)` returns the action object `action_obj` to use in the check `check_obj`. The `setAction` method identifies the action you want to use in a check.

**See Also** [ModelAdvisor.Action](#) — Create custom actions  
[Customizing the Model Advisor on page 1](#) — Describes how to create custom checks

# ModelAdvisor.Paragraph.setAlign

---

**Purpose** Specify paragraph alignment

**Syntax** `setAlign(paragraph, alignment)`

**Description** `setAlign(paragraph, alignment)` specifies the alignment of text. Possible values are:

- 'left' (default)
- 'right'
- 'center'

**Example**

```
report_paragraph = ModelAdvisor.Paragraph;  
setAlign(report_paragraph, 'center');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

|                        |                                                                                                                                                                                                                                                                                                                                                                                            |                   |                                                           |                   |                                                                                                                                                                                                                             |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-----------------------------------------------------------|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>         | Specify bold text                                                                                                                                                                                                                                                                                                                                                                          |                   |                                                           |                   |                                                                                                                                                                                                                             |
| <b>Syntax</b>          | <code>setBold(text, mode)</code>                                                                                                                                                                                                                                                                                                                                                           |                   |                                                           |                   |                                                                                                                                                                                                                             |
| <b>Description</b>     | <code>setBold(text, mode)</code> specifies whether text should be formatted in bold font.                                                                                                                                                                                                                                                                                                  |                   |                                                           |                   |                                                                                                                                                                                                                             |
| <b>Input Arguments</b> | <table><tr><td><code>text</code></td><td>Instantiation of the <code>ModelAdvisor.Text</code> class</td></tr><tr><td><code>mode</code></td><td>A Boolean value indicating bold formatting of text:<ul style="list-style-type: none"><li>• <code>true</code> — Format the text in bold font.</li><li>• <code>false</code> — Do not format the text in bold font.</li></ul></td></tr></table> | <code>text</code> | Instantiation of the <code>ModelAdvisor.Text</code> class | <code>mode</code> | A Boolean value indicating bold formatting of text: <ul style="list-style-type: none"><li>• <code>true</code> — Format the text in bold font.</li><li>• <code>false</code> — Do not format the text in bold font.</li></ul> |
| <code>text</code>      | Instantiation of the <code>ModelAdvisor.Text</code> class                                                                                                                                                                                                                                                                                                                                  |                   |                                                           |                   |                                                                                                                                                                                                                             |
| <code>mode</code>      | A Boolean value indicating bold formatting of text: <ul style="list-style-type: none"><li>• <code>true</code> — Format the text in bold font.</li><li>• <code>false</code> — Do not format the text in bold font.</li></ul>                                                                                                                                                                |                   |                                                           |                   |                                                                                                                                                                                                                             |
| <b>Example</b>         | <pre>t1 = ModelAdvisor.Text('This is some text'); setBold(t1, 'true');</pre>                                                                                                                                                                                                                                                                                                               |                   |                                                           |                   |                                                                                                                                                                                                                             |
| <b>See Also</b>        | Customizing the Model Advisor on page 1 — Describes how to create custom checks                                                                                                                                                                                                                                                                                                            |                   |                                                           |                   |                                                                                                                                                                                                                             |

# ModelAdvisor.Action.setCallbackFcn

---

**Purpose** Specify action callback function

**Syntax** setCallbackFcn(action\_obj, @handle)

**Description** setCallbackFcn(action\_obj, @handle) specifies the handle to the callback function, handle, to use with the action object, action\_obj.

## Example

---

**Note** The following example is a fragment of code from the `sl_customization.m` file for the demo model, `slvndemo_mdadv`. The example does not execute as shown without the additional content found in the `sl_customization.m` file.

---

```
rec = ModelAdvisor.Check('mathworks.example.optimizationSettings');
% Define an automatic fix action for this check
modifyAction = ModelAdvisor.Action;
modifyAction.setCallbackFcn(@modifyOptimizationSetting);
modifyAction.Name = 'Modify Settings';
modifyAction.Description = ['Modify model configuration optimization' ...
    ' settings that can impact safety'];
modifyAction.Enable = true;
rec.setAction(modifyAction);
```

## See Also

“Defining Check Actions” on page 20-19 — Describes how to create custom actions  
Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“setActionenable” — Set enable/disable status for check action



|                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                        |                                                            |                     |                                     |                      |                                                                                                                                                                                        |                    |                                                                                                                                                                                                                                                                                                    |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|------------------------------------------------------------|---------------------|-------------------------------------|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>         | Specify callback function for check                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                        |                                                            |                     |                                     |                      |                                                                                                                                                                                        |                    |                                                                                                                                                                                                                                                                                                    |
| <b>Syntax</b>          | <code>setCallbackFcn(check_obj, @handle, context, style)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                        |                                                            |                     |                                     |                      |                                                                                                                                                                                        |                    |                                                                                                                                                                                                                                                                                                    |
| <b>Description</b>     | <code>setCallbackFcn(check_obj, @handle, context, style)</code> specifies the callback function to use with the check, <code>check_obj</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                        |                                                            |                     |                                     |                      |                                                                                                                                                                                        |                    |                                                                                                                                                                                                                                                                                                    |
| <b>Input Arguments</b> | <table><tr><td><code>check_obj</code></td><td>Instantiation of the <code>ModelAdvisor.Check</code> class</td></tr><tr><td><code>handle</code></td><td>Handle to a check callback function</td></tr><tr><td><code>context</code></td><td>Context for checking the model or subsystem:<ul style="list-style-type: none"><li>• 'None' — No special requirements.</li><li>• 'PostCompile' — The model must be compiled.</li></ul></td></tr><tr><td><code>style</code></td><td>Type of callback function:<ul style="list-style-type: none"><li>• 'StyleOne' — Simple check callback function, for formatting results using template</li><li>• 'StyleTwo' — Detailed check callback function</li><li>• 'StyleThree' — Check callback functions with hyperlinked results</li></ul></td></tr></table> | <code>check_obj</code> | Instantiation of the <code>ModelAdvisor.Check</code> class | <code>handle</code> | Handle to a check callback function | <code>context</code> | Context for checking the model or subsystem: <ul style="list-style-type: none"><li>• 'None' — No special requirements.</li><li>• 'PostCompile' — The model must be compiled.</li></ul> | <code>style</code> | Type of callback function: <ul style="list-style-type: none"><li>• 'StyleOne' — Simple check callback function, for formatting results using template</li><li>• 'StyleTwo' — Detailed check callback function</li><li>• 'StyleThree' — Check callback functions with hyperlinked results</li></ul> |
| <code>check_obj</code> | Instantiation of the <code>ModelAdvisor.Check</code> class                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                        |                                                            |                     |                                     |                      |                                                                                                                                                                                        |                    |                                                                                                                                                                                                                                                                                                    |
| <code>handle</code>    | Handle to a check callback function                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                        |                                                            |                     |                                     |                      |                                                                                                                                                                                        |                    |                                                                                                                                                                                                                                                                                                    |
| <code>context</code>   | Context for checking the model or subsystem: <ul style="list-style-type: none"><li>• 'None' — No special requirements.</li><li>• 'PostCompile' — The model must be compiled.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                        |                                                            |                     |                                     |                      |                                                                                                                                                                                        |                    |                                                                                                                                                                                                                                                                                                    |
| <code>style</code>     | Type of callback function: <ul style="list-style-type: none"><li>• 'StyleOne' — Simple check callback function, for formatting results using template</li><li>• 'StyleTwo' — Detailed check callback function</li><li>• 'StyleThree' — Check callback functions with hyperlinked results</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                        |                                                            |                     |                                     |                      |                                                                                                                                                                                        |                    |                                                                                                                                                                                                                                                                                                    |

## Example

```
% --- sample check 1
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.Title = 'Check Simulink block font';
rec.TitleTips = 'Example style three callback';
rec.setCallbackFcn(@SampleStyleThreeCallback, 'None', 'StyleThree');
```

# ModelAdvisor.Check.setCallbackFcn

---

## **See Also**

“Creating Callback Functions and Results” on page 20-22 —  
Describes how to create check callback functions

Customizing the Model Advisor on page 1 — Describes how to create  
custom checks

**Purpose** Specify check used in task

**Syntax** `setCheck(task, check_ID)`

**Description** `setCheck(task, check_ID)` specifies the check to use in the task.

You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution appears** in the **By Product > Simulink folder** and in the **By Task > Model Referencing** folder in the Model Advisor tree.

When adding checks as tasks, the Model Advisor uses the task properties instead of the check properties, except for `Visible` and `LicenseName`.

## Input Arguments

|                       |                                                              |
|-----------------------|--------------------------------------------------------------|
| <code>task</code>     | Instantiation of the <code>ModelAdvisor.Task</code> class    |
| <code>check_ID</code> | A unique string that identifies the check to use in the task |

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
setCheck(MAT1, 'com.mathworks.sample.Check1');
```

# ModelAdvisor.FormatTemplate.setCheckText

---

**Purpose** Add description of check to result

**Syntax** `setCheckText(ft_obj, text)`

**Description** `setCheckText(ft_obj, text)` is an optional method that adds *text* or a model advisor template object as the first item in the report. Use this method to add information describing the overall check.

**Input Arguments** *ft\_obj*  
A handle to a template object.

*text*

A string or a handle to a formatting object.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

*text* appears as the first line in the analysis result.

**Examples** Create a list object, *ft*, and add a line of text to the result:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setCheckText(ft, ['Identify unconnected lines, input ports,...
    'and output ports in the model']);
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

# ModelAdvisor.Table.setColHeading

---

**Purpose** Specify table column title

**Syntax** `setColHeading(table, column, heading)`

**Description** `setColHeading(table, column, heading)` specifies that the column header of column is set to heading.

**Input Arguments**

|                      |                                                                             |
|----------------------|-----------------------------------------------------------------------------|
| <code>table</code>   | Instantiation of the <code>ModelAdvisor.Table</code> class                  |
| <code>column</code>  | An integer specifying the column number                                     |
| <code>heading</code> | A string, element object, or object array specifying the table column title |

**Examples**

```
table1 = ModelAdvisor.Table(2, 3);
setColHeading(table1, 1, 'Header 1');
setColHeading(table1, 2, 'Header 2');
setColHeading(table1, 3, 'Header 3');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Table.setColHeadingAlign

---

**Purpose** Specify column title alignment

**Syntax** `setColHeadingAlign(table, column, alignment)`

**Description** `setColHeadingAlign(table, column, alignment)` specifies the alignment of the column heading.

**Input Arguments**

|                               |                                                                                                                                                                                                                                  |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>table</code>            | Instantiation of the <code>ModelAdvisor.Table</code> class                                                                                                                                                                       |
| <code>column</code>           | An integer specifying the column number                                                                                                                                                                                          |
| <code><i>alignment</i></code> | Alignment of the column heading. <i>alignment</i> can have one of the following values: <ul style="list-style-type: none"><li>• <code>left</code> (default)</li><li>• <code>right</code></li><li>• <code>center</code></li></ul> |

## Examples

```
table1 = ModelAdvisor.Table(2, 3);
setColHeading(table1, 1, 'Header 1');
setColHeadingAlign(table1, 1, 'center');
setColHeading(table1, 2, 'Header 2');
setColHeadingAlign(table1, 2, 'center');
setColHeading(table1, 3, 'Header 3');
setColHeadingAlign(table1, 3, 'center');
```

## See Also

Customizing the Model Advisor on page 1 — Describes how to create custom checks

**Purpose** Specify text color

**Syntax** `setColor(text, color)`

**Description** `setColor(text, color)` sets the text color to *color*.

**Input Arguments**

|                    |                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>text</code>  | Instantiation of the <code>ModelAdvisor.Text</code> class                                                                                                                                                                                                                                                                                                                                 |
| <code>color</code> | An enumerated string specifying the color of the text. Possible formatting options include: <ul style="list-style-type: none"><li>• <code>normal</code> (default) — Text is default color.</li><li>• <code>pass</code> — Text is green.</li><li>• <code>warn</code> — Text is yellow.</li><li>• <code>fail</code> — Text is red.</li><li>• <code>keyword</code> — Text is blue.</li></ul> |

**Example**

```
t1 = ModelAdvisor.Text('This is a warning');
setColor(t1, 'warn');
```

# ModelAdvisor.InputParameter.setColSpan

---

**Purpose** Specify number of columns for input parameter

**Syntax** `setColSpan(input_param, [start_col end_col])`

**Description** `setColSpan(input_param, [start_col end_col])` specifies the number of columns that the parameter occupies. Use the `setColSpan` method to specify where you want an input parameter located in the layout grid when there are multiple input parameters.

|                        |                          |                                                                                                       |
|------------------------|--------------------------|-------------------------------------------------------------------------------------------------------|
| <b>Input Arguments</b> | <code>input_param</code> | Instantiation of the <code>ModelAdvisor.InputParameter</code> class                                   |
|                        | <code>start_col</code>   | A positive integer representing the first column that the input parameter occupies in the layout grid |
|                        | <code>end_col</code>     | A positive integer representing the last column that the input parameter occupies in the layout grid  |

**Example**

```
inputParam2 = ModelAdvisor.InputParameter;  
inputParam2.Name = 'Standard font size';  
inputParam2.Value='12';  
inputParam2.Type='String';  
inputParam2.Description='sample tooltip';  
inputParam2.setRowSpan([2 2]);  
inputParam2.setColSpan([1 1]);
```



# ModelAdvisor.FormatTemplate.setColTitles

---

## Purpose

Add column titles to table

## Syntax

```
setColTitles(ft_obj, {col_title_1, col_title_2, ...})
```

## Description

`setColTitles(ft_obj, {col_title_1, col_title_2, ...})` is method you must use when you create a template object that is a table type. Use it to specify the titles of the columns in the table.

---

**Note** Before adding data to a table, you must specify column titles.

---

## Input Arguments

*ft\_obj*

A handle to a template object.

*col\_title\_N*

A cell of strings or handles to formatting objects, specifying the column titles.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

The order of the *col\_title\_N* inputs determines which column the title is in. If you do not add data to the table, the Model Advisor does not display the table in the result.

## Examples

Create a table object, `ft`, and specify two column titles:

```
ft = ModelAdvisor.FormatTemplate('TableTemplate');  
setColTitle(ft, {'Index', 'Block Name'});
```

## See Also

Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

# ModelAdvisor.Table.setColWidth

---

**Purpose** Specify column widths

**Syntax** `setColWidth(table, column, width)`

**Description** `setColWidth(table, column, width)` specifies the column.

The `setColWidth` method specifies the table column widths relative to the entire table width. If column widths are [1 2 3], the second column is twice the width of the first column, and the third column is three times the width of the first column. Unspecified columns have a default width of 1. For example:

```
setColWidth(1, 1);  
setColWidth(3, 2);
```

specifies [1 1 2] column widths.

## Input Arguments

|                     |                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------|
| <code>table</code>  | Instantiation of the <code>ModelAdvisor.Table</code> class                                       |
| <code>column</code> | An integer specifying column number                                                              |
| <code>width</code>  | An integer or array of integers specifying the column widths, relative to the entire table width |

## Example

```
table1 = ModelAdvisor.Table(2, 3)  
setColWidth(table1, 1, 1);  
setColWidth(table1, 3, 2);
```

## See Also

Customizing the Model Advisor on page 1 — Describes how to create custom checks

**Purpose** Add cell to table

**Syntax**  
`setEntry(table, row, column, string)`  
`setEntry(table, row, column, content)`

**Description**  
`setEntry(table, row, column, string)` adds a string to a cell in a table.  
`setEntry(table, row, column, content)` adds an object specified by content to a cell in a table.

**Input Arguments**

|                      |                                                                               |
|----------------------|-------------------------------------------------------------------------------|
| <code>table</code>   | Instantiation of the <code>ModelAdvisor.Table</code> class                    |
| <code>row</code>     | An integer specifying the row                                                 |
| <code>column</code>  | An integer specifying the column                                              |
| <code>string</code>  | A string representing the contents of the entry                               |
| <code>content</code> | An element object or object array specifying the content of the table entries |

**Example** Create two tables and insert table2 into the first cell of table1:

```
table1 = ModelAdvisor.Table(1, 1);  
table2 = ModelAdvisor.Table(2, 3);  
. . .  
setEntry(table1, 1, 1, table2);
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Table.setEntryAlign

---

**Purpose** Specify table cell alignment

**Syntax** `setEntryAlign(table, row, column, alignment)`

**Description** `setEntryAlign(table, row, column, alignment)` specifies the cell alignment of the designated cell.

**Input Arguments**

|                               |                                                                                                                                                                                                       |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>table</code>            | Instantiation of the <code>ModelAdvisor.Table</code> class                                                                                                                                            |
| <code>row</code>              | An integer specifying row number                                                                                                                                                                      |
| <code>column</code>           | An integer specifying column number                                                                                                                                                                   |
| <code><i>alignment</i></code> | A string specifying the cell alignment. Possible values are: <ul style="list-style-type: none"><li>• <code>left</code> (default)</li><li>• <code>right</code></li><li>• <code>center</code></li></ul> |

**Example**

```
table1 = ModelAdvisor.Table(2,3);
setHeading(table1, 'New Table');
.
.
.
setEntry(table1, 1, 1, 'First Entry');
setEntryAlign(table1, 1, 1, 'center');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Table.setHeading

---

**Purpose** Specify table title

**Syntax** `setHeading(table, title)`

**Description** `setHeading(table, title)` specifies the table title.

**Input Arguments**

|                    |                                                                          |
|--------------------|--------------------------------------------------------------------------|
| <code>table</code> | Instantiation of the <code>ModelAdvisor.Table</code> class               |
| <code>title</code> | A string, element object, or object array that specifies the table title |

**Example**

```
table1 = ModelAdvisor.Table(2, 3);  
setHeading(table1, 'New Table');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Table.setHeadingAlign

---

**Purpose** Specify table title alignment

**Syntax** `setHeadingAlign(table, alignment)`

**Description** `setHeadingAlign(table, alignment)` specifies the alignment for the table title.

**Input Arguments**

|                               |                                                                                                                                                                                                              |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>table</code>            | Instantiation of the <code>ModelAdvisor.Table</code> class                                                                                                                                                   |
| <code><i>alignment</i></code> | A string specifying the table title alignment. Possible values are: <ul style="list-style-type: none"><li>• <code>left</code> (default)</li><li>• <code>right</code></li><li>• <code>center</code></li></ul> |

**Example**

```
table1 = ModelAdvisor.Table(2, 3);
setHeading(table1, 'New Table');
setHeadingAlign(table1, 'center');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Image.setHyperlink

---

**Purpose** Specify hyperlink location

**Syntax** setHyperlink(image, url)

**Description** setHyperlink(image, url) specifies the target location of the hyperlink associated with image.

|                        |       |                                               |
|------------------------|-------|-----------------------------------------------|
| <b>Input Arguments</b> | image | Instantiation of the ModelAdvisor.Image class |
|                        | url   | A string specifying the target URL            |

**Example**

```
matlab_logo=ModelAdvisor.Image;  
setHyperlink(matlab_logo, 'http://www.mathworks.com');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Text.setHyperlink

---

|                        |                                                                                                                                                                                                                       |                   |                                                           |                  |                                                        |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-----------------------------------------------------------|------------------|--------------------------------------------------------|
| <b>Purpose</b>         | Specify hyperlinked text                                                                                                                                                                                              |                   |                                                           |                  |                                                        |
| <b>Syntax</b>          | <code>setHyperlink(text, url)</code>                                                                                                                                                                                  |                   |                                                           |                  |                                                        |
| <b>Description</b>     | <code>setHyperlink(text, url)</code> creates a hyperlink from the text to the specified URL.                                                                                                                          |                   |                                                           |                  |                                                        |
| <b>Input Arguments</b> | <table><tr><td><code>text</code></td><td>Instantiation of the <code>ModelAdvisor.Text</code> class</td></tr><tr><td><code>url</code></td><td>A string that specifies the target location of the URL</td></tr></table> | <code>text</code> | Instantiation of the <code>ModelAdvisor.Text</code> class | <code>url</code> | A string that specifies the target location of the URL |
| <code>text</code>      | Instantiation of the <code>ModelAdvisor.Text</code> class                                                                                                                                                             |                   |                                                           |                  |                                                        |
| <code>url</code>       | A string that specifies the target location of the URL                                                                                                                                                                |                   |                                                           |                  |                                                        |
| <b>Examples</b>        | <pre>t1 = ModelAdvisor.Text('MathWorks home page'); setHyperlink(t1, 'http://www.mathworks.com');</pre>                                                                                                               |                   |                                                           |                  |                                                        |
| <b>See Also</b>        | Customizing the Model Advisor on page 1 — Describes how to create custom checks                                                                                                                                       |                   |                                                           |                  |                                                        |



# ModelAdvisor.Image.setImageSource

---

|                        |                                                                                     |
|------------------------|-------------------------------------------------------------------------------------|
| <b>Purpose</b>         | Specify image location                                                              |
| <b>Syntax</b>          | <code>setImageSource(image_obj, source)</code>                                      |
| <b>Description</b>     | <code>setImageSource(image_obj, source)</code> specifies the location of the image. |
| <b>Input Arguments</b> | <code>image_obj</code> Instantiation of the <code>ModelAdvisor.Image</code> class   |
|                        | <code>source</code> A string specifying the location of the image file              |
| <b>See Also</b>        | Customizing the Model Advisor on page 1 — Describes how to create custom checks     |

# ModelAdvisor.FormatTemplate.setInformation

---

|                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>         | Add description of subcheck to result                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Syntax</b>          | <code>setInformation(<i>ft_obj</i>, <i>text</i>)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b>     | <code>setInformation(<i>ft_obj</i>, <i>text</i>)</code> is an optional method that adds <i>text</i> as the first item after the subcheck title. Use this method to add information describing the subcheck.                                                                                                                                                                                                                                                                                        |
| <b>Input Arguments</b> | <p><i>ft_obj</i></p> <p>A handle to a template object.</p> <p><i>text</i></p> <p>A string or a handle to a formatting object, that describes the subcheck.</p> <p>Valid formatting objects are: <code>ModelAdvisor.Image</code>, <code>ModelAdvisor.LineBreak</code>, <code>ModelAdvisor.List</code>, <code>ModelAdvisor.Paragraph</code>, <code>ModelAdvisor.Table</code>, and <code>ModelAdvisor.Text</code>.</p> <p>The Model Advisor displays <i>text</i> after the title of the subcheck.</p> |
| <b>Examples</b>        | <p>Create a list object, <i>ft</i>, and specify a subcheck title and description:</p> <pre>ft = ModelAdvisor.FormatTemplate('ListTemplate'); setSubTitle(ft, ['Check for constructs in the model '...     'that are not supported when generating code']); setInformation(ft, ['Identify blocks that should not '...     'be used for code generation.']);</pre>                                                                                                                                   |
| <b>See Also</b>        | <p>Customizing the Model Advisor on page 1 — Describes how to create custom checks</p> <p>“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results</p>                                                                                                                                                                                                                                                                                                     |

# ModelAdvisor.Check.setInputParameters

---

**Purpose** Specify input parameters for check

**Syntax** setInputParameters(check\_obj, params)

**Description** setInputParameters(check\_obj, params) specifies ModelAdvisor.InputParameter objects (params) to be used as input parameters to a check (check\_obj).

|                        |           |                                                      |
|------------------------|-----------|------------------------------------------------------|
| <b>Input Arguments</b> | check_obj | Instantiation of the ModelAdvisor.Check class        |
|                        | params    | A cell array of ModelAdvisor.InputParameters objects |

**Examples**

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
inputParam1 = ModelAdvisor.InputParameter;
inputParam2 = ModelAdvisor.InputParameter;
inputParam3 = ModelAdvisor.InputParameter;
setInputParameters(rec, {inputParam1,inputParam2,inputParam3});
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks  
ModelAdvisor.InputParameter— Add input parameters to custom checks

# ModelAdvisor.Check.setInputParametersLayoutGrid

---

**Purpose** Specify layout grid for input parameters

**Syntax** `setInputParametersLayoutGrid(check_obj, [row col])`

**Description** `setInputParametersLayoutGrid(check_obj, [row col])` specifies the layout grid for input parameters in the Model Advisor. Use the `setInputParametersLayoutGrid` method if there are multiple input parameters.

|                        |                        |                                                            |
|------------------------|------------------------|------------------------------------------------------------|
| <b>Input Arguments</b> | <code>check_obj</code> | Instantiation of the <code>ModelAdvisor.Check</code> class |
|                        | <code>row</code>       | Number of rows in the layout grid                          |
|                        | <code>col</code>       | Number of columns in the layout grid                       |

**Example**

```
% --- sample check 1
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.Title = 'Check Simulink block font';
rec.TitleTips = 'Example style three callback';
rec.setCallbackFcn(@SampleStyleThreeCallback, 'None', 'StyleThree');
rec.setInputParametersLayoutGrid([3 2]);
```

**See Also** `ModelAdvisor.InputParameter` — Add input parameters to custom checks  
`Customizing the Model Advisor on page 1` — Describes how to create custom checks

**Purpose** Italicize text

**Syntax** `setItalic(text, mode)`

**Description** `setItalic(text, mode)` specifies whether text should be italicized.

**Input Arguments**

|                   |                                                                                                                                                                                                           |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>text</code> | Instantiation of the <code>ModelAdvisor.Text</code> class                                                                                                                                                 |
| <code>mode</code> | A Boolean value indicating italic formatting of text: <ul style="list-style-type: none"><li>• <code>true</code> — Italicize the text.</li><li>• <code>false</code> — Do not italicize the text.</li></ul> |

**Example**

```
t1 = ModelAdvisor.Text('This is some text');
setItalic(t1, 'true');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.FormatTemplate.setListObj

---

**Purpose** Add list of hyperlinks to model objects

**Syntax** `setListObj(ft_obj, {model_obj})`

**Description** `setListObj(ft_obj, {model_obj})` is an optional method that generates a bulleted list of hyperlinks to model objects. *ft\_obj* is a handle to a list template object. *model\_obj* is a cell array of handles or full paths to blocks, or model objects that the Model Advisor displays as a bulleted list of hyperlinks in the report.

**Examples** Create a list object, `ft`, and add a list of the blocks found in the model:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');

% Find all the blocks in the system
allBlocks = find_system(system);

% Add the blocks to a list
setListObj(ft, allBlocks);
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

# ModelAdvisor.FormatTemplate.setRecAction

---

|                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>         | Add Recommended Action section and text                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Syntax</b>          | <code>setRecAction(<i>ft_obj</i>, {<i>text</i>})</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b>     | <code>setRecAction(<i>ft_obj</i>, {<i>text</i>})</code> is an optional method that adds a Recommended Action section to the report. Use this method to describe how to fix the check.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Input Arguments</b> | <p><i>ft_obj</i></p> <p>A handle to a template object.</p> <p><i>text</i></p> <p>A cell array of strings or handles to formatting objects, that describes the recommended action to fix the issues reported by the check.</p> <p>Valid formatting objects are: <code>ModelAdvisor.Image</code>, <code>ModelAdvisor.LineBreak</code>, <code>ModelAdvisor.List</code>, <code>ModelAdvisor.Paragraph</code>, <code>ModelAdvisor.Table</code>, and <code>ModelAdvisor.Text</code>.</p> <p>The Model Advisor displays the recommended action as a separate section below the list or table in the report.</p> |

**Examples** Create a list object, `ft`, find Gain blocks in the model, and recommend changing them:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
% Find all Gain blocks
gainBlocks = find_system(gcs, 'BlockType','Gain');

% Find Gain blocks with expression evaluates to 1
for idx = 1:length(gainBlocks)
    gainObj = get_param(gainBlocks(idx), 'Object');
    resGain = slResolve(gainObj.Gain, gainObj.getFullName);
    if ~isempty(resGain)
        % Find the first index that computes to 1
    end
end
```

# ModelAdvisor.FormatTemplate.setRecAction

---

```
        if ~isempty(find(resGain == 1, 1))
            setRecAction(ft, {'If you are using these blocks '...
                'as buffers, you should replace them with '...
                'Signal Conversion blocks'});
        end
    end
end
```

## See Also

Customizing the Model Advisor on page 1 — Describes how to create custom checks

“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results



**Purpose** Add See Also section and links

**Syntax**

```
setRefLink(ft_obj, {'standard'})  
setRefLink(ft_obj, {'url', 'standard'})
```

**Description** `setRefLink(ft_obj, {'standard'})` is an optional method that adds a See Also section above the table or list in the result. Use this method to add references to standards. `ft_obj` is a handle to a template object. `standard` is a cell array of strings that you want to display in the result. If you include more than one cell, the Model Advisor displays the strings in a bulleted list.

`setRefLink(ft_obj, {'url', 'standard'})` generates a list of links in the See Also section. `url` is a string that indicates the location to link to. You must provide the full link including the protocol. For example, `http:\www.mathworks.com` is a valid link, while `www.mathworks.com` is not a valid link. You can create a link to any protocol that is valid URL, such as a web site address, a full path to a file, or a relative path to a file.

---

**Note** `setRefLink` expects a cell array of cell arrays for the second input.

---

**Examples** Create a list object, `ft`, and add a related standard:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');  
setRefLink(ft, {'IEC 61508-3, Table A.3 (3) 'Language subset'});
```

Create a list object, `ft`, and add a list of related standards:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');  
setRefLink(ft, {  
    'IEC 61508-3, Table A.3 (2) 'Strongly typed programming language'',...  
    'IEC 61508-3, Table A.3 (3) 'Language subset''});
```

# ModelAdvisor.FormatTemplate.setRefLink

---

## **See Also**

Customizing the Model Advisor on page 1 — Describes how to create custom checks

“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

# ModelAdvisor.Text.setRetainSpaceReturn

---

## Purpose

Retain spacing and returns in text

## Syntax

```
setRetainSpaceReturn(text, mode)
```

## Description

`setRetainSpaceReturn(text, mode)` specifies whether the text must retain the spaces and carriage returns.

## Input Arguments

`text`

Instantiation of the `ModelAdvisor.Text` class

`mode`

A Boolean value indicating whether to preserve spaces and carriage returns in the text:

- `true` (default) — Preserve spaces and carriage returns.
- `false` — Do not preserve spaces and carriage returns.

