

Requirements Toolbox™

User's Guide



MATLAB® & SIMULINK®

R2023a



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Requirements Toolbox™ User's Guide

© COPYRIGHT 2017–2023 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

September 2017	Online only	New for Version 1.0 (Release 2017b)
March 2018	Online only	Revised for Version 1.1 (Release 2018a)
September 2018	Online only	Revised for Version 1.2 (Release R2018b)
March 2019	Online only	Revised for Version 1.3 (Release R2019a)
September 2019	Online Only	Revised for Version 1.4 (Release 2019b)
March 2020	Online only	Revised for Version 1.5 (Release 2020a)
September 2020	Online only	Revised for Version 1.6 (Release 2020b)
March 2021	Online only	Revised for Version 1.7 (Release 2021a)
September 2021	Online only	Revised for Version 1.8 (Release 2021b)
March 2022	Online only	Revised for Version 2.0 (Release 2022a)
September 2022	Online only	Revised for Version 2.1 (Release 2022b)
March 2023	Online only	Revised for Version 2.2 (Release 2023a)

Requirements Definition

Author Requirements in MATLAB or Simulink	1-2
Author and Edit Requirements Content by Using Microsoft Word	1-4
Customize Requirements Browser View	1-4
Filter Requirements Content	1-4
Requirement Types	1-6
Built-in Requirement Types	1-6
Custom Requirement Types	1-6
Set the Requirement Type	1-6
Import Requirements from Third-Party Applications	1-8
Add Requirements to the Path	1-8
Select an Import Mode	1-8
Differences Between Importing and Direct Linking	1-11
Import Requirements from Microsoft Office Documents	1-12
Import Options for Microsoft Word Documents	1-12
Import Options for Microsoft Excel Spreadsheets	1-14
Import Requirements from ReqIF Files	1-17
Choosing an Import Mapping	1-17
Importing Requirements	1-19
Importing Links	1-23
Mapping ReqIF Attributes in Requirements Toolbox	1-24
Navigate from Referenced Requirements to Requirements in Third-Party Applications	1-27
Use Stereotypes when Importing from ReqIF Files	1-28
Create Profile During Import	1-28
Use Existing Profile During Import	1-30
Adjust Stereotype Mapping	1-32
Import Requirements from IBM DOORS Next	1-35
Configure IBM DOORS Next Session	1-35
Import DOORS Next Requirements	1-36
Update Referenced Requirements	1-39
Navigate from Referenced Requirements to Requirements in DOORS Next	1-40
Linking with Referenced Requirements	1-41
Import Requirements from IBM Rational DOORS	1-42
Configure IBM Rational DOORS Session	1-42
Import an Entire Requirements Module	1-42
Import a Subset of Requirements from a Module	1-44

Update the Requirement Set	1-45
Navigate Between Referenced Requirements and Requirements in IBM Rational DOORS	1-45
Define Custom Document Interface for Importing Requirements	1-47
Define Custom Document Interface	1-47
Define Custom Version of Built-In Document Interface	1-58
Use Custom Document Interface During Import	1-59
Export Requirements to ReqIF Files	1-61
Choosing an Export Mapping	1-61
Exporting Requirements	1-63
Exporting Links	1-64
Define Requirements Hierarchy	1-66
Requirement Sets	1-66
Custom Attributes of Requirement Sets	1-66
Create Requirement Set Hierarchies by Using the Requirements Toolbox API	1-68
Customize Requirements and Links by Using Stereotypes	1-71
Create a Profile and Define Stereotypes	1-71
Add Properties to Stereotypes	1-73
Assign Profiles and Use Stereotypes	1-73
Edit Profiles and Stereotypes	1-75
Manage and Remove Profiles	1-77
Define Custom Requirement and Link Types and Properties	1-78
Define Custom Requirement and Link Types	1-78
Define Custom Properties for Requirements and Links	1-79
Choose a Built-in Type as a Base Behavior	1-80
Add Custom Attributes to Requirements	1-81
Define Custom Attributes for Requirement Sets	1-81
Set Custom Attribute Values for Requirements	1-82
Edit Custom Attributes	1-83
Custom Attributes for Referenced Requirements	1-83
Import Custom Attributes	1-84
Customize Requirement Index Numbering	1-85
Disable Index Numbering	1-85
Set the Index to a Specified Value	1-86
Programmatically Customize Index Numbering	1-86
Update Imported Requirements	1-89
Update a Requirement Set	1-89
Update Requirements with Change Tracking Enabled	1-89
Considerations for Microsoft Word Documents	1-90
Filter Requirements and Links in the Requirements Editor	1-91
Create Views	1-91
Define Requirement and Link Filters	1-91
Apply Views	1-92
Manage Views	1-93

Share Views	1-93
Import and Edit Requirements from a Microsoft Word Document	1-95
Export Requirement Sets and Link Sets to Previous Versions of Requirements Toolbox	1-98
Export Requirement Sets	1-98
Export Link Sets	1-98
Round-Trip Importing and Exporting for ReqIF Files	1-99
Considerations for Importing Requirements	1-99
Edit Imported Content	1-99
Link Requirements to Items in MATLAB and Simulink	1-101
Considerations for Exporting Requirements	1-101
Best Practices and Guidelines for ReqIF Round-Trip Workflows	1-103
Managing Requirement Custom IDs	1-103
Guidelines for Updating Referenced Requirements Content	1-103
Guidelines for Editing Referenced Requirements Content	1-103
Guidelines for Adding Details to Imported Requirements	1-103
Guidelines for Exporting Requirements to ReqIF Files	1-103
Manage Custom Attributes for Requirements by Using the Requirements Toolbox API	1-105
Create and Edit Attribute Mappings	1-110
Edit the Attribute Mapping for Imported Requirements	1-110
Specify Default ReqIF Requirement Type	1-112
Specify ReqIF Template	1-112
Use Callbacks to Customize Requirement Import Behavior	1-113
Assign Code to Callbacks	1-113
Customize Requirement Import Behavior	1-115
Execute Code When Loading and Saving Requirement Sets	1-117
Assign Code to Callbacks	1-117
Customize Requirement Set Load and Save Behavior	1-118
Import Requirements from IBM Rational DOORS by Using the API ..	1-119
Import Requirements from a Microsoft Excel Document	1-125
Export Requirement and Link Information to Excel	1-131
Use Command-Line API to Document Simulink Model in Requirements Editor	1-136
Capture Model as Requirements and Create Links	1-138
Import Requirements and Reuse Existing Links	1-143
Programmatically Repair Broken Links	1-145

Use a Requirements Table Block to Create Formal Requirements	2-2
Define Formal Requirements	2-2
Create Expressions for Formal Requirements	2-3
Use Functions in Requirements Table Blocks	2-5
Define Block Data	2-5
Observe or Generate Model Behavior	2-6
Add Assumptions to Requirements	2-10
Use Assumptions in an Example Model	2-10
Create Requirements Table Blocks Programmatically	2-13
Create a Requirements Table Block	2-13
Specify Requirements Table Block Name and Column Headers	2-13
Define Data	2-14
Add and Modify Requirements	2-14
Manage Requirements as slreq.Requirement Objects	2-15
Identify Inconsistent and Incomplete Formal Requirement Sets	2-17
Analyze the Block	2-17
Analyze Rigid and Flexible Requirements	2-20
Debug Requirements Table Blocks	2-22
Add a Breakpoint to a Requirement Set	2-22
View the Simulation Status at the Breakpoints	2-24
Watch Data Values During Simulation	2-25
Resolve Cell Errors	2-26
Configure Properties of Formal Requirements	2-27
Open the Requirements in the Requirements Editor	2-27
Manage Requirements In the Requirements Editor and Property Inspector	2-28
Link Requirements	2-30
Create a Traceability Matrix and Traceability Diagram	2-31
Control Requirement Execution by Using Temporal Logic	2-33
Using the Duration Column	2-33
Use Temporal Logic Operators	2-35
Detect Data Changes by Using Requirements Table Blocks	2-39
Change Detection Operators	2-39
Example of Requirements Table Block with Change Detection	2-40
Leverage Evaluation Order of Formal Requirements	2-43
Example Using Requirement Order	2-43
Establish Hierarchy in Requirements Table Blocks	2-45
Add Child Rows	2-45
Add Semantic Rows	2-46
Specify Requirements Table Block Properties	2-50
Requirements Table Block Properties	2-50

Fixed-Point Properties	2-51
Description and Document Link Properties	2-52
Define Data in Requirements Table Blocks	2-53
Create and Delete Data	2-53
Set General Data Properties	2-53
Set Limit Range Properties	2-57
Set Logging Properties	2-57
Set Description Properties	2-58
Set Data Types in Requirements Table Blocks	2-59
Specify Data Types	2-59
Inheriting Data Types	2-60
Specify Built-In Data Types	2-60
Fixed-Point Designer Data Properties	2-61
Enumerated Data Types	2-65
Bus Objects	2-65
Expression Data Types	2-65
Specify Size of Requirements Table Block Data	2-67
Inherit Size from Specified Source	2-67
Customize Data Sizes	2-67
What Is a Specification Model?	2-69
Use Specification Models in Requirements-Based Testing	2-69
Construct a Specification Model	2-70
Iterate Through the Steps	2-74
Export Tests from Models That Contain Requirements Table Blocks with Simulink Design Verifier	2-75
Construct the Model and Generate Tests	2-75
Export the Tests to the Test Manager	2-76
Run the Tests	2-78
Inspect Test Failures	2-78
Model Dice Game Rules Using a Requirements Table Block	2-80
Analyze Formal Requirements of an Electric Vehicle Charger Locking Mechanism	2-83
Define Formal Requirements with a Duration	2-89
Use Specification Models for Requirements-Based Testing	2-91
Prove Properties with Requirements Table Blocks	2-102

Requirements Traceability and Consistency

3

Link Blocks and Requirements	3-2
Work with Simulink Annotations	3-4

Track Requirement Links with a Traceability Matrix	3-5
Generate a Traceability Matrix	3-5
Modify the Traceability Matrix View	3-9
Work with Links in the Traceability Matrix	3-13
Export the Traceability Matrix	3-16
Work Programmatically with a Traceability Matrix	3-16
Visualize Links with Traceability Diagrams	3-17
Generate Traceability Diagrams	3-17
Use the Traceability Diagram	3-21
Modify the Traceability Diagram View	3-22
Export the Diagram	3-24
Assess Allocation and Impact	3-26
Assess Requirements Allocation	3-26
Visualize Change Propagation	3-28
Create and Store Links	3-31
Link Objects, Sources, and Destinations	3-31
Link Storage	3-31
Linkable Items	3-32
Create Links	3-33
Link Types	3-34
View and Edit Links	3-37
Delete Links and Link Sets	3-37
Load and Resolve Links	3-39
Load Artifacts and Associated Link Sets	3-39
Load Registered Requirement Sets	3-39
Unresolved Links	3-40
Unload Link Information	3-41
Define Custom Requirement and Link Types by Using sl_customization	
Files	3-42
Create and Register Custom Requirement and Link Types	3-42
Set the Type in the Requirements Editor	3-43
Add Custom Attributes to Links	3-44
Define Custom Attributes for Link Sets	3-44
Set Custom Attribute Values for Links	3-45
Edit Custom Attributes	3-46
Requirements Consistency Checks	3-47
Check Requirements Consistency in Model Advisor	3-47
Manage Navigation Backlinks in External Requirements Documents ..	3-51
Insert Backlinks in External Requirements Documents	3-51
Requirements Traceability for Code Generated from MATLAB Code ...	3-53
Include Requirement Comments in Generated Code	3-53
View Comments in Generated Code	3-55
Navigate to Requirements from Code Generation Report	3-56
Link from Sequence Diagrams	3-58
Create Links	3-58

View Links	3-59
Use Command-Line API to Update or Repair Requirements Links	3-61
Manage Custom Attributes for Links by Using the Requirements Toolbox API	3-73
Make Requirements Fully Traceable with a Traceability Matrix	3-78
Modeling System Architecture of Small UAV	3-91
Model-Based Systems Engineering for Space-Based Applications	3-97

Requirements-Based Verification

4

Review Requirements Implementation Status	4-2
Implement Functional Requirements by Linking to Model Elements	4-2
Run Link Implementation Analysis	4-3
View the Implementation Status	4-4
Review Requirements Verification Status	4-6
Verify Functional Requirements	4-6
Run Link Verification Analysis	4-7
Display Verification Status	4-7
Update Verification Status by Running Tests or Analyses	4-8
Include Verification Status in Report	4-9
Validate Requirements by Analyzing Model Properties	4-10
Justify Requirements	4-17
Linking to a Test Script	4-20
Linking to a Test Script Using the Outgoing Links Editor	4-20
Linking to a Test Script Using the API	4-23
Integrating Results from a MATLAB Unit Test Case	4-26
Include Results from External Sources in Verification Status	4-28
How to Populate Verification Results from External Sources	4-28
Linking to a Result File	4-31
Open Example Files	4-31
Create and Register a Custom Link Type	4-32
Create a Requirement Link	4-33
View the Verification Status	4-35
Integrating Results from a Custom-Authored MATLAB Script as a Test	4-37
Integrating Results from an External Result File	4-41

Integrating Results from a Custom Authored MUnit Script as a Test . . .	4-45
Fix Requirements-Based Testing Issues	4-49

Change Tracking and Team-Based Workflows

5

Requirements-Based Development in Projects	5-2
Organizing Requirements, Models, and Tests	5-2
Track Changes to Requirement Links	5-3
Enable Change Tracking for Requirement Links	5-3
Run Change Tracking Analysis	5-3
Review Changes to Requirements, Test Objects, and MATLAB Code Lines	5-3
Resolve Change Issues	5-7
Add Comments to Links	5-8
Manually Check for Using Links Change Tracking	5-9
Compare Requirement Sets	5-10
Compare two Requirements Toolbox requirement sets	5-10
Review Changes in Source-Controlled Files	5-10
Compare Link Sets	5-11
Report Requirements Information	5-12
Report Navigation Links	5-14
Three-Way AutoMerge Solution for Requirement Set and Link Set	5-15
Configure Git Environment for AutoMerge	5-15
Select and Merge Branches in Git	5-15
Limitations	5-15
Merge Requirement Set and Link Set Files	5-17
Track Changes to Test Cases in Requirements Editor	5-21
Track Changes to MATLAB Code Using Requirements Editor	5-24
Verify Safety Requirements Linked to Test Steps Using Functional Requirements	5-29
Publish and Save Printable Report of Comparison Results	5-36
Publish Report from Comparison Tool	5-36
Publish Report from Command Line	5-36

6

Configure Requirements Toolbox for Interaction with Microsoft Office and IBM DOORS	6-2
Configure Requirements Toolbox for Microsoft Office	6-2
Configure Requirements Toolbox for IBM Rational DOORS	6-2
Configure an IBM DOORS Next Server for Integration with Requirements Toolbox	6-2
Requirements Link Storage	6-4
Save Requirements Links in External Storage	6-4
Load Requirements Links from External Storage	6-5
Move Internally Stored Requirements Links to External Storage	6-5
Move Externally Stored Requirements Links to the Model File	6-5
External Storage	6-6
Guidelines for External Storage of Requirements Links	6-6
Copying Model Objects and their Linked Requirements	6-7
Supported Requirements Document Types	6-8
Requirements Settings	6-10
Selection Linking Tab	6-10
Filter Requirements with User Keywords	6-11
Migrating Requirements Management Interface Data to Requirements Toolbox	6-16

Microsoft Office Traceability

7

Link to Requirements in Microsoft Word Documents	7-2
Link a Requirement in Word to a Simulink Block	7-2
Link to Requirements in Microsoft Excel	7-7
Link and Navigate to Requirements in Excel from Simulink blocks	7-7
Navigate to Requirements in Microsoft Office Documents from Simulink	7-11
Enable Linking from Microsoft Office Documents to Simulink Objects ...	7-11
Insert Navigation Objects in Microsoft Office Documents	7-12
Customize Microsoft Office Navigation Objects	7-12
Navigate Between Microsoft Office Requirement and Model	7-13
Managing Requirements for Fault-Tolerant Fuel Control System (Microsoft Office)	7-15

Requirements Traceability with IBM Rational DOORS

8

Configure Requirements Toolbox for IBM Rational DOORS Software . . .	8-2
Manually Install Additional Files for DOORS Software	8-2
Address DXL Errors	8-3
Link and Trace Requirements with IBM DOORS Next	8-4
Configure IBM DOORS Next Session	8-4
Linking with Referenced Requirements	8-5
Directly Linking DOORS Next Requirements	8-6
Specifying and Updating the IBM DOORS Next Configuration	8-12
Navigate to Requirements in IBM Rational DOORS Databases from	
Simulink	8-15
Enable Linking from IBM Rational DOORS Databases to Simulink Objects	8-15
Insert Navigation Objects into IBM Rational DOORS Requirements	8-15
Navigate Between IBM Rational DOORS Requirement and Model Object	8-17
Why Add Navigation Objects to IBM Rational DOORS Requirements?	8-18
Customize IBM Rational DOORS Navigation Objects	8-18
Synchronize Simulink Models with IBM Rational DOORS Databases by using Surrogate Modules	8-19
Synchronize a Simulink Model to Create a Surrogate Module	8-19
Create Links Between Surrogate Module and Formal Module in an IBM Rational DOORS Database	8-20
Resynchronize IBM Rational DOORS Surrogate Module to Reflect Model Changes	8-21
Navigate with the Surrogate Module	8-21
Customize IBM Rational DOORS Synchronization	8-23
Synchronization with IBM Rational DOORS Surrogate Modules	8-28
Advantages of Synchronizing Your Model with a Surrogate Module	8-29
Working with IBM Rational DOORS 9 Requirements	8-30
Managing Requirements for Fault-Tolerant Fuel Control System (IBM Rational DOORS)	8-39