## Example

```
t1 = ModelAdvisor.Text('MathWorks home page');  
setRetainSpaceReturn(t1, 'true');
```

## See Also

Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Table.setRowHeading

---

**Purpose** Specify table row title

**Syntax** `setRowHeading(table, row, heading)`

**Description** `setRowHeading(table, row, heading)` specifies a title for the designated table row.

|                        |                      |                                                                          |
|------------------------|----------------------|--------------------------------------------------------------------------|
| <b>Input Arguments</b> | <code>table</code>   | Instantiation of the <code>ModelAdvisor.Table</code> class               |
|                        | <code>row</code>     | An integer specifying row number                                         |
|                        | <code>heading</code> | A string, element object, or object array specifying the table row title |

**Example**

```
table1 = ModelAdvisor.Table(2,3);
setRowHeading(table1, 1, 'Row 1 Title');
setRowHeading(table1, 2, 'Row 2 Title');
setRowHeading(table1, 3, 'Row 3 Title');
```

**See Also** [Customizing the Model Advisor on page 1](#) — Describes how to create custom checks

# ModelAdvisor.Table.setRowHeadingAlign

---

**Purpose** Specify table row title alignment

**Syntax** `setRowHeadingAlign(table, row, alignment)`

**Description** `setRowHeadingAlign(table, row, alignment)` specifies the alignment for the designated table row.

**Input Arguments**

|                        |                                                                                                                                                                                                       |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>table</code>     | Instantiation of the <code>ModelAdvisor.Table</code> class                                                                                                                                            |
| <code>row</code>       | An integer specifying row number.                                                                                                                                                                     |
| <code>alignment</code> | A string specifying the cell alignment. Possible values are: <ul style="list-style-type: none"><li>• <code>left</code> (default)</li><li>• <code>right</code></li><li>• <code>center</code></li></ul> |

**Examples**

```
table1 = ModelAdvisor.Table(2, 3);
setRowHeading(table1, 1, 'Row 1 Title');
setRowHeadingAlign(table1, 1, 'center');
setRowHeading(table1, 2, 'Row 2 Title');
setRowHeadingAlign(table1, 2, 'center');
setRowHeading(table1, 3, 'Row 3 Title');
setRowHeadingAlign(table1, 3, 'center');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.InputParameter.setRowSpan

---

**Purpose** Specify rows for input parameter

**Syntax** `setRowSpan(input_param, [start_row end_row])`

**Description** `setRowSpan(input_param, [start_row end_row])` specifies the number of rows that the parameter occupies. Specify where you want an input parameter located in the layout grid when there are multiple input parameters.

## Input Arguments

|                          |                                                                                                    |
|--------------------------|----------------------------------------------------------------------------------------------------|
| <code>input_param</code> | The input parameter object                                                                         |
| <code>start_row</code>   | A positive integer representing the first row that the input parameter occupies in the layout grid |
| <code>end_row</code>     | A positive integer representing the last row that the input parameter occupies in the layout grid  |

## Examples

```
inputParam2 = ModelAdvisor.InputParameter;  
inputParam2.Name = 'Standard font size';  
inputParam2.Value='12';  
inputParam2.Type='String';  
inputParam2.Description='sample tooltip';  
inputParam2.setRowSpan([2 2]);  
inputParam2.setColSpan([1 1]);
```

# ModelAdvisor.FormatTemplate.setSubBar

---

**Purpose** Add line between subcheck results

**Syntax** `setSubBar(ft_obj, value)`

**Description** `setSubBar(ft_obj, value)` is an optional method that adds lines between results for subchecks. *ft\_obj* is a handle to a template object. *value* is a boolean value that specifies when the Model Advisor includes a line between subchecks in the check results. By default, the value is true, and the Model Advisor displays the bar. The Model Advisor does not display the bar when you set the value to false.

**Examples** Create a list object, `ft`, turn off the subbar:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');  
setSubBar(ft, false);
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

# ModelAdvisor.FormatTemplate.setSubResultStatus

---

**Purpose** Add status to check or subcheck result

**Syntax** `setSubResultStatus(ft_obj, 'status')`

**Description** `setSubResultStatus(ft_obj, 'status')` is an optional method that displays the status in the result. Use this method to display the status of the check or subcheck in the result. *ft\_obj* is a handle to a template object. *status* is a string identifying the status of the check. Valid strings are:

Pass

Warn

Fail

**Examples** Create a list object, *ft*, and add a passing status:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');  
setSubResultStatus(ft, 'Pass');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results



# ModelAdvisor.FormatTemplate.setSubResultStatusText

---

## Purpose

Add text below status in result

## Syntax

```
setSubResultStatusText(ft_obj, message)
```

## Description

`setSubResultStatusText(ft_obj, message)` is an optional method that displays text below the status in the result. Use this method to describe the status.

## Input Arguments

*ft\_obj*

A handle to a template object.

*message*

A string or a handle to a formatting object that the Model Advisor displays below the status in the report.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

## Examples

Create a list object, `ft`, add a passing status and a description of why the check passed:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setSubResultStatus(ft, 'Pass');
setSubResultStatusText(ft, ['Constructs that are not supported when '...
    'generating code were not found in the model or subsystem']);
```

## See Also

Customizing the Model Advisor on page 1 — Describes how to create custom checks

“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

# ModelAdvisor.Text.setSubscript

---

|                        |                                                                                                                                                                                                                                                                                                                                                                                         |                   |                                                           |                   |                                                                                                                                                                                                                          |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-----------------------------------------------------------|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>         | Specify subscripted text                                                                                                                                                                                                                                                                                                                                                                |                   |                                                           |                   |                                                                                                                                                                                                                          |
| <b>Syntax</b>          | <code>setSubscript(text, mode)</code>                                                                                                                                                                                                                                                                                                                                                   |                   |                                                           |                   |                                                                                                                                                                                                                          |
| <b>Description</b>     | <code>setSubscript(text, mode)</code> indicates whether to make text subscript.                                                                                                                                                                                                                                                                                                         |                   |                                                           |                   |                                                                                                                                                                                                                          |
| <b>Input Arguments</b> | <table><tr><td><code>text</code></td><td>Instantiation of the <code>ModelAdvisor.Text</code> class</td></tr><tr><td><code>mode</code></td><td>A Boolean value indicating subscripted formatting of text:<ul style="list-style-type: none"><li>• <code>true</code> — Make the text subscript.</li><li>• <code>false</code> — Do not make the text subscript.</li></ul></td></tr></table> | <code>text</code> | Instantiation of the <code>ModelAdvisor.Text</code> class | <code>mode</code> | A Boolean value indicating subscripted formatting of text: <ul style="list-style-type: none"><li>• <code>true</code> — Make the text subscript.</li><li>• <code>false</code> — Do not make the text subscript.</li></ul> |
| <code>text</code>      | Instantiation of the <code>ModelAdvisor.Text</code> class                                                                                                                                                                                                                                                                                                                               |                   |                                                           |                   |                                                                                                                                                                                                                          |
| <code>mode</code>      | A Boolean value indicating subscripted formatting of text: <ul style="list-style-type: none"><li>• <code>true</code> — Make the text subscript.</li><li>• <code>false</code> — Do not make the text subscript.</li></ul>                                                                                                                                                                |                   |                                                           |                   |                                                                                                                                                                                                                          |
| <b>Example</b>         | <pre>t1 = ModelAdvisor.Text('This is some text'); setSubscript(t1, 'true');</pre>                                                                                                                                                                                                                                                                                                       |                   |                                                           |                   |                                                                                                                                                                                                                          |
| <b>See Also</b>        | Customizing the Model Advisor on page 1 — Describes how to create custom checks                                                                                                                                                                                                                                                                                                         |                   |                                                           |                   |                                                                                                                                                                                                                          |

|                        |                                                                                                                                                                                                                                                                                                                                                                                               |                   |                                                           |                   |                                                                                                                                                                                                                                |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-----------------------------------------------------------|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>         | Specify superscripted text                                                                                                                                                                                                                                                                                                                                                                    |                   |                                                           |                   |                                                                                                                                                                                                                                |
| <b>Syntax</b>          | <code>setSuperscript(text, mode)</code>                                                                                                                                                                                                                                                                                                                                                       |                   |                                                           |                   |                                                                                                                                                                                                                                |
| <b>Description</b>     | <code>setSuperscript(text, mode)</code> indicates whether to make text subscript.                                                                                                                                                                                                                                                                                                             |                   |                                                           |                   |                                                                                                                                                                                                                                |
| <b>Input Arguments</b> | <table><tr><td><code>text</code></td><td>Instantiation of the <code>ModelAdvisor.Text</code> class</td></tr><tr><td><code>mode</code></td><td>A Boolean value indicating superscripted formatting of text:<ul style="list-style-type: none"><li>• <code>true</code> — Make the text superscript.</li><li>• <code>false</code> — Do not make the text superscript.</li></ul></td></tr></table> | <code>text</code> | Instantiation of the <code>ModelAdvisor.Text</code> class | <code>mode</code> | A Boolean value indicating superscripted formatting of text: <ul style="list-style-type: none"><li>• <code>true</code> — Make the text superscript.</li><li>• <code>false</code> — Do not make the text superscript.</li></ul> |
| <code>text</code>      | Instantiation of the <code>ModelAdvisor.Text</code> class                                                                                                                                                                                                                                                                                                                                     |                   |                                                           |                   |                                                                                                                                                                                                                                |
| <code>mode</code>      | A Boolean value indicating superscripted formatting of text: <ul style="list-style-type: none"><li>• <code>true</code> — Make the text superscript.</li><li>• <code>false</code> — Do not make the text superscript.</li></ul>                                                                                                                                                                |                   |                                                           |                   |                                                                                                                                                                                                                                |
| <b>Example</b>         | <pre>t1 = ModelAdvisor.Text('This is some text'); setSuperscript(t1, 'true');</pre>                                                                                                                                                                                                                                                                                                           |                   |                                                           |                   |                                                                                                                                                                                                                                |
| <b>See Also</b>        | Customizing the Model Advisor on page 1 — Describes how to create custom checks                                                                                                                                                                                                                                                                                                               |                   |                                                           |                   |                                                                                                                                                                                                                                |

# ModelAdvisor.FormatTemplate.setSubTitle

---

**Purpose** Add title for subcheck in result

**Syntax** `setSubTitle(ft_obj, title)`

**Description** `setSubTitle(ft_obj, title)` is an optional method that adds a subcheck result title. Use this method when you create subchecks to distinguish between them in the result.

**Input Arguments** *ft\_obj*  
A handle to a template object.

*title*  
A string or a handle to a formatting object specifying the title of the subcheck.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

**Examples** Create a list object, `ft`, and add a subcheck title:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setSubTitle(ft, ['Check for constructs in the model '...
    'that are not supported when generating code']);
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

# ModelAdvisor.FormatTemplate.setTableInfo

---

**Purpose** Add data to table

**Syntax** `setTableInfo(ft_obj, {data})`

**Description** `setTableInfo(ft_obj, {data})` is an optional method that creates a table. *ft\_obj* is a handle to a table template object. *data* is a cell array of strings or objects specifying the information in the body of the table. The Model Advisor creates hyperlinks to objects. If you do not add data to the table, the Model Advisor does not display the table in the result.

---

**Note** Before creating a table, you must specify column titles using the `setColTitle` method.

---

**Examples** Create a table object, `ft`, add column titles, and add data to the table:

```
ft = ModelAdvisor.FormatTemplate('TableTemplate');
setColTitle(ft, {'Index', 'Block Name'});
setTableInfo(ft, {'1', 'Gain'});
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

# ModelAdvisor.FormatTemplate.setTableTitle

---

**Purpose** Add title to table

**Syntax** `setTableTitle(ft_obj, title)`

**Description** `setTableTitle(ft_obj, title)` is an optional method that adds a title to a table.

**Input Arguments** *ft\_obj*  
A handle to a template object.

*title*

A string or a handle to a formatting object specifying the title of the table.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

The title appears above the table. If you do not add data to the table, the Model Advisor does not display the table and title in the result.

**Examples** Create a table object, `ft`, and add a table title:

```
ft = ModelAdvisor.FormatTemplate('TableTemplate');  
setTableTitle(ft, 'Table of fonts and styles used in model');
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks  
“Formatting Model Advisor Results” on page 20-38 — Describes how to format Model Advisor results

**Purpose** Specify list type

**Syntax** setType(list\_obj, listType)

**Description** setType(list\_obj, listType) specifies the type of list the ModelAdvisor.List constructor creates.

**Input Arguments**

|          |                                                                                                        |
|----------|--------------------------------------------------------------------------------------------------------|
| list_obj | Instantiation of the ModelAdvisor.List class                                                           |
| listType | Specifies the list type: <ul style="list-style-type: none"><li>• numbered</li><li>• bulleted</li></ul> |

**Example**

```
subList = ModelAdvisor.List();
subList.setType('numbered')
subList.addItem(ModelAdvisor.Text('Sub entry 1', {'pass','bold'}));
subList.addItem(ModelAdvisor.Text('Sub entry 2', {'pass','bold'}));
```

**See Also** Customizing the Model Advisor on page 1 — Describes how to create custom checks

# ModelAdvisor.Text.setUnderlined

---

|                        |                                                                                                                                                                                                                                                                                                                                                                              |                   |                                                           |                   |                                                                                                                                                                                                               |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-----------------------------------------------------------|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>         | Underline text                                                                                                                                                                                                                                                                                                                                                               |                   |                                                           |                   |                                                                                                                                                                                                               |
| <b>Syntax</b>          | <code>setUnderlined(text, mode)</code>                                                                                                                                                                                                                                                                                                                                       |                   |                                                           |                   |                                                                                                                                                                                                               |
| <b>Description</b>     | <code>setUnderlined(text, mode)</code> indicates whether to underline text.                                                                                                                                                                                                                                                                                                  |                   |                                                           |                   |                                                                                                                                                                                                               |
| <b>Input Arguments</b> | <table><tr><td><code>text</code></td><td>Instantiation of the <code>ModelAdvisor.Text</code> class</td></tr><tr><td><code>mode</code></td><td>A Boolean value indicating underlined formatting of text:<ul style="list-style-type: none"><li>• <code>true</code> — Underline the text.</li><li>• <code>false</code> — Do not underline the text.</li></ul></td></tr></table> | <code>text</code> | Instantiation of the <code>ModelAdvisor.Text</code> class | <code>mode</code> | A Boolean value indicating underlined formatting of text: <ul style="list-style-type: none"><li>• <code>true</code> — Underline the text.</li><li>• <code>false</code> — Do not underline the text.</li></ul> |
| <code>text</code>      | Instantiation of the <code>ModelAdvisor.Text</code> class                                                                                                                                                                                                                                                                                                                    |                   |                                                           |                   |                                                                                                                                                                                                               |
| <code>mode</code>      | A Boolean value indicating underlined formatting of text: <ul style="list-style-type: none"><li>• <code>true</code> — Underline the text.</li><li>• <code>false</code> — Do not underline the text.</li></ul>                                                                                                                                                                |                   |                                                           |                   |                                                                                                                                                                                                               |
| <b>Example</b>         | <pre>t1 = ModelAdvisor.Text('This is some text'); setUnderlined(t1, 'true');</pre>                                                                                                                                                                                                                                                                                           |                   |                                                           |                   |                                                                                                                                                                                                               |
| <b>See Also</b>        | Customizing the Model Advisor on page 1 — Describes how to create custom checks                                                                                                                                                                                                                                                                                              |                   |                                                           |                   |                                                                                                                                                                                                               |



**Purpose**

Collect signal range coverage information for model object

**Syntax**

```
[min, max] = sigrangeinfo(cvdo, object)
[min, max] = sigrangeinfo(cvdo, object, portID)
```

**Description**

[min, max] = sigrangeinfo(cvdo, object) returns the minimum and maximum signal values output by the model component object within the cvdata object cvdo.

[min, max] = sigrangeinfo(cvdo, object, portID) returns the minimum and maximum signal values associated with the output port portID of the Simulink block object.

**Input Arguments**

cvdo

cvdata object

object

An object in the model or Stateflow chart that received decision coverage. Valid values for object include the following:

**Object Specification****Description**

BlockPath

Full path to a model or block

BlockHandle

Handle to a model or block

s1Obj

Handle to a Simulink API object

sfID

Stateflow ID

sfObj

Handle to a Stateflow API object

{BlockPath, sfID}

Cell array with the path to a Stateflow block and the ID of an object contained in that chart

## Object Specification

{BlockPath, sfObj}

## Description

Cell array with the path to a Stateflow chart and a Stateflow object API handle contained in that chart

[BlockHandle, sfID]

Array with a Stateflow chart handle and the ID of an object contained in that chart

portID

Output port of the block object

## Output Arguments

max

Maximum signal values output by the model component object within the cvdata object cvdo. If object outputs a vector, min and max are also vectors.

min

Minimum signal values output by the model component object within the cvdata object cvdo. If object outputs a vector, min and max are also vectors.

## Examples

Collect signal range data for the Product block in the slvnvdemo\_cv\_small\_controller model:

```
mdl = 'slvnvdemo_cv_small_controller';
open_system(mdl)
testObj = cvtest(mdl) %Create test spec object
testObj.settings.sigrange = 1; %Enable MC/DC coverage
data = cvsim(testObj) %Simulate the model
blk_handle = get_param([mdl, '/Product'], 'Handle');
[minVal, maxVal] = sigrangeinfo(data, blk_handle) %Get signal range data
```

## See Also

[conditioninfo](#) | [cvsim](#) | [decisioninfo](#) | [mcdcinfo](#) | [tableinfo](#)

---

|                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>         | Display lookup table coverage information for model object                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Syntax</b>          | <pre>coverage = tableinfo(cvdo, object) coverage = tableinfo(cvdo, object, ignore_descendants) [coverage, exeCounts] = tableinfo(cvdo, object) [coverage, exeCounts, brkEquality] = tableinfo(cvdo, object)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b>     | <p><code>coverage = tableinfo(cvdo, object)</code> returns lookup table coverage results from the cvdata object <code>cvdo</code> for the model component <code>object</code>.</p> <p><code>coverage = tableinfo(cvdo, object, ignore_descendants)</code> returns lookup table coverage results for <code>object</code>, depending on the value of <code>ignore_descendants</code>.</p> <p><code>[coverage, exeCounts] = tableinfo(cvdo, object)</code> returns lookup table coverage results and the execution count for each interpolation/extrapolation interval in the lookup table block <code>object</code>.</p> <p><code>[coverage, exeCounts, brkEquality] = tableinfo(cvdo, object)</code> returns lookup table coverage results, the execution count for each interpolation/extrapolation interval, and the execution counts for breakpoint equality.</p> |
| <b>Input Arguments</b> | <p><code>cvdo</code><br/>cvdata object</p> <p><code>ignore_descendants</code><br/>Logical value specifying whether to ignore the coverage of descendant objects</p> <ul style="list-style-type: none"><li>1 — Ignore coverage of descendant objects</li><li>0 — Collect coverage for descendant objects</li></ul> <p><code>object</code><br/>Full path or handle to a lookup table block or a model containing a lookup table block.</p>                                                                                                                                                                                                                                                                                                                                                                                                                            |

## Output Arguments

`brkEquality`

A cell array containing vectors that identify the number of times during simulation that the lookup table block input was equivalent to a breakpoint value. Each vector represents the breakpoints along a different lookup table dimension.

`coverage`

The value of `coverage` is a two-element vector of form `[covered_intervals total_intervals]`, the elements of which are:

|                                |                                                                      |
|--------------------------------|----------------------------------------------------------------------|
| <code>covered_intervals</code> | Number of interpolation/extrapolation intervals satisfied for object |
| <code>total_intervals</code>   | Total number of interpolation/extrapolation intervals for object     |

`coverage` is empty if `cvdo` does not contain lookup table coverage results for object.

`execounts`

An array having the same dimensionality as the lookup table block; its size has been extended to allow for the lookup table extrapolation intervals.

## Examples

Collect lookup table coverage for the `slvndemo_cv_small_controller` model and determine the percentage of interpolation/extrapolation intervals coverage collected for the Gain Table block in the Gain subsystem:

```
mdl = 'slvndemo_cv_small_controller';
open_system(mdl)
testObj = cvtest(mdl) %Create test spec object
testObj.settings.tableExec = 1; %Enable lookup table coverage
data = cvsim(testObj) %Simulate the model
```

```
blk_handle = get_param([mdl, '/Gain/Gain Table'], 'Handle');
cov = tableinfo(data, blk_handle) %Retrieve l/u table coverage
percent_cov = 100 * cov(1) / cov(2) %Percent MC/DC outcomes covered
```

## Alternatives

To collect lookup coverage for a model:

- 1 Open the model.
- 2 In the Model Editor, select **Tools > Coverage Settings**.
- 3 On the **Coverage** tab, under **Coverage Metrics**, select **Look-up Table Coverage**.
- 4 On the **Results** and **Report** tabs, select the desired options.
- 5 Click **OK** to close the Coverage Settings dialog box.
- 6 Simulate the model and review the lookup table coverage in the report.

## See Also

conditioninfo | cvsim | decisioninfo | mcdcinfo | sigrangeinfo

## How To

- “Lookup Table Coverage” on page 15-6

# ModelAdvisor.ListViewParameter.Attributes property

---

**Purpose** Attributes to display in Model Advisor Report Explorer

**Values** Cell array

**Default:** {} (empty cell array)

**Description** The `Attributes` property specifies the attributes to display in the center pane of the Model Advisor Results Explorer.

## Example

```
% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
myLVParam.Data = get_param(searchResult,'object');
myLVParam.Attributes = {'FontName'}; % name is default property
```

# ModelAdvisor.Check.CallbackContext property

---

**Purpose** Model or subsystem context

**Values** 'PostCompile'  
'None' (default)

**Description** The `CallbackContext` property specifies the context for checking the model or subsystem.

'None' No special requirements for the model before checking.

'Postcompile' The model must be compiled.

# ModelAdvisor.Check.CallbackHandle property

---

|                    |                                                                                  |
|--------------------|----------------------------------------------------------------------------------|
| <b>Purpose</b>     | Callback function handle for check                                               |
| <b>Values</b>      | Function handle.<br>An empty handle [ ] is the default.                          |
| <b>Description</b> | The CallbackHandle property specifies the handle to the check callback function. |



# ModelAdvisor.Check.CallbackStyle property

---

|                    |                                                                                                                                                                                                                                                                                                                               |            |                                |            |                                  |              |                                                  |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|--------------------------------|------------|----------------------------------|--------------|--------------------------------------------------|
| <b>Purpose</b>     | Callback function type                                                                                                                                                                                                                                                                                                        |            |                                |            |                                  |              |                                                  |
| <b>Values</b>      | 'StyleOne' (default)<br>'StyleTwo'<br>'StyleThree'                                                                                                                                                                                                                                                                            |            |                                |            |                                  |              |                                                  |
| <b>Description</b> | The CallbackStyle property specifies the type of the callback function.<br><br><table><tr><td>'StyleOne'</td><td>Simple check callback function</td></tr><tr><td>'StyleTwo'</td><td>Detailed check callback function</td></tr><tr><td>'StyleThree'</td><td>Check callback function with hyperlinked results</td></tr></table> | 'StyleOne' | Simple check callback function | 'StyleTwo' | Detailed check callback function | 'StyleThree' | Check callback function with hyperlinked results |
| 'StyleOne'         | Simple check callback function                                                                                                                                                                                                                                                                                                |            |                                |            |                                  |              |                                                  |
| 'StyleTwo'         | Detailed check callback function                                                                                                                                                                                                                                                                                              |            |                                |            |                                  |              |                                                  |
| 'StyleThree'       | Check callback function with hyperlinked results                                                                                                                                                                                                                                                                              |            |                                |            |                                  |              |                                                  |

# ModelAdvisor.ListViewParameter.Data property

---

**Purpose** Objects in Model Advisor Result Explorer

**Values** Array of Simulink objects  
**Default:** [] (empty array)

**Description** The Data property specifies the objects displayed in the Model Advisor Result Explorer.

**Example**

```
% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
myLVParam.Data = get_param(searchResult, 'object');
```

# ModelAdvisor.Action.Description property

---

**Purpose**

Message in **Action** box

**Values**

String

**Default:** '' (null string)

**Description**

The **Description** property specifies the message displayed in the **Action** box.

**Example**

```
% define action (fix) operation
myAction = ModelAdvisor.Action;
%Specify a callback function for the action
myAction.setCallbackFcn(@sampleActionCB);
myAction.Name='Fix block fonts';
myAction.Description=...
    'Click the button to update all blocks with specified font';
```

# ModelAdvisor.FactoryGroup.Description property

---

**Purpose** Description of folder

**Values** String  
**Default:** '' (null string)

**Description** The Description property provides information about the folder. Details about the folder are displayed in the right pane of the Model Advisor.

**Example**

```
% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
rec.Description='Demo Factory Group';
```

# ModelAdvisor.Group.Description property

---

**Purpose** Description of folder

**Values** String

**Default:** '' (null string)

**Description** The Description property provides information about the folder. Details about the folder are displayed in the right pane of the Model Advisor.

**Example**

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');  
MAG.Description='This is my group';
```

# ModelAdvisor.InputParameter.Description property

---

**Purpose** Description of input parameter

**Values** String.

**Default:** '' (null string)

**Description** The Description property specifies a description of the input parameter. Details about the check are displayed in the right pane of the Model Advisor.

**Example**

```
% define input parameters
inputParam2 = ModelAdvisor.InputParameter;
inputParam2.Name = 'Standard font size';
inputParam2.Value='12';
inputParam2.Type='String';
inputParam2.Description='sample tooltip';
```

# ModelAdvisor.Task.Description property

---

**Purpose** Description of task

**Values** String

**Default:** '' (null string)

**Description** The Description property is a description of the task that the Model Advisor displays in the **Analysis** box.

When adding checks as tasks, the Model Advisor uses the task Description property instead of the check TitleTips property.

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.DisplayName='Example task 1';  
MAT1.Description='This is the first example task.'
```

```
MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');  
MAT2.DisplayName='Example task 2';  
MAT2.Description='This is the second example task.'
```

```
MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');  
MAT3.DisplayName='Example task 3';  
MAT3.Description='This is the third example task.'
```

# ModelAdvisor.FactoryGroup.DisplayName property

---

**Purpose** Name of folder

**Values** String  
**Default:** '' (null string)

**Description** The DisplayName specifies the name of the folder that is displayed in the Model Advisor.

**Examples**

```
% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
rec.DisplayName='Demo Factory Group';
```



# ModelAdvisor.Group.DisplayName property

---

|                    |                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Name of folder                                                                                           |
| <b>Values</b>      | String<br><b>Default:</b> '' (null string)                                                               |
| <b>Description</b> | The DisplayName specifies the name of the folder that is displayed in the Model Advisor.                 |
| <b>Examples</b>    | <pre>MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');<br/>MAG.DisplayName='My Group';</pre> |

# ModelAdvisor.Task.DisplayName property

---

**Purpose** Name of task

**Values** String

**Default:** '' (null string)

**Description** The `DisplayName` property specifies the name of the task. The Model Advisor displays each custom task in the tree using the name of the task. Therefore, you should specify a unique name for each task. When you specify the same name for multiple tasks, the Model Advisor generates a warning.

When adding checks as tasks, the Model Advisor uses the task `DisplayName` property instead of the check `Title` property.

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.DisplayName='Example task with input parameter and auto-fix ability';
```

```
MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');  
MAT2.DisplayName='Example task 2';
```

```
MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');  
MAT3.DisplayName='Example task 3';
```

# ModelAdvisor.Check.Enable property

---

**Purpose**

Indicate whether user can enable or disable check

**Values**

true (default)  
false

**Description**

The Enable property specifies whether the user can enable or disable the check.

|       |                               |
|-------|-------------------------------|
| true  | Display the check box control |
| false | Hide the check box control    |

# ModelAdvisor.Task.Enable property

---

**Purpose** Indicate if user can enable and disable task

**Values** true (default)  
false

**Description** The Enable property specifies whether the user can enable or disable a task.

|                |                                        |
|----------------|----------------------------------------|
| true (default) | Display the check box control for task |
| false          | Hide the check box control for task    |

When adding checks as tasks, the Model Advisor uses the task Enable property instead of the check Enable property.

**Example**

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.Enable = 'false';
```

# ModelAdvisor.InputParameter.Entries property

---

|                    |                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Drop-down list entries                                                                                                                                                                                                    |
| <b>Values</b>      | Depends on the value of the Type property.                                                                                                                                                                                |
| <b>Description</b> | <p>The Entries property is valid only when the Type property is one of the following:</p> <ul style="list-style-type: none"><li>• Enum</li><li>• ComboBox</li><li>• PushButton</li></ul>                                  |
| <b>Examples</b>    | <pre>inputParam3 = ModelAdvisor.InputParameter;<br/>inputParam3.Name='Valid font';<br/>inputParam3.Type='Combobox';<br/>inputParam3.Description='sample tooltip';<br/>inputParam3.Entries={'Arial', 'Arial Black'};</pre> |

# ModelAdvisor.Check.ID property

---

**Purpose** Identifier for check

**Values** String  
**Default:** '' (null string)

**Description** The ID property specifies a permanent, unique identifier for the check. Note the following about the ID property:

- You must specify this property.
- The value of ID must remain constant.
- The Model Advisor generates an error if ID is not unique.
- Tasks and factory group definitions must refer to checks by ID.

# ModelAdvisor.FactoryGroup.ID property

---

**Purpose** Identifier for folder

**Values** String

**Description** The ID property specifies a permanent, unique identifier for the folder.

---

## Note

- You must specify this field.
  - The value of ID must remain constant.
  - The Model Advisor generates an error if ID is not unique.
  - Group definitions must refer to other groups by ID.
-

# ModelAdvisor.Group.ID property

---

**Purpose** Identifier for folder

**Values** String

**Description** The ID property specifies a permanent, unique identifier for the folder.

---

**Note**

- You must specify this field.
  - The value of ID must remain constant.
  - The Model Advisor generates an error if ID is not unique.
  - Group definitions must refer to other groups by ID.
-



# ModelAdvisor.Task.ID property

---

**Purpose** Identifier for task

**Values** String

**Default:** '' (null string)

**Description** The ID property specifies a permanent, unique identifier for the task.

---

## Note

- The Model Advisor automatically assigns a string to ID if you do not specify it.
  - The value of ID must remain constant.
  - The Model Advisor generates an error if ID is not unique.
  - Group definitions must refer to tasks using ID.
- 

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.ID='Task_ID_1234';
```

# ModelAdvisor.Check.LicenseName property

---

**Purpose** Product license names required to display and run check

**Values** Cell array of product license names  
{}(empty cell array) (default)

**Description** The `LicenseName` property specifies a cell array of names for product licenses required to display and run the check.

When the Model Advisor starts, it tests whether the product license exists. If you do not meet the license requirements, the Model Advisor does not display the check.

The Model Advisor performs a checkout of the product licenses when you run the custom check. If you do not have the product licenses available, you see an error message that the required license is not available.

---

**Tip** To find the correct text for license strings, type `help license` at the MATLAB command line.

---

# ModelAdvisor.Task.LicenseName property

---

## Purpose

Product license names required to display and run task

## Values

Cell array of product license names

**Default:** {} (empty cell array)

## Description

The `LicenseName` property specifies a cell array of names for product licenses required to display and run the check.

When the Model Advisor starts, it tests whether the product license exists. If you do not meet the license requirements, the Model Advisor does not display the check.

The Model Advisor performs a checkout of the product licenses when you run the custom check. If you do not have the product licenses available, you see an error message that the required license is not available.

If you specify `ModelAdvisor.Check.LicenseName`, the Model Advisor displays the check when the union of both properties is true.

---

**Tip** To find the correct text for license strings, type `help license` at the MATLAB command line.

---

# ModelAdvisor.Check.ListViewVisible property

---

**Purpose** Status of **Explore Result** button

**Values** false (default)  
true

**Description** The `ListViewVisible` property is a Boolean value that sets the status of the **Explore Result** button.

|       |                                           |
|-------|-------------------------------------------|
| true  | Display the <b>Explore Result</b> button. |
| false | Hide the <b>Explore Result</b> button.    |

**Example**

```
% add 'Explore Result' button  
rec.ListViewVisible = true;
```

# ModelAdvisor.FactoryGroup.MAObj property

---

|                    |                                                                            |
|--------------------|----------------------------------------------------------------------------|
| <b>Purpose</b>     | Model Advisor object                                                       |
| <b>Values</b>      | Handle to a Simulink.ModelAdvisor object                                   |
| <b>Description</b> | The MAObj property specifies a handle to the current Model Advisor object. |

# ModelAdvisor.Group.MAObj property

---

|                    |                                                                            |
|--------------------|----------------------------------------------------------------------------|
| <b>Purpose</b>     | Model Advisor object                                                       |
| <b>Values</b>      | Handle to Simulink.ModelAdvisor object                                     |
| <b>Description</b> | The MAObj property specifies a handle to the current Model Advisor object. |

# ModelAdvisor.Task.MAObj property

---

|                    |                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Model Advisor object                                                                                                                                                                          |
| <b>Values</b>      | Handle to a Simulink.ModelAdvisor object                                                                                                                                                      |
| <b>Description</b> | <p>The MAObj property specifies the current Model Advisor object.</p> <p>When adding checks as tasks, the Model Advisor uses the task MAObj property instead of the check MAObj property.</p> |

## cv.cvdatagroup.name property

---

|                    |                                                                                  |
|--------------------|----------------------------------------------------------------------------------|
| <b>Purpose</b>     | cv.cvdatagroup object name                                                       |
| <b>Values</b>      | name                                                                             |
| <b>Description</b> | The name property specifies the name of the cv.cvdatagroup object.               |
| <b>Examples</b>    | <pre>cvdg = cvsimref(topModelName, cvtg);<br/>cvdg.name = 'My_Data_Group';</pre> |



|                    |                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | cv.cvtestgroup object name                                                                                                                                                |
| <b>Value</b>       | name                                                                                                                                                                      |
| <b>Description</b> | The name property specifies the name of the cv.cvtestgroup object.                                                                                                        |
| <b>Examples</b>    | <pre>cvto1 = cvtest('TopModel'); cvto2 = cvtest('SubModel1'); cvto3 = cvtest('SubModel2'); cvtg = cv.cvtestgroup(cvto1, cvto2, cvto3); cvtg.name = 'My_Test_Group';</pre> |
| <b>See Also</b>    | cvtest                                                                                                                                                                    |

# ModelAdvisor.Action.Name property

---

|                    |                                                                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Action button label                                                                                                                                                                               |
| <b>Values</b>      | String<br><b>Default:</b> '' (null string)                                                                                                                                                        |
| <b>Description</b> | The Name property specifies the label for the action button. This property is required.                                                                                                           |
| <b>Example</b>     | <pre>% define action (fix) operation myAction = ModelAdvisor.Action; %Specify a callback function for the action myAction.setCallbackFcn(@sampleActionCB); myAction.Name='Fix block fonts';</pre> |

# ModelAdvisor.InputParameter.Name property

---

|                    |                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Input parameter name                                                                                                                                                                                        |
| <b>Values</b>      | String.<br><b>Default:</b> '' (null string)                                                                                                                                                                 |
| <b>Description</b> | The Name property specifies the name of the input parameter in the custom check.                                                                                                                            |
| <b>Examples</b>    | <pre>inputParam2 = ModelAdvisor.InputParameter;<br/>inputParam2.Name = 'Standard font size';<br/>inputParam2.Value='12';<br/>inputParam2.Type='String';<br/>inputParam2.Description='sample tooltip';</pre> |

# ModelAdvisor.ListViewParameter.Name property

---

**Purpose** Drop-down list entry

**Values** String

**Default:** '' (null string)

**Description** The Name property specifies an entry in the **Show** drop-down list in the Model Advisor Result Explorer.

**Examples**

```
% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
```

# ModelAdvisor.Check.Result property

---

**Purpose** Results cell array

**Values** Cell array

**Default:** {} (empty cell array)

**Description** The `Result` property specifies the cell array for storing the results that are returned by the callback function specified in `CallbackHandle`.

---

**Tip** To set the icon associated with the check, use the `Simulink.ModelAdvisor setCheckResultStatus` and `setCheckErrorSeverity` methods.

---

# ModelAdvisor.Check.Title property

---

**Purpose** Name of check

**Values** String  
**Default:** '' (null string)

**Description** The Title property specifies the name of the check in the Model Advisor. The Model Advisor displays each custom check in the tree using the title of the check. Therefore, you should specify a unique title for each check. When you specify the same title for multiple checks, the Model Advisor generates a warning.

**Example**

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');  
rec.Title = 'Check Simulink block font';
```

# ModelAdvisor.Check.TitleTips property

---

**Purpose**

Description of check

**Values**

String

**Default:** '' (null string)

**Description**

The TitleTips property specifies a description of the check. Details about the check are displayed in the right pane of the Model Advisor.