Simulink Traceability Between Model Objects

9

Link Model Objects	9-2
Link Objects in the Same Model	9-2
Link Objects in Different Models	9-2
Link Test Cases to Requirements Documents	9-3
Establish Requirements Traceability for Testing	9-3
Link Simulink Data Dictionary Entries to Requirements	9-7

Link Signal Builder Blocks to Requirements and Simulink Model Objects	9-9
Link Signal Editor Blocks to Requirements Documents	9-9
Link Signal Builder Blocks to Model Objects	9-10
Requirements Links for Library Blocks and Reference Blocks	9-13
Introduction to Library Blocks and Reference Blocks	9-13
Library Blocks and Requirements	9-13
Copy Library Blocks with Requirements	9-13
Manage Requirements on Reference Blocks	9-13
Manage Requirements Inside Reference Blocks	9-14
Links from Requirements to Library Blocks	9-15
Navigate to Requirements from Model	9-16
Navigate from Model Object	9-16
Navigate from System Requirements Block	9-16
Link to Requirements Modeled in Simulink	9-18

MATLAB Code Traceability

10

Requirements Traceability for MATLAB Code	10-2
Create Links to MATLAB Code or Plain-Text External Code	10-2
Verify Requirements with MATLAB Tests	10-4
View and Edit Links and Linked Line Ranges	10-6
Save Links	10-7
Delete Links and Unused Line Ranges	10-7
Associate Traceability Information with MATLAB Code Lines in Simulink	10-9

URL and Custom Traceability

11

Requirement Links and Link Types	11-2
Requirements Traceability Links	11-2
Supported Model Objects for Requirements Linking	11-2
Links and Link Types	11-2
Link Type Properties	11-3
Outgoing Links Editor	11-6
Custom Link Types	11-8
Create a Custom Requirements Link Type	11-8
Implement Custom Link Types	11-13
Why Create a Custom Link Type?	11-14
Custom Link Type Functions	11-14
Custom Link Type Registration	11-15
Custom Link Type Synchronization	11-15

12

Review and Maintain Requirements Links

Highlight Model Objects with Requirements	12-2
Highlight Model Objects with Requirements Using Model Editor	12-2
Highlight Model Objects with Requirements Using Model Explorer	12-3
Navigate to Simulink Objects from External Documents	12-4
Provide Unique Object Identifiers	12-4
Use the rmiobjnavigate Function	12-4
Determine the Navigation Command	12-4
Use the ActiveX Navigation Control	12-4
Typical Code Sequence for Establishing Navigation Controls	12-5
View Requirements Details for a Selected Block	12-6
Identify Blocks with Links	12-6
Configure Settings	12-6
View Requirements Details	12-6
Create a Requirement Annotation	12-7
Generate Code for Models with Requirements Links	12-8
How Requirements Information Is Included in Generated Code	12-9
Create and Customize Requirements Traceability Reports	12-11
Create Requirements Traceability Report for Model	12-11
Customize Requirements Traceability Report for Model	12-12
Create Requirements Traceability Report for A Project	12-26
Validate Requirements Links	12-27
Validate Requirements Links in a Model	12-27
Validate Requirements Links in a Requirements Document	12-31
Validation of Requirements Links	12-33
Delete Requirements Links from Simulink Objects	12-35
Delete a Single Link from a Simulink Object	12-35
Delete All Links from a Simulink Object	12-35
Delete All Links from Multiple Simulink Objects	12-35
Document Path Storage	12-36
Relative (Partial) Path Example	12-36
Relative (No) Path Example	12-36
Absolute Path Example	12-36
How to Include Linked Requirements Details in Generated Report . . .	12-38
Managing Requirements Without Modifying Simulink Model Files . . .	12-45
Compare Requirement Sets Using Comparison Tool	12-50

- Test Model Against Requirements and Report Results** 13-2
 - Requirements - Test Traceability Overview 13-2
 - Display the Requirements 13-2
 - Link Requirements to Tests 13-3
 - Run the Test 13-4
 - Report the Results 13-5

- Analyze Models for Standards Compliance and Design Errors** 13-7
 - Standards and Analysis Overview 13-7
 - Check Model for Style Guideline Violations and Design Errors 13-7

- Perform Functional Testing and Analyze Test Coverage** 13-9
 - Incrementally Increase Test Coverage Using Test Case Generation 13-9

- Analyze Code and Test Software-in-the-Loop** 13-12
 - Code Analysis and Testing Software-in-the-Loop Overview 13-12
 - Analyze Code for Defects, Metrics, and MISRA C:2012 13-12
 - Test Code Against Model Using Software-in-the-Loop Testing 13-17

Requirements Definition

- “Author Requirements in MATLAB or Simulink” on page 1-2
- “Requirement Types” on page 1-6
- “Import Requirements from Third-Party Applications” on page 1-8
- “Import Requirements from Microsoft Office Documents” on page 1-12
- “Import Requirements from ReqIF Files” on page 1-17
- “Use Stereotypes when Importing from ReqIF Files” on page 1-28
- “Import Requirements from IBM DOORS Next” on page 1-35
- “Import Requirements from IBM Rational DOORS” on page 1-42
- “Define Custom Document Interface for Importing Requirements” on page 1-47
- “Export Requirements to ReqIF Files” on page 1-61
- “Define Requirements Hierarchy” on page 1-66
- “Create Requirement Set Hierarchies by Using the Requirements Toolbox API” on page 1-68
- “Customize Requirements and Links by Using Stereotypes” on page 1-71
- “Define Custom Requirement and Link Types and Properties” on page 1-78
- “Add Custom Attributes to Requirements” on page 1-81
- “Customize Requirement Index Numbering” on page 1-85
- “Update Imported Requirements” on page 1-89
- “Filter Requirements and Links in the Requirements Editor” on page 1-91
- “Import and Edit Requirements from a Microsoft Word Document” on page 1-95
- “Export Requirement Sets and Link Sets to Previous Versions of Requirements Toolbox” on page 1-98
- “Round-Trip Importing and Exporting for ReqIF Files” on page 1-99
- “Best Practices and Guidelines for ReqIF Round-Trip Workflows” on page 1-103
- “Manage Custom Attributes for Requirements by Using the Requirements Toolbox API” on page 1-105
- “Create and Edit Attribute Mappings” on page 1-110
- “Use Callbacks to Customize Requirement Import Behavior” on page 1-113
- “Execute Code When Loading and Saving Requirement Sets” on page 1-117
- “Import Requirements from IBM Rational DOORS by Using the API” on page 1-119
- “Import Requirements from a Microsoft Excel Document” on page 1-125
- “Export Requirement and Link Information to Excel” on page 1-131
- “Use Command-Line API to Document Simulink Model in Requirements Editor” on page 1-136
- “Capture Model as Requirements and Create Links” on page 1-138
- “Import Requirements and Reuse Existing Links” on page 1-143
- “Programmatically Repair Broken Links” on page 1-145

Author Requirements in MATLAB or Simulink

In Requirements Toolbox™, you organize your requirements in groups called requirement sets. In each requirement set, you can create additional levels of hierarchy if you need to further describe a requirement's details.

In this tutorial, you use the **Requirements Editor** to create a requirement set, organize related requirements, and add requirements to the set. If you have Simulink, you can also use the Requirements Perspective to author requirements without leaving the Simulink Editor. For more information about using the Requirements Perspective, see “Link Blocks and Requirements” on page 3-2.

Suppose that you are writing requirements for a controller model of an automobile cruise control system. You develop these requirements using your company's numbering standard (R1, R2, and so on).