**Example**

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');  
rec.Title = 'Check Simulink block font';  
rec.TitleTips = 'Example style three callback';
```

# ModelAdvisor.InputParameter.Type property

---

**Purpose** Input parameter type

**Values** String.

**Default:** '' (null string)

**Description** The Type property specifies the type of input parameter.

Use the Type property with the Value and Entries properties to define input parameters.

Valid values are listed in the following table.

| Type     | Data Type  | Default Value           | Description                                                                                                                                                                                                         |
|----------|------------|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bool     | Boolean    | false                   | A check box                                                                                                                                                                                                         |
| ComboBox | Cell array | First entry in the list | A drop-down menu <ul style="list-style-type: none"><li>• Use Entries to define the entries in the list.</li><li>• Use Value to indicate a specific entry in the menu or to enter a value not in the list.</li></ul> |
| Enum     | Cell array | First entry in the list | A drop-down menu <ul style="list-style-type: none"><li>• Use Entries to define the entries in the list.</li><li>• Use Value to indicate a specific entry in the list.</li></ul>                                     |



# ModelAdvisor.InputParameter.Type property

| Type       | Data Type | Default Value     | Description                                                                                               |
|------------|-----------|-------------------|-----------------------------------------------------------------------------------------------------------|
| PushButton | N/A       | N/A               | A button<br>When you click the button, the callback function specified by <code>Entries</code> is called. |
| String     | String    | ' ' (null string) | A text box                                                                                                |

## Examples

```
% define input parameters
inputParam1 = ModelAdvisor.InputParameter;
inputParam1.Name = 'Skip font checks.';
inputParam1.Type = 'Bool';
inputParam1.Value = false;
```

# ModelAdvisor.Check.Value property

---

**Purpose** Status of check

**Values** 'true' (default)  
'false'

**Description** The Value property specifies the initial status of the check.

'true' Check is enabled  
'false' Check is disabled

**Examples**

```
% hide all checks that do not belong to Demo group
if ~(strcmp(checkCellArray{i}.Group, 'Demo'))
    checkCellArray{i}.Visible = false;
    checkCellArray{i}.Value = false;
end
```

# ModelAdvisor.InputParameter.Value property

---

|                    |                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Value of input parameter                                                                                                                                                                                                                                                    |
| <b>Values</b>      | Depends on the Type property.                                                                                                                                                                                                                                               |
| <b>Description</b> | <p>The Value property specifies the initial value of the input parameter. This property is valid only when the Type property is one of the following:</p> <ul style="list-style-type: none"><li>• 'Bool'</li><li>• 'String'</li><li>• 'Enum'</li><li>• 'ComboBox'</li></ul> |
| <b>Example</b>     | <pre>% define input parameters inputParam1 = ModelAdvisor.InputParameter; inputParam1.Name = 'Skip font checks.'; inputParam1.Type = 'Bool'; inputParam1.Value = false;</pre>                                                                                               |

# ModelAdvisor.Task.Value property

---

## Purpose

Status of task

## Values

'true' (default) — Initial status of task is enabled

'false' — Initial status of task is disabled

## Description

The Value property indicates the initial status of a task—whether it is enabled or disabled.

When adding checks as tasks, the Model Advisor uses the task Value property instead of the check Value property.

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.Value = 'false';
```

# ModelAdvisor.Check.Visible property

---

## Purpose

Indicate to display or hide check

## Values

'true' (default)  
'false'

## Description

The Visible property specifies whether the Model Advisor displays the check.

|         |                   |
|---------|-------------------|
| 'true'  | Display the check |
| 'false' | Hide the check    |

## Examples

```
% hide all checks that do not belong to Demo group
if ~(strcmp(checkCellArray{i}.Group, 'Demo'))
    checkCellArray{i}.Visible = false;
    checkCellArray{i}.Value = false;
end
```

# ModelAdvisor.Task.Visible property

---

**Purpose** Indicate to display or hide task

**Values** 'true' (default) — Display task in the Model Advisor  
'false' — Hide task

**Description** The `Visible` property specifies whether the Model Advisor displays the task.

---

## Caution

When adding checks as tasks, you cannot specify both the task and check `Visible` properties, you must specify one or the other. If you specify both properties, the Model Advisor generates an error when the check `Visible` property is `false`.

---

**Example**

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.Visible = 'false';
```

# Block Reference

---

# System Requirements

**Purpose** List system requirements in Simulink diagrams

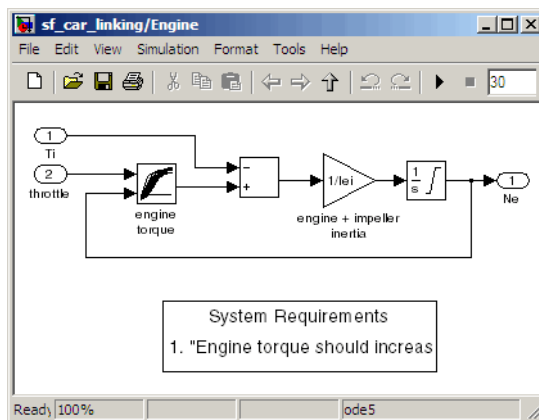
**Library** Simulink Verification and Validation

**Description** The System Requirements block lists all the system requirements associated with the model or subsystem depicted in the current diagram. It does not list requirements associated with individual blocks in the diagram.



You can place this block anywhere in a diagram. It is not connected to other Simulink blocks. You can only have one System Requirements block in a diagram.

When you drag the System Requirements block from the Library Browser into your Simulink diagram, it is automatically populated with the system requirements, as shown.



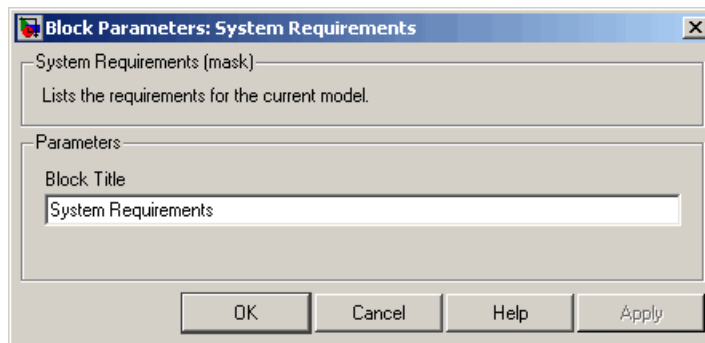
Each of the listed requirements is an active link to the actual requirements document. When you double-click on a requirement name, the associated requirements document opens in its editor window, scrolled to the target location.



If the System Requirements block exists in a diagram, it automatically updates the requirements listing as you add, modify, or delete requirements for the model or subsystem.

## Dialog Box and Parameters

To access the Block Parameters dialog box for the System Requirements block, right-click on the System Requirements block and, from the resulting pop-up menu, select **Mask Parameters**. The Block Parameters dialog box opens, as shown.



The Block Parameters dialog box for the System Requirements block contains one parameter.

### Block Title

The title of the system requirements list in the diagram. The default title is **System Requirements**. You can type a customized title, for example, **Engine Requirements**.

# System Requirements

---

# Model Advisor Checks

---

- “Simulink® Verification and Validation Checks” on page 27-2
- “DO-178B Checks” on page 27-4
- “IEC 61508 Checks” on page 27-67
- “MathWorks Automotive Advisory Board Checks” on page 27-87
- “Requirements Consistency Checks” on page 27-163

## Simulink Verification and Validation Checks

| In this section...                                                   |
|----------------------------------------------------------------------|
| “Simulink® Verification and Validation Checks Overview” on page 27-2 |
| “Modeling Standards Checks Overview” on page 27-3                    |

### Simulink Verification and Validation Checks Overview

Simulink Verification and Validation checks facilitate designing and troubleshooting models from which code is generated for applications that must meet safety or mission-critical requirements, modeling guidelines, or requirements consistency.

The Model Advisor performs a checkout of the Simulink Verification and Validation license when you run the Simulink Verification and Validation checks.

For descriptions of the modeling standards checks, see

- “DO-178B Checks” on page 27-4
- “IEC 61508 Checks” on page 27-67
- “MathWorks Automotive Advisory Board Checks” on page 27-87

For descriptions of the requirements consistency checks, see “Requirements Consistency Checks” on page 27-163.

### See Also

- “Consulting the Model Advisor” in the Simulink documentation
- “Simulink Checks” in the Simulink reference documentation
- “Real-Time Workshop Checks” in the Real-Time Workshop documentation

## Modeling Standards Checks Overview

Modeling standards checks facilitate designing and troubleshooting models from which code is generated for applications that must meet safety or mission-critical requirements or MathWorks Automotive Advisory Board (MAAB) modeling guidelines.

The Model Advisor performs a checkout of the Simulink Verification and Validation license when you run the modeling standards checks.

For descriptions of the modeling standards checks, see

- “DO-178B Checks” on page 27-4
- “IEC 61508 Checks” on page 27-67
- “MathWorks Automotive Advisory Board Checks” on page 27-87

### See Also

- “Consulting the Model Advisor” in the Simulink documentation
- “Simulink Checks” in the Simulink reference documentation
- “Real-Time Workshop Checks” in the Real-Time Workshop documentation

## DO-178B Checks

**In this section...**

“DO-178B Checks Overview” on page 27-5

“Check safety-related optimization settings” on page 27-6

“Check safety-related diagnostic settings for solvers” on page 27-10

“Check safety-related diagnostic settings for sample time” on page 27-13

“Check safety-related diagnostic settings for signal data” on page 27-16

“Check safety-related diagnostic settings for parameters” on page 27-19

“Check safety-related diagnostic settings for data used for debugging” on page 27-22

“Check safety-related diagnostic settings for data store memory” on page 27-24

“Check safety-related diagnostic settings for type conversions” on page 27-26

“Check safety-related diagnostic settings for signal connectivity” on page 27-28

“Check safety-related diagnostic settings for bus connectivity” on page 27-30

“Check safety-related diagnostic settings that apply to function-call connectivity” on page 27-32

“Check safety-related diagnostic settings for compatibility” on page 27-34

“Check safety-related diagnostic settings for model initialization” on page 27-36

“Check safety-related diagnostic settings for model referencing” on page 27-38

“Check safety-related model referencing settings” on page 27-41

“Check safety-related code generation settings” on page 27-43

“Check safety-related diagnostic settings for saving” on page 27-50

“Check for model objects that do not link to requirements” on page 27-52

“Check for proper usage of Math blocks” on page 27-53

“Check state machine type of Stateflow charts” on page 27-54

**In this section...**

“Check Stateflow charts for ordering of states and transitions” on page 27-56

“Check Stateflow debugging settings” on page 27-57

“Check for proper usage of For Iterator blocks” on page 27-59

“Check for proper usage of While Iterator blocks” on page 27-60

“Display model version information” on page 27-62

“Check for proper usage of blocks that compute absolute values” on page 27-63

“Check for proper usage of Relational Operator blocks” on page 27-65

## DO-178B Checks Overview

DO-178B checks facilitate designing and troubleshooting models from which code is generated for applications that must meet safety or mission-critical requirements.

The Model Advisor performs a checkout of the Simulink Verification and Validation license when you run the DO-178B checks.

### See Also

- “Consulting the Model Advisor” in the Simulink documentation
- “Simulink Checks” in the Simulink reference documentation
- “Real-Time Workshop Checks” in the Real-Time Workshop documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related optimization settings

Check model configuration for optimization settings that can impact safety.

### Description

This check verifies that model optimization configuration parameters are set optimally for generating code for a safety-related application. Although highly optimized code is desirable for most real-time systems, some optimizations can have undesirable side effects that impact safety.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                            | Recommended Action                                                                                                                                                                                     |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Block reduction optimization is selected. This optimization can remove blocks from generated code, resulting in requirements with no associated code and violations for traceability requirements. (See DO-178B, Section 6.3.4e—Source code is traceable to low-level requirements.)                 | Clear the <b>Block reduction</b> check box on the <b>Optimization</b> pane of the Configuration Parameters dialog box or set the parameter <code>BlockReduction</code> to off.                         |
| Implementation of logic signals as Boolean data is cleared. Strong data typing is recommended for safety-related code. (See DO-178B, Section 6.3.1e—High-level requirements conform to standards, DO-178B, Section 6.3.2e—Low-level requirements conform to standards, and MISRA-C:2004, Rule 12.6.) | Select <b>Implement logic signals as boolean data (vs. double)</b> on the <b>Optimization</b> pane of the Configuration Parameters dialog box or set the parameter <code>BooleanDataType</code> to on. |



| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Recommended Action                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The model includes blocks that depend on elapsed or absolute time and is configured to minimize the amount of memory allocated for the timers. Such a configuration limits the number of days the application can execute before a timer overflow occurs. Many aerospace products are powered on continuously and timers should not assume a limited lifespan. (See DO-178B, Section 6.3.1g—Algorithms are accurate, DO-178B, Section 6.3.2g—Algorithms are accurate, and MISRA-C:2004, Rule 12.11.)</p> | <p>Set <b>Application lifespan (days)</b> on the <b>Optimization</b> pane of the Configuration Parameters dialog box or set the parameter <code>LifeSpan</code> to <code>inf</code>.</p>                                                                                                                                                                                                                                                                         |
| <p>The optimization that ignores integer downcasts in folded expressions is selected. This optimization can remove blocks that typecast data from generated code, resulting in incorrect behavior due to overflows of integer data and requirements without associated code. (See DO-178B, Section 6.3.1g—Algorithms are accurate, DO-178B, Section 6.3.2g—Algorithms are accurate, and MISRA-C:2004, Rule 10.1.)</p>                                                                                       | <p>If you have a Real-Time Workshop license, set the parameter <code>EnforceIntegerDowncast</code> to <code>on</code>.</p>                                                                                                                                                                                                                                                                                                                                       |
| <p>The optimization that suppresses the generation of initialization code for root-level inports and outports that are set to zero is selected. For safety-related code, you should explicitly initialize all variables. (See DO-178B, Section 6.3.3b—Software architecture is consistent and MISRA-C:2004, Rule 9.1.)</p>                                                                                                                                                                                  | <p>If you have a Real-Time Workshop Embedded Coder license, and you are using an ERT-based system target file, clear the <b>Remove root level I/O zero initialization</b> check box on the <b>Optimization</b> pane of the Configuration Parameters dialog box or set the parameter <code>ZeroExternalMemoryAtStartup</code> to <code>on</code>. Alternatively, integrate external, hand-written code that initializes all I/O variables to zero explicitly.</p> |

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Recommended Action                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The optimization that suppresses the generation of initialization code for internal work structures, such as block states and block outputs that are set to zero, is selected. For safety-related code, you should explicitly initialize all variables. (See DO-178B, Section 6.3.3b—Software architecture is consistent and MISRA-C:2004, Rule 9.1.)</p>                                                                                                                                                                                                                                          | <p>If you have a Real-Time Workshop Embedded Coder license, and you are using an ERT-based system target file, clear the <b>Remove internal data zero initialization</b> check box on the <b>Optimization</b> pane of the Configuration Parameters dialog box or set the parameter <code>ZeroInternalMemoryAtStartup</code> to on. Alternatively, integrate external, hand-written code that initializes all state variables to zero explicitly.</p> |
| <p>The optimization that suppresses generation of code resulting from floating-point to integer conversions that wrap out-of-range values is cleared. You must avoid overflows for safety-related code. When this optimization is off and your model includes blocks that disable the <b>Saturate on overflow</b> parameter, the code generator wraps out-of-range values for those blocks. This can result in unreachable and, therefore, untestable code. (See DO-178B, Section 6.3.1g—Algorithms are accurate, DO-178B, Section 6.3.2g—Algorithms are accurate, and MISRA-C:2004, Rule 12.11.)</p> | <p>If you have a Real-Time Workshop license, select <b>Remove code from floating-point to integer conversions that wraps out-of-range values</b> on the <b>Optimization</b> pane of the Configuration Parameters dialog box or set the parameter <code>EfficientFloat2IntCast</code> to on.</p>                                                                                                                                                      |
| <p>The optimization that suppresses generation of code that guards against division by zero for fixed-point data is selected. You must avoid division-by-zero exceptions in safety-related code. (See DO-178B, Section 6.3.1g—Algorithms are accurate, DO-178B, Section 6.3.2g—Algorithms are accurate, and MISRA C 2004, Rule 21.1.)</p>                                                                                                                                                                                                                                                             | <p>If you have a Real-Time Workshop Embedded Coder license, and you are using an ERT-based system target file, clear the <b>Remove code that protects against division arithmetic exceptions</b> check box on the <b>Optimization</b> pane of the Configuration Parameters dialog box or set the parameter <code>NoFixptDivByZeroProtection</code> to off.</p>                                                                                       |

### Action Results

Clicking **Modify Settings** configures model optimization settings that can impact safety.

**See Also**

- Optimization Pane in the Simulink graphical user interface documentation
- Optimizing a Model for Code Generation in the Real-Time Workshop documentation
- Tips for Optimizing the Generated Code in the Real-Time Workshop Embedded Coder documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related diagnostic settings for solvers

Check model configuration for diagnostic settings that apply to solvers and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to solvers are set optimally for generating code for a safety-related application.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                             | Recommended Action                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic for detecting automatic breakage of algebraic loops is set to none or warning. The breaking of algebraic loops can affect the predictability of the order of block execution. For safety-related applications, a model developer needs to know when such breaks occur. (See DO-178B, Section 6.3.3e – Software architecture conforms to standards.)</p>                                                             | <p>Set <b>Algebraic loop</b> on the <b>Diagnostics &gt; Solver</b> pane of the Configuration Parameters dialog box or set the parameter AlgebraicLoopMsg to error. Consider breaking such loops explicitly with Unit Delay blocks to ensure that execution order is predictable. At a minimum, verify that the results of loops breaking automatically are acceptable.</p>                    |
| <p>The diagnostic for detecting automatic breakage of algebraic loops for Model blocks, atomic subsystems, and enabled subsystems is set to none or warning. The breaking of algebraic loops can affect the predictability of the order of block execution. For safety-related applications, a model developer needs to know when such breaks occur. (See DO-178B, Section 6.3.3e – Software architecture conforms to standards.)</p> | <p>Set <b>Minimize algebraic loop</b> on the <b>Diagnostics &gt; Solver</b> pane of the Configuration Parameters dialog box or set the parameter ArtificialAlgebraicLoopMsg to error. Consider breaking such loops explicitly with Unit Delay blocks to ensure that execution order is predictable. At a minimum, verify that the results of loops breaking automatically are acceptable.</p> |

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                             | Recommended Action                                                                                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic for detecting potential conflict in block execution order is set to none or warning. For safety-related applications, block execution order must be predictable. A model developer needs to know when conflicting block priorities exist. (See DO-178B, Section 6.3.3b – Software architecture is consistent.)</p>                                                                                                                  | <p>Set <b>Block priority violation</b> on the <b>Diagnostics &gt; Solver</b> pane of the Configuration Parameters dialog box or set the parameter BlockPriorityViolationMsg to error.</p>            |
| <p>The diagnostic for detecting whether a model contains an S-function that has not been specified explicitly to inherit sample time is set to none or warning. These settings can result in unpredictable behavior. A model developer needs to know when such an S-function exists in a model so it can be modified to produce predictable behavior. (See DO-178B, Section 6.3.3e – Software architecture conforms to standards.)</p>                | <p>Set <b>Unspecified inheritability of sample times</b> on the <b>Diagnostics &gt; Solver</b> pane of the Configuration Parameters dialog box or set the parameter UnknownTs1nhSupMsg to error.</p> |
| <p>The diagnostic for detecting whether the Simulink software automatically modifies the solver, step size, or simulation stop time is set to none or warning. Such changes can affect the operation of generated code. For safety-related applications, it is better to detect such changes so a model developer can explicitly set the parameters to known values. (See DO-178B, Section 6.3.3e – Software architecture conforms to standards.)</p> | <p>Set <b>Automatic solver parameter selection</b> on the <b>Diagnostics &gt; Solver</b> pane of the Configuration Parameters dialog box or set the parameter SolverPrmCheckMsg to error.</p>        |
| <p>The diagnostic for detecting when a name is used for more than one state in the model is set to none. State names within a model should be unique. For safety-related applications, it is better to detect name clashes so a model developer can correct them. (See DO-178B, Section 6.3.3b – Software architecture is consistent.)</p>                                                                                                            | <p>Set <b>State name clash</b> on the <b>Diagnostics &gt; Solver</b> pane of the Configuration Parameters dialog box or set the parameter StateNameClashWarn to warning.</p>                         |

### **Action Results**

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

### **See Also**

- Diagnostics Pane: Solver in the Simulink graphical user interface documentation
- Diagnosing Simulation Errors in the Simulink documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related diagnostic settings for sample time

Check model configuration for diagnostic settings that apply to sample time and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to sample times are set optimally for generating code for a safety-related application.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Recommended Action                                                                                                                                                                                                 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic for detecting when a source block, such as a Sine Wave block, inherits a sample time (specified as -1) is set to none or warning. The use of inherited sample times for a source block can result in unpredictable execution rates for the source block and blocks connected to it. For safety-related applications, source blocks should have explicit sample times to prevent incorrect execution sequencing. (See DO-178B, Section 6.3.3e – Software architecture conforms to standards.)</p> | <p>Set <b>Source block specifies -1 sample time</b> on the <b>Diagnostics &gt; Sample Time</b> pane of the Configuration Parameters dialog box or set the parameter <code>InheritedTsInSrcMsg</code> to error.</p> |
| <p>The diagnostic for detecting whether the input for a discrete block, such as the Unit Delay block, is a continuous signal is set to none or warning. Signals with continuous sample times should not be used for embedded real-time code. (See DO-178B, Section 6.3.3e – Software architecture conforms to standards.)</p>                                                                                                                                                                                      | <p>Set <b>Discrete used as continuous</b> on the <b>Diagnostics &gt; Sample Time</b> pane of the Configuration Parameters dialog box or set the parameter <code>DiscreteInheritContinuousMsg</code> to error.</p>  |

| <b>Condition</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | <b>Recommended Action</b>                                                                                                                                                                                                                         |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic for detecting invalid rate transitions between two blocks operating in multitasking mode is set to none or warning. Such rate transitions should not be used for embedded real-time code. (See DO-178B, Section 6.3.3b – Software architecture is consistent.)</p>                                                                                                                                                                                              | <p>Set <b>Multitask rate transition</b> on the <b>Diagnostics &gt; Sample Time</b> pane of the Configuration Parameters dialog box or set the parameter <code>MultiTaskRateTransMsg</code> to error.</p>                                          |
| <p>The diagnostic for detecting subsystems that can cause data corruption or nondeterministic behavior is set to none or warning. This diagnostic detects whether conditionally executed multirate subsystems (enabled, triggered, or function-call subsystems) operate in multitasking mode. Such subsystems can corrupt data and behave unpredictably in real-time environments that allow preemption. (See DO-178B, Section 6.3.3b – Software architecture is consistent.)</p> | <p>Set <b>Multitask conditionally executed subsystem</b> on the <b>Diagnostics &gt; Sample Time</b> pane of the Configuration Parameters dialog box or set the parameter <code>MultiTaskCondExecSysMsg</code> to error.</p>                       |
| <p>The diagnostic for checking sample time consistency between a Signal Specification block and the connected destination block is set to none or warning. An over-specified sample time can result in an unpredictable execution rate. (See DO-178B, Section 6.3.3e – Software architecture conforms to standards.)</p>                                                                                                                                                          | <p>Set <b>Enforce sample times specified by Signal Specification blocks</b> on the <b>Diagnostics &gt; Sample Time</b> pane of the Configuration Parameters dialog box or set the parameter <code>SigSpecEnsureSampleTimeMsg</code> to error.</p> |

**Action Results**

Clicking **Modify Settings** configures model diagnostic settings that apply to sample time and that can impact safety.



**See Also**

- [Diagnostics Pane: Sample Time in the Simulink graphical user interface documentation](#)
- [Diagnosing Simulation Errors in the Simulink documentation](#)
- [Radio Technical Commission for Aeronautics \(RTCA\) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard](#)

## Check safety-related diagnostic settings for signal data

Check model configuration for diagnostic settings that apply to signal data and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to signal data are set optimally for generating code for a safety-related application.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Recommended Action                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic that specifies how the Simulink software resolves signals associated with <code>Simulink.Signal</code> objects in the MATLAB workspace is set to <code>Explicit</code> and <code>implicit</code> or <code>Explicit</code> and <code>warn implicit</code>. For safety-related applications, model developers should be required to define signal resolution explicitly. (See DO-178B, Section 6.3.3b – Software architecture is consistent.)</p>                | <p>Set <b>Signal resolution</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>SignalResolutionControl</code> to <code>Explicit</code> only. This provides predictable operation by requiring users to define each signal and block setting that must resolve to <code>Simulink.Signal</code> objects in the workspace.</p> |
| <p>The Product block diagnostic that detects a singular matrix while inverting one of its inputs in matrix multiplication mode is set to <code>none</code> or <code>warning</code>. Division by a singular matrix can result in numeric exceptions when executing generated code. This is not acceptable in safety-related systems. (See DO-178B, Section 6.3.1g – Algorithms are accurate, DO-178B, Section 6.3.2g – Algorithms are accurate, and MISRA C 2004, Rule 21.1.)</p> | <p>Set <b>Division by singular matrix</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>CheckMatrixSingularityMsg</code> to <code>error</code>.</p>                                                                                                                                                                        |

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                              | Recommended Action                                                                                                                                                                                             |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic that detects when the Simulink software cannot infer the data type of a signal during data type propagation is set to none or warning. For safety-related applications, model developers must ensure that all data types are specified correctly. (See DO-178B, Section 6.3.1e – High-level requirements conform to standards, DO-178B and Section 6.3.2e – Low-level requirements conform to standards.)</p>                        | <p>Set <b>Underspecified data types</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>UnderSpecifiedDataTypeMsg</code> to error.</p> |
| <p>The diagnostic that detects whether the value of a signal or parameter is too large to be represented by the signal or parameter's data type is set to none or warning. Undetected numeric overflows can result in unexpected application behavior. (See DO-178B, Section 6.3.1g – Algorithms are accurate, DO-178B, Section 6.3.2g – Algorithms are accurate, and MISRA-C:2004, Rule 21.1.)</p>                                                    | <p>Set <b>Detect overflow</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>IntegerOverflowMsg</code> to error.</p>                  |
| <p>The diagnostic that detects when the value of a block output signal is Inf or NaN at the current time step is set to none or warning. When this type of block output signal condition occurs, numeric exceptions can result, and numeric exceptions are not acceptable in safety-related applications. (See DO-178B, Section 6.3.1g – Algorithms are accurate, DO-178B, Section 6.3.2g – Algorithms are accurate, and MISRA-C:2004, Rule 21.1.)</p> | <p>Set <b>Inf or NaN block output</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>SignalInfNanChecking</code> to error.</p>        |

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Recommended Action                                                                                                                                                                                                    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic that detects Simulink object names that begin with <code>rt</code> is set to <code>none</code> or <code>warning</code>. This diagnostic prevents name clashes with generated signal names that have an <code>rt</code> prefix. (See DO-178B, Section 6.3.1e – High-level requirements conform to standards, and DO-178B, Section 6.3.2e – Low-level requirements conform to standards.)</p>                                               | <p>Set "<b>rt</b>" prefix for identifiers on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>RTPrefix</code> to <code>error</code>.</p>          |
| <p>The diagnostic that detects simulation range checking is set to <code>none</code> or <code>warning</code>. This diagnostic detects when signals exceed their specified ranges during simulation. Simulink compares the signal values that a block outputs with the specified range and the block data type. (See DO-178B, Section 6.3.1g – Algorithms are accurate, DO-178B, Section 6.3.2g – Algorithms are accurate, and MISRA-C:2004, Rule 21.1.)</p> | <p>Set <b>Simulation range checking</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>SignalRangeChecking</code> to <code>error</code>.</p> |

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to signal data and that can impact safety.

### See Also

- Diagnostics Pane: Data Validity in the Simulink graphical user interface documentation
- Diagnosing Simulation Errors in the Simulink documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related diagnostic settings for parameters

Check model configuration for diagnostic settings that apply to parameters and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to parameters are set optimally for generating code for a safety-related application.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                           | Recommended Action                                                                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic that detects when a parameter downcast occurs is set to none or warning. A downcast to a lower signal range can result in numeric overflows of parameters, resulting in unexpected behavior. (See DO-178B, Section 6.3.1g – Algorithms are accurate, DO-178B, Section 6.3.2g – Algorithms are accurate, and MISRA-C:2004, Rule 21.1.)</p>                                                                         | <p>Set <b>Detect downcast</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter ParameterDowncastMsg to error.</p>   |
| <p>The diagnostic that detects when a parameter underflow occurs is set to none or warning. When the data type of a parameter does not have sufficient resolution, the parameter value is zero instead of the specified value. This can lead to incorrect operation of generated code. (See DO-178B, Section 6.3.1g – Algorithms are accurate, DO-178B, Section 6.3.2g – Algorithms are accurate, and MISRA-C:2004, Rule 21.1.)</p> | <p>Set <b>Detect underflow</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter ParameterUnderflowMsg to error.</p> |

| <b>Condition</b>                                                                                                                                                                                                                                                                                                                                                             | <b>Recommended Action</b>                                                                                                                                                                                       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic that detects when a parameter overflow occurs is set to none or warning. Numeric overflows can result in unexpected application behavior and should be detected and corrected in safety-related applications. (See DO-178B, Section 6.3.1g – Algorithms are accurate, DO-178B, Section 6.3.2g – Algorithms are accurate, and MISRA-C:2004, Rule 21.1.)</p> | <p>Set <b>Detect overflow</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>ParameterOverflowMsg</code> to error.</p>                 |
| <p>The diagnostic that detects when a parameter loses precision is set to none or warning. Not detecting such errors can result in a parameter being set to an incorrect value in the generated code. (See DO-178B, Section 6.3.1g – Algorithms are accurate, DO-178B, Section 6.3.2g – Algorithms are accurate, and MISRA-C:2004, Rules 10.1, 10.2, 10.3, and 10.4.)</p>    | <p>Set <b>Detect precision loss</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>ParameterPrecisionLossMsg</code> to error.</p>      |
| <p>The diagnostic that detects when an expression with tunable variables is reduced to its numerical equivalent is set to none or warning. This can result in a tunable parameter unexpectedly not being tunable in generated code. (See DO-178B, Section 6.3.1g – Algorithms are accurate and DO-178B, Section 6.3.2g – Algorithms are accurate.)</p>                       | <p>Set <b>Detect loss of tunability</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>ParameterTunabilityLossMsg</code> to error.</p> |

**Action Results**

Clicking **Modify Settings** configures model diagnostic settings that apply to parameters and that can impact safety.

**See Also**

- Diagnostics Pane: Data Validity in the Simulink graphical user interface documentation

- Diagnosing Simulation Errors in the Simulink documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related diagnostic settings for data used for debugging

Check model configuration for diagnostic settings that apply to data used for debugging and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to debugging are set optimally for generating code for a safety-related application.

See

- DO-178B, Section 6.3.1e – High-level requirements conform to standards
- DO-178B and Section 6.3.2e – Low-level requirements conform to standards

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                    | Recommended Action                                                                                                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The diagnostic that enables model verification blocks is set to <code>Use local settings</code> or <code>Enable all</code> . Such blocks should be disabled because they are assertion blocks, which are for verification only. Model developers should not use assertions in embedded code. | Set <b>Model Verification block enabling</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>AssertControl</code> to <code>Disable All</code> . |

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to data used for debugging and that can impact safety.

### See Also

- Diagnostics Pane: Data Validity in the Simulink graphical user interface documentation
- Diagnosing Simulation Errors in the Simulink documentation



- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related diagnostic settings for data store memory

Check model configuration for diagnostic settings that apply to data store memory and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to data store memory are set optimally for generating code for a safety-related application.

See DO-178B, Section 6.3.3b – Software architecture is consistent.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                | Recommended Action                                                                                                                                                                                                                 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic that detects whether the model attempts to read data from a data store in which it has not stored data in the current time step is set to a value other than <code>Enable all as errors</code>. Reading data before it is written can result in use of stale data or data that is not initialized.</p> | <p>Set <b>Detect read before write</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>ReadBeforeWriteMsg</code> to <code>Enable all as errors</code>.</p> |
| <p>The diagnostic that detects whether the model attempts to store data in a data store, after previously reading data from it in the current time step, is set to a value other than <code>Enable all as errors</code>. Writing data after it is read can result in use of stale or incorrect data.</p>                 | <p>Set <b>Detect write after read</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>WriteAfterReadMsg</code> to <code>Enable all as errors</code>.</p>   |

| Condition                                                                                                                                                                                                                                                                               | Recommended Action                                                                                                                                                                                                           |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The diagnostic that detects whether the model attempts to store data in a data store twice in succession in the current time step is set to a value other than <code>Enable all as errors</code> . Writing data twice in one time step can result in unpredictable data.                | Set <b>Detect write after write</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>WriteAfterWriteMsg</code> to <code>Enable all as errors</code> . |
| The diagnostic that detects when one task reads data from a Data Store Memory block to which another task writes data is set to <code>none</code> or <code>warning</code> . Reading or writing data in different tasks in multitask mode can result in corrupted or unpredictable data. | Set <b>Multitask data store</b> on the <b>Diagnostics &gt; Data Validity</b> pane of the Configuration Parameters dialog box or set the parameter <code>MultiTaskDSMMsg</code> to <code>error</code> .                       |

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to data store memory and that can impact safety.

### See Also

- Diagnostics Pane: Data Validity in the Simulink graphical user interface documentation
- Diagnosing Simulation Errors in the Simulink documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related diagnostic settings for type conversions

Check model configuration for diagnostic settings that apply to type conversions and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to type conversions are set optimally for generating code for a safety-related application.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Recommended Action                                                                                                                                                                                                          |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic that detects Data Type Conversion blocks used where no type conversion is necessary is set to none. The Simulink software might remove unnecessary Data Type Conversion blocks from generated code. This might result in requirements without corresponding code. The removal of such blocks need to be detected so model developers can remove the unnecessary blocks explicitly. (See DO-178B, Section 6.3.1g – Algorithms are accurate and DO-178B, Section 6.3.2g – Algorithms are accurate.)</p> | <p>Set <b>Unnecessary type conversions</b> on the <b>Diagnostics &gt; Type Conversion</b> pane of the Configuration Parameters dialog box or set the parameter <code>UnnecessaryDatatypeConvMsg</code> to warning.</p>      |
| <p>The diagnostic that detects vector-to-matrix or matrix-to-vector conversions at block inputs is set to none or warning. When the Simulink software automatically converts between vector and matrix dimensions, unintended operations or unpredictable behavior can occur. (See DO-178B, Section 6.3.1g – Algorithms are accurate and DO-178B, Section 6.3.2g – Algorithms are accurate.)</p>                                                                                                                        | <p>Set <b>Vector/matrix block input conversion</b> on the <b>Diagnostics &gt; Type Conversion</b> pane of the Configuration Parameters dialog box or set the parameter <code>VectorMatrixConversionMsg</code> to error.</p> |

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                    | Recommended Action                                                                                                                                                                                                        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic that detects when a 32-bit integer value is converted to a floating-point value is set to none. This type of conversion can result in a loss of precision due to truncation of the least significant bits for large integer values. (See DO-178B, Section 6.3.1g – Algorithms are accurate and DO-178B, Section 6.3.2g – Algorithms are accurate, and MISRA C 2004, Rules 10.1, 10.2, 10.3, and 10.4.)</p> | <p>Set <b>32-bit integer to single precision float conversion</b> on the <b>Diagnostics &gt; Type Conversion</b> pane of the Configuration Parameters dialog box or set the parameter Int32ToFloatConvMsg to warning.</p> |

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to type conversions and that can impact safety.

### See Also

- Diagnostics Pane: Type Conversion in the Simulink graphical user interface documentation
- Data Type Conversion block in the Simulink reference documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related diagnostic settings for signal connectivity

Check model configuration for diagnostic settings that apply to signal connectivity and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to signal connectivity are set optimally for generating code for a safety-related application.

See

- DO-178B, Section 6.3.1e – High-level requirements conform to standards
- DO-178B, Section 6.3.2e – Low-level requirements conform to standards

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                              | Recommended Action                                                                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic that detects virtual signals that have a common source signal but different labels is set to none or warning. This diagnostic pertains to virtual signals only and has no effect on generated code. However, signal label mismatches can lead to confusion during model reviews.</p> | <p>Set <b>Signal label mismatch</b> on the <b>Diagnostics &gt; Connectivity</b> pane of the Configuration Parameters dialog box or set the parameter <code>SignalLabelMismatchMsg</code> to error.</p>      |
| <p>The diagnostic that detects when the model contains a block with an unconnected input signal is set to none or warning. This must be detected because code is not generated for unconnected block inputs.</p>                                                                                       | <p>Set <b>Unconnected block input ports</b> on the <b>Diagnostics &gt; Connectivity</b> pane of the Configuration Parameters dialog box or set the parameter <code>UnconnectedInputMsg</code> to error.</p> |

| Condition                                                                                                                                                                                                          | Recommended Action                                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The diagnostic that detects when the model contains a block with an unconnected output signal is set to none or warning. This must be detected because dead code can result from unconnected block output signals. | Set <b>Unconnected block output ports</b> on the <b>Diagnostics &gt; Connectivity</b> pane of the Configuration Parameters dialog box or set the parameter UnconnectedOutputMsg to error. |
| The diagnostic that detects unconnected signal lines and unmatched Goto or From blocks is set to none or warning. This error must be detected because code is not generated for unconnected lines.                 | Set <b>Unconnected line</b> on the <b>Diagnostics &gt; Connectivity</b> pane of the Configuration Parameters dialog box or set the parameter UnconnectedLineMsg to error.                 |

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to signal connectivity and that can impact safety.

### See Also

- Diagnostics Pane: Connectivity in the Simulink graphical user interface documentation
- Signal Basics in the Simulink documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related diagnostic settings for bus connectivity

Check model configuration for diagnostic settings that apply to bus connectivity and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to bus connectivity are set optimally for generating code for a safety-related application.

See DO-178B, Section 6.3.3b – Software architecture is consistent.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                               | Recommended Action                                                                                                                                                                                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The diagnostic that detects whether a Model block's root Output block is connected to a bus but does not specify a bus object is set to none or warning. For a bus signal to cross a model boundary, the signal must be defined as a bus object to ensure compatibility with higher level models that use a model as a reference model. | Set <b>Unspecified bus object at root Output block</b> on the <b>Diagnostics &gt; Connectivity</b> pane of the Configuration Parameters dialog box or set the parameter <code>RootOutputRequireBusObject</code> to error.                                                                          |
| The diagnostic that detects whether the name of a bus element matches the name specified by the corresponding bus object is set to none or warning. This diagnostic prevents the use of incompatible buses in a bus-capable block such that the output names are inconsistent.                                                          | Set <b>Element name mismatch</b> on the <b>Diagnostics &gt; Connectivity</b> pane of the Configuration Parameters dialog box or set the parameter <code>BusObjectLabelMismatch</code> to error.                                                                                                    |
| The diagnostic that detects when some blocks treat a signal as a mux/vector, while other blocks treat the signal as a bus, is set to none or warning. When the Simulink software automatically converts a muxed signal to a bus, it is possible for                                                                                     | <ul style="list-style-type: none"> <li>Set <b>Mux blocks used to create bus signals</b> on the <b>Diagnostics &gt; Connectivity</b> pane of the Configuration Parameters dialog box to error, or set the parameter <code>StrictBusMsg</code> to <code>ErrorOnBusTreatedAsVector</code>.</li> </ul> |



| Condition                                                          | Recommended Action                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>an unintended operation or unpredictable behavior to occur.</p> | <ul style="list-style-type: none"> <li>Set “Bus signal treated as vector” on the <b>Diagnostics &gt; Connectivity</b> pane of the Configuration Parameters dialog box to error, or the parameter <code>StrictBusMsg</code> to <code>ErrorOnBusTreatedAsVector</code>.</li> </ul> <p>You can use the Model Advisor or the <code>sl_replace_mux</code> utility function to replace all Mux blocks used as bus creators with a Bus Creator block.</p> |