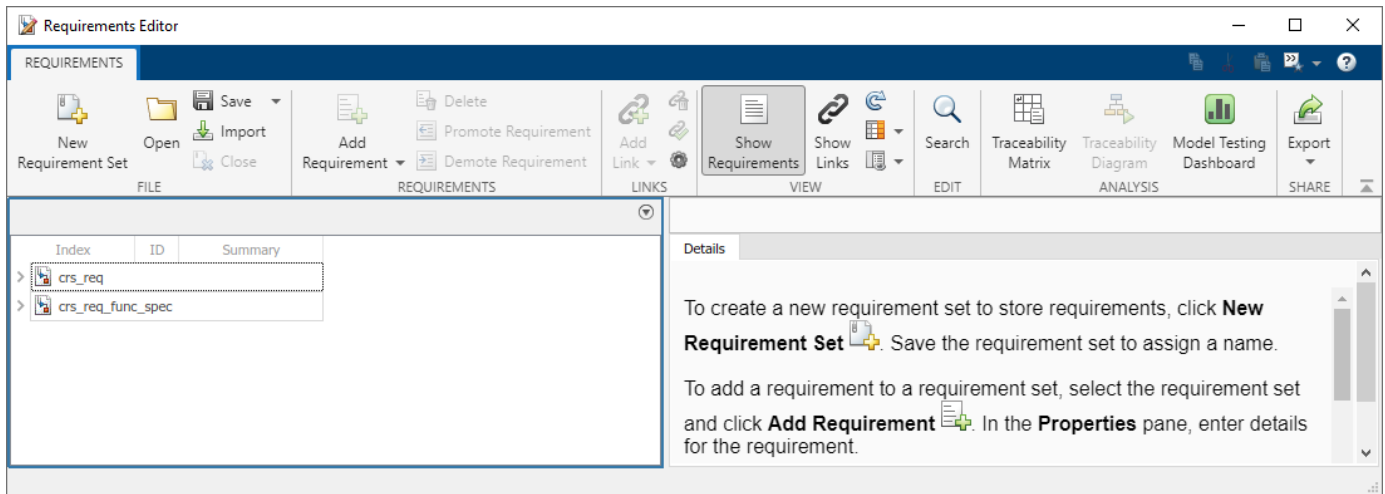
ID and Description	Rationale
R1: The maximum input throttle is 100%	The maximum value of the throttle from the acceleration pedal can be no greater than 100%.
R2: Cruise control has a speed operation range	Cruise control has a minimum and maximum operating speed.
R2.1: The vehicle speed must be at least 40 km/h	The speed of the vehicle must be at least 40 km/h for the cruise control system to engage.
R2.2: The vehicle speed cannot be greater than 100 km/h	The maximum operational speed of the cruise control system for the vehicle is 100 km/h.

These requirements capture functionality modeled in a model called `crs_controller`.

- 1 Open the project that includes the model and supporting files. At the MATLAB command prompt, enter:

```
slreqCCProjectStart
```
- 2 Open the requirement set `crs_req` in the **Requirements Editor**. At the command prompt, enter:

```
slreq.open("crs_req")
```
- 3 The **Requirements Editor** displays the requirements arranged by requirement set. The project has two requirement sets: `crs_req_func_spec` and `crs_req`.



- 4 Add a requirement set. From the **Requirements Editor** toolbar, click **New Requirement Set**.
- 5 Save the requirement sets to external files. Save your requirement set to a writable location and name it `cruise_control_reqset.slreqx`.
- 6 Add a requirement to your requirement set by selecting the requirement set and clicking **Add Requirement**.
- 7 In the right pane, under **Properties**, enter the details for the requirement. Enter the details for the requirement:

- **Custom ID:** R1
- **Summary:** Max input throttle %
- **Description:** The maximum input throttle is 100%.







If you do not specify a custom ID, the **Requirements Editor** numbers requirements in order. Custom IDs enable you to use your company standards for labeling requirements and to set the numeric order. (Custom IDs cannot contain a # character.) You can also use an ID to help locate a requirement when searching. Keywords aid in searching for a requirement.

- 8 Create the requirement R2. Click **Add Requirement**. Enter the details for the requirement:

- **Custom ID:** R2
- **Summary:** Cruise control speed operation range
- **Description:** Cruise control has a minimum and maximum operating speed.

- 9 Create child requirements for R2 by selecting R2 and clicking **Add Requirement > Add Child Requirement**. Enter the details for the requirement:

- **Custom ID:** R2.1
- **Summary:** Minimum vehicle speed
- **Description:** The speed of the vehicle must be at least 40 km/h for the cruise control system to engage.

Index	ID	Summary
>  crs_req		
>  crs_req_func_spec		
▼  cruise_control_reqset*		
 1	R1	Max input throttle %
▼  2	R2	Cruise control speed operation range
 2.1	R2.1	Minimum vehicle speed

Repeat this step to add other child requirements to R2.

You can rearrange the hierarchy by using  **Promote Requirement** or  **Demote Requirement**.


Author and Edit Requirements Content by Using Microsoft Word

To author and edit the **Description** and **Rationale** fields of your requirements, open Microsoft® Word from within the **Requirements Editor** or the Requirements Perspective View.

Note This functionality is available only on Microsoft Windows® platforms.


Using Microsoft Word to edit rich text requirements enables you to:

- Spell-check requirements content.
- Resize images.
- Insert and edit equations.
- Insert and edit tables.

On the Edit field toolbar, in either the **Description** or **Rationale** fields, click the  icon. Save the changes to your requirements content within Microsoft Word to see them reflected in Requirements Toolbox.

When you use Microsoft Word to edit requirements content, you cannot edit requirements in the built-in editor.

Customize Requirements Browser View

You can view or hide columns in the **Requirements Editor** when you click  **Columns > Select Attributes**. Add, remove, and reorder attribute columns in the Column Selector. The view configuration is saved across sessions. You can export view settings to a MAT-file by using the `slreq.exportViewSettings` function and import them by using the `slreq.importViewSettings` function. You can reset view configurations by using the `slreq.resetViewSettings` function.

Filter Requirements Content

You can search requirements content by clicking **Search**. You can find specific requirements within loaded requirement sets based on requirement attributes and descriptions.

Specify Filter Text Strings — As you enter text in the **Search** text box, the Requirements Browser performs a dynamic search and displays the results. The search operation applies only to attributes you choose to display in the Requirements Browser.

The text strings you enter must be consistent with the guidelines described in the following sections.

Case Sensitivity — By default, the Requirements Browser ignores case as it filters.

If you want the Requirements Browser to respect case sensitivity, put that text string in quotation marks.

Specify Attributes and Attribute Values — To restrict the filtering to requirements with a specific attribute, type the attribute name, followed by a colon. The Requirements Browser displays only the requirements that have that attribute.

To filter for requirements for which a specific attribute has a specific value, type the attribute name, followed by a colon (:), then the value. For example, to filter the contents to display only the requirements where the **Summary** attribute has a value that includes **Aircraft**, enter **Summary: Aircraft** (alternatively, you could put the whole string in quotation marks to enforce case sensitivity).

Wildcards and MATLAB Expressions Are Not Supported — The Requirements Browser does not recognize wildcard characters, such as *. For example, searching **fuel*** returns no results, even if requirements contain the text string **fuel**.

Also, if you specify a MATLAB expression in the **Search** text box, the Requirements Browser interprets that string as literal text, not as a MATLAB expression.

Requirement Types

Each requirement or referenced requirement has a requirement type that specifies its role. The requirement type is specified by the `Type` property of the `slreq.Requirement` object or `slreq.Reference` object.

Built-in Requirement Types

You can use these built-in requirement types when authoring or importing requirements:

- **Functional:** Use functional requirements to capture required functional behavior for the design. Requirements Toolbox calculates the implementation and verification status for functional requirements based on the requirement links.
- **Container:** Use container requirements to organize your requirements in groups and create a hierarchy. Requirements Toolbox does not include container requirements when calculating the implementation and verification status of the requirement set. However, any functional requirements under a container requirement contribute to the calculation of the statuses.
- **Informational:** Use informational requirements to capture non-functional behavior or other supplemental information. Requirements Toolbox does not include informational requirements or any requirements under them when calculating the implementation and verification status of the requirement set.

For more information about implementation and verification status, see “Review Requirements Implementation Status” on page 4-2 and “Review Requirements Verification Status” on page 4-6.

Custom Requirement Types

You can create custom requirements by extending one of the built-in types to define a requirement type that aligns with your project. For example, you can create custom types for system or stakeholder requirements. You can use stereotypes to define custom requirement types that have custom properties, or define custom requirement types by using an `sl_customization` file. For more information, see “Define Custom Requirement and Link Types and Properties” on page 1-78.

Custom requirement types must use one of the built-in types as the base behavior. The custom requirement type inherits some functionality from the built-in type, including how the requirement type contributes to the implementation and verification statuses. For more information, see the “Choose a Built-in Type as a Base Behavior” on page 1-80 section of “Define Custom Requirement and Link Types and Properties” on page 1-78.

Set the Requirement Type

When you create or import requirements in Requirements Toolbox, you can specify the requirement type in the **Requirements Editor** by clicking **Show Requirements** and, in the right pane, under **Properties**, selecting the type from the **Type** list. You can then choose from the built-in requirement types or your custom requirement types.

See Also

Requirements Editor

More About

- “Review Requirements Implementation Status” on page 4-2
- “Review Requirements Verification Status” on page 4-6
- “Define Custom Requirement and Link Types and Properties” on page 1-78

Import Requirements from Third-Party Applications

You can author requirements in third-party applications and import them to Requirements Toolbox. When you import requirements, you can migrate the requirements and manage them in Requirements Toolbox, or import the requirements as references to requirements called referenced requirements and continue to manage them in the third-party application. Supported applications include:

- Microsoft Word and Microsoft Excel®. See “Import Requirements from Microsoft Office Documents” on page 1-12.
- IBM® Rational® DOORS®. See “Import Requirements from IBM Rational DOORS” on page 1-42.
- IBM DOORS Next. See “Import Requirements from IBM DOORS Next” on page 1-35.
- Applications that use the Requirements Interchange Format (ReqIF™). See “Import Requirements from ReqIF Files” on page 1-17.