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to bus connectivity and that can impact safety.

### See Also

- Diagnostics Pane: Connectivity in the Simulink graphical user interface documentation
- `Simulink.Bus` in the Simulink reference documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related diagnostic settings that apply to function-call connectivity

Check model configuration for diagnostic settings that apply to function-call connectivity and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to function-call connectivity are set optimally for generating code for a safety-related application.

DO-178B, Section 6.3.3b – Software architecture is consistent

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                                                                                | Recommended Action                                                                                                                                                                                                        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The diagnostic that detects incorrect use of a function-call subsystem is set to none or warning. If this condition is undetected, incorrect code might be generated.                                                                                                                                                                                                                    | Set <b>Invalid function-call connection</b> on the <b>Diagnostics &gt; Connectivity</b> pane of the Configuration Parameters dialog box or set the parameter <code>InvalidFcnCallConMsg</code> to error.                  |
| The diagnostic that specifies whether the Simulink software has to compute inputs of a function-call subsystem directly or indirectly while executing the subsystem is set to <code>Use local settings</code> or <code>Disable all</code> . This diagnostic detects unpredictable data coupling between a function-call subsystem and the inputs of the subsystem in the generated code. | Set <b>Context-dependent inputs</b> on the <b>Diagnostics &gt; Connectivity</b> pane of the Configuration Parameters dialog box or set the parameter <code>FcnCallInpInsideContextMsg</code> to <code>Enable all</code> . |

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to function-call connectivity and that can impact safety.

**See Also**

- Diagnostics Pane: Connectivity in the Simulink graphical user interface documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related diagnostic settings for compatibility

Check model configuration for diagnostic settings that affect compatibility and that might impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to compatibility are set optimally for generating code for a safety-related application.

See DO-178B, Section 6.3.3b – Software architecture is consistent and MISRA-C:2004, Rule 9.1.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                           | Recommended Action                                                                                                                                                                                         |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic that detects when a block has not been upgraded to use features of the current release is set to none or warning. An S-function written for an earlier version might not be compatible with the current version and generated code could operate incorrectly.</p> | <p>Set <b>S-function upgrades needed</b> on the <b>Diagnostics &gt; Compatibility</b> pane of the Configuration Parameters dialog box or set the parameter <code>SFcnCompatibilityMsg</code> to error.</p> |

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that affect compatibility and that might impact safety.

### See Also

- Diagnosing Simulation Errors in the Simulink documentation
- Diagnostics Pane: Compatibility in the Simulink graphical user interface documentation

- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related diagnostic settings for model initialization

In the model configuration, check diagnostic settings that affect model initialization and might impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to initialization are set optimally for generating code for a safety-related application.

See DO-178B, Section 6.3.3b – Software architecture is consistent and MISRA C® 2004, Rule 9.1.

### Analysis Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Recommended Action                                                                                                                                                                                                                                  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The <b>Check undefined subsystem initial output</b> diagnostic is cleared. This diagnostic specifies whether the Simulink software displays a warning if the model contains a conditionally executed subsystem, in which a block with a specified initial condition drives an Outport block with an undefined initial condition. A conditionally executed subsystem could have an output that is not initialized. If undetected, this condition can produce behavior that is nondeterministic.</p> | <p>In the Configuration Parameters dialog box, on the <b>Diagnostics &gt; Data Validity</b> pane, select <b>Check undefined subsystem initial output</b> or set the parameter <code>CheckSSInitialOutputMsg</code> to on.</p>                       |
| <p>The diagnostic that detects potential initial output differences from earlier releases is cleared. A conditionally executed subsystem could have an output that is not initialized. If undetected, this condition can produce behavior that is nondeterministic.</p>                                                                                                                                                                                                                               | <p>In the Configuration Parameters dialog box, on the <b>Diagnostics &gt; Compatibility</b> pane, select <b>Check preactivation output of execution context</b> or set the parameter <code>CheckExecutionContextPreStartOutputMsg</code> to on.</p> |
| <p>The diagnostic that detects potential output differences from earlier releases is cleared. A conditionally executed subsystem could have an output that is not initialized and feeds into a</p>                                                                                                                                                                                                                                                                                                    | <p>In the Configuration Parameters dialog box, on the <b>Diagnostics &gt; Compatibility</b> pane, select <b>Check runtime output of execution context</b> or set the parameter</p>                                                                  |

| Condition                                                                                                                            | Recommended Action                           |
|--------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|
| block with a tunable parameter. If undetected, this condition can cause the behavior of the downstream block to be nondeterministic. | CheckExecutionContextRuntimeOutputMsg to on. |

### Action Results

To configure the diagnostic settings that affect model initialization and that might impact safety, click **Modify Settings**.

### See Also

- “Diagnosing Simulation Errors” in the Simulink documentation
- “Diagnostics Pane: Data Validity” in the Simulink graphical user interface documentation
- For information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard, Radio Technical Commission for Aeronautics (RTCA)

## Check safety-related diagnostic settings for model referencing

Check model configuration for diagnostic settings that apply to model referencing and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to model referencing are set optimally for generating code for a safety-related application.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Recommended Action                                                                                                                                                                                                              |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The diagnostic that detects a mismatch between the version of the model that creates or refreshes a Model block and the current version of the referenced model is set to error or warning. The detection occurs during load and update operations. When you get the latest version of the referenced model from the software configuration management system, rather than an older version that was used in a previous simulation, if this diagnostic is set to error, the simulation is aborted. If the diagnostic is set to warning, a warning message is issued. To resolve the issue, the user must resave the model being simulated, which may not be the desired action. (See DO-178B, Section 6.3.3b – Software architecture is consistent.)</p> | <p>Set <b>Model block version mismatch</b> on the <b>Diagnostics &gt; Model Referencing</b> pane of the Configuration Parameters dialog box or set the parameter <code>ModelReferenceVersionMismatchMessage</code> to none.</p> |
| <p>The diagnostic that detects port and parameter mismatches during model loading and updating is set to none or warning. If undetected, such mismatches can lead to incorrect simulation results because the parent and referenced models have different</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | <p>Set <b>Port and parameter mismatch</b> on the <b>Diagnostics &gt; Model Referencing</b> pane of the Configuration Parameters dialog box or set the parameter <code>ModelReferenceIOMismatchMessage</code> to error.</p>      |



| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Recommended Action                                                                                                                                                                                                                                                      |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>interfaces. (See DO-178B, Section 6.3.3b – Software architecture is consistent.)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                                         |
| <p>The <b>Model configuration mismatch</b> diagnostic is set to none or error. This diagnostic checks whether the configuration parameters of a model referenced by the current model match the current model's configuration parameters or are inappropriate for a referenced model. Some diagnostics for referenced models are not supported in simulation mode. Setting this diagnostic to error can prevent simulations from running. Some differences in configurations can lead to incorrect simulation results and mismatches between simulation and target code generation. (See DO-178B, Section 6.3.3b – Software architecture is consistent.)</p> | <p>Set <b>Model configuration mismatch</b> on the <b>Diagnostics &gt; Model Referencing</b> pane of the Configuration Parameters dialog box or set the parameter <code>ModelReferenceCSMismatchMessage</code> to warning.</p>                                           |
| <p>The diagnostic that detects invalid internal connections to the current model's root-level Inport and Outport blocks is set to none or warning. When this condition is detected, the Simulink software might automatically insert hidden blocks into the model to correct the condition. The hidden blocks can result in generated code that has no traceable requirements. Setting the diagnostic to error forces model developers to correct the referenced models manually. (See DO-178B, Section 6.3.3b – Software architecture is consistent.)</p>                                                                                                   | <p>Set <b>Invalid root Inport/Outport block connection</b> on the <b>Diagnostics &gt; Model Referencing</b> pane of the Configuration Parameters dialog box or set the parameter <code>ModelReferenceIOMessage</code> to error.</p>                                     |
| <p>The diagnostic that detects whether To Workspace or Scope blocks are logging data in a referenced model is set to none or warning. Data logging is not supported for To Workspace and Scope blocks in referenced models. (See DO-178B, Section 6.3.1d –</p>                                                                                                                                                                                                                                                                                                                                                                                               | <p>Set <b>Unsupported data logging</b> on the <b>Diagnostics &gt; Model Referencing</b> pane of the Configuration Parameters dialog box or set the parameter <code>ModelReferenceDataLoggingMessage</code> to error.<br/>To log data, remove the blocks and log the</p> |

| <b>Condition</b>                                                                                             | <b>Recommended Action</b>                                                               |
|--------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| High-level requirements are verifiable and DO-178B, Section 6.3.2d – Low-level requirements are verifiable.) | referenced model signals. For more information, see “Logging Referenced Model Signals”. |

### **Action Results**

Clicking **Modify Settings** configures model diagnostic settings that apply to model referencing and that can impact safety.

### **See Also**

- Diagnosing Simulation Errors in the Simulink documentation
- Diagnostics Pane: Model Referencing in the Simulink graphical user interface documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard
- “Logging Referenced Model Signals” in the Simulink documentation

## Check safety-related model referencing settings

Check model configuration for model referencing settings that can impact safety.

### Description

This check verifies that model configuration parameters for model referencing are set optimally for generating code for a safety-related application.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Recommended Action                                                                                                                                                                                                                                 |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The referenced model is configured such that its target is rebuilt whenever you update, simulate, or generate code for the model, or if the Simulink software detects any changes in known dependencies. These configuration settings can result in unnecessary regeneration of the code, resulting in changing only the date of the file and slowing down the build process when using model references. (See DO-178B, Section 6.3.1b – High-level requirements are accurate and consistent and DO-178B, Section 6.3.2b – Low-level requirements are accurate and consistent.)</p> | <p>Set “Rebuild options” on the <b>Model Referencing</b> pane of the Configuration Parameters dialog box or set the parameter <code>UpdateModelReferenceTargets</code> to <code>Never</code> or <code>If</code> any changes detected.</p>          |
| <p>The diagnostic that detects whether a target needs to be rebuilt is set to <code>None</code> or <code>Warn</code> if targets require rebuild. For safety-related applications, an error should alert model developers that the parent and referenced models are inconsistent. This diagnostic parameter is available only if <b>Rebuild options</b> is set to <code>Never</code>. (See DO-178B, Section 6.3.1b – High-level requirements are accurate and consistent and DO-178B, Section 6.3.2b – Low-level requirements are accurate and consistent.)</p>                         | <p>Set “Never rebuild targets diagnostic” on the <b>Model Referencing</b> pane of the Configuration Parameters dialog box or set the parameter <code>CheckModelReferenceTargetMessage</code> to <code>Error</code> if targets require rebuild.</p> |

| <b>Condition</b>                                                                                                                                                                                                                                                               | <b>Recommended Action</b>                                                                                                                                                                                                                         |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The ability to pass scalar root input by value is on. This capability should be off because scalar values can change during a time step and result in unpredictable data. (See DO-178B, Section 6.3.3b – Software architecture is consistent.)</p>                          | <p>Set “Pass fixed-size scalar root inputs by value for Real-Time Workshop” on the <b>Model Referencing</b> pane of the Configuration Parameters dialog box or set the parameter <code>ModelReferencePassRootInputsByReference</code> to off.</p> |
| <p>The model is configured to minimize algebraic loop occurrences. This configuration is incompatible with the recommended setting of <b>Single output/update function</b> for embedded systems code. (See DO-178B, Section 6.3.3b – Software architecture is consistent.)</p> | <p>Set “Minimize algebraic loop occurrences” on the <b>Model Referencing</b> pane of the Configuration Parameters dialog box or set the parameter <code>ModelReferenceMinAlgLoopOccurrences</code> to off.</p>                                    |

**Action Results**

Clicking **Modify Settings** configures model referencing settings that can impact safety.

**See Also**

- Model Dependencies in the Simulink documentation
- Model Referencing Pane in the Simulink graphical user interface documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related code generation settings

Check model configuration for code generation settings that can impact safety.

### Description

This check verifies that model configuration parameters for code generation are set optimally for a safety-related application.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                             | Recommended Action                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The option to include comments in the generated code is cleared. Comments are necessary for good traceability between the code and the model. (See DO-178B, Section 6.3.4e – Source code is traceable to low-level requirements.)                                     | Select <b>Include comments</b> on the <b>Real-Time Workshop &gt; Comments</b> pane of the Configuration Parameters dialog box or set the parameter <code>GenerateComments</code> to on.                                |
| The option to include comments that describe the code for blocks is cleared. Comments are necessary for good traceability between the code and the model. (See DO-178B, Section 6.3.4e – Source code is traceable to low-level requirements.)                         | Select <b>Simulink block / Stateflow object comments</b> on the <b>Real-Time Workshop &gt; Comments</b> pane of the Configuration Parameters dialog box or set the parameter <code>SimulinkBlockComments</code> to on. |
| The option to include comments that describe the code for blocks eliminated from a model is cleared. Comments are necessary for good traceability between the code and the model. (See DO-178B, Section 6.3.4e – Source code is traceable to low-level requirements.) | Select <b>Show eliminated blocks</b> on the <b>Real-Time Workshop &gt; Comments</b> pane of the Configuration Parameters dialog box or set the parameter <code>ShowEliminatedStatement</code> to on.                   |

| Condition                                                                                                                                                                                                                                                                                                                                                                     | Recommended Action                                                                                                                                                                                                                     |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The option to include the names of parameter variables and source blocks as comments in the model parameter structure declaration in <i>model_prm.h</i> is cleared. Comments are necessary for good traceability between the code and the model. (See DO-178B, Section 6.3.4e – Source code is traceable to low-level requirements.)</p>                                   | <p>Select <b>Verbose comments for SimulinkGlobal storage class</b> on the <b>Real-Time Workshop &gt; Comments</b> pane of the Configuration Parameters dialog box or set the parameter <code>ForceParamTrailComments</code> to on.</p> |
| <p>The option to include requirement descriptions assigned to Simulink blocks as comments is cleared. Comments are necessary for good traceability between the code and the model. (See DO-178B, Section 6.3.4e – Source code is traceable to low-level requirements.)</p>                                                                                                    | <p>Select <b>Requirements in block comments</b> on the <b>Real-Time Workshop &gt; Comments</b> pane of the Configuration Parameters dialog box or set the parameter <code>ReqsInCode</code> to on.</p>                                 |
| <p>The option to generate nonfinite data and operations is selected. Support for nonfinite numbers is inappropriate for real-time embedded systems. (See DO-178B, Section 6.3.1c – High-level requirements are compatible with target computer and DO-178B, Section 6.3.2c – Low-level requirements are compatible with target computer.)</p>                                 | <p>Clear <b>Support: non-finite numbers</b> on the <b>Real-Time Workshop &gt; Interface</b> pane of the Configuration Parameters dialog box or set the parameter <code>SupportNonFinite</code> to off.</p>                             |
| <p>The option to generate and maintain integer counters for absolute and elapsed time is selected. Support for absolute time is inappropriate for real-time safety-related systems. (See DO-178B, Section 6.3.1c – High-level requirements are compatible with target computer and DO-178B, Section 6.3.2c – Low-level requirements are compatible with target computer.)</p> | <p>Clear <b>Support: absolute time</b> on the <b>Real-Time Workshop &gt; Interface</b> pane of the Configuration Parameters dialog box or set the parameter <code>SupportAbsoluteTime</code> to off.</p>                               |

| Condition                                                                                                                                                                                                                                                                                                                                                                          | Recommended Action                                                                                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The option to generate code for blocks that use continuous time is selected. Support for continuous time is inappropriate for real-time safety-related systems. (See DO-178B, Section 6.3.1c – High-level requirements are compatible with target computer and DO-178B, Section 6.3.2c – Low-level requirements are compatible with target computer.)</p>                       | <p>Clear <b>Support: continuous time</b> on the <b>Real-Time Workshop &gt; Interface</b> pane of the Configuration Parameters dialog box or set the parameter <code>SupportContinuousTime</code> to off.</p>          |
| <p>The option to generate code for noninlined S-functions is selected. This option requires support of nonfinite numbers, which is inappropriate for real-time safety-related systems. (See DO-178B, Section 6.3.1c – High-level requirements are compatible with target computer and DO-178B, Section 6.3.2c – Low-level requirements are compatible with target computer.)</p>   | <p>Clear <b>Support: non-inlined S-functions</b> on the <b>Real-Time Workshop &gt; Interface</b> pane of the Configuration Parameters dialog box or set the parameter <code>SupportNonInlinedSFcns</code> to off.</p> |
| <p>The option to generate model function calls compatible with the main program module of the GRT target is selected. This option is inappropriate for real-time safety-related systems. (See DO-178B, Section 6.3.1c – High-level requirements are compatible with target computer and DO-178B, Section 6.3.2c – Low-level requirements are compatible with target computer.)</p> | <p>Clear <b>GRT compatible call interface</b> on the <b>Real-Time Workshop &gt; Interface</b> pane of the Configuration Parameters dialog box or set the parameter <code>GRTInterface</code> to off.</p>              |

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                     | Recommended Action                                                                                                                                                                                                                        |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The option to generate the <i>model_update</i> function is cleared. Having a single call to the output and update functions simplifies the interface to the real-time operating system (RTOS) and simplifies verification of the generated code. (See DO-178B, Section 6.3.1c – High-level requirements are compatible with target computer and DO-178B, Section 6.3.2c – Low-level requirements are compatible with target computer.)</p> | <p>Select <b>Single output/update function</b> on the <b>Real-Time Workshop &gt; Interface</b> pane of the Configuration Parameters dialog box or set the parameter <code>CombineOutputUpdateFcns</code> to on.</p>                       |
| <p>The option to generate the <i>model_terminate</i> function is selected. This function deallocates dynamic memory, which is not appropriate for real-time safety-related systems. (See DO-178B, Section 6.3.1c – High-level requirements are compatible with target computer and DO-178B, Section 6.3.2c – Low-level requirements are compatible with target computer.)</p>                                                                 | <p>Clear <b>Terminate function required</b> on the <b>Real-Time Workshop &gt; Interface</b> pane of the Configuration Parameters dialog box or set the parameter <code>IncludeMdlTerminateFcn</code> to off.</p>                          |
| <p>The option to log or monitor error status is cleared. If you do not select this option, the Real-Time Workshop product generates extra code that might not be reachable for testing. (See DO-178B, Section 6.3.1c – High-level requirements are compatible with target computer and DO-178B, Section 6.3.2c – Low-level requirements are compatible with target computer.)</p>                                                             | <p>Select <b>Suppress error status in real-time model data structure</b> on the <b>Real-Time Workshop &gt; Interface</b> pane of the Configuration Parameters dialog box or set the parameter <code>SuppressErrorStatus</code> to on.</p> |



| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Recommended Action                                                                                                                                                                                                                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>MAT-file logging is selected. This option adds extra code for logging test points to a MAT-file, which is not supported by embedded targets. Use this option only in test harnesses. (See DO-178B, Section 6.3.1c – High-level requirements are compatible with target computer and DO-178B, Section 6.3.2c – Low-level requirements are compatible with target computer.)</p>                                                                                                            | <p>Clear <b>MAT-file logging</b> on the <b>Real-Time Workshop &gt; Interface</b> pane of the Configuration Parameters dialog box or set the parameter <code>MatFileLogging</code> to off.</p>                                                    |
| <p>The option that specifies the style for parenthesis usage is set to <code>Minimum (Rely on C/C++ operators precedence)</code> or to <code>Nominal (Optimize for readability)</code>. For safety-related applications, explicitly specify precedence with parentheses. (See DO-178B, Section 6.3.1c – High-level requirements are compatible with target computer, DO-178B, Section 6.3.2c – Low-level requirements are compatible with target computer, and MISRA-C:2004, Rule 12.1.)</p> | <p>Set <b>Parenthesis level</b> on the <b>Real-Time Workshop &gt; Code</b> pane of the Configuration Parameters dialog box or set the parameter <code>ParenthesesLevel</code> to <code>Maximum (Specify precedence with parentheses)</code>.</p> |
| <p>The option that specifies whether to preserve operand order is cleared. This option increases the traceability of the generated code. (See DO-178B, Section 6.3.4e – Source code is traceable to low-level requirements.)</p>                                                                                                                                                                                                                                                             | <p>Select <b>Preserve operand order in expression</b> on the <b>Real-Time Workshop &gt; Code</b> pane of the Configuration Parameters dialog box or set the parameter <code>PreserveExpressionOrder</code> to on.</p>                            |

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Recommended Action                                                                                                                                                                                                         |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The option that specifies whether to preserve empty primary condition expressions in <code>if</code> statements is cleared. This option increases the traceability of the generated code. ( See DO-178B, Section 6.3.4e – Source code is traceable to low-level requirements.)</p>                                                                                                                                                                                         | <p>Select <b>Preserve condition expression in if statement</b> on the <b>Real-Time Workshop &gt; Code</b> pane of the Configuration Parameters dialog box or set the parameter <code>PreserveIfCondition</code> to on.</p> |
| <p>The minimum number of characters specified for generating name mangling strings is less than four. You can use this option to minimize the likelihood that parameter and signal names will change during code generation when the model changes. Use of this option assists with minimizing code differences between file versions, decreasing the effort to perform code reviews. (See DO-178B, Section 6.3.4e – Source code is traceable to low-level requirements.)</p> | <p>Set <b>Minimum mangle length</b> on the <b>Real-Time Workshop &gt; Symbols</b> pane of the Configuration Parameters dialog box or the parameter <code>MangleLength</code> to a value of 4 or greater.</p>               |

### Action Results

Clicking **Modify Settings** configures model code generation settings that can impact safety.

### Limitations

This check requires a Real-Time Workshop Embedded Coder license and an ERT-based system target file.

### See Also

- Real-Time Workshop Pane: Comments in the Real-Time Workshop reference documentation
- Real-Time Workshop Pane: Symbols in the Real-Time Workshop reference documentation

- Real-Time Workshop Pane: Interface in the Real-Time Workshop reference documentation
- Real-Time Workshop Pane: Code Style in the Real-Time Workshop Embedded Coder reference documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check safety-related diagnostic settings for saving

Check model configuration for diagnostic settings that apply to saving model files

### Description

This check verifies that model configuration parameters are set optimally for saving a model for a safety-related application.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                | Recommended Action                                                                                                                                                                                                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The diagnostic that detects whether a model contains disabled library links before the model is saved is set to none or warning. If this condition is undetected, incorrect code might be generated. (See DO-178B, Section 6.3.3b - Software architecture is consistent.)                                | Set <b>Block diagram contains disabled library links</b> on the <b>Diagnostics &gt; Saving&gt;</b> pane of the Configuration Parameters dialog box or set the parameter <code>SaveWithDisabledLinkMsg</code> to error.           |
| The diagnostic that detects whether a model contains library links that are using parameters not in a mask before the model is saved is set to none or warning. If this condition is undetected, incorrect code might be generated. (See DO-178B, Section 6.3.3b - Software architecture is consistent.) | Set <b>Block diagram contains parameterized library links</b> on the <b>Diagnostics &gt; Saving&gt;</b> pane of the Configuration Parameters dialog box or set the parameter <code>SaveWithParameterizedLinkMsg</code> to error. |

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to saving a model file.

### See Also

- Disabling Links to Library Blocks in the Simulink documentation
- Identifying Disabled Library Links in the Simulink documentation
- Saving a Model in the Simulink documentation

- Model Parameters in the Simulink documentation
- Diagnostics Pane: Saving in the Simulink documentation

## Check for model objects that do not link to requirements

Check whether Simulink blocks and Stateflow objects link to a requirements document.

### Description

This check verifies whether Simulink blocks and Stateflow objects link to a document containing engineering requirements for traceability.

### Analysis Results and Recommended Actions

| Condition                                                                                                                                                                                                      | Recommended Action                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| Blocks do not link to a requirements document. (See DO-178B, Section 6.3.1f - High-level requirements trace to system requirements, Section 6.3.2f - Low-level requirements trace to high-level requirements.) | Link to requirements document. See Chapter 4, “Creating and Managing Requirements Links”. |

### Limitations

When you run this check, the Model Advisor does not follow library links or look under masks. The Model Advisor reviews all top-level blocks in the system.

### Tip

Run this check from the top model or subsystem that you want to check.

### See Also

Requirements Linking and Traceability on page 1

## Check for proper usage of Math blocks

Check whether math operators require nonfinite number support.

### Description

This check verifies that Math Function blocks do not use math operations that need nonfinite number support with real-time embedded targets.

### Analysis Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                        | Recommended Action                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Math Function blocks using <code>log</code> (natural logarithm), <code>log10</code> (base 10 logarithm), and <code>rem</code> (Remainder) operators that require nonfinite number support. (See DO-178B, Section 6.3.1g - Algorithms are accurate, Section 6.3.2g - Algorithms are accurate, and MISRA-C:2004, Rule 21.1)</p> | <p>When using the Math Function block with a <code>log</code> or <code>log10</code> function, you must protect the input to the block in the model such that it is not less than or equal to zero. Otherwise, the output can produce a NaN or <code>-Inf</code> and result in a run-time error in the generated code.</p> <p>When using the Math Function block with a <code>rem</code> function, you must protect the second input to the block such that it is not equal to zero. Otherwise the output can produce a <code>Inf</code> or <code>-Inf</code> and result in a run-time error in the generated code.</p> |

### Tips

With embedded systems, you must take care when using blocks that could produce nonfinite outputs such as NaN, `Inf` or `-Inf`. Your design must protect the inputs to these blocks in order to avoid run-time errors in the embedded system.

### See Also

Math Function block in the Simulink documentation

## Check state machine type of Stateflow charts

Identify whether Stateflow charts are all Mealy or all Moore charts.

### Description

Compares the state machine type of all Stateflow charts to the type that you specify in the input parameters.

See hisf\_0001: Mealy and Moore Semantics and DO-178B, sections 6.3.1b: High-level requirements are accurate and consistent, 6.3.1e: High-level requirements conform to standards, 6.3.2b: Low-level requirements are accurate and consistent, 6.3.2e: Low-level requirements conform to standards, 6.3.3b: Software architecture is consistent, and 6.3.3e: Software architecture conform to standards

### Input Parameters

#### Common

Check whether charts use the same state machine type, and are all Mealy or all Moore charts.

#### Mealy

Check whether all charts are Mealy charts.

#### Moore

Check whether all charts are Moore charts.

### Analysis Results and Recommended Actions

| Condition                                                                                                                                                                                                                      | Recommended Action                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The input parameter is set to <b>Common</b> and charts in the model use any of the following:</p> <ul style="list-style-type: none"> <li>• Classic state machine types.</li> <li>• Multiple state machine types.</li> </ul> | <p>For each chart, in the Chart Properties dialog box, specify <b>State Machine Type</b> to either <b>Mealy</b> or <b>Moore</b>. Use the same state machine type for all charts in the model.</p> |



| <b>Condition</b>                                                                                  | <b>Recommended Action</b>                                                                               |
|---------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| The input parameter is set to <b>Mealy</b> and charts in the model use other state machine types. | For each chart, in the Chart Properties dialog box, specify <b>State Machine Type</b> to <b>Mealy</b> . |
| The input parameter is set to <b>Moore</b> and charts in the model use other state machine types. | For each chart, in the Chart Properties dialog box, specify <b>State Machine Type</b> to <b>Moore</b> . |

### See Also

- “Building Mealy and Moore Charts” in the Stateflow documentation.
- “Stateflow Chart Considerations” in the Simulink documentation.

## Check Stateflow charts for ordering of states and transitions

Identify Stateflow charts that have **User specified state/transition execution order** cleared.

### Description

Identify Stateflow charts that have **User specified state/transition execution order** cleared, and therefore do not use explicit ordering of parallel states and transitions.

### Analysis Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                            | Recommended Action                                                                                                            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| Stateflow charts have <b>User specified state/transition execution order</b> cleared.<br>(See hisf_0002: User Specified State/Transition Execution Order and DO-178B, sections 6.3.3b - Software architecture is consistent and 6.3.3e - Software architecture conform to standards) | For the specified charts, in the Chart Properties dialog box, select <b>User specified state/transition execution order</b> . |

### Action Results

Clicking **Modify** selects **User specified state/transition execution order** for the specified charts.

### See Also

- “Transition Testing Order in Multilevel State Hierarchy” in the Stateflow documentation.
- “Execution Order for Parallel States” in the Stateflow documentation.
- “Stateflow Chart Considerations” in the Simulink documentation.

## Check Stateflow debugging settings

Identify whether Stateflow debugging options are cleared.

### Description

Identify whether the following debugging options are cleared, which might lead to unreachable code and indeterminate execution time:

- **Enable debugging/animation**
- **Enable overflow detection (with debugging)**
- **State Inconsistency**
- **Transition Conflict**
- **Data Range**
- **Detect Cycles**

### Analysis Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Recommended Action                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Any of the following debugging options are cleared:</p> <ul style="list-style-type: none"> <li>• <b>Enable debugging/animation</b></li> <li>• <b>Enable overflow detection (with debugging)</b></li> <li>• <b>State Inconsistency</b></li> <li>• <b>Transition Conflict</b></li> <li>• <b>Data Range</b></li> <li>• <b>Detect Cycles</b></li> </ul> <p>(See hisf_0011: Stateflow Debugging Settings and DO-178B, sections 6.3.1b : High-level requirements are accurate and consistent, 6.3.1e : High-level requirements conform to standards, 6.3.2b : Low-level</p> | <p>Select the debugging options. In the Configuration Parameters dialog box, select:</p> <ul style="list-style-type: none"> <li>• <b>Simulation</b><br/><b>Target &gt; General &gt; Enable debugging/animation</b></li> <li>• <b>Simulation</b><br/><b>Target &gt; General &gt; Enable overflow detection (with debugging)</b></li> </ul> <p>In the Stateflow Debugging dialog box, select:</p> <ul style="list-style-type: none"> <li>• <b>State Inconsistency</b></li> <li>• <b>Transition Conflict</b></li> </ul> |

| <b>Condition</b>                                                                                    | <b>Recommended Action</b>                                                                          |
|-----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| requirements are accurate and consistent, and 6.3.2e : Low-level requirements conform to standards) | <ul style="list-style-type: none"><li>• <b>Data Range</b></li><li>• <b>Detect Cycles</b></li></ul> |

**Action Results**

Clicking **Modify** selects the specified debugging options.

**See Also**

“Stateflow Chart Considerations” in the Simulink documentation.

## Check for proper usage of For Iterator blocks

Check for For Iterator blocks that have variable loops.

### Description

This check verifies that a model does not use variable loops with For Iterator blocks.

See

- DO-178B Section 6.3.1e – High-level requirements conform to standards
- DO-178B Section 6.3.2e – Low-level requirements conform to standards
- MISRA-C:2004, Rule 13.6

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                          | Recommended Action                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The model combines the use of variable iteration values with a For Iterator block. The use of variable for loops can lead to unpredictable execution time and, in the case of external iteration variables, infinite loops.</p> | <p>To avoid the use of variable for loops, do one of the following:</p> <ul style="list-style-type: none"> <li>• Set the <b>Iteration limit source</b> parameter of the For Iterator block to <code>internal</code>.</li> <li>• If the <b>Iteration limit source</b> parameter of the For Iterator block must be <code>external</code>, use a Constant, Probe, or Width block as the source.</li> <li>• Avoid selecting the <b>Set next i (iteration variable) externally</b> parameter of the For Iterator block.</li> </ul> |

### See Also

- For Iterator Subsystem block in the Simulink reference documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check for proper usage of While Iterator blocks

Check for While Iterator blocks that cause infinite loops.

### Description

This check verifies that a model does not include infinite loops with While Iterator blocks.

See

- DO-178B Section 6.3.1e – High-level requirements conform to standards
- DO-178B Section 6.3.2e – Low-level requirements conform to standards
- MISRA-C:2004, Rule 21.1

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                      | Recommended Action                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The model combines the use of a While Iterator block with an unlimited number of iterations. An unlimited number of iterations can lead to infinite loops in real-time code, which can lead to execution time overruns.</p> | <p>To avoid infinite loops:</p> <ul style="list-style-type: none"> <li>• Set the <b>Maximum number of iterations</b> parameter of the While Iterator block to a positive integer value.</li> <li>• Consider selecting the <b>Show iteration number port</b> parameter of the While Iterator block and observe the iteration value during simulation to determine whether the maximum number of iterations is being reached. If the loop reaches the maximum number of iterations, verify whether the output values of the While Iterator block are correct.</li> </ul> |

### See Also

- While Iterator Subsystem block in the Simulink reference documentation

- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Display model version information

Display model version information in your report.

### Description

This check displays the following information for the current model:

- Version number
- Author
- Date
- Model checksum

### Results and Recommended Actions

| Condition                                                  | Recommended Action                                                    |
|------------------------------------------------------------|-----------------------------------------------------------------------|
| Could not retrieve model version and checksum information. | This summary is provided for your information. No action is required. |

### See Also

- Validating Generated Code in the Real-Time Workshop documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard



## Check for proper usage of blocks that compute absolute values

Check for absolute value blocks that have unreachable code or produce overflows.

### Description

This check verifies whether the model includes a block that attempts to compute the absolute value of a Boolean or unsigned integer value.

See

- DO-178B Section 6.3.1d – High-level requirements are verifiable
- DO-178B Section 6.3.2d – Low-level requirements are verifiable
- DO-178B Section 6.3.1g – Algorithms are accurate
- DO-178B Section 6.3.2g – Algorithms are accurate
- MISRA-C:2004, Rule 14.1
- MISRA-C:2004, Rule 21.1

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Recommended Action                                                                                                                                                                                                                                                       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The model includes a block that:</p> <ul style="list-style-type: none"> <li>• Computes an absolute value and the input signal of the block is a Boolean value or an unsigned integer. Use of Boolean and unsigned data types might result in code that is unreachable and cannot be tested.</li> <li>• Computes an absolute value of a signed integer and <b>Saturate on integer overflow</b> is not selected for that block. Taking the absolute value of full scale negative integer value results in an overflow.</li> </ul> | <ul style="list-style-type: none"> <li>• To avoid unreachable code, change the input to the Absolute Value block to a signed input type.</li> <li>• To avoid overflows, select the <b>Saturate on integer overflow</b> check box of the Absolute Value block.</li> </ul> |