Note Microsoft Windows platforms support importing requirements from all applications listed above. To import requirements from third-party applications on a Mac or Linux® platform, you must use IBM DOORS Next or an application that uses ReqIF.

Add Requirements to the Path

Add requirements documents to the MATLAB path or project path. You can:

- Copy the requirements document to the MATLAB current folder.
- Add the parent folder of the requirements document to the MATLAB path.
- Update the Requirements Toolbox path preference to always use the relative path.

For more information on setting path preferences for requirements documents, see “Document Path Storage” on page 12-36.

Select an Import Mode

When you import requirements from third-party applications to Requirements Toolbox, you can migrate the requirements to Requirements Toolbox or continue to manage your requirements in the third-party application.

When you migrate your requirements to Requirements Toolbox, you no longer need to use the third-party application to make changes to your requirements.

If you choose to manage your requirements in the third-party application, you continue to make changes to requirements in the third-party application. Then, you can update the referenced requirements in Requirements Toolbox to bring changes that were made in the third-party application after the previous import. When you make changes in third-party application, the imported referenced requirements are outdated in Requirements Toolbox until you update them. Requirements Toolbox notifies you when a newer version of the source document is available.

Both import modes give you access to Requirements Toolbox analyses, such as change tracking (see “Track Changes to Requirement Links” on page 5-3), implementation status (see “Review Requirements Implementation Status” on page 4-2), and verification status (see “Review Requirements Verification Status” on page 4-6).

Migrate Requirements to Requirements Toolbox

If you want to migrate your requirements from the external requirements management application to Requirements Toolbox, when you import the requirements, clear the selection **Allow updates from external source**.

The screenshot shows the 'Importing Requirements' dialog box with the following settings:

- Source document:**
 - Document type: Microsoft Word Document
 - Document location: C:\Users\ahoward\MATLAB\Projects\examples\Cru
- Content:**
 - Plain text (selected)
 - Rich text (include graphics and tables)
- Requirement Identification:**
 - Content is imported to match the document outline of section headings.
 - Within a section you can identify additional items:
 - Use bookmarks to identify items and serve as custom IDs
 - Identify items by occurrences of search pattern (REGEXP)
 - Ignore outline numbers in section headers
- Destination(s):**
 - Requirement Set: \examples\CruiseRequirementsExample\crs_req.slreqx
 - Allow updates from external source

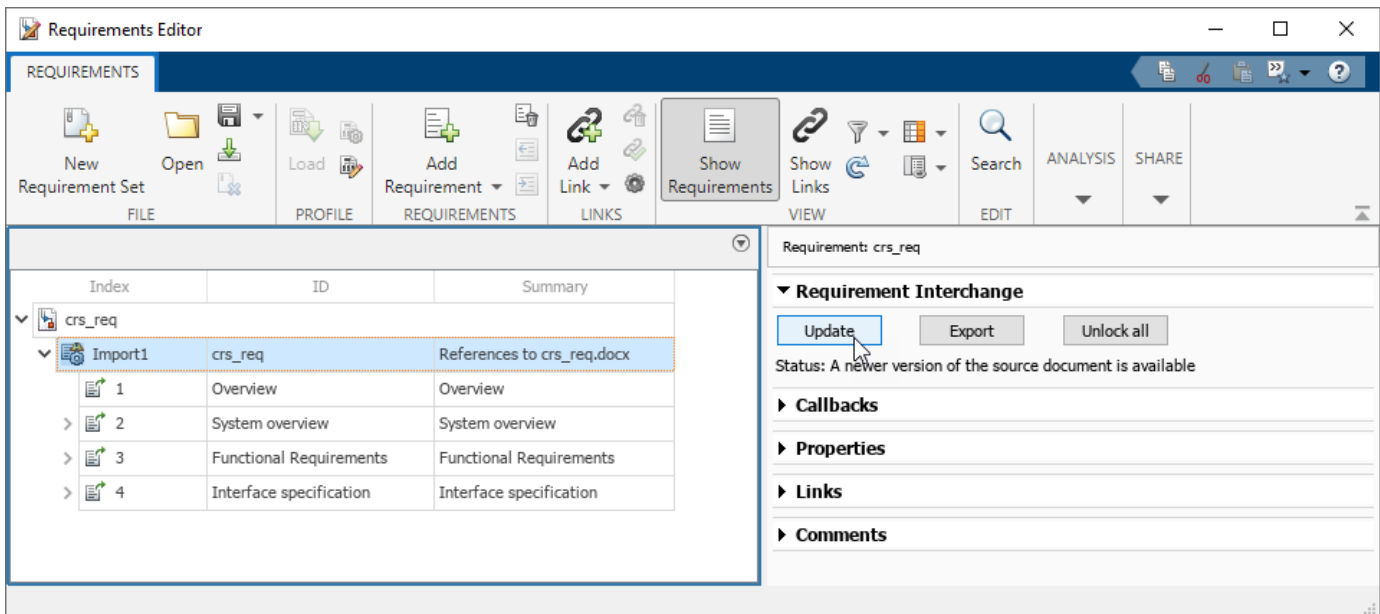
Requirements are imported as `slreq.Requirement` objects and are represented by the requirement icon (📄) in the **Requirements Editor** and in the Traceability Matrix. Importing requirements as `slreq.Requirement` objects allows you to freely edit, add, delete, and rearrange requirements. Updates you make to the requirements in the third-party application are not reflected in Requirements Toolbox.

Note You can export your requirements back to third-party applications that support ReqIF files by exporting requirements that are stored in Requirements Toolbox to a ReqIF file.

Manage Imported Requirements with External Applications

If you want to continue to manage your imported requirement with an external application, select **Allow updates from external source** when you import the requirements. The requirements are imported as referenced requirements (sl req.Reference objects).

If someone makes changes to the external source document, you can update the referenced requirements in Requirements Toolbox. In the **Requirements Editor**, select the top import node, represented by the Import node icon (🔗). In the right pane, under **Requirement Interchange**, click **Update**. You will be prompted to select the latest version of the file. For more information, see “Update Imported Requirements” on page 1-89.



Referenced requirements are locked for editing by default. Locked requirements are represented by the locked referenced requirement icon (🔒) in the **Requirements Editor**. To unlock an individual referenced requirement, navigate to it in the **Requirements Editor**, and, in the right pane, under **Properties**, click **Unlock**. Unlocked requirements are represented by the unlocked referenced requirement icon (🔓) in the **Requirements Editor**. Unlock all referenced requirements by navigating to the top Import node and, in the right pane, under **Requirement Interchange**, clicking **Unlock all**. You cannot delete referenced requirements or change their hierarchy within Requirements Toolbox, even after unlocking them. You cannot relock requirements after you unlock them, except by updating the entire referenced requirement set. Updating the referenced requirements overwrites changes made after the referenced requirements were unlocked.

You can register custom attributes for a requirement set that contains referenced requirements in Requirements Toolbox. To set the custom attribute value for a referenced requirement, you must unlock that requirement. For more information on registering custom attributes and setting their values for requirements, see “Add Custom Attributes to Requirements” on page 1-81. When you register custom attributes within Requirements Toolbox and set referenced requirement custom attribute values, those values are retained when you update the referenced requirements from the external source. However, if you modify custom attribute values that were imported from the external source, the update operation will overwrite modifications made to unlocked referenced requirements.

However, some third-party applications also allow you to create custom attributes. If you have attributes with the same name in the requirement set and in the external source document, when you update the referenced requirements from the external source, the local values are overwritten with the attribute values defined in the external source document.

When working with referenced requirements, you can navigate to the requirement in the external source document by clicking **Show in document** in the **Properties** pane.

Note To navigate from referenced requirements imported from ReqIF files to the original external source document, see “Navigate from Referenced Requirements to Requirements in Third-Party Applications” on page 1-27.

If there is a change in the source document's file name or location, right-click the top node of the requirement set and select **Update source document name or location**.

Differences Between Importing and Direct Linking

Requirements Toolbox also supports direct linking to requirements stored externally in Microsoft Word, Microsoft Excel, IBM Rational DOORS, and IBM DOORS Next.

When you create direct links from requirements in third-party applications to items in MATLAB or Simulink, the requirements are not covered by analysis tools provided by Requirements Toolbox. Additionally, depending on how the direct links to external requirements were created, you might not have visible backlinks to navigate to the linked item in MATLAB or Simulink. For example, when you link to requirements in Microsoft Word by creating a link to a bookmark or a heading, no navigation object is added to the Microsoft Word document. (See “Link to Requirements in Microsoft Word Documents” on page 7-2.) There is no indication when a direct link becomes unresolved unless you run consistency checks.

When you import requirements to Requirements Toolbox and then create links instead of creating direct links, you gain access to Requirements Toolbox analysis tools, such as implementation status, verification status, and change tracking. Additionally, Requirements Toolbox provides full link source and destination traceability and navigation. There is full indication when a link becomes unresolved.