### **See Also**

- Abs block in the Simulink reference documentation
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## Check for proper usage of Relational Operator blocks

Check for relational operator blocks that compare data types or equate floating-point types.

### Description

This check verifies that a model does not use the == or ~= operator with a relational operator block to compare floating-point signals.

See

- DO-178B Section 6.3.1g – Algorithms are accurate
- DO-178B Section 6.3.2g – Algorithms are accurate
- MISRA-C:2004, Rule 12.6
- MISRA-C:2004, Rule 13.3

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                         | Recommended Action                                                                                                                            |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| The model includes a relational operator block that uses the == or ~= operator to compare floating-point signals. Because of floating-point precision issues, the use of these operators on floating-point signals is unreliable. | Change the data type of the signal or rework the model to eliminate the need to use the relational operator block with the == or ~= operator. |

### See Also

Descriptions of the following blocks in the Simulink reference documentation

- Relational Operator block in the Simulink reference documentation
- Compare To Constant block in the Simulink documentation
- Compare To Zero block in the Simulink documentation
- Detect Change block in the Simulink documentation

- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard

## IEC 61508 Checks

### In this section...

- “IEC 61508 Checks Overview” on page 27-67
- “Display model metrics and complexity report” on page 27-69
- “Check for unconnected objects” on page 27-70
- “Check for fully defined interface” on page 27-71
- “Check for questionable constructs” on page 27-73
- “Check usage of Stateflow constructs” on page 27-75
- “Check state machine type of Stateflow charts” on page 27-79
- “Check for model objects that do not link to requirements” on page 27-81
- “Display configuration management data” on page 27-82
- “Check usage of Simulink constructs” on page 27-83

## IEC 61508 Checks Overview

IEC 61508 checks facilitate designing and troubleshooting Simulink models and subsystems and the code that you generate from it for applications that need to comply with IEC 61508-3.

The Model Advisor performs a checkout of the Simulink Verification and Validation license when you run the IEC 61508 checks.

### Tips

If your model uses model referencing, run the IEC 61508 checks on all referenced models before running them on the top-level model.

### See Also

- IEC 61508–3 Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 3: Software requirements
- Developing Models and Code That Comply with the IEC 16508 Standard in the Real-Time Workshop Embedded Coder documentation

- “Consulting the Model Advisor” in the Simulink documentation
- “Simulink Checks” in the Simulink reference documentation
- “Real-Time Workshop Checks” in the Real-Time Workshop documentation

## Display model metrics and complexity report

Display number of elements and name, level, and depth of subsystems for the model or subsystem.

### Description

The IEC 61508 standard recommends the usage of size and complexity metrics to assess the software under development. This check provides model metrics information for the model. The provided information can be used to inspect whether the size or complexity of the model or subsystem exceeds given limits. The check displays:

- A block count for each Simulink block type contained in the given model.
- The maximum subsystem depth of the given model.
- A count of Stateflow constructs in the given model (if applicable).
- Name, level, and depth of the subsystems contained in the given model (if applicable).

See IEC 61508-3, Table A.9 (5) – Software complexity metrics.

### Results and Recommended Actions

| Condition | Recommended Action                                                    |
|-----------|-----------------------------------------------------------------------|
| N/A       | This summary is provided for your information. No action is required. |

### See Also

- sldiagnostics in the Simulink documentation
- “Cyclomatic Complexity” in the Stateflow documentation

## Check for unconnected objects

Identify unconnected lines, input ports, and output ports in the model.

### Description

Unconnected objects are likely to cause problems propagating signal attributes such as data, type, sample time, and dimensions.

Ports connected to Ground or Terminator blocks pass this check.

See IEC 61508-3, Table A.3 (3) — Language subset.

### Results and Recommended Actions

| Condition                                                                            | Recommended Action                                                                                                                                                                                                    |
|--------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| There are unconnected lines, input ports, or output ports in the model or subsystem. | <ul style="list-style-type: none"><li>• Double-click an element in the list of unconnected items to locate the item in the model diagram.</li><li>• Properly connect the objects identified in the results.</li></ul> |

### See Also

“Working with Signals” in the Simulink documentation



## Check for fully defined interface

Identify root model Inport blocks that do not have fully defined attributes.

### Description

Using root model Inport blocks that do not have fully define dimensions, sample time, or data type can lead to undesired simulation results. Simulink back-propagates dimensions, sample times, and data types from downstream blocks unless you explicitly assign these values.

See IEC 61508-3, Table B.9 (5) – Fully defined interface.

### Results and Recommended Actions

| Condition                                                                                                                              | Recommended Action                                                                                                                                                         |
|----------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The model has root-level Inport blocks that have undefined attributes, such as an inherited sample time, data type, or port dimension. | Explicitly define all root-level Inport block attributes identified in the results. Double-click an element from the list of underspecified items to locate the condition. |

### Tips

The following configurations pass this check:

- Inport blocks with inherited port dimensions in conjunction with the usage of bus objects
- Inport blocks with automatically inherited data types in conjunction with bus objects
- Inport blocks with inherited sample times in conjunction with the **Periodic sample time constraint** menu set to Ensure sample time independent

### See Also

- Working with Data Types in the Simulink documentation
- Determining Output Signal Dimensions in the Simulink documentation

- Specifying Sample Time in the Simulink documentation

## Check for questionable constructs

Identify blocks not supported by code generation or not recommended for deployment.

### Description

This check partially identifies model constructs that are not suited for code generation or not recommended for production code generation as identified in the Simulink Block Support tables for Real-Time Workshop and Real-Time Workshop Embedded Coder. If you are using blocks with support notes for code generation, review the information and follow the given advice.

See IEC 61508-3, Table A.3 (3) – Language subset.

### Results and Recommended Actions

| Condition                                                                                      | Recommended Action                                                                                                                                                               |
|------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The model or subsystem contains blocks that should not be used for code generation.            | Consider replacing the blocks listed in the results. Double-click an element from the list of questionable items to locate condition.                                            |
| The model or subsystem contains blocks that should not be used for production code deployment. | Consider replacing the blocks listed in the results. Double-click an element from the list of questionable items to locate condition.                                            |
| The model or subsystem contains Gain blocks whose value equals 1.                              | If you are using Gain blocks as buffers, consider replacing them with Signal Conversion blocks. Double-click an element from the list of questionable items to locate condition. |

### Limitation

This check might not identify all instances of noncompliance with the Real-Time Workshop and Real-Time Workshop Embedded Coder “Simulink Built-In Blocks That Support Code Generation” tables.

**See Also**

- “Simulink Built-In Blocks That Support Code Generation” tables in the Real-Time Workshop documentation for Real-Time Workshop and Real-Time Workshop Embedded Coder
- “Developing Models for Code Generation” in the Real-Time Workshop Embedded Coder documentation

## Check usage of Stateflow constructs

Identify usage of Stateflow constructs that might impact safety.

### Description

This check identifies instances of Stateflow software being used in a way that can impact an application's safety, including:

- Use of strong data typing
- Port name mismatches
- Scope of data objects and events
- Formatting of state action statements
- Ordering of states and transitions
- Unreachable code
- Indeterminate execution time

### Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                   | Recommended Action                                                                                                                                                                                                                                                                                                                                                |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>A Stateflow chart is not configured for strong data typing on boundaries between a Simulink model and the Stateflow chart. See (IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' and MISRA-C:2004, Rules 10.1, 10.2, 10.3, and 10.4)</p> | <p>In the Chart properties dialog box, select <b>Use Strong Data Typing with Simulink I/O</b> for the Stateflow chart. When you select this check box, the Stateflow chart accepts input signals of any data type that Simulink models support, provided that the type of the input signal matches the type of the corresponding Stateflow input data object.</p> |

| <b>Condition</b>                                                                                                                                                                                                                                                                            | <b>Recommended Action</b>                                                                                                                                                                                                           |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Signals have names that differ from those of their corresponding Stateflow ports. (See IEC 61508-3, Table A.3 (3) 'Language subset')</p>                                                                                                                                                 | <ul style="list-style-type: none"> <li>• Check whether the ports are connected properly and, if not, correct the connections.</li> <li>• Change the names of the signals or the Stateflow ports so that the names match.</li> </ul> |
| <p>Events are not defined in the Stateflow hierarchy at the chart level or below. (See IEC 61508-3, Table A.3 (3) 'Language subset' and Table B.9 (2) 'Information hiding/encapsulation')</p>                                                                                               | <p>Define events at the chart level or below.</p>                                                                                                                                                                                   |
| <p>Local data is not defined in the Stateflow hierarchy at the chart level or below. (See IEC 61508-3, Table A.3 (3) 'Language subset' and Table B.9 (2) 'Information hiding/encapsulation')</p>                                                                                            | <p>Define local data at the chart level or below.</p>                                                                                                                                                                               |
| <p>A new line is missing from a state action after:</p> <ul style="list-style-type: none"> <li>• An entry (en), during (du), or exit (ex) statement</li> <li>• The semicolon (;) at the end of an assignment statement</li> </ul> <p>(See IEC 61508-3, Table A.3 (3) 'Language subset')</p> | <p>Add missing new lines.</p>                                                                                                                                                                                                       |

| <b>Condition</b>                                                                                                                                                                                                                                                                                                                                                                                                                                   | <b>Recommended Action</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Stateflow charts have <b>User specified state/transition execution order</b> cleared.<br>(See hisf_0002: User Specified State/Transition Execution Order and IEC 61508-3, Table A.3 (3) 'Language subset')                                                                                                                                                                                                                                         | For the specified charts, in the Chart Properties dialog box, select <b>User specified state/transition execution order</b> .                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Any of the following debugging options are cleared: <ul style="list-style-type: none"> <li>• <b>Enable debugging/animation</b></li> <li>• <b>Enable overflow detection (with debugging)</b></li> <li>• <b>State Inconsistency</b></li> <li>• <b>Transition Conflict</b></li> <li>• <b>Data Range</b></li> <li>• <b>Detect Cycles</b></li> </ul> (See hisf_0011: Stateflow Debugging Settings and IEC 61508-3, Table A.7 (2) 'Simulation/modeling') | Select the debugging options. In the Configuration Parameters dialog box, select: <ul style="list-style-type: none"> <li>• <b>Simulation Target &gt; General &gt; Enable debugging/animation</b></li> <li>• <b>Simulation Target &gt; General &gt; Enable overflow detection (with debugging)</b></li> </ul> In the Stateflow Debugging dialog box, select: <ul style="list-style-type: none"> <li>• <b>State Inconsistency</b></li> <li>• <b>Transition Conflict</b></li> <li>• <b>Data Range</b></li> <li>• <b>Detect Cycles</b></li> </ul> |

### See Also

See the following topics in the Stateflow documentation:

- “Strong Data Typing with Simulink I/O”
- “Property Fields”
- “Defining Events”

- “Defining Data”
- “Labeling States”

See “Stateflow Chart Considerations” in the Simulink documentation.



## Check state machine type of Stateflow charts

Identify whether Stateflow charts are all Mealy or all Moore charts.

### Description

Compares the state machine type of all Stateflow charts to the type that you specify in the input parameters.

See hisf\_0001: Mealy and Moore Semantics and IEC 61508-3, Table A.7 (2) 'Simulation/modeling'.

### Input Parameters

#### Common

Check whether charts use the same state machine type, and are all Mealy or all Moore charts.

#### Mealy

Check whether all charts are Mealy charts.

#### Moore

Check whether all charts are Moore charts.

### Analysis Results and Recommended Actions

| Condition                                                                                                                                                                                                                      | Recommended Action                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The input parameter is set to <b>Common</b> and charts in the model use any of the following:</p> <ul style="list-style-type: none"> <li>• Classic state machine types.</li> <li>• Multiple state machine types.</li> </ul> | <p>For each chart, in the Chart Properties dialog box, specify <b>State Machine Type</b> to either <b>Mealy</b> or <b>Moore</b>. Use the same state machine type for all charts in the model.</p> |
| <p>The input parameter is set to <b>Mealy</b> and charts in the model use other state machine types.</p>                                                                                                                       | <p>For each chart, in the Chart Properties dialog box, specify <b>State Machine Type</b> to <b>Mealy</b>.</p>                                                                                     |
| <p>The input parameter is set to <b>Moore</b> and charts in the model use other state machine types.</p>                                                                                                                       | <p>For each chart, in the Chart Properties dialog box, specify <b>State Machine Type</b> to <b>Moore</b>.</p>                                                                                     |

### **See Also**

- “Building Mealy and Moore Charts” in the Stateflow documentation.
- “Stateflow Chart Considerations” in the Simulink documentation.

## Check for model objects that do not link to requirements

Check whether Simulink blocks and Stateflow objects link to a requirements document.

### Description

This check verifies whether Simulink blocks and Stateflow objects link to a document containing engineering requirements for traceability.

### Analysis Results and Recommended Actions

| Condition                                                                                                                                                                                                      | Recommended Action                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| Blocks do not link to a requirements document. (See IEC 61508-3, Table A.1 (1) 'Computer-aided specification tools', Table A.2 (8) 'Computer-aided specification tools', and Table A.8 (1) 'Impact analysis'.) | Link to requirements document. See Chapter 4, "Creating and Managing Requirements Links". |

### Limitations

When you run this check, the Model Advisor does not follow library links or look under masks. The Model Advisor reviews all top-level blocks in the system.

### Tip

Run this check from the top model or subsystem that you want to check.

### See Also

Requirements Linking and Traceability on page 1

## Display configuration management data

Display model configuration and checksum information.

### Description

This informer check displays the following information for the current model:

- Model version number
- Model author
- Date
- Model checksum

See IEC 61508-3, Table A.8 (5) – Software configuration management.

### Results and Recommended Actions

| Condition                                                  | Recommended Action                                                    |
|------------------------------------------------------------|-----------------------------------------------------------------------|
| Could not retrieve model version and checksum information. | This summary is provided for your information. No action is required. |

### See Also

- “How Simulink Helps You Manage Model Versions” in the Simulink documentation
- Model Change Log in the Simulink Report Generator documentation
- Simulink.BlockDiagram.getChecksum in the Simulink documentation
- Simulink.SubSystem.getChecksum in the Simulink documentation

## Check usage of Simulink constructs

Identify usage of Simulink constructs that might impact safety.

### Description

Blocks that you use incorrectly can result in unreachable code, incorrect or unpredictable results, infinite loops, and unpredictable execution times in generated code.

This check inspects your model for proper usage of:

- Abs blocks
- Blocks that compute relational operators including Relational Operator, Compare To Constant, Compare To Zero, and Detect Change blocks
- While Iterator blocks
- For Iterator blocks

See

- IEC 61508-3, Table A.3 (2) – Strongly typed programming language
- IEC 61508-3, Table A.3 (3) – Language subset
- IEC 61508-3, Table A.4 (3) – Defensive programming
- IEC 61508-3, Table B.8 (3) – Control Flow Analysis
- MISRA-C:2004, Rule 13.3
- MISRA-C:2004, Rule 13.6
- MISRA-C:2004, Rule 14.1
- MISRA-C:2004, Rule 21.1

## Results and Recommended Actions

| Condition                                                                                                                                                                                                                                                                                                                                                           | Recommended Action                                                                                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The model or subsystem contains an Abs block that is operating on a Boolean or an unsigned input data type. This condition results in unreachable simulation pathways through the model and might result in unreachable code.</p>                                                                                                                                | <ul style="list-style-type: none"> <li>• Change the input of the Abs block to a signed input type.</li> <li>• Remove the Abs from the model.</li> </ul> |
| <p>The model or subsystem contains an Abs block that is operating on a signed integer value, and the <b>Saturate on integer overflow</b> check box is cleared. For signed data types, the absolute value of the most negative value is problematic since it is not representable by the data type. This condition results in an overflow in the generated code.</p> | <p>Select the <b>Saturate on integer overflow</b> check box of the specified Abs blocks.</p>                                                            |
| <p>The model or subsystem contains a block computing a relational operator that is operating on different data types. The condition can lead to unpredictable results in the generated code.</p>                                                                                                                                                                    | <p>For the specified blocks, use common data types as inputs.</p>                                                                                       |
| <p>The model or subsystem contains a block computing a relational operator that is not generating Boolean data as its output. This condition violates strong data typing rules and can lead to unpredictable results in the generated code.</p>                                                                                                                     | <p>Set the <b>Output data type</b> to boolean in the <b>Block Parameters &gt; Signal Attributes</b> pane for the specified blocks.</p>                  |

| Condition                                                                                                                                                                                                                                                                                                                                       | Recommended Action                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The model or subsystem contains a block computing a relational operator that uses the == or ~= operator to compare floating-point signals. The use of these operators on floating-point signals is unreliable and unpredictable because of floating-point precision issues, and can lead to unpredictable results in the generated code.</p> | <p>For the specified blocks, do one of the following:</p> <ul style="list-style-type: none"> <li>• Change the signal data type.</li> <li>• Rework the model to eliminate the need to use == or ~= operators on floating-point signals.</li> </ul>                                                                                                                                                                                                                                                                   |
| <p>The model or subsystem contains a While Iterator block that has unlimited iterations. This condition can lead to infinite loops in the generated code.</p>                                                                                                                                                                                   | <p>For the specified While Iterator blocks:</p> <ul style="list-style-type: none"> <li>• Set the <b>Maximum number of iterations (-1 for unlimited)</b> parameter to a positive integer value.</li> <li>• Consider selecting the <b>Show iteration number port</b> check box and observe the iteration value during simulation.</li> </ul>                                                                                                                                                                          |
| <p>The model or subsystem contains a For Iterator block that has variable iterations. This condition can lead to unpredictable execution times or infinite loops in the generated code.</p>                                                                                                                                                     | <p>For the specified For Iterator blocks, do one of the following:</p> <ul style="list-style-type: none"> <li>• Set the <b>Iteration limit source</b> parameter to <code>internal</code>.</li> <li>• If the <b>Iteration limit source</b> parameter must be <code>external</code>, use a Constant, Probe, or Width block as the source.</li> <li>• Clear the <b>Set next i (iteration variable) externally</b> check box.</li> <li>• Consider selecting the <b>Show iteration variable</b> check box and</li> </ul> |

| <b>Condition</b> | <b>Recommended Action</b>                      |
|------------------|------------------------------------------------|
|                  | observe the iteration value during simulation. |

**See Also**

Descriptions of the following blocks in the Simulink reference documentation:

- Abs block
- Relational Operator block
- Compare To Constant block
- Compare To Zero block
- Detect Change block
- While Iterator block
- For Iterator block



## MathWorks Automotive Advisory Board Checks

### In this section...

“MathWorks Automotive Advisory Board Checks Overview” on page 27-89

“Check for difference in font and font sizes” on page 27-91

“Check transition orientations in flow charts” on page 27-93

“Check for display of nondefault block attributes” on page 27-94

“Check for proper labeling on signal lines” on page 27-95

“Check for propagated labels on signal lines” on page 27-97

“Check default transition placement in Stateflow charts” on page 27-99

“Check return value assignments of graphical functions in Stateflow charts” on page 27-100

“Check entry formatting of states in Stateflow charts” on page 27-101

“Check usage of return values from a graphical function in Stateflow charts” on page 27-102

“Check for pointers in Stateflow charts” on page 27-103

“Check for event broadcasts in Stateflow charts” on page 27-104

“Check for MATLAB expressions in Stateflow charts” on page 27-105

“Check for blocks that do not using one-based indexing” on page 27-106

“Check for invalid file names” on page 27-108

“Check for invalid model directory names ” on page 27-110

“Check for blocks that are not discrete ” on page 27-111

“Check for prohibited sink blocks” on page 27-112

“Check for invalid port positioning and configuration” on page 27-113

“Check for mismatches between names of ports and corresponding signals” on page 27-115

“Check whether block names do not appear below blocks” on page 27-116

“Check for systems that mix primitive blocks and subsystems” on page 27-117

**In this section...**

“Check whether model has unconnected block input ports, output ports, or signal lines” on page 27-119

“Check for improperly positioned Trigger and Enable blocks” on page 27-120

“Check whether annotations have drop shadows” on page 27-121

“Check whether tunable parameters specify expressions, data type conversions, or indexing operations” on page 27-122

“Check whether Stateflow events are defined at the chart level or below” on page 27-124

“Check whether Stateflow data objects with local scope are defined at the chart level or below” on page 27-125

“Check interface signals and parameters” on page 27-126

“Check for exclusive states, default states, and substate validity” on page 27-127

“Check optimization parameters for Boolean data types” on page 27-129

“Check model diagnostic settings” on page 27-130

“Check the display attributes of block names” on page 27-133

“Check icon display attributes for port blocks” on page 27-134

“Check whether subsystem block names include invalid characters” on page 27-135

“Check whether Inport and Outport block names include invalid characters” on page 27-137

“Check whether signal line names include invalid characters” on page 27-139

“Check whether block names include invalid characters” on page 27-141

“Check Trigger and Enable block port names” on page 27-143

“Check for Simulink diagrams that have nonstandard appearance attributes” on page 27-144

“Check visibility of port block names” on page 27-147

“Check for direction of subsystem blocks” on page 27-149

**In this section...**

“Check for proper position of constants used in Relational Operator blocks” on page 27-150

“Check for use of tunable parameters in Stateflow” on page 27-151

“Check for proper use of Switch blocks” on page 27-152

“Check for proper use of signal buses and Mux block usage” on page 27-153

“Check for bitwise operations in Stateflow charts” on page 27-155

“Check for comparison operations in Stateflow charts” on page 27-157

“Check for unary minus operations on unsigned integers in Stateflow charts” on page 27-158

“Check for equality operations between floating-point expressions in Stateflow charts” on page 27-159

“Check for mismatches between Stateflow ports and associated signal names” on page 27-161

“Check for proper scope of From and Goto blocks” on page 27-162

## **MathWorks Automotive Advisory Board Checks Overview**

MathWorks Automotive Advisory Board (MAAB) checks facilitate designing and troubleshooting models from which code is generated for automotive applications.

The Model Advisor performs a checkout of the Simulink Verification and Validation license when you run the MAAB checks.

### **See Also**

- “Consulting the Model Advisor” in the Simulink documentation
- “Simulink Checks” in the Simulink reference documentation
- “Real-Time Workshop Checks” in the Real-Time Workshop documentation

- “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation
- The MathWorks Automotive Advisory Board on the MathWorks Web site, which lists downloads for the latest version of *Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow*

## Check for difference in font and font sizes

Check for difference in font and font sizes.

### Description

With the exception of free text annotations within a model, text elements, such as block names, block annotations, and signal labels, must have the same font style and font size. Select a font style and font size that is legible and portable (convertible between platforms), such as Arial or Helvetica 12 point.

This guideline facilitates

- Readability
- Workflow

See MAAB guideline db\_0043: Simulink font and font size.

### Input Parameters

#### Font Name

Apply the specified font to all text elements. Available fonts include Helvetica (default), Arial, Arial Black, Mangal, or Modern.

#### Font Size

Apply the specified font size to all text elements. Available sizes include -1, 6, 8, 9, 10 (default), 12, 14, 16, 18, 20, 22, and 24.

#### Font Angle

Apply the specified font angle to all text elements. Available angles include auto (default), normal, italic, and oblique.

#### Font Weight

Apply the specified font weight to all text elements. Available weights include auto (default), normal, light, demi, and bold.

## Results and Recommended Actions

| Condition                                                                              | Recommended Action                                                                                                                                                                               |
|----------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The fonts or font sizes for text elements in the model are not consistent or portable. | Specify values for the font parameters and click <b>Modify all Fonts</b> , or manually change the fonts and font sizes of text elements in the model such that they are consistent and portable. |

### Action Results

Clicking **Modify all Fonts** changes the font and font size of all text elements in the model according to the values you specify for the font parameters.

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check transition orientations in flow charts

Check transition orientations in flow charts.

### Description

The following rules apply to transitions in flow charts:

- Draw transition conditions horizontally.
- Draw transitions with a condition action vertically.

Loop constructs are exceptions to these rules.

This guideline facilitates

- Readability
- Workflow
- Verification and validation

See MAAB guideline db\_0132: Transitions in Flowcharts.

### Results and Recommended Actions

| Condition                                                                                                                            | Recommended Action |
|--------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| The model includes a transition with a condition that is not drawn horizontally or a transition action that is not drawn vertically. | Modify the model.  |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for display of nondefault block attributes

Check for display of nondefault block attributes.

### Description

Model diagrams should display block parameters that have values other than default values. One way of displaying this information is by using the **Block Annotation** tab in the Block Properties dialog box.

This guideline facilitates

- Readability
- Verification and validation

See MAAB guideline db\_0140: Display of basic block parameters.

### Results and Recommended Actions

| Condition                                                                                       | Recommended Action                                                                                     |
|-------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| Block parameters that have values other than default values do not appear in the model display. | Use the <b>Block Annotation</b> tab in the Block Properties dialog to add block parameter annotations. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation



## Check for proper labeling on signal lines

Check for proper labeling on signal lines.

### Description

You should use a label to identify:

- Signals originating from the following blocks (the block icon exception noted below applies to all blocks listed except Inport, Bus Selector, Demux, and Selector):

Bus Selector block (tool forces labeling)

Chart block (Stateflow)

Constant block

Data Store Read block

Demux block

From block

Inport block

Selector block

Subsystem block

---

**Block Icon Exception** If a signal label is visible in the display of the icon for the originating block, you do not have to display a label for the connected signal unless the signal label is needed elsewhere due to a rule for signal destinations.

---

- Signals connected to one of the following destination blocks (directly or indirectly with a basic block that performs an operation that is not transformative):

Bus Creator block

Chart block (Stateflow)

Data Store Write block

Goto block

Mux block

Outport block

Subsystem block

- Any signal of interest.

This guideline facilitates

- Readability
- Workflow
- Verification and validation
- Code generation

See MAAB guideline na\_0008: Display of labels on signals.

### Results and Recommended Actions

| Condition                                                                                                                    | Recommended Action                                                                                                                                             |
|------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Signals coming from Bus Selector, Chart, Constant, Data Store Read, Demux, From, Inport, or Selector blocks are not labeled. | Double-click the line that represents the signal. After the text cursor appears, enter a name and click anywhere outside the label to exit label editing mode. |

### See Also

- “Signal Labels” in the Simulink documentation
- “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for propagated labels on signal lines

Check for propagated labels on signal lines.

### Description

You should propagate a signal label from its source rather than enter the signal label explicitly (manually) if the signal originates from:

- An Inport block in a nested subsystem. However, if the nested subsystem is a library subsystem, you can explicitly label the signal coming from the Inport block to accommodate reuse of the library block.
- A basic block that performs a nontransformative operation.
- A Subsystem or Stateflow Chart block. However, if the connection originates from the output of an instance of the library block, you can explicitly label the signal to accommodate reuse of the library block.

This guideline facilitates

- Readability
- Workflow
- Verification and validation
- Code generation

See MAAB guideline na\_0009: Entry versus propagation of signal labels.

### Results and Recommended Actions

| Condition                                                                                | Recommended Action                                                                                                                           |
|------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| The model includes signal labels that were entered explicitly, but should be propagated. | Use the open angle bracket (<) character to mark signal labels that should be propagated and remove the labels that were entered explicitly. |

### See Also

- “Signal Labels” in the Simulink documentation
- “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check default transition placement in Stateflow charts

Check default transition placement in Stateflow charts.

### Description

In a Stateflow chart, you should connect the default transition at the top of the state and place the destination state of the default transition above other states in the hierarchy.

Properly position the default transition and its destination state for:

- Readability

See MAAB guideline jc\_0531: Placement of the default transition.

### Results and Recommended Actions

| Condition                                                                                                         | Recommended Action                                                                                                                 |
|-------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| The default transition for a Stateflow chart is not connected at the top of the state.                            | Move the default transition to the top of the state chart.                                                                         |
| The destination state of a Stateflow chart's default transition is lower than other states in the same hierarchy. | Adjust the position of the default transition's destination state such that the state is above other states in the same hierarchy. |

### See Also

- “Defining Transitions Between States” in the Stateflow documentation
- “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check return value assignments of graphical functions in Stateflow charts

Identify graphical functions with multiple assignments of return values in Stateflow charts.

### Description

The return value from a Stateflow graphical function must be set in only one place.

This guideline facilitates:

- Workflow
- Code generation

See MAAB guideline jc\_0511: Setting the return value from a graphical function.

### Results and Recommended Actions

| Condition                                                                            | Recommended Action                                                                    |
|--------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| The return value from a Stateflow graphical function is assigned in multiple places. | Modify the specified graphical function so that its return value is set in one place. |

### See Also

- “Graphical Functions” in the Stateflow documentation
- “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check entry formatting of states in Stateflow charts

Identify missing line breaks between entry action (en), during action (du), and exit action (ex) entries in states. Identify missing line breaks after semicolons (;) in statements.

### Description

Start a new line after the entry, during, and exit entries, and after the completion of a statement “;”.

This guideline facilitates:

- Readability

See MAAB guideline jc\_0501: Format of entries in a State block.

### Results and Recommended Actions

| Condition                              | Recommended Action                   |
|----------------------------------------|--------------------------------------|
| An entry (en) is not on a new line.    | Add a new line after the entry.      |
| A during (du) is not on a new line.    | Add a new line after the during.     |
| An exit (ex) is not on a new line.     | Add a new line after the exit.       |
| Multiple statements found on one line. | Add a new line after each statement. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check usage of return values from a graphical function in Stateflow charts

Identify calls to graphical functions in conditional expressions.

### Description

Do not use the return value of a graphical function in a comparison operation.

This guideline facilitates:

- Readability

See MAAB guideline jc\_0521: Use of the return value from graphical functions.

### Analysis Results and Recommended Actions

| Condition                                                     | Recommended Action                                                                                                                                |
|---------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| Conditional expressions contain calls to graphical functions. | Assign return values of graphical functions to intermediate variables. Use these intermediate variables in the specified conditional expressions. |

### See Also

- “Graphical Functions” in the Stateflow documentation
- “Using Graphical Functions to Extend Actions” in the Stateflow documentation
- “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation



## Check for pointers in Stateflow charts

Identify pointer operations on custom code variables.

### Description

Pointers to custom code variables are not allowed.

This guideline facilitates:

- Readability
- Workflow
- Verification and Validation
- Code Generation

See MAAB guideline jm\_0011: Pointers in Stateflow.

### Analysis Results and Recommended Actions

| Condition                                     | Recommended Action                                                         |
|-----------------------------------------------|----------------------------------------------------------------------------|
| Custom code variables use pointer operations. | Modify the specified chart to remove the dependency on pointer operations. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for event broadcasts in Stateflow charts

Identify undirected event broadcasts that might cause recursion during simulation and generate inefficient code.

### Description

Event broadcasts in Stateflow charts must be directed.

This guideline facilitates:

- Readability
- Workflow
- Verification and Validation
- Code Generation

See MAAB guideline jm\_0012: Event broadcasts

### Analysis Results and Recommended Actions

| Condition                        | Recommended Action                                                                                                                                                                                                          |
|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Event broadcasts are undirected. | Rearchitect the diagram to use directed event broadcasting. Use the send syntax or qualified event names to direct the event to a particular state. Use multiple send statements to direct an event to more than one state. |

### See Also

- “Broadcasting Events in Actions” in the Stateflow documentation
- “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for MATLAB expressions in Stateflow charts

Identify Stateflow objects that use MATLAB expressions that are not suitable for code generation.

### Description

Do not use MATLAB functions, instructions, and operators in Stateflow objects.

This guideline facilitates:

- Readability
- Workflow
- Verification and Validation
- Code Generation

See MAAB guideline db\_0127: MATLAB commands in Stateflow.

### Analysis Results and Recommended Actions

| Condition                                 | Recommended Action                                   |
|-------------------------------------------|------------------------------------------------------|
| Stateflow objects use MATLAB expressions. | Replace all MATLAB expressions in Stateflow objects. |

### See Also

- “Using MATLAB Functions and Data in Actions” in the Stateflow documentation
- “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for blocks that do not using one-based indexing

Check for blocks that do not use one-based indexing.

### Description

One-based indexing ([1, 2, 3,...]) is used for the following:

| Product   | Items                                                                                                                                                                                                                                                                                                                                |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MATLAB    | <ul style="list-style-type: none"> <li>• Workspace variables and structures</li> <li>• Local variables of MATLAB functions</li> <li>• Global variables</li> </ul>                                                                                                                                                                    |
| Simulink  | <ul style="list-style-type: none"> <li>• Signal vectors and matrices</li> <li>• Parameter vectors and matrices</li> <li>• S-function input and output signal vectors and matrices in MATLAB-code</li> <li>• S-function parameter vectors and matrices in MATLAB-code</li> <li>• S-function local variables in MATLAB-code</li> </ul> |
| Stateflow | <ul style="list-style-type: none"> <li>• Input and output signal vectors and matrices</li> <li>• Parameter vectors and matrices</li> <li>• Local variables</li> </ul>                                                                                                                                                                |

Zero-based indexing ([0, 1, 2, ...]) is used for the following:

| Product   | Items                                                                                                                                                                                                                                                                                                                        |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Simulink  | <ul style="list-style-type: none"> <li>• Signal vectors and matrices</li> <li>• S-function input and output signal vectors and matrices in C code</li> <li>• S-function input parameters in C code</li> <li>• S-function parameter vectors and matrices in C code</li> <li>• S-function local variables in C code</li> </ul> |