See Also

slreq.import | **Requirements Editor**

More About

- “Import Requirements from Microsoft Office Documents” on page 1-12
- “Import Requirements from IBM DOORS Next” on page 1-35
- “Import Requirements from IBM Rational DOORS” on page 1-42
- “Import Requirements from ReqIF Files” on page 1-17

Import Requirements from Microsoft Office Documents

You can author requirements in Microsoft Word and Microsoft Excel and import them into Requirements Toolbox. When you import the requirements, you can allow updates from the Microsoft Office documents, or you can import them without allowing updates. To read more about these import modes, see “Select an Import Mode” on page 1-8.

Note You can only import requirements from Microsoft Office on Microsoft Windows platforms.

To import requirements from a Microsoft Office document:

- 1 Open the **Requirements Editor**. At the MATLAB command line, enter:
`slreq.editor`
- 2 Click **Import**.
- 3 Set the **Document type** to Microsoft Word Document or Microsoft Excel Spreadsheet.
- 4 Next to the **Document Location** field, click **Browse** and select the desired file.
- 5 Set the import options. To read more about import options for Microsoft Office documents, see “Import Options for Microsoft Word Documents” on page 1-12 and “Import Options for Microsoft Excel Spreadsheets” on page 1-14. To read more about importing requirement sets or referenced requirements, see “Select an Import Mode” on page 1-8.
- 6 Click **Import** to import the requirements.

Import Options for Microsoft Word Documents

You can import requirements in plain and rich text formats from Microsoft Word documents. Use the rich text format to import requirements that contain content such as graphics and tables.

When you import requirements from Microsoft Word documents, the section headers and numbers populate the **ID** and **Summary** fields and the section body populates the **Description** field. To ignore section numbers in the imported requirements, select **Ignore outline numbers in section headers**. If you select **Allow updates from external source**, it is recommended to ignore outline numbers to prevent unexpected behavior that can occur if the section numbers change when you make changes to your Microsoft Word document and then update the imported requirements. For example, when you insert a new section in the middle of your document, some of the outline numbers in the section headers change to reflect the new section numbering. When you update the requirement set, Requirements Toolbox deletes the referenced requirements that correspond to sections whose outline numbers changed and re-inserts them with the updated numbering. This can create some unexpected change issues.

The imported requirements hierarchy matches the Microsoft Word document headings hierarchy.

When you import requirements, it is recommended to select **Use bookmarks to identify items and serve as custom IDs** because the bookmarks are persistently stored in the document and cannot be duplicated.

You can import requirements selectively when you select **Identify items by occurrences of search pattern (REGEXP)** and enter a regular expression search pattern. To read more about regular expressions, see “Regular Expressions”.

Note If you do not have images in your requirements document, consider importing your requirements as plain text to prevent some issues related to font, style, or whitespace differences.

Import Options for Microsoft Excel Spreadsheets

You can import requirements in plain and rich text formats from Microsoft Excel spreadsheets. The plain text format imports only text and associates each column of your spreadsheet to a requirement property. The rich text format imports graphics, layouts, and captures multicell ranges.

Note If your Excel spreadsheet contains cells that are grouped and the group is collapsed, any requirements in cells that are not visible are not imported.

When you import requirements from Microsoft Excel files, you can identify requirements by specifying rows and columns, or you use a regular expression search pattern.

The screenshot shows the 'Importing Requirements' dialog box with the following settings:

- Source document:**
 - Document type: Microsoft Excel Spreadsheet
 - Document location: C:\Users\ahoward\Downloads\slvndemo_
 - Sheet name: Test Scenarios
 - Use worksheet name as item ID prefix
- Content:**
 - Text only (better performance)
 - Include graphics and layout
- Requirement Identification:**
 - Specify rows and columns (Value: ,A,B,C, 2-17)
 - Use search pattern (REGEXP)
- Destination(s):**
 - Requirement Set: dows\slvndemo_FuelSys_TestScenarios.slreqx
 - Allow updates from external source

Identify Requirements by Specifying Rows and Columns

To identify requirements by specifying rows and columns, in the Importing Requirements dialog, under **Requirement Identification**, select **Specify rows and columns**.

Importing requirements with this method allows you to map columns to requirements properties and custom attributes when you click **Configure columns**. Under each column, you can select an item from the list. You must select a column to map to either **Summary** or **Description**. If you select <Custom Attribute>, a custom attribute is registered for the requirement set with the custom attribute name specified by the column name. To read more about custom attributes for requirements, see “Add Custom Attributes to Requirements” on page 1-81.

Each column is imported as a separate specified property or custom attribute, with the exception of the **Description** and **Rationale** properties, which can combine multiple adjacent columns. When you select multiple columns for **Description** and **Rationale**, the value from each cell is concatenated into one field.

Test Scenarios in slvndemo_FuelSys_TestScenarios.xlsx Manually specify header row

Configure how to use each column by choosing an option from the drop-down list

	Requirements	Test Scenario	Test Description	Expectation
Use As:	Description	Description	<Custom Attribute>	<Ignore>
2:	1.1 Normal Mode of Operati...			
3:	Normal operation	The simulation is run with a th...	During the normal mode of operatio...	
4:	1.1.1 Stoichiometric mixture ...	The simulation is run with a th...	The stoichiometric mixture target sh...	
5:	1.1.2 Oxygen Sensor (EGO) 1...	The simulation is run with a th...		
6:	1.1.3 High Oxygen Level	The simulation is run with a th...	If the EGO sensor determines a high ...	
7:	1.1.4 Fuel-rich Mixture	The simulation is run with a th...	If the EGO sensor determines a low ...	
8:	1.2 Failure Mode of Operation			
9:	1.2.1. Single Sensor Failure			
10:	1.2.1.1 Throttle sensor failure	During normal mode of opera...	The System detects the Throttle Sens...	
11:	1.2.1.2 Throttle sensor failure	During normal mode of opera...	The System detects the Throttle Sens...	
12:	1.2.1.3 MAP Sensor failure	During normal mode of opera...	The System detects the Manifold Ab...	
13:	1.2.1.4 MAP Sensor failure	During normal mode of opera...	The System detects the Manifold Ab...	

Specify rows to import

From: To:

If you cannot map one of the columns in the spreadsheet to a column that holds unique requirement custom IDs, the import operation automatically generates unique custom IDs based on the rows in the spreadsheet. These custom IDs might not be persistent. If you explicitly select a column that does not have unique custom IDs, you cannot update the requirements document later.

You can exclude contents by ignoring columns and selecting only a range of rows to import. To ignore a column, select <Ignore> from the drop-down menu at the top of that column. To import only a range of rows, under **Specify rows to import**, enter the row number to start at and end at.

Note You cannot maintain the hierarchy from your Microsoft Excel file when, under **Requirement Identification**, you select **Specify rows and columns**.

Identify Requirements by Regular Expression Search Pattern

To identify requirement by using a regular expression search pattern, in the Importing Requirements dialog, under **Requirement Identification**, select **Use search pattern (REGEXP)**. To read more about regular expressions, see “Regular Expressions”.

The main advantage to using a regular expression search pattern is that you can retain the existing hierarchy when you import requirements from an Excel document if the matched requirement IDs are hierarchical. For example, the pattern `R[\d\.]+` will match requirements with IDs `R1`, `R1.1`, `R2`, and so on, and `R1.1` will be recognized as a child of `R1`. Additionally, you can selectively import requirements by importing only requirements that match the regular expression.

See Also

`slreq.import`

Related Examples

- “Import and Edit Requirements from a Microsoft Word Document” on page 1-95

More About

- “Import Requirements from Third-Party Applications” on page 1-8

Import Requirements from ReqIF Files

Many third-party requirements management applications can export and import requirements using the ReqIF format. You can import requirements from a ReqIF file as references to a third-party source called referenced requirements, which are represented as `slreq.Reference` objects, or as requirements in new requirement sets, which are represented as `slreq.Requirement` objects. For more information about choosing which import mode to use, see “Select an Import Mode” on page 1-8.

Choosing an Import Mapping

Before you import requirements and links from ReqIF files, decide which mapping to use. The mapping determines how ReqIF requirement and link data maps to Requirements Toolbox requirement and link data. Alternatively, you can map ReqIF data to stereotypes. For more information, see “Use Stereotypes when Importing from ReqIF Files” on page 1-28.

ReqIF represents requirements as `SpecObject` objects and links as `SpecRelation` objects that relate `SpecObject` objects. Each `SpecObjectType` object specifies the associated `SpecObject` object and the `SpecRelationType` objects classify each `SpecRelation` object. The `SpecObjectType` and `SpecRelationType` objects define attributes to store requirements and link information. The `SpecObject` and `SpecRelation` objects contain values for these attributes.

This table shows the relationship between requirements and links in Requirements Toolbox and their ReqIF counterparts.

Item	Representation in Requirements Toolbox	Representation in ReqIF
Requirement	<ul style="list-style-type: none"> <code>slreq.Requirement</code> object <code>slreq.Reference</code> object 	<code>SpecObject</code> object
Requirement type	<ul style="list-style-type: none"> Type property of <code>slreq.Requirement</code> object Type property of <code>slreq.Reference</code> object 	<code>longName</code> attribute of the <code>SpecObjectType</code> object
Requirement attributes	<ul style="list-style-type: none"> <code>slreq.Requirement</code> properties <code>slreq.Reference</code> properties Custom attributes for requirement sets on page 1-81 	<ul style="list-style-type: none"> <code>SpecObjectType</code> objects define attributes <code>SpecObject</code> objects define attribute values
Link	<code>slreq.Link</code> object	<code>SpecRelation</code> object
Link type	Type property of <code>slreq.Link</code> object	<code>longName</code> attribute of the <code>SpecRelationType</code> object
Link attributes	<code>slreq.Link</code> properties and custom attributes on page 3-73	<ul style="list-style-type: none"> <code>SpecRelationType</code> objects define attributes <code>SpecRelation</code> objects define attribute values

For more information about ReqIF data organization, see the Exchange Document Content section in Requirements Interchange Format (ReqIF) Version 1.2lo.

When you import requirements and links from a ReqIF file, you can choose the import mode that you use based on how the import process maps the requirements from ReqIF to Requirements Toolbox. The import process maps the `SpecObject` objects to `slreq.Requirement` objects or `slreq.Reference` objects, depending on the import mode, and `SpecRelation` objects to `slreq.Link` objects. The imported requirement type, properties, and imported link type depend on the import mapping that you choose.

Requirements Toolbox provides built-in import mappings for some third-party applications that use ReqIF:

- IBM Rational DOORS
- IBM DOORS Next
- Polarion®
- PREEvision
- Jama

When you import requirements from ReqIF files generated by other requirements management applications, you can use a generic attribute mapping.

After you import requirements, you can edit the attribute mappings. See “Mapping ReqIF Attributes in Requirements Toolbox” on page 1-24.

Note If you experience problems navigating from requirements in Polarion to items in MATLAB or Simulink due to changes to navigation URLs enforced by Polarion, you may need to apply a configuration change. Open the `polarion.properties` file found in the `<polarion_installation>/polarion/configuration/` folder and modify these lines by replacing `localhost` with the externally known name for your server:

- `repo=http://localhost:80/repo/`
 - `base.url=http://localhost:80/`
-

Using the Built-In Mapping During Import

When you import a ReqIF file and use the built-in mapping for the third-party tool that generated the file, Requirements Toolbox imports the `SpecObject` objects as requirements with **Type** set to `Functional` regardless of the associated `SpecObjectType` object. If the `SpecObjectType` objects define additional attributes in the third-party tool, the attributes map to built-in requirements properties, including `Custom ID` or `ID`, `Summary`, `Description` and revision information. The remaining attributes map to new custom attributes. For more information about requirement custom attributes, see “Add Custom Attributes to Requirements” on page 1-81.

After you import the requirements, you can map the `SpecObjectType` objects to requirement types. You can also edit the `SpecObjectType` object attribute mappings to requirement properties. See “Mapping ReqIF Attributes in Requirements Toolbox” on page 1-24.

When you import links using the built-in mapping, Requirements Toolbox imports `SpecRelation` objects as links and maps the `SpecRelationType` objects to link types in Requirements Toolbox. If a

`SpecRelationType` in the ReqIF file is not defined in the import mapping, then `SpecRelation` objects with that type import as links with **Type** set to `Related` to. For more information about link types, see “Link Types” on page 3-34.

Using the Generic Mapping During Import

When you import a ReqIF file and use the generic mapping, Requirements Toolbox imports the `SpecObject` objects as requirements with **Type** set to `Functional`. The `SpecObjectType` object attributes map to the `CustomID` or `ID`, `Description`, and `Summary` requirement properties, and to new custom attributes. For more information about requirement custom attributes, see “Add Custom Attributes to Requirements” on page 1-81.

After you import the requirements, you can map the `SpecObjectType` objects to requirement types. You can also edit the `SpecObjectType` object attribute mappings to match your desired requirement properties. See “Mapping ReqIF Attributes in Requirements Toolbox” on page 1-24.

When you import links by using the generic mapping, the `SpecRelation` objects import as links with **Type** set to `Related` to. For more information about link types, see “Link Types” on page 3-34.

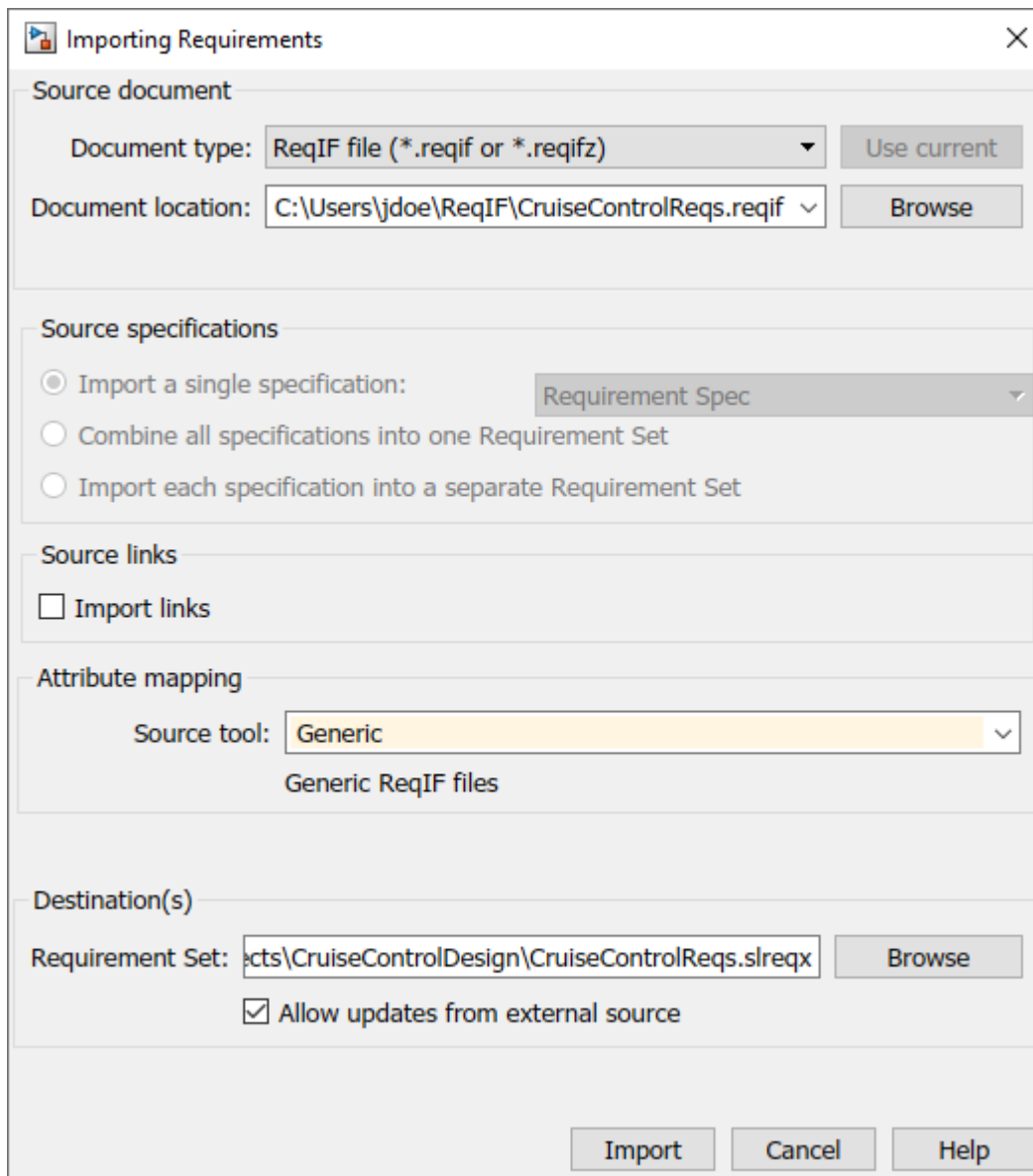
Importing Requirements

You can import requirements in the **Requirements Editor**. Requirements in ReqIF files belong to specifications.

Tip To import images associated with requirements, use the third-party tool to export the requirements as a `.reqifz` file and then import the file to Requirements Toolbox.

- 1 Open the **Requirements Editor** with one of these approaches:
 - At the MATLAB command line, enter:

```
slreq.editor
```
 - In the MATLAB **Apps** tab, under **Verification, Validation, and Test**, click the **Requirements Editor** app.
 - In the Simulink **Apps** tab, under **Model Verification, Validation, and Test**, click the **Requirements Editor** app.
- 2 In the **Requirements Editor**, click **Import**.
- 3 In the Importing Requirements dialog, set **Document type**, to ReqIF file (`*.reqif` or `*.reqifz`).
- 4 Next to **Document location**, click **Browse** and select the ReqIF file.