| Stateflow | <ul style="list-style-type: none"> <li>• Variables and structures in custom C code</li> </ul>                                                                                                                                                                                                                                |
| C code    | <ul style="list-style-type: none"> <li>• Local variables and structures</li> <li>• Global variables</li> </ul>                                                                                                                                                                                                               |

This guideline facilitates

- Readability
- Workflow
- Code generation

See MAAB guideline db\_0112: Indexing.

## Results and Recommended Actions

| Condition                                                       | Recommended Action                                                   |
|-----------------------------------------------------------------|----------------------------------------------------------------------|
| Blocks in your model are not configured for one-based indexing. | Using block parameters, configure all blocks for one-based indexing. |

## See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for invalid file names

Check for files residing in the same folder as the model that have illegal file names.

### Description

This guideline facilitates

- Readability
- Workflow

See MAAB guideline ar\_0001: Filenames.

### Results and Recommended Actions

| Condition                                              | Recommended Action                                                         |
|--------------------------------------------------------|----------------------------------------------------------------------------|
| The file name contains illegal characters.             | Rename the file. Allowed characters are a–z, A–Z, 0–9, and underscore (_). |
| The file name starts with a number.                    | Rename the file.                                                           |
| The file name starts with an underscore ("_").         | Rename the file.                                                           |
| The file name ends with an underscore ("_").           | Rename the file.                                                           |
| The file extension contains one (or more) underscores. | Change the file extension.                                                 |
| The file name has consecutive underscores.             | Rename the file.                                                           |
| The file name contains more than one dot (".").        | Rename the file.                                                           |

**See Also**

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for invalid model directory names

Checks model directory and subdirectory names for invalid characters.

### Description

This guideline facilitates

- Readability
- Workflow

See MAAB guideline ar\_0002: Directory names.

### Results and Recommended Actions

| Condition                                           | Recommended Action                                                              |
|-----------------------------------------------------|---------------------------------------------------------------------------------|
| The directory name contains illegal characters.     | Rename the directory. Allowed characters are a–z, A–Z, 0–9, and underscore (_). |
| The directory name starts with a number.            | Rename the directory.                                                           |
| The directory name starts with an underscore ("_"). | Rename the directory.                                                           |
| The directory name ends with an underscore ("_").   | Rename the directory.                                                           |
| The directory name has consecutive underscores.     | Rename the directory.                                                           |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation



## Check for blocks that are not discrete

Check for blocks that are not discrete.

### Description

You cannot include continuous blocks in controller models.

This guideline facilitates

- Readability
- Workflow
- Code generation

See MAAB guideline jm\_0001: Prohibited Simulink standard blocks inside controllers.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                            | Recommended Action                                                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Continuous blocks — Derivative, Integrator, State-Space, Transfer Fcn, Transfer Delay, Variable Time Delay, Variable Transport Delay, and Zero-Pole — are not permitted in models representing discrete controllers. | Replace continuous blocks with the equivalent blocks discretized in the s-domain by using the Discretizing library, as explain in “How to Discretize Blocks from the Simulink Model” in the Simulink documentation. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for prohibited sink blocks

Check for prohibited Simulink sink blocks.

### Description

You must design controller models from discrete blocks. Sink blocks, such as the Scope block, are not allowed.

This guideline facilitates

- Readability
- Workflow

See MAAB guideline hd\_0001: Prohibited Simulink sinks.

### Results and Recommended Actions

| Condition                                              | Recommended Action                 |
|--------------------------------------------------------|------------------------------------|
| Sink blocks are not permitted in discrete controllers. | Remove sink blocks from the model. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for invalid port positioning and configuration

Check whether the model contains ports with invalid position and configuration.

### Description

In models, ports must comply with the following rules:

- Place Inport blocks on the left side of the diagram. Move the Inport block right only to prevent signal crossings.
- Place Outport blocks on the right side of the diagram. Move the Outport block left only to prevent signal crossings.
- Avoid using duplicate Inport blocks at the subsystem level if possible.
- Do not use duplicate Inport blocks at the root level.

This guideline facilitates

- Readability

See MAAB guideline db\_0042: Port block in Simulink models.

### Results and Recommended Actions

| Condition                                                                   | Recommended Action                             |
|-----------------------------------------------------------------------------|------------------------------------------------|
| Inport blocks are too far to the right and result in left-flowing signals.  | Move the specified Inport blocks to the left.  |
| Outport blocks are too far to the left and result in right-flowing signals. | Move the specified Output blocks to the right. |

| <b>Condition</b>                           | <b>Recommended Action</b>                                                                                                                                                                                                                                               |
|--------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ports do not have the default orientation. | Modify the model diagram such that signal lines for output ports enter the side of the block and signal lines for input ports exit the right side of the block.                                                                                                         |
| Ports are duplicate Inport blocks.         | <ul style="list-style-type: none"><li data-bbox="872 475 1307 569">• If the duplicate Inport blocks are in a subsystem, remove them where possible.</li><li data-bbox="872 586 1307 644">• If the duplicate Inport blocks are at the root level, remove them.</li></ul> |

**See Also**

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for mismatches between names of ports and corresponding signals

Check for mismatches between names of ports and corresponding signals.

### Description

Use matching names for ports and their corresponding signals.

This guideline facilitates

- Readability
- Workflow
- Simulation

See MAAB guideline jm\_0010: Port block names in Simulink models.

### Prerequisite

Prerequisite MAAB guidelines for this check are:

- db\_0042: Port block in Simulink models
- na\_0005: Port block name visibility in Simulink models

### Results and Recommended Actions

| Condition                                                      | Recommended Action                                                                |
|----------------------------------------------------------------|-----------------------------------------------------------------------------------|
| Ports have names that differ from their corresponding signals. | Change the port name or the signal name to match the correct name for the signal. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check whether block names do not appear below blocks

Check whether block names do not appear below blocks.

### Description

If shown, the name of all blocks should appear below the blocks.

This guideline facilitates

- Readability
- Workflow

See MAAB guideline db\_0142: Position of block names.

### Results and Recommended Actions

| Condition                                              | Recommended Action                                    |
|--------------------------------------------------------|-------------------------------------------------------|
| Blocks have names that do not appear below the blocks. | Set the name of the block to appear below the blocks. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for systems that mix primitive blocks and subsystems

Check for systems that mix primitive blocks and subsystems.

### Description

You must design every level of a model with building blocks of the same type, for example, only subsystems or only primitive (basic) blocks.

This guideline facilitates

- Readability
- Workflow
- Verification and validation

See MAAB guideline db\_0143: Similar block types on the model levels.

### Results and Recommended Actions

| Condition                                                                 | Recommended Action                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A level in the model includes both subsystem blocks and primitive blocks. | <ul style="list-style-type: none"> <li>• Move nonvirtual blocks into the subsystem.</li> <li>• If possible, replace blocks at the identified level of the model hierarchy with blocks that you can place at any module level. Such blocks include Inport, Outport, Enable (not at highest model level), Trigger (not at highest model level), Mux, Demux, Bus Selector, Bus Creator, Selector, Ground, Terminator, From, Goto, Switch, Multiport Switch, Merge, Unit Delay, Rate Transition, Type Conversion, Data Store Memory, If, and Switch Case.</li> </ul> |

**See Also**

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation



## Check whether model has unconnected block input ports, output ports, or signal lines

Check whether model has unconnected input ports, output ports, or signal lines.

### Description

All unconnected inputs should be connected to ground blocks. All unconnected outputs should be connected to terminator blocks. Respecting the guideline eliminates error messages.

See MAAB guideline db\_0081: Unconnected signals, block inputs and block outputs.

### Results and Recommended Actions

| Condition                                  | Recommended Action                                                                             |
|--------------------------------------------|------------------------------------------------------------------------------------------------|
| Blocks have unconnected inputs or outputs. | Connect unconnected lines to blocks specified by the design or to Ground or Terminator blocks. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for improperly positioned Trigger and Enable blocks

Check for improperly positioned Trigger and Enable blocks.

### Description

Locate blocks that define subsystems as conditional or iterative at the top of the subsystem diagram.

This guideline facilitates

- Readability
- Workflow
- Verification and validation

See MAAB guideline db\_0146: Triggered, enabled, conditional Subsystems.

### Results and Recommended Actions

| Condition                                                                                          | Recommended Action                                                                         |
|----------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| Trigger , Enable, and Action Port blocks are not centered in the upper third of the model diagram. | Move the Trigger, Enable, and Action Port blocks to the correct area of the model diagram. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check whether annotations have drop shadows

Check whether annotations have drop shadows.

### Description

Annotations should not have a drop shadow for readability.

This guideline facilitates

- Readability

See MAAB guideline jm\_0013: Annotations.

### Results and Recommended Actions

| Condition                         | Recommended Action                                         |
|-----------------------------------|------------------------------------------------------------|
| Annotations display drop shadows. | Clear the <b>Format &gt; Show Drop Shadow</b> menu option. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check whether tunable parameters specify expressions, data type conversions, or indexing operations

Check whether tunable parameters specify expressions, data type conversions, or indexing operations.

### Description

To ensure that a parameter is tunable, you must enter the basic block without the use of MATLAB calculations or scripting. For example, omit

- Expressions
- Data type conversions
- Selections of rows or columns

This guideline facilitates

- Readability
- Workflow
- Code generation

See MAAB guideline db\_0110: Tunable parameters in basic blocks.

### Results and Recommended Actions

| Condition                                                                                                  | Recommended Action                                                                                                                                                                                     |
|------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Blocks have a tunable parameter that specifies an expression, data type conversion, or indexing operation. | In each case, move the calculation outside of the block, for example, by performing the calculation with a series of Simulink blocks, or precompute the value in the base workspace as a new variable. |

**See Also**

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check whether Stateflow events are defined at the chart level or below

Check whether Stateflow events are defined at the chart level or below.

### Description

All events of a Stateflow chart must be defined at the chart level or lower. Events cannot be at the machine level; that is, charts cannot interact with local events.

This guideline facilitates

- Readability
- Workflow
- Verification and validation

See MAAB guideline db\_0126: Scope of events.

### Results and Recommended Actions

| Condition                                                       | Recommended Action                            |
|-----------------------------------------------------------------|-----------------------------------------------|
| An event in a chart is not defined at the chart level or below. | Define the event at the chart level or below. |

### See Also

- “Defining Events” in the Stateflow documentation.
- “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check whether Stateflow data objects with local scope are defined at the chart level or below

Check whether Stateflow data objects with local scope are defined at the chart level or below.

### Description

You must define all local data of a Stateflow block on the chart level or below in the object hierarchy. You cannot define local variables on the machine level; however, parameters and constants are allowed at the machine level.

This guideline facilitates

- Readability
- Workflow
- Verification and validation

See MAAB guideline db\_0125: Scope of internal signals and local auxiliary variables.

### Results and Recommended Actions

| Condition                                                                         | Recommended Action                             |
|-----------------------------------------------------------------------------------|------------------------------------------------|
| Local data is not defined in the Stateflow hierarchy at the chart level or below. | Define local data at the chart level or below. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check interface signals and parameters

Check whether labeled Stateflow and Simulink input and output signals are strongly typed.

### Description

Strong data typing between Stateflow and Simulink input and output signals is required.

This guideline facilitates

- Readability
- Workflow
- Verification and validation

See MAAB guideline db\_0122: Stateflow and Simulink interface signals and parameters.

### Results and Recommended Actions

| Condition                                                        | Recommended Action                                                                            |
|------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| A Stateflow chart does not use strong data typing with Simulink. | Select the <b>Use Strong Data Typing with Simulink I/O</b> check box for the specified block. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation



## Check for exclusive states, default states, and substate validity

Check states in state machines.

### Description

In state machines:

- There must be at least two exclusive states.
- A state cannot have only one substate.
- The initial state of a hierarchical level with exclusive states is clearly defined by a default transition.

This guideline facilitates

- Readability
- Workflow
- Verification and validation

See MAAB guideline db\_0137: States in state machines.

### Prerequisite

A prerequisite MAAB guideline for this check is db\_0149: Flowchart patterns for condition actions.

### Results and Recommended Actions

| Condition                                | Recommended Action                                                                  |
|------------------------------------------|-------------------------------------------------------------------------------------|
| A system is underspecified.              | Validate that the intended design is properly represented in the Stateflow diagram. |
| Chart has only one exclusive (OR) state. | Make the state a parallel state, or add another exclusive (OR) state.               |

| <b>Condition</b>                                | <b>Recommended Action</b>                                             |
|-------------------------------------------------|-----------------------------------------------------------------------|
| Chart does not have a default state defined.    | Define a default state.                                               |
| Chart has multiple default states defined.      | Define only one default state. Make the others nondefault.            |
| State has only one exclusive (OR) substate.     | Make the state a parallel state, or add another exclusive (OR) state. |
| State does not have a default substate defined. | Define a default substate.                                            |
| State has multiple default substates defined.   | Define only one default substate, make the others nondefault.         |

**See Also**

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check optimization parameters for Boolean data types

Check the optimization parameter for Boolean data types.

### Description

Optimization for Boolean data types is required

This guideline facilitates

- Workflow
- Code generation

See MAAB guideline jc\_0011: Optimization parameters for Boolean data types.

### Prerequisite

A prerequisite MAAB guideline for this check is na\_0002: Appropriate implementation of fundamental logical and numerical operations.

### Results and Recommended Actions

| Condition                                                                                           | Recommended Action                                                                                                                                    |
|-----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| Configuration setting for <b>Implement logic signals as boolean data (vs. double)</b> is incorrect. | Select the <b>Implement logic signals as boolean data (vs. double)</b> check box in the Configuration Parameters dialog box <b>Optimization</b> pane. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check model diagnostic settings

Check the model diagnostics configuration parameter settings.

### Description

You should enable the following diagnostics:

**Algebraic loop**

**Minimize algebraic loop**

**Inf or NaN block output**

**Duplicate data store names**

**Unconnected block input ports**

**Unconnected block output ports**

**Unconnected line**

**Unspecified bus object at root Outport block**

**Mux blocks used to create bus signals**

**Element name mismatch**

**Invalid function-call connection**

This guideline facilitates

- Workflow
- Code generation

Diagnostics not listed in the Results and Recommended Actions section below can be set to any value.

See MAAB guideline jc\_0021: Model diagnostic settings.

## Results and Recommended Actions

| Condition                                                                                                              | Recommended Action                                                                                                                                                                                                                                                                                                                          |
|------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Algebraic loop</b> is set to none.                                                                                  | Set <b>Algebraic loop</b> on the <b>Diagnostics &gt; Solver</b> pane of the Configuration Parameters dialog box to error or warning. Otherwise, Simulink might attempt to automatically break the algebraic loops, which can affect execution order of the blocks.                                                                          |
| <b>Minimize algebraic loop</b> is set to none.                                                                         | Set <b>Minimize algebraic loop</b> on the <b>Diagnostics &gt; Solver</b> pane of the Configuration Parameters dialog box to error or warning. Otherwise, Simulink might attempt to automatically break the algebraic loops for reference models and atomic subsystems, which can affect the execution order for those models or subsystems. |
| <b>Inf or NaN block output</b> is set to none, which can result in numerical exceptions in the generated code.         | Set <b>Inf or NaN block output</b> on the <b>Diagnostics &gt; Data Validity &gt; Signals</b> pane of the Configuration Parameters dialog box to error or warning.                                                                                                                                                                           |
| <b>Duplicate data store names</b> is set to none, which can result in nonunique variable naming in the generated code. | Set <b>Duplicate data store names</b> on the <b>Diagnostics &gt; Data Validity &gt; Signals</b> pane of the Configuration Parameters dialog box to error or warning.                                                                                                                                                                        |
| <b>Unconnected block input ports</b> is set to none, which prevents code generation.                                   | Set <b>Unconnected block input ports</b> on the <b>Diagnostics &gt; Data Validity &gt; Signals</b> pane of the Configuration Parameters dialog box to error or warning.                                                                                                                                                                     |
| <b>Unconnected block output ports</b> is set to none, which can lead to dead code.                                     | Set <b>Unconnected block output ports</b> on the <b>Diagnostics &gt; Data Validity &gt; Signals</b> pane of the Configuration Parameters dialog box to error or warning.                                                                                                                                                                    |

| Condition                                                                                                                                                           | Recommended Action                                                                                                                                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Unconnected line</b> is set to none, which prevents code generation.</p>                                                                                      | <p>Set <b>Unconnected line</b> on the <b>Diagnostics &gt; Connectivity &gt; Signals</b> pane of the Configuration Parameters dialog box to error or warning.</p>                                                                                                          |
| <p><b>Unspecified bus object at root Output block</b> is set to none, which can lead to an unspecified interface if the model is referenced from another model.</p> | <p>Set <b>Unspecified bus object at root Output block</b> on the <b>Diagnostics &gt; Connectivity &gt; Buses</b> pane of the Configuration Parameters dialog box to error or warning.</p>                                                                                 |
| <p><b>Mux blocks used to create bus signals</b> is set to none, which can lead to an unintended bus being created in the model.</p>                                 | <p>Set <b>Mux blocks used to create bus signals</b> on the <b>Diagnostics &gt; Connectivity &gt; Buses</b> pane of the Configuration Parameters dialog box to error or warning.</p>                                                                                       |
| <p><b>Element name mismatch</b> is set to none, which can lead to an incorrect interface in the generated code.</p>                                                 | <p>Set <b>Element name mismatch</b> on the <b>Diagnostics &gt; Connectivity &gt; Buses</b> pane of the Configuration Parameters dialog box to error or warning.</p>                                                                                                       |
| <p><b>Invalid function-call connection</b> is set to none, which can lead to an error in the operation of the generated code.</p>                                   | <p>Set <b>Invalid function-call connection</b> on the <b>Diagnostics &gt; Connectivity &gt; Function Calls</b> pane of the Configuration Parameters dialog box to error or warning, since this condition can lead to an error in the operation of the generated code.</p> |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check the display attributes of block names

Check the display attributes of block names.

### Description

Block names should be displayed when providing descriptive information. Block names should not be displayed if the block function is known from its appearance.

This guideline facilitates

- Readability

See MAAB guideline jc\_0061: Display of block names.

### Results and Recommended Actions

| Condition                      | Recommended Action                                                              |
|--------------------------------|---------------------------------------------------------------------------------|
| Block name is not descriptive. | These block names should be modified to be more descriptive or not be shown.    |
| Block name is not displayed.   | These block names should be shown since they appear to have a descriptive name. |
| Block name is obvious.         | These block names should not be displayed.                                      |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check icon display attributes for port blocks

Check the **Icon display** setting for Inport and Outport blocks.

### Description

The **Icon display** setting is required.

This guideline facilitates

- Readability

See MAAB guideline jc\_0081: Icon display for Port block.

### Results and Recommended Actions

| Condition                                     | Recommended Action                                                                      |
|-----------------------------------------------|-----------------------------------------------------------------------------------------|
| The <b>Icon display</b> setting is incorrect. | Set the <b>Icon display</b> to Port number for the specified Inport and Outport blocks. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation



## Check whether subsystem block names include invalid characters

Check whether subsystem block names include invalid characters.

### Description

The names of all subsystem blocks are required.

This guideline facilitates

- Readability
- Workflow
- Code generation

See MAAB guideline jc\_0201: Usable characters for Subsystem names.

### Results and Recommended Actions

| Condition                                            | Recommended Action                                                                              |
|------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| The subsystem name contains illegal characters.      | Rename the subsystem. Allowed characters include a–z, A–Z, 0–9, underscore (_), and period (.). |
| The subsystem name starts with a number.             | Rename the subsystem.                                                                           |
| The subsystem name starts with an underscore ("_").  | Rename the subsystem.                                                                           |
| The subsystem name ends with an underscore ("_").    | Rename the subsystem.                                                                           |
| The subsystem name contains consecutive underscores. | Rename the subsystem.                                                                           |
| The subsystem name has consecutive underscores.      | Rename the subsystem.                                                                           |
| The subsystem name has blank spaces.                 | Rename the subsystem.                                                                           |

### **Tips**

Use underscores to separate parts of a subsystem name instead of spaces.

### **See Also**

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check whether Inport and Outport block names include invalid characters

Check whether Inport and Outport block names include invalid characters.

### Description

The names of all Inport and Outport blocks are required.

This guideline facilitates

- Readability
- Workflow
- Code generation

See MAAB guideline jc\_0211: Usable characters for Inport blocks and Outport blocks.

### Results and Recommended Actions

| Condition                                        | Recommended Action                                                                          |
|--------------------------------------------------|---------------------------------------------------------------------------------------------|
| The block name contains illegal characters.      | Rename the block. Allowed characters include a–z, A–Z, 0–9, underscore (_), and period (.). |
| The block name starts with a number.             | Rename the block.                                                                           |
| The block name starts with an underscore ("_").  | Rename the block.                                                                           |
| The block name ends with an underscore ("_").    | Rename the block.                                                                           |
| The block name contains consecutive underscores. | Rename the block.                                                                           |
| The block name has consecutive underscores.      | Rename the block.                                                                           |
| The block name has blank spaces.                 | Rename the block.                                                                           |

### **Tips**

Use underscores to separate parts of a block name instead of spaces.

### **See Also**

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check whether signal line names include invalid characters

Check whether signal line names include invalid characters.

### Description

The names of all signal lines are required.

This guideline facilitates

- Readability
- Workflow
- Code generation

See MAAB guideline jc\_0221: Usable characters for signal line names.

### Results and Recommended Actions

| Condition                                              | Recommended Action                                                                                |
|--------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| The signal line name contains illegal characters.      | Rename the signal line. Allowed characters include a–z, A–Z, 0–9, underscore (_), and period (.). |
| The signal line name starts with a number.             | Rename the signal line.                                                                           |
| The signal line name starts with an underscore ("_").  | Rename the signal line.                                                                           |
| The signal line name ends with an underscore ("_").    | Rename the signal line.                                                                           |
| The signal line name contains consecutive underscores. | Rename the signal line.                                                                           |
| The signal line name has consecutive underscores.      | Rename the signal line.                                                                           |

| <b>Condition</b>                             | <b>Recommended Action</b> |
|----------------------------------------------|---------------------------|
| The signal line name has blank spaces.       | Rename the signal line.   |
| The signal line name has control characters. | Rename the signal line.   |

**Tips**

Use underscores to separate parts of a signal line name instead of spaces.

**See Also**

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check whether block names include invalid characters

Check whether block names include invalid characters.

### Description

The names of all blocks are required.

This guideline facilitates

- Readability
- Workflow
- Code generation

This guideline does not apply to subsystem blocks.

See MAAB guideline jc\_0231: Usable characters for block names.

### Prerequisite

A prerequisite MAAB guideline for this check is jc\_0201: Usable characters for Subsystem names.

### Results and Recommended Actions

| Condition                                   | Recommended Action                                                                          |
|---------------------------------------------|---------------------------------------------------------------------------------------------|
| The block name contains illegal characters. | Rename the block. Allowed characters include a–z, A–Z, 0–9, underscore (_), and period (.). |
| The block name starts with a number.        | Rename the block.                                                                           |
| The block name has blank spaces.            | Rename the block.                                                                           |
| The block name has double byte characters.  | Rename the block.                                                                           |

### **Tips**

Carriage returns are allowed in block names.

### **See Also**

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation



## Check Trigger and Enable block port names

Check Trigger and Enable block port names.

### Description

Block port names should match the name of the signal triggering the subsystem.

This guideline facilitates

- Readability

See MAAB guideline jc\_0281: Naming of Trigger Port block and Enable Port block.

### Results and Recommended Actions

| Condition                                                                     | Recommended Action                                  |
|-------------------------------------------------------------------------------|-----------------------------------------------------|
| Trigger block does not match the name of the signal to which it is connected. | Match Trigger block names to the connecting signal. |
| Enable block does not match the name of the signal to which it is connected.  | Match Enable block names to the connecting signal.  |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for Simulink diagrams that have nonstandard appearance attributes

Check model appearance setting attributes.

### Description

Model appearance settings are required to conform to the guidelines when the model is released.

This guideline facilitates

- Readability
- Workflow

See MAAB guideline na\_0004: Simulink model appearance.

### Results and Recommended Actions

| Condition                                     | Recommended Action                                                                                                                                                                                    |
|-----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Diagrams do not have white backgrounds.       | Select <b>Format &gt; Screen Color &gt; Automatic</b> .                                                                                                                                               |
| Diagrams do not have zoom factor set to 100%. | Select <b>View &gt; Normal (100%)</b> .                                                                                                                                                               |
| The toolbar is not visible.                   | Select <b>View &gt; Toolbar</b> .                                                                                                                                                                     |
| The status bar is not visible.                | Select <b>View &gt; Status Bar</b> .                                                                                                                                                                  |
| Block backgrounds are not white.              | Blocks should have black foregrounds with white backgrounds. Click the specified block and select <b>Format &gt; Foreground Color &gt; Black</b> and <b>Format &gt; Background Color &gt; White</b> . |
| <b>Wide Nonscalar Lines</b> is cleared.       | Select <b>Format &gt; Port/Signal Displays &gt; Wide Nonscalar Lines</b> .                                                                                                                            |

| <b>Condition</b>                                                  | <b>Recommended Action</b>                                                      |
|-------------------------------------------------------------------|--------------------------------------------------------------------------------|
| <b>Viewer Indicators</b> is cleared.                              | Select <b>Format &gt; Port/Signal Displays &gt; Viewer Indicators</b> .        |
| <b>Testpoint Indicators</b> is cleared.                           | Select <b>Format &gt; Port/Signal Displays &gt; Testpoint Indicators</b> .     |
| <b>Port Data Types</b> is selected.                               | Clear <b>Format &gt; Port/Signal Displays &gt; Port Data Types</b> .           |
| <b>Storage Class</b> is selected.                                 | Clear <b>Format &gt; Port/Signal Displays &gt; Storage Class</b> .             |
| <b>Signal Dimensions</b> is selected.                             | Clear <b>Format &gt; Port/Signal Displays &gt; Signal Dimensions</b> .         |
| <b>Model Browser</b> is selected.                                 | Clear <b>View &gt; Model Browser Options &gt; Model Browser</b> .              |
| <b>Sorted Order</b> is selected.                                  | Clear <b>Format &gt; Block Displays &gt; Sorted Order</b> .                    |
| <b>Model Block Version</b> is selected.                           | Clear <b>Format &gt; Block Displays &gt; Model Block Version</b> .             |
| <b>Model Block I/O Mismatch</b> is selected.                      | Clear <b>Format &gt; Block Displays &gt; Model Block I/O Mismatch</b> .        |
| <b>Execution Context Indicator</b> is selected.                   | Clear <b>Format &gt; Block Displays &gt; Execution Context Indicator</b> .     |
| <b>Sample Time Colors</b> is selected.                            | Clear <b>Format &gt; Port/Signal Displays &gt; Sample Time Colors</b> .        |
| <b>Library Link Display</b> is set to <b>User</b> or <b>All</b> . | Select <b>Format &gt; Library Link Display &gt; None</b> .                     |
| <b>Linearization Indicators</b> is cleared.                       | Select <b>Format &gt; Port/Signal Displays &gt; Linearization Indicators</b> . |

**See Also**

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check visibility of port block names

Check the visibility of port block names.

### Description

An organization applying the MAAB guidelines must select one of the following alternatives to enforce:

- The name of port blocks are not hidden.
- The name of port blocks must be hidden.

This guideline facilitates

- Readability

---

**Note** This check does not look in masked subsystems.

---

See MAAB guideline na\_0005: Port block name visibility in Simulink models.

### Input Parameters

#### All Port names should be shown (Format/Show Name)

Select this check box if all ports should show the name, including subsystems.

### Results and Recommended Actions

| Condition                                                                                                             | Recommended Action                                                                          |
|-----------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| Blocks do not show their name and the <b>All Port names should be shown (Format/Show Name)</b> check box is selected. | Change the format of the specified blocks to show names according to the input requirement. |

| <b>Condition</b>                                                                                              | <b>Recommended Action</b>                                                                   |
|---------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| Blocks show their name and the <b>All Port names should be shown (Format/Show Name)</b> check box is cleared. | Change the format of the specified blocks to hide names according to the input requirement. |
| Subsystem blocks do not show their port names.                                                                | Set the subsystem parameter <b>Show port labels</b> to a value other than none.             |
| Subsystem blocks show their port names.                                                                       | Set the subsystem parameter <b>Show port labels</b> to none.                                |

**See Also**

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for direction of subsystem blocks

Check the orientation of subsystem blocks.

### Description

Subsystem inputs must be located on the left side of the block, and outputs must be located on the right side of the block.

This guideline facilitates

- Readability

See MAAB guideline jc\_0111: Direction of Subsystem.

### Results and Recommended Actions

| Condition                                            | Recommended Action                                                                                               |
|------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| Subsystem blocks are not in the correct orientation. | Change the subsystem blocks to have the correct orientation, with inports on the left and outports on the right. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for proper position of constants used in Relational Operator blocks

Check the position of Constant blocks used in Relational Operator blocks.

### Description

When the relational operator is used to compare a signal to a constant value, the constant input should be the second, lower input.

This guideline facilitates

- Readability
- Code generation

See MAAB guideline jc\_0131: Use of Relational Operator block.

### Results and Recommended Actions

| Condition                                                                   | Recommended Action                                  |
|-----------------------------------------------------------------------------|-----------------------------------------------------|
| Relational Operator blocks have a Constant block on the first, upper input. | Move the Constant block to the second, lower input. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation



## Check for use of tunable parameters in Stateflow

Check for use of tunable parameters in Stateflow charts.

### Description

Include tunable parameters in a Stateflow chart as inputs from the Simulink model.

This guideline facilitates

- Readability
- Workflow
- Code generation

See MAAB guideline jc\_0541: Use of tunable parameters in Stateflow.

### Results and Recommended Actions

| Condition                                                                                                 | Recommended Action                                                                              |
|-----------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| Stateflow charts reference Simulink data objects, which should be used as inputs from the Simulink model. | Make the Simulink data objects inputs from the Simulink model to the specified Stateflow chart. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for proper use of Switch blocks

Check for proper use of Switch blocks.

### Description

This check verifies that the Switch block's control input (the second input) is a Boolean value and that the block is configured to pass the first input when the control input is nonzero.

This guideline facilitates

- Readability
- Workflow

See MAAB guideline jc\_0141: Use of the Switch block.

### Results and Recommended Actions

| Condition                                                                                     | Recommended Action                                                                       |
|-----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| The Switch block's control input (second input) is not a Boolean value.                       | Change the data type of the control input to Boolean.                                    |
| The Switch block is not configured to pass the first input when the control input is nonzero. | Set the block parameter <b>Criteria for passing first input</b> to <code>u2 ~=0</code> . |

### See Also

- See the description of the Switch block in the Simulink documentation.
- “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for proper use of signal buses and Mux block usage

Check for proper use of signal busses and Mux block usage.

### Description

This check verifies whether a model is using signal buses and Mux blocks properly.

This guideline facilitates

- Readability
- Workflow

See MAAB guidelinenena\_0010: Grouping data flows into signals.

### Results and Recommended Actions

| Condition                                                                                                                | Recommended Action                                                    |
|--------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| The individual scalar input signals for a Mux block do not have common functionality, data types, dimensions, and units. | Modify the scalar input signals such that the specifications match.   |
| The output of a Mux block is not a vector.                                                                               | Change the output of the Mux block to a vector.                       |
| All inputs to a Mux block are not scalars.                                                                               | Make sure that all input signals to Mux blocks are scalars.           |
| The input for a Bus Selector block is not a bus signal.                                                                  | Make sure that the input for all Bus Selector blocks is a bus signal. |

### See Also

- “Using Composite Signals” in the Simulink documentation.

- “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for bitwise operations in Stateflow charts

Identify bitwise operators (&, |, and ^) in Stateflow charts. If you select **Enable C-bit operations** for a chart, only bitwise operators in expressions containing Boolean data types are reported. Otherwise, all bitwise operators are reported for the chart.

### Description

Do not use bitwise operators in Stateflow charts, unless you enable bitwise operations.

This guideline facilitates:

- Simulation
- Code Generation

See MAAB guideline na\_0001: Bitwise Stateflow operators.

### Analysis Results and Recommended Actions

| Condition                                                                                                                                       | Recommended Action                                                                                                                                                                                                                                                                                                |
|-------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Stateflow charts with <b>Enable C-bit operations</b> selected use bitwise operators (&,  , and ^) in expressions containing Boolean data types. | Do not use Boolean data types in the specified expressions.                                                                                                                                                                                                                                                       |
| The Model Advisor could not determine the data types in expressions with bitwise operations.                                                    | To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions.                                                                                                                                                                                                    |
| Stateflow charts with <b>Enable C-bit operations</b> cleared use bitwise operators (&,  , and ^).                                               | To fix this issue, do any of the following: <ul style="list-style-type: none"> <li>• Modify the expressions to replace bitwise operators.</li> <li>• If not using Boolean data types, consider enabling bitwise operations. In the Chart properties dialog box, select <b>Enable C-bit operations</b>.</li> </ul> |

### See Also

- “Binary and Bitwise Operations” in the Stateflow documentation
- “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for comparison operations in Stateflow charts

Identify comparison operations with different data types in Stateflow objects.

### Description

Comparisons should be made between variables of the same data types.

This guideline facilitates:

- Workflow
- Code Generation

See MAAB guideline na\_0013: Comparison operation in Stateflow

### Analysis Results and Recommended Actions

| Condition                                                                                       | Recommended Action                                                                                             |
|-------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| Comparison operations with different data types were found.                                     | Revisit the specified operations to avoid comparison operations with different data types.                     |
| The Model Advisor could not determine the data types in expressions with comparison operations. | To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for unary minus operations on unsigned integers in Stateflow charts

Identify unary minus operations applied to unsigned integers in Stateflow objects.

### Description

Do not perform unary minus operations on unsigned integers in Stateflow objects.

This guideline facilitates:

- Readability
- Workflow
- Code Generation

See MAAB guideline jc\_0451: Use of unary minus on unsigned integers in Stateflow

### Analysis Results and Recommended Actions

| Condition                                                                                        | Recommended Action                                                                                             |
|--------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| Unary minus operations are applied to unsigned integers in Stateflow objects.                    | Modify the specified objects to remove dependency on unary minus operations.                                   |
| The Model Advisor could not determine the data types in expressions with unary minus operations. | To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation



## Check for equality operations between floating-point expressions in Stateflow charts

Identify equal to operations (==) in expressions where at least one side of the expression is a floating-point variable or constant.

### Description

Do not use equal to operations with floating-point data types. You can use equal to operations with integer data types.

This guideline facilitates:

- Workflow
- Verification and Validation
- Code Generation

See MAAB guideline jc\_0481: Use of hard equality comparisons for floating point numbers in Stateflow

### Analysis Results and Recommended Actions

| Condition                                                                                                                    | Recommended Action                                                                                                                                                                                     |
|------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Expressions use equal to operations (==) where at least one side of the expression is a floating-point variable or constant. | Modify the specified expressions to avoid equal to operations between floating-point expressions. If an equal to operation is required, a margin of error should be defined and used in the operation. |
| The Model Advisor could not determine the data types in expressions with equality operations.                                | To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions.                                                                                         |

**See Also**

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for mismatches between Stateflow ports and associated signal names

Check for mismatches between Stateflow ports and associated signal names.

### Description

The name of Stateflow input and output should be the same as the corresponding signal. This guideline is required for:

- Readability
- Workflow

See MAAB guideline db\_0123: Stateflow port names.

### Results and Recommended Actions

| Condition                                                                         | Recommended Action                                             |
|-----------------------------------------------------------------------------------|----------------------------------------------------------------|
| Signals have names that differ from those of their corresponding Stateflow ports. | Change the names of either the signals or the Stateflow ports. |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Check for proper scope of From and Goto blocks

Check the scope of From and Goto blocks.

### Description

You can use global scope for controlling flow. However, From and Goto blocks must use local scope for signal flows.

This guideline facilitates

- Readability
- Workflow
- Code generation

See MAAB guideline na\_0011: Scope of Goto and From blocks.

### Results and Recommended Actions

| Condition                                                 | Recommended Action                                                                                                                                          |
|-----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| From and Goto blocks are not configured with local scope. | <ul style="list-style-type: none"><li>• Make sure the ports are connected correctly.</li><li>• Change the scope of the specified blocks to local.</li></ul> |

### See Also

“MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow” in the Simulink documentation

## Requirements Consistency Checks

**In this section...**

“Identify requirement links with missing documents” on page 27-164

“Identify requirement links that specify invalid locations within documents” on page 27-165

“Identify selection-based links having descriptions that do not match their requirements document text” on page 27-166

“Identify requirement links with inconsistent path types and preferences” on page 27-168

## Identify requirement links with missing documents

Ensure that requirements link to existing documents.

### Description

You used the Requirements Management Interface (RMI) to associate a design requirements document with a part of your model design and the interface cannot find the specified document.

### Results and Recommended Actions

| Condition                                                                                                          | Recommended Action                                                                                                                      |
|--------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| The requirements document associated with a part of your model design is not accessible at the specified location. | Open the Requirements dialog box and correct the path name of the requirements document or move the document to the specified location. |

### Tips

If your model has links to a DOORS requirements document, to run this check, the DOORS software must be open and you must be logged in.

### See Also

Chapter 6, “Keeping Requirements Information Up to Date”

## Identify requirement links that specify invalid locations within documents

Ensure that requirements link to valid locations (e.g., bookmarks, line numbers, anchors) within documents.

### Description

You used the Requirements Management Interface (RMI) to associate a location in a design requirements document (a bookmark, line number, or anchor) with a part of your model design and the interface cannot find the specified location in the specified document.

### Results and Recommended Actions

| Condition                                                                                                | Recommended Action                                                                                    |
|----------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| The location in the requirements document associated with a part of your model design is not accessible. | Open the Requirements dialog box and correct the location reference within the requirements document. |

### Tips

If your model has links to a DOORS requirements document, to run this check, the DOORS software must be open and you must be logged in.

If your model has links to a Microsoft Word or Microsoft Excel document, to run this check, those applications must be closed on your computer.

### See Also

Chapter 6, “Keeping Requirements Information Up to Date”

## Identify selection-based links having descriptions that do not match their requirements document text

Ensure that descriptions of selection-based links use the same text found in their requirements documents.

### Description

You used selection-based linking of the Requirements Management Interface (RMI) to label requirements in the model's **Requirements** menu with text that appears in the corresponding requirements document. This check helps you manage traceability by identifying requirement descriptions in the menu that are not synchronized with text in the documents.

### Results and Recommended Actions

| Condition                                                                                                              | Recommended Action                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Selection-based links have descriptions that differ from their corresponding selections in the requirements documents. | If the difference reflects a change in the requirements document, click <b>Update</b> in the Model Advisor results to replace the current description in the selection-based link with the text from the requirements document (the external description). Alternatively, you can right-click the object in the model window, select <b>Edit/Add Links</b> from the <b>Requirements</b> menu, and use the Requirements dialog box that appears to synchronize the text. |

### Tips

If your model has links to a DOORS requirements document, to run this check, the DOORS software must be open and you must be logged in.

If your model has links to a Microsoft Word or Microsoft Excel document, to run this check, those applications must be closed on your computer.



**See Also**

Chapter 4, “Creating and Managing Requirements Links”

## Identify requirement links with inconsistent path types and preferences

Check that requirement paths are of the type selected in the preferences.

### Description

You are using the Requirements Management Interface (RMI) and the paths specifying the location of your requirements documents differ from the file reference type set as your preference.

### Results and Recommended Actions

| Condition                                                                                                                                                                                                             | Recommended Action                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>The paths indicating the location of requirements documents use a file reference type that differs from the preference specified in the Requirements Settings dialog box, on the <b>Selection Linking</b> tab.</p> | <p>Change the preferred document file reference type or the specified paths by doing one of the following:</p> <ul style="list-style-type: none"> <li>• Click <b>Fix</b> to change the current path to the valid path.</li> <li>• In the model window, select <b>Tools &gt; Requirements &gt; Settings</b>, select the <b>Selection Linking</b> tab, and change the value for the <b>Document file reference</b> option.</li> </ul> |

### See Also

Chapter 4, “Creating and Managing Requirements Links”

# Examples

---

Use this list to find examples in the documentation.

## Requirements Management Interface

- “Before You Start: Configuring the RMI for One-Way Linking” on page 3-2
- “Linking to Requirements with Selection-Based Linking” on page 3-4
- “Tutorial: Linking to Requirements in Microsoft Word Documents” on page 3-7
- “Navigating from a Model to a Requirements Document” on page 3-7
- “Creating a Link from a Model Object to a Microsoft Word Requirements Document” on page 3-8
- “Adding Requirement Links to Multiple Model Objects Simultaneously” on page 3-9
- “Tutorial: Linking to Requirements in IBM Rational DOORS Databases” on page 3-11
- “Creating Requirements Reports” on page 3-12
- “Tutorial: Managing Requirements Links to Microsoft® Excel Workbooks” on page 4-3
- “Tutorial: Creating Links to MuPAD Notebooks” on page 4-8
- “Tutorial: Linking Signal Builder Blocks to Requirements” on page 4-10
- “Highlighting Requirements in a Model” on page 5-2
- “Navigating to Requirements from a Model” on page 5-5
- “Creating a Default Requirements Report” on page 5-7
- “Customizing a Requirements Report Using the RMI Settings” on page 5-16
- “Applying a User Tag to a Requirement” on page 5-21
- “Filtering, Highlighting, and Reporting with User Tags” on page 5-23
- “Applying User Tags During Selection-Based Linking” on page 5-25
- “Running Consistency Checks” on page 6-2
- “Fixing Inconsistent Links” on page 6-4
- “Deleting a Single Link from a Simulink Object” on page 6-9
- “Deleting All Links from a Simulink Object” on page 6-9
- “Deleting Links from Multiple Simulink Objects” on page 6-10
- “Managing Requirements in Linked Libraries” on page 6-11
- “Configuring the RMI to Insert Navigation Controls” on page 8-3
- “Enabling Two-Way Linking for Microsoft Office Documents” on page 9-3
- “Inserting Navigation Controls in Microsoft Office Requirements Documents” on page 9-5
- “Navigating Between a Microsoft Word Requirement and a Model” on page 9-6

“Troubleshooting Simulink Navigation Controls in Microsoft Office 2007”  
on page 9-7

“Creating a Custom Link Requirement Type” on page 10-7

Chapter 11, “Including Requirements Information with Generated Code”

## **Requirements Management Interface (DOORS Version)**

“Tutorial: Synchronizing a Simulink Model to Create a Surrogate Module”  
on page 7-5

“Tutorial: Creating Links Between the Surrogate Module and Formal  
Module in a DOORS Database During Synchronization” on page 7-7

“Tutorial: Resynchronizing to Include All Simulink Objects” on page 7-12

“Tutorial: Updating the Surrogate Module to Reflect Model Changes” on  
page 7-16

“Navigating with the Surrogate Module” on page 7-18

“Navigating Between Requirements and the Surrogate Module in the  
DOORS Database” on page 7-18

“Two-Way Navigation with the Surrogate Module” on page 7-19

“Navigating from a Simulink Object to a Requirement via the Surrogate  
Module” on page 7-19

“Navigating from a Requirement to the Model via the Surrogate Module”  
on page 7-20

“Configuring the Requirements Management Interface for DOORS  
Software” on page 8-3

“Enabling Two-Way Linking for IBM Rational DOORS Databases” on  
page 8-6

“Inserting Navigation Controls into DOORS Requirements” on page 8-8

“Navigating Between a DOORS Requirement and a Model Object” on page  
8-10

## **Verification Manager**

“Example: Using the Check Static Lower Bound Block to Check for  
Out-of-Bounds Signal” on page 12-3

“Opening the Verification Manager” on page 13-2

“Enabling and Disabling Model Verification Blocks Using the Verification Manager” on page 13-8

“Using Enabling and Disabling Tools in the Verification Manager” on page 13-11

Chapter 14, “Linking Verification Blocks to Requirements Documents Using the Verification Manager”

## **Model Coverage**

“Creating and Running Test Cases” on page 15-32

“Creating a Model with Embedded MATLAB Function Block Decisions” on page 15-39

“Understanding Embedded MATLAB Function Block Model Coverage” on page 15-43

“Enabling the Colored Diagram Display” on page 15-56

“Displaying Model Coverage with Model Coloring” on page 15-57

“Accessing Coverage Information for Colored Blocks” on page 15-59

“Coverage Summary” on page 17-3

“Details” on page 17-5

“Cyclomatic Complexity” on page 17-13

“Decisions Analyzed” on page 17-15

“Conditions Analyzed” on page 17-17

“MCDC Analysis” on page 17-17

“Cumulative Coverage” on page 17-19

“N-Dimensional Lookup Table” on page 17-21

“Block Reduction” on page 17-28

“Signal Range Analysis” on page 17-30

“Signal Size Coverage for Variable-Dimension Signals” on page 17-32

“Simulink® Design Verifier Coverage” on page 17-34

“External MATLAB File Coverage Reports” on page 17-40

“Subsystem Coverage Reports” on page 17-45

“Coverage Script Example” on page 18-11

## Model Advisor Check

- “Model Advisor Code Example: Registering Custom Checks and Process Callbacks” on page 20-7
- “Model Advisor Code Example: Check Definition Function” on page 20-15
- “Model Advisor Code Example: Input Parameter Definition” on page 20-17
- “Model Advisor Code Example: List View Definition” on page 20-19
- “Model Advisor Code Example: Action Definition” on page 20-20
- “Model Advisor Code Example: Informational Check Callback Function” on page 20-24
- “Model Advisor Code Example: Basic Check with Pass/Fail Status” on page 20-26
- “Model Advisor Code Example: Check With Subchecks and Actions” on page 20-29
- “Model Advisor Code Example: Action Callback Function” on page 20-37
- “Model Advisor Code Example: Formatted Output” on page 20-41

## Model Advisor Organization

- “How To Organize Checks and Folders Using the Model Advisor Configuration Editor” on page 21-10
- “Model Advisor Code Example: Registering Custom Tasks and Folders” on page 21-15
- “Model Advisor Code Example: Task Definition Function” on page 21-17
- “Model Advisor Code Example: Group Definition” on page 21-19





## A

- Abs block
  - model coverage for 15-10
- ActiveX controls
  - deleting from Microsoft Excel documents 9-12
  - enabling, in the Microsoft Office Trust Center 9-6
  - field codes 9-8
  - RMI use in requirements documents 10-18
  - troubleshooting 9-7

## B

- block reduction
  - model coverage and 15-30

## C

- categorical lists of functions 24-1
- classes
  - cv.cvdatagroup 25-15
  - cv.cvtestgroup 25-17
  - ModelAdvisor.Action 25-54
  - ModelAdvisor.Check 25-56
  - ModelAdvisor.FactoryGroup 25-60
  - ModelAdvisor.FormatTemplate 25-62
  - ModelAdvisor.Group 25-70
  - ModelAdvisor.Image 25-72
  - ModelAdvisor.InputParameter 25-74
  - ModelAdvisor.LineBreak 25-77
  - ModelAdvisor.List 25-79
  - ModelAdvisor.ListViewParameter 25-81
  - ModelAdvisor.Paragraph 25-84
  - ModelAdvisor.Root 25-86
  - ModelAdvisor.Table 25-88
  - ModelAdvisor.Task 25-90
  - ModelAdvisor.Text 25-93
- closing Signal Builder Requirements pane 13-7
- colored diagram model coverage display

- enabling 15-56
  - how it works 15-56
- Combinatorial Logic block
  - model coverage for 15-10
- condition coverage
  - description 15-4
  - Embedded MATLAB Function blocks 15-53
  - statements in Embedded MATLAB Function block 15-38
- conditional input branch execution
  - model coverage and 15-30
- conditioninfo function 25-11
- constructors
  - cv.cvdatagroup 25-16
  - cv.cvtestgroup 25-18
  - ModelAdvisor.Action 25-55
  - ModelAdvisor.Check 25-59
  - ModelAdvisor.FactoryGroup 25-61
  - ModelAdvisor.FormatTemplate 25-69
  - ModelAdvisor.Group 25-71
  - ModelAdvisor.Image 25-73
  - ModelAdvisor.InputParameter 25-75
  - ModelAdvisor.LineBreak 25-78
  - ModelAdvisor.List 25-80
  - ModelAdvisor.ListViewParameter 25-83
  - ModelAdvisor.Paragraph 25-85
  - ModelAdvisor.Root 25-87
  - ModelAdvisor.Table 25-89
  - ModelAdvisor.Task 25-92
  - ModelAdvisor.Text 25-94
- Coverage Settings dialog box 15-32
  - accessing 16-2
  - Coverage tab 16-3
  - Options tab 16-12
  - Report tab 16-8
  - Results tab 16-6
- cv.cvdatagroup class 25-15
- cv.cvdatagroup constructor 25-16
- cv.cvdatagroup.allNames method 25-9
- cv.cvdatagroup.get method 25-41

- cv.cvdatagroup.getAll method 25-43
- cv.cvdatagroup.name property 25-184
- cv.cvtestgroup class 25-17
- cv.cvtestgroup constructor 25-18
- cv.cvtestgroup.add method 25-2
- cv.cvtestgroup.allNames method 25-10
- cv.cvtestgroup.get method 25-42
- cv.cvtestgroup.name property 25-185
- cvexit function 25-19
- cvhtml function 25-20
  - model coverage 18-8
- cvload function 25-23
  - model coverage 18-10
- cvmodelview function 25-24
- cvsave function 25-26
  - model coverage 18-9
- cvsim function 25-28
  - model coverage 18-6
- cvsimref function 25-31
- cvtest function 25-34
  - model coverage 18-3
- cyclomatic complexity
  - description 15-3

## D

- Dead Zone block
  - model coverage for 15-11
- decision coverage
  - description 15-4
  - Embedded MATLAB Function blocks 15-52
  - statements in Embedded MATLAB Function blocks 15-37
- decisioninfo function 25-37
- defining Model Advisor checks 20-11
- defining Model Advisor folders 21-18
- defining Model Advisor tasks 21-15
- demos
  - Model Advisor customization demo 21-20
  - simcovdemo model coverage demo 15-32

- Direct Lookup Table (n-D) block
  - model coverage for 15-12
- disabling Model Verification blocks across test groups 13-11
- Discrete-Time Integrator block
  - model coverage for 15-13
- DO-178B
  - Model Advisor checks 27-5
- DOORS Requirements Management Interface
  - block type descriptions 7-13
  - creating one-way links to 3-11
  - definition for object 7-2
  - from Simulink to DOORS 7-19
  - hierarchical numbers 7-13
  - inserting navigation controls into 8-8
  - navigating between model and 8-10
  - object identifiers 7-13
  - opening the object in Simulink, Stateflow, or MATLAB 7-20
  - overview 8-2
  - saving formal modules 7-17
  - synchronizing models with DOORS 7-5
  - synchronizing objects with DOORS formal module 7-5
  - viewing requirements 7-18
- DOORS software
  - installing 8-3
  - manual installation 8-3

## E

- Embedded MATLAB Function block
  - condition coverage 15-53
  - condition coverage statements 15-38
  - decision coverage 15-52
  - decision coverage statements 15-37
  - MCDC coverage 15-53
  - MCDC coverage statements 15-38
  - model coverage 15-37
  - model coverage example 15-39

- model coverage for 15-14
- model coverage for Simulink Design Verifier
  - functions 15-25 15-38
- types of model coverage 15-37
- understanding model coverage for 15-43
- Embedded MATLAB functions
  - model coverage reports 17-10
  - Simulink Design Verifier coverage for 15-7
- Enabled and Triggered Subsystem block
  - model coverage for 15-14
- Enabled Subsystem block
  - model coverage for 15-15
- enabling Model Verification blocks across test groups 13-11

## F

- Fcn block
  - model coverage for 15-16
- field codes
  - requirements in Microsoft Word 9-8
- filtering
  - requirements 5-21
  - settings for 5-27
- For Iterator block
  - model coverage for 15-17
- For Iterator Subsystem block
  - model coverage for 15-17
- formal modules
  - creating links to surrogate modules during synchronization 7-7
- functions
  - categories 24-1
  - conditioninfo 25-11
  - cvexit 25-19
  - cvhtml 25-20
  - cvload 25-23
  - cvmodelview 25-24
  - cvsave 25-26
  - cvsim 25-28

- cvsimref 25-31
- cvtest 25-34
- decisioninfo 25-37
- getCoverageInfo 25-44
- mcdcinfo 25-50
- Model Advisor customization API 23-5 24-3
- Model Advisor formatting API 23-8 24-5
- Model Advisor result template API 23-7 24-4
- model coverage 23-3 24-2
- Requirements Management Interface 23-2
- rmi 25-98
- rmidocrename 25-104
- rmitag 25-106
- sigrangeinfo 25-153
- tableinfo 25-155

## G

- getCoverageInfo function 25-44

## H

- highlighting
  - requirements in a model 5-2

## I

- icons for Model Verification blocks in Verification Manager 13-8
- IEC 61508
  - Model Advisor checks 27-67
- If block
  - model coverage for 15-17
- If Subsystem block
  - model coverage for 15-17
- Interpolation Using Prelookup block
  - model coverage for 15-17

## L

- linked libraries

- requirements in 6-11
- Logical Operator block
  - model coverage for 15-18
- Lookup Table (2-D) block
  - model coverage for 15-19
- Lookup Table (n-D) block
  - model coverage for 15-20
- Lookup Table block
  - in model coverage report 17-21
  - model coverage for 15-19
- lookup table coverage
  - description 15-6
- Lookup Table model coverage
  - n-dimensional 17-28
  - three-dimensional example 17-25
  - two-dimensional example 17-21

## M

- MathWorks Automotive Advisory Board
  - Model Advisor checks 27-87
- MCDC coverage
  - description 15-4
  - Embedded MATLAB Function blocks 15-53
  - statements in Embedded MATLAB Function blocks 15-38
- MCDC table
  - condition cases 17-18
- mcdcinfo function 25-50
- methods
  - cv.cvdatagroup.allNames 25-9
  - cv.cvdatagroup.get 25-41
  - cv.cvdatagroup.getAll 25-43
  - cv.cvtestgroup.add 25-2
  - cv.cvtestgroup.allNames 25-10
  - cv.cvtestgroup.get 25-42
  - ModelAdvisor.Action.setCallbackFcn 25-112
  - ModelAdvisor.Check.getID 25-49
  - ModelAdvisor.Check.setAction 25-109
  - ModelAdvisor.Check.setCallbackFcn 25-113
  - ModelAdvisor.Check.setInputParameters 25-131
  - ModelAdvisor.Check.setInputParameters-LayoutGrid 25-132
  - ModelAdvisor.FactoryGroup.addCheck 25-3
  - ModelAdvisor.FormatTemplate.addRow 25-7
  - ModelAdvisor.FormatTemplate.-setCheckText 25-116
  - ModelAdvisor.FormatTemplate.-setColTitles 25-121
  - ModelAdvisor.FormatTemplate.-setInformation 25-130
  - ModelAdvisor.FormatTemplate.-setListObj 25-134
  - ModelAdvisor.FormatTemplate.-setRecAction 25-135
  - ModelAdvisor.FormatTemplate.-setRefLink 25-137
  - ModelAdvisor.FormatTemplate.-setSubBar 25-143
  - ModelAdvisor.FormatTemplate.-setSubResultStatus 25-144
  - ModelAdvisor.FormatTemplate.-setSubResultStatusText 25-145
  - ModelAdvisor.FormatTemplate.-setSubTitle 25-148
  - ModelAdvisor.FormatTemplate.-setTableInfo 25-149
  - ModelAdvisor.FormatTemplate.-setTableTitle 25-150
  - ModelAdvisor.Group.AddGroup 25-4
  - ModelAdvisor.Group.AddTask 25-8
  - ModelAdvisor.Image.setHyperlink 25-127
  - ModelAdvisor.Image.setImageSource 25-129
  - ModelAdvisor.InputParameter.setColSpan 25-120
  - ModelAdvisor.InputParameter.setRowSpan 25-142
  - ModelAdvisor.List.addItem 25-5
  - ModelAdvisor.List.setType 25-151
  - ModelAdvisor.Paragraph.addItem 25-6
  - ModelAdvisor.Paragraph.setAlign 25-110
  - ModelAdvisor.Root.publish 25-96

- ModelAdvisor.Root.register 25-97
- ModelAdvisor.Table.getEntry 25-48
- ModelAdvisor.Table.setColHeading 25-117
- ModelAdvisor.Table.setColHeadingAlign 25-118
- ModelAdvisor.Table.setColWidth 25-122
- ModelAdvisor.Table.setEntry 25-123
- ModelAdvisor.Table.setEntryAlign 25-124
- ModelAdvisor.Table.setHeading 25-125
- ModelAdvisor.Table.setHeadingAlign 25-126
- ModelAdvisor.Table.setRowHeading 25-140
- ModelAdvisor.Table.setRowHeadingAlign 25-141
- ModelAdvisor.Task.setCheck 25-115
- ModelAdvisor.Text.setBold 25-111
- ModelAdvisor.Text.setColor 25-119
- ModelAdvisor.Text.setHyperlink 25-128
- ModelAdvisor.Text.setItalic 25-133
- ModelAdvisor.Text.setRetainSpace-  
Return 25-139
- ModelAdvisor.Text.setSubscript 25-146
- ModelAdvisor.Text.setSuperscript 25-147
- ModelAdvisor.Text.setUnderlined 25-152
- Microsoft Excel
  - deleting ActiveX controls from 9-12
  - managing requirements links in 4-3
- Microsoft Office Trust Center
  - enabling ActiveX controls 9-6
- Microsoft Word
  - linking to requirements in 3-7
  - requirements documents, linking to 3-8 to  
3-9
  - troubleshooting ActiveX controls 9-7
- MinMax block
  - model coverage for 15-20
- model
  - synchronizing to DOORS surrogate  
module 7-2
- Model Advisor
  - requirements consistency checks 6-2
- Model Advisor checks
  - DO-178B 27-5
  - IEC 61508 27-67
  - MathWorks Automotive Advisory  
Board 27-87
  - requirements consistency 27-163
- Model Advisor customization API functions 23-5
- Model Advisor customization classes 24-3
- Model Advisor customizations
  - creating check callback functions 20-22
  - defining custom checks 20-11
  - defining custom folders 21-18
  - defining custom tasks 21-15
  - defining process callback functions 20-8
  - formatting Model Advisor results 20-38
  - registering custom checks 20-6
  - registering custom tasks and folders 21-13
  - slvrvdemo\_mdladv demo 21-20
  - workflow overview 19-4
- Model Advisor formatting API functions 23-8
- Model Advisor formatting classes 24-5
- Model Advisor result template class 23-7 24-4
- Model block
  - model coverage for 15-21
- model coverage
  - Abs block 15-10
  - block reduction 17-28
  - block reduction effect on 15-30
  - colored Simulink diagram display 15-56 to  
15-57
  - Combinatorial Logic block 15-10
  - condition coverage 15-4
  - conditional input branch execution effect  
on 15-30
  - conditions analyzed table 17-17
  - cumulative coverage 17-19
  - cyclomatic complexity 15-3 17-13
  - Dead Zone block 15-11
  - decision coverage 15-4
  - Decisions analyzed table 17-15
  - Direct Lookup Table (n-D) block 15-12
  - Discrete-Time Integrator block 15-13

- Embedded MATLAB Function block 15-14
    - Simulink Design Verifier functions 15-25
      - 15-38
  - Embedded MATLAB Function blocks 15-37
  - Embedded MATLAB functions 17-10
  - Enabled and Triggered Subsystem block 15-14
  - Enabled Subsystem block 15-15
  - enabling colored Simulink diagram
    - display 15-56
  - Fcn block 15-16
  - For Iterator block 15-17
  - For Iterator Subsystem block 15-17
  - HTML settings 16-9
  - If block 15-17
  - If Subsystem block 15-17
  - Interpolation Using Prelookup block 15-17
  - introduction 15-2
  - Logical Operator block 15-18
  - Lookup Table (2-D) block 15-19
  - Lookup Table (n-D) block 15-20
  - Lookup Table block 15-19
  - Lookup Table block report 17-21
  - lookup table coverage 15-6
  - MCDC analysis 17-17
  - MCDC coverage 15-4
  - MCDC table 17-18
  - MinMax block 15-20
  - Model block 15-21
  - model objects that receive 15-8
  - Multiport Switch block 15-21
  - n-dimensional Lookup Table 17-28
  - Proof Assumption block 15-22
  - Proof Objective block 15-22
  - Rate Limiter block 15-22
  - Relay block 15-23
  - Saturation block 15-24
  - settings in dialog 16-2
  - signal range analysis 17-30
  - signal range coverage 15-6
  - signal size coverage 15-6
  - signal size, for variable dimensions
    - signals 17-32
  - Simulink Design Verifier blocks and functions 17-34
  - Simulink Design Verifier coverage 15-7
  - Simulink Design Verifier functions 17-10
  - Simulink optimizations and 15-30
  - Switch block 15-25
  - SwitchCase Action Subsystem block 15-26
  - SwitchCase block 15-26
  - Test Condition block 15-26
  - Test Objective block 15-26
  - three-dimensional Lookup Table
    - example 17-25
  - triggered models 15-27
  - Triggered Subsystem block 15-28
  - two-dimensional Lookup Table 17-21
  - types 15-3
  - understanding report 17-2
  - While Iterator block 15-29
  - While Iterator Subsystem block 15-29
  - workflow 15-32
- model coverage demo
    - simcovdemo 15-32
  - model coverage functions 23-3 24-2
    - cvhtml 18-8
    - cvload 18-10
    - cvsave 18-9
    - cvsim 18-6
    - cvtest 18-3
  - model objects
    - adding requirements links to multiple 3-9
    - linking to requirements from 3-8
  - Model Verification blocks
    - block appearance 13-9
    - disabling for test groups 13-8
    - enabling for test groups 13-8
    - icons 13-8
    - parameter settings 12-3

- using individually 12-2
- ModelAdvisor.Action class 25-54
- ModelAdvisor.Action constructor 25-55
- ModelAdvisor.Action.Description
  - property 25-163
- ModelAdvisor.Action.Name property 25-186
- ModelAdvisor.Action.setCallbackFcn
  - method 25-112
- ModelAdvisor.Check class 25-56
- ModelAdvisor.Check constructor 25-59
- ModelAdvisor.Check.CallbackContext
  - property 25-159
- ModelAdvisor.Check.CallbackFunction
  - property 25-160
- ModelAdvisor.Check.CallbackStyle
  - property 25-161
- ModelAdvisor.Check.Enable property 25-171
- ModelAdvisor.Check.getID method 25-49
- ModelAdvisor.Check.ID property 25-174
- ModelAdvisor.Check.LicenseName
  - property 25-178
- ModelAdvisor.Check.ListViewVisible
  - property 25-180
- ModelAdvisor.Check.Result property 25-189
- ModelAdvisor.Check.setAction
  - method 25-109
- ModelAdvisor.Check.setCallbackFcn
  - method 25-113
- ModelAdvisor.Check.setInputParameters
  - method 25-131
- ModelAdvisor.Check.setInputParameters-  
LayoutGrid method 25-132
- ModelAdvisor.Check.Title property 25-190
- ModelAdvisor.Check.TitleTips
  - property 25-191
- ModelAdvisor.Check.Value property 25-194
- ModelAdvisor.Check.Visible property 25-197
- ModelAdvisor.FactoryGroup class 25-60
- ModelAdvisor.FactoryGroup constructor 25-61
- ModelAdvisor.FactoryGroup.addCheck
  - method 25-3
- ModelAdvisor.FactoryGroup.Description
  - property 25-164
- ModelAdvisor.FactoryGroup.DisplayName
  - property 25-168
- ModelAdvisor.FactoryGroup.ID
  - property 25-175
- ModelAdvisor.FactoryGroup.MAObj
  - property 25-181
- ModelAdvisor.FormatTemplate class 25-62
- ModelAdvisor.FormatTemplate
  - constructor 25-69
- ModelAdvisor.FormatTemplate.addRow
  - method 25-7
- ModelAdvisor.FormatTemplate.setCheckText
  - method 25-116
- ModelAdvisor.FormatTemplate.setColTitles
  - method 25-121
- ModelAdvisor.FormatTemplate.setInformation
  - method 25-130
- ModelAdvisor.FormatTemplate.setListObj
  - method 25-134
- ModelAdvisor.FormatTemplate.setRecAction
  - method 25-135
- ModelAdvisor.FormatTemplate.setRefLink
  - method 25-137
- ModelAdvisor.FormatTemplate.setSubBar
  - method 25-143
- ModelAdvisor.FormatTemplate.-  
setSubResultStatus method 25-144
- ModelAdvisor.FormatTemplate.-  
setSubResultStatusText method 25-145
- ModelAdvisor.FormatTemplate.setSubTitle
  - method 25-148
- ModelAdvisor.FormatTemplate.setTableInfo
  - method 25-149
- ModelAdvisor.FormatTemplate.setTableTitle
  - method 25-150
- ModelAdvisor.Group class 25-70

- ModelAdvisor.Group constructor 25-71
- ModelAdvisor.Group.AddGroup method 25-4
- ModelAdvisor.Group.AddTask method 25-8
- ModelAdvisor.Group.Description
  - property 25-165
- ModelAdvisor.Group.DisplayName
  - property 25-169
- ModelAdvisor.Group.ID property 25-176
- ModelAdvisor.Group.MAObj property 25-182
- ModelAdvisor.Image class 25-72
- ModelAdvisor.Image constructor 25-73
- ModelAdvisor.Image.Input ArgumentsInput
  - ArgumentsInput ArgumentsInput
  - ArgumentsInput ArgumentsInput
  - ArgumentsInput ArgumentsInput
  - ArgumentsInput ArgumentsInput
  - ArgumentsInput ArgumentsInput
  - ArgumentsInput ArgumentsInput
  - ArgumentsInput ArgumentsInput
  - ArgumentsInput ArgumentsInput
  - ArgumentsInput ArgumentsInput
  - ArgumentsInput ArgumentsInput
  - Arguments method 25-127
- ModelAdvisor.Image.setImageSource
  - method 25-129
- ModelAdvisor.InputParameter class 25-74
- ModelAdvisor.InputParameter
  - constructor 25-75
- ModelAdvisor.InputParameter.Description
  - property 25-166
- ModelAdvisor.InputParameter.Entries
  - property 25-173
- ModelAdvisor.InputParameter.Name
  - property 25-187
- ModelAdvisor.InputParameter.setColSpan
  - method 25-120
- ModelAdvisor.InputParameter.setRowSpan
  - method 25-142
- ModelAdvisor.InputParameter.Type
  - property 25-192
- ModelAdvisor.InputParameter.Value
  - property 25-195
- ModelAdvisor.LineBreak class 25-77
- ModelAdvisor.LineBreak constructor 25-78
- ModelAdvisor.List class 25-79
- ModelAdvisor.List constructor 25-80
- ModelAdvisor.List.addItem method 25-5
- ModelAdvisor.List.setType method 25-151
- ModelAdvisor.ListViewParameter class 25-81
- ModelAdvisor.ListViewParameter
  - constructor 25-83
- ModelAdvisor.ListViewParameter.Attributes
  - property 25-158
- ModelAdvisor.ListViewParameter.Data
  - property 25-162
- ModelAdvisor.ListViewParameter.Name
  - property 25-188
- ModelAdvisor.Paragraph class 25-84
- ModelAdvisor.Paragraph constructor 25-85
- ModelAdvisor.Paragraph.addItem
  - method 25-6
- ModelAdvisor.Paragraph.setAlign
  - method 25-110
- ModelAdvisor.Root class 25-86
- ModelAdvisor.Root constructor 25-87
- ModelAdvisor.Root.publish method 25-96
- ModelAdvisor.Root.register method 25-97
- ModelAdvisor.Table class 25-88
- ModelAdvisor.Table constructor 25-89
- ModelAdvisor.Table.getEntry method 25-48
- ModelAdvisor.Table.setColHeading
  - method 25-117
- ModelAdvisor.Table.setColHeadingAlign
  - method 25-118
- ModelAdvisor.Table.setColWidth
  - method 25-122
- ModelAdvisor.Table.setEntry method 25-123
- ModelAdvisor.Table.setEntryAlign
  - method 25-124
- ModelAdvisor.Table.setHeading
  - method 25-125



ModelAdvisor.Table.setHeadingAlign  
     method 25-126  
 ModelAdvisor.Table.setRowHeading  
     method 25-140  
 ModelAdvisor.Table.setRowHeadingAlign  
     method 25-141  
 ModelAdvisor.Task class 25-90  
 ModelAdvisor.Task constructor 25-92  
 ModelAdvisor.Task.Description  
     property 25-167  
 ModelAdvisor.Task.DisplayName  
     property 25-170  
 ModelAdvisor.Task.Enable property 25-172  
 ModelAdvisor.Task.ID property 25-177  
 ModelAdvisor.Task.LicenseName  
     property 25-179  
 ModelAdvisor.Task.MAObj property 25-183  
 ModelAdvisor.Task.setCheck method 25-115  
 ModelAdvisor.Task.Value property 25-196  
 ModelAdvisor.Task.Visible property 25-198  
 ModelAdvisor.Text class 25-93  
 ModelAdvisor.Text constructor 25-94  
 ModelAdvisor.Text.setBold method 25-111  
 ModelAdvisor.Text.setColor method 25-119  
 ModelAdvisor.Text.setHyperlink  
     method 25-128  
 ModelAdvisor.Text.setItalic method 25-133  
 ModelAdvisor.Text.setRetainSpaceReturn  
     method 25-139  
 ModelAdvisor.Text.setSubscript  
     method 25-146  
 ModelAdvisor.Text.setSuperscript  
     method 25-147  
 ModelAdvisor.Text.setUnderlined  
     method 25-152  
 models  
     highlighting requirements in 5-2  
     linking to requirements from 3-9  
     navigating to requirements documents  
         from 5-5

        navigating to, from external  
             documents 10-17  
         running test cases 15-32  
 Multiport Switch block  
     model coverage for 15-21  
 MuPAD notebooks  
     linking from models to 4-8

## N

navigating  
     between model and DOORS 8-10  
     from model to requirements documents 5-5  
 navigation controls  
     in requirements 8-2 9-2  
 notebooks, MuPAD  
     linking from models to 4-8

## O

opening a Signal Builder block 13-4  
 operating system requirements 1-3

## P

parameters for Model Verification blocks 12-3  
 Proof Assumption block  
     model coverage for 15-22  
 Proof Objective block  
     model coverage for 15-22  
 properties  
     cv.cvdatagroup.name 25-184  
     cv.cvtestgroup.name 25-185  
     ModelAdvisor.Action.Description 25-163  
     ModelAdvisor.Action.Name 25-186  
     ModelAdvisor.Check.CallbackContext 25-159  
     ModelAdvisor.Check.CallbackFunction 25-160  
     ModelAdvisor.Check.CallbackStyle 25-161  
     ModelAdvisor.Check.Enable 25-171  
     ModelAdvisor.Check.ID 25-174  
     ModelAdvisor.Check.LicenseName 25-178

ModelAdvisor.Check.ListViewVisible 25-180  
 ModelAdvisor.Check.Result 25-189  
 ModelAdvisor.Check.Title 25-190  
 ModelAdvisor.Check.TitleTips 25-191  
 ModelAdvisor.Check.Value 25-194  
 ModelAdvisor.Check.Visible 25-197  
 ModelAdvisor.FactoryGroup.Description 25-164  
 ModelAdvisor.FactoryGroup.DisplayName 25-168  
 ModelAdvisor.FactoryGroup.ID 25-175  
 ModelAdvisor.FactoryGroup.MAObj 25-181  
 ModelAdvisor.Group.Description 25-165  
 ModelAdvisor.Group.DisplayName 25-169  
 ModelAdvisor.Group.ID 25-176  
 ModelAdvisor.Group.MAObj 25-182  
 ModelAdvisor.InputParameter.-  
     Description 25-166  
 ModelAdvisor.InputParameter.Entries 25-173  
 ModelAdvisor.InputParameter.Name 25-187  
 ModelAdvisor.InputParameter.Type 25-192  
 ModelAdvisor.InputParameter.Value 25-195  
 ModelAdvisor.ListViewParameter.-  
     Attributes 25-158  
 ModelAdvisor.ListViewParameter.Data 25-162  
 ModelAdvisor.ListViewParameter.Name 25-188  
 ModelAdvisor.Task.Description 25-167  
 ModelAdvisor.Task.DisplayName 25-170  
 ModelAdvisor.Task.Enable 25-172  
 ModelAdvisor.Task.ID 25-177  
 ModelAdvisor.Task.LicenseName 25-179  
 ModelAdvisor.Task.MAObj 25-183  
 ModelAdvisor.Task.Value 25-196  
 ModelAdvisor.Task.Visible 25-198

## R

Rate Limiter block  
     model coverage for 15-22  
 Relay block  
     model coverage for 15-23  
 report

model coverage HTML options 16-9  
 understanding model coverage report 17-2  
 reports  
     model coverage 17-3  
         block reduction 17-28  
         conditions analyzed 17-17  
         coverage summary 17-3  
         cumulative coverage 17-19  
         cyclomatic complexity 17-13  
         decisions analyzed 17-15  
         details 17-5  
         Lookup Table block coverage 17-21  
         MCDC analysis 17-17  
         sections 17-3  
         Signal range analysis 17-30  
         signal size 17-32  
         Simulink Design Verifier blocks and  
             functions 17-34  
 requirements  
     adding navigation controls to 8-2 9-2  
     adding to test groups 14-1  
     adding, to multiple objects 3-9  
     applying user tags with 5-21  
     default reports 3-12  
     deleting  
         all links from an object 6-9  
         from multiple objects 6-10  
         one at a time 6-9  
     filtering  
         settings for 5-27  
     filtering with user tags 5-21  
     fixing inconsistent links to 6-4  
     for Model Verification block settings 14-1  
     highlighting 5-2  
     identifying inconsistent links to 6-4  
     in generated code 11-1  
     in linked libraries 6-11  
     in subsystems 5-2  
     inserting navigation controls into 8-8  
     linking from Signal Builder blocks to 4-10

- navigating to 5-5
  - navigating to, from System Requirements
    - block 5-5
  - one-way linking to 3-2
  - reports
    - creating default 5-7
    - customizing with Simulink Report Generator software 5-16
    - customizing with the RMI 5-16
    - sections 5-8
  - RMI for DOORS 8-2
  - running consistency checks for 6-2
  - selection-based linking to 3-4
  - two-way linking 8-6 9-3
  - viewing for test groups 14-3
  - requirements consistency
    - Model Advisor checks 27-163
  - requirements documents
    - ActiveX controls in 10-18
    - creating index 10-15
    - custom link types 10-2
      - creating 10-7
      - properties 10-5
      - registering 10-3
    - custom links
      - properties 10-4
    - linking to, from model objects 3-8 to 3-9
    - opening from Simulink model 3-7
    - resolving paths to 6-7
    - supported types 2-4
  - requirements links 2-3
  - Requirements Management Interface
    - configuring for one-way linking 3-2
    - configuring for two-way linking 8-3
    - configuring to insert navigation controls 8-3
    - default requirements report 3-12
    - overview 2-2
    - registering custom requirements
      - documents 10-3
  - Requirements Management Interface for DOORS
    - block type descriptions 7-13
    - definition of object in DOORS 7-2
    - from Simulink to DOORS 7-19
    - hierarchical numbers 7-13
    - object identifiers 7-13
    - opening the object in Simulink or Stateflow 7-20
    - overview 8-2
    - saving formal modules 7-17
    - synchronizing models with DOORS 7-5
    - synchronizing objects with DOORS formal module 7-5
    - viewing requirements 7-18
  - Requirements pane for Verification Manager 14-1
  - Requirements Settings dialog box
    - Filters tab 5-27
  - RMI. *See* Requirements Management Interface
  - rmi function 25-98
  - rmidocrename function 25-104
  - rmitag function 25-106
- S**
- Saturation block
    - model coverage for 15-24
  - selection-based linking 3-4
  - Signal Builder block
    - linking to requirements from 4-10
    - opening 13-4
  - Signal Builder dialog box
    - closing Verification Manager Requirements pane 13-7
  - signal range analysis report in model
    - coverage 17-30
  - signal range coverage
    - description 15-6
  - signal size coverage
    - description 15-6
  - sigrangeinfo function 25-153

- simcovdemo
    - model coverage demo 15-32
  - Simulink
    - optimizations
      - model coverage and 15-30
  - Simulink Design Verifier coverage
    - description 15-7
  - Simulink Design Verifier functions
    - model coverage reports 17-10
  - slvnvdemo\_mdldv
    - Model Advisor customization demo 21-20
  - subsystems
    - highlighting requirements in 5-2
  - surrogate modules
    - characteristics 7-14
    - creating links to formal modules during synchronization 7-7
  - Switch block
    - model coverage for 15-25
  - SwitchCase Action Subsystem block
    - model coverage for 15-26
  - SwitchCase block
    - model coverage for 15-26
  - synchronization
    - advantages 7-4
    - creating links between surrogate and formal modules during 7-7
    - customizing level of detail 7-11
    - definition 7-2
    - resynchronizing 7-10 7-12
    - settings 7-8
    - Simulink model to DOORS surrogate module 7-2
  - synchronizing models with DOORS 7-5
  - system requirements 1-3
    - IBM Rational DOORS 1-3
    - MATLAB 1-3
    - Microsoft Excel 1-3
    - Microsoft Word 1-3
    - operating system 1-3
    - Simulink 1-3
    - Stateflow 1-3
  - System Requirements block 5-5 26-2
- T**
- tableinfo function 25-155
  - test case commands 15-32
  - Test Condition block
    - model coverage for 15-26
  - test groups
    - adding requirements 14-1
    - disabling Model Verification blocks 13-8
    - enabling Model Verification blocks 13-8
    - Model Verification blocks enabled
      - across 13-11
  - Test Objective block
    - model coverage for 15-26
  - triggered models
    - model coverage for 15-27
  - Triggered Subsystem block
    - model coverage for 15-28
  - two-way linking
    - advantages of 8-2 9-2
    - configuring RMI for 8-3
    - configuring the RMI for 8-3
    - enabling 8-6 9-3
- U**
- user tags
    - applying with requirements 5-21
    - definition 5-21
- V**
- variable-dimension signals
    - model coverage for 17-32
  - verification blocks
    - example of use 12-3
    - icons 13-8

- requirements for test groups 14-1
- stopping simulation 12-4
- Verification Manager
  - closing Requirements pane 13-7
  - disabling Model Verification blocks for test groups 13-8
  - enabled/disabled block appearance 13-9
  - enabling Model Verification blocks for test groups 13-8
  - flat display 13-8
  - hierarchical display 13-8

- icons for Model Verification blocks 13-8
- opening 13-2
- Requirements pane 14-1

## **W**

- While Iterator block
  - model coverage for 15-29
- While Iterator Subsystem block
  - model coverage for 15-29