- 5 Under **Attribute mapping**, in the **Source tool** drop-down, select the desired attribute mapping. See “Choosing an Import Mapping” on page 1-17.
- 6 Under **Destination(s)**, click **Browse**. Enter the file name, select the location to save the new requirement set, and click **Save**.
- 7 Select whether to allow updates to the imported requirements. If you want to continue to manage your imported requirements in the third-party tool, select **Allow updates from external source**, which imports the requirements as referenced requirements. If you want to migrate your requirements to Requirements Toolbox, clear **Allow updates from external source**. For more information about import options, see “Select an Import Mode” on page 1-8.
- 8 Click **Import** to import the requirements.

The imported requirements maintain the requirement hierarchy.

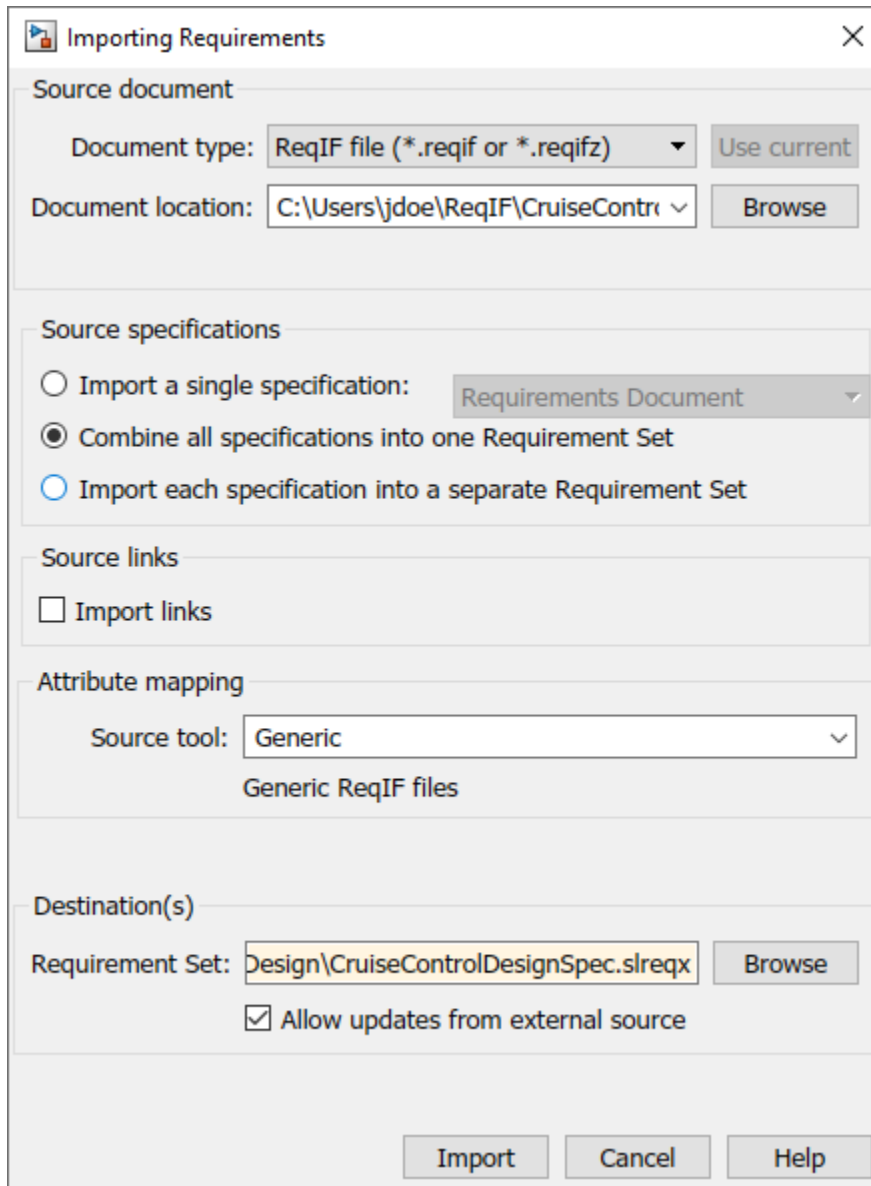
Importing Requirements from a ReqIF File with Multiple Specifications

If you import a ReqIF file that contains multiple source specifications, you can select options in the **Source specifications** section in the Importing Requirements dialog. You can:

- Select a single ReqIF source specification to import into a requirement set. In the Importing Requirements dialog, under **Source specifications**, select **Import a single specification** and choose a specification from the list.
- Combine ReqIF source specifications into one requirement set. In the Importing Requirements dialog, under **Source specifications**, select **Combine all specifications into one Requirement Set**.

If you select **Allow updates from external source**, then each specification is imported into a separate Import node. You can update each Import node independently. Otherwise, each source specification imports as a parent requirement and all requirements in the specification import as its children.

- Import each ReqIF source specification into a separate requirement set. In the Importing Requirements dialog, under **Source specifications**, select **Import each specification into a separate Requirement Set**. Under **Destination(s)**, next to **Folder**, click **Browse** and select a destination folder location to save the requirement sets in.



The resulting requirement set file names are the same as the source specification name. If you have an existing requirement set file with the same name as one of the source specifications in the selected destination, it is overwritten.

Tip For large ReqIF files, consider importing each source specification into a separate requirement set. This can help reduce file conflicts and help you track differences in individual requirement sets.

When deciding which import method to use for a ReqIF file that contains multiple source specifications, consider if you are importing links and if you plan to export back to ReqIF. For more information, see “Importing Links” on page 1-23 and “Considerations for ReqIF Files with Multiple Specifications” on page 1-99.

Importing Links

When you import a ReqIF file to a requirement set, you can import links as well. To import links, in the Importing Requirements dialog, under **Source links**, select **Import links** to preserve the links from the ReqIF file. After the import, the Requirements Toolbox link set files contain links between requirements and other Model-Based Design items.

The screenshot shows the 'Importing Requirements' dialog box with the following settings:

- Source document:**
 - Document type: ReqIF file (*.reqif or *.reqifz)
 - Document location: C:\Users\jdoe\ReqIF\CruiseControlDesignSpe
- Source specifications:**
 - Import a single specification: Requirements Document (selected)
 - Combine all specifications into one Requirement Set (unselected)
 - Import each specification into a separate Requirement Set (unselected)
- Source links:**
 - Import links (checked)
- Attribute mapping:**
 - Source tool: Generic
- Destination(s):**
 - Requirement Set: CruiseControlDesign\CruiseControlDesignSpec.slreqx
 - Allow updates from external source (checked)

ReqIF files represent links as a `SpecRelation` object that relates two `SpecObject` objects. You can only import links if the ReqIF file contains at least one `SpecRelation` object.

Importing Links from a ReqIF File with Multiple Source Specifications

When you import links from a ReqIF file with multiple source specifications, how you import the source specifications affects the link import. If you:

- Import a single specification into a requirement set, Requirements Toolbox only imports the `SpecRelation` objects that link `SpecObject` objects within that specification. This import might omit some links from the ReqIF file during import.
- Combine multiple ReqIF source specifications into one requirement set, the resolved links import into one link set.
- Import each ReqIF source specification into a separate requirement set, the resolved links import into separate link sets.

Importing Links from a ReqIF File Generated by Requirements Toolbox

If you link a requirement in Requirements Toolbox to an item that is not contained in the requirement set, such as a Simulink block, and then export the requirement and associated links to a ReqIF file, the export process inserts a `SpecObject` object into the ReqIF file that serves as a proxy object for the linked item. If the linked item is one of the supported types, the proxy object has a `SpecObjectType` `longName` value that describes the linked object type. For more information, see “Exporting Links” on page 1-64.

When you re-import this ReqIF file, the software reconstructs the links that relate the proxy `SpecObject` objects and requirements for proxy objects of the supported types. Links between the proxy `SpecObject` objects that have the `SpecObjectType` `longName` attribute set to `Requirement` cannot be reconstructed.

To reconstruct the links when you import a ReqIF file, in the Importing Requirements dialog:


- 1 Under **Source specifications**, select either **Combine all Specifications into one Requirement Set** or **Import each specification into a separate Requirement Set**.
- 2 Under **Source links**, select **Import Links**.

The reconstructed links use the Requirements Toolbox default link storage. For more information, see “Requirements Link Storage” on page 6-4. The reconstructed links are appended to the link set for the artifact that contains the link source. If the link set is not available, it is created with the same base file name as the artifact and stored in the same folder as the artifact.

Mapping ReqIF Attributes in Requirements Toolbox


ReqIF represents a requirement as a `SpecObject` object with a `SpecObjectType` object that defines requirement attributes. When you import requirements from a ReqIF file, the attributes map to requirement properties or custom attributes according to the import mapping that you choose. See “Choosing an Import Mapping” on page 1-17.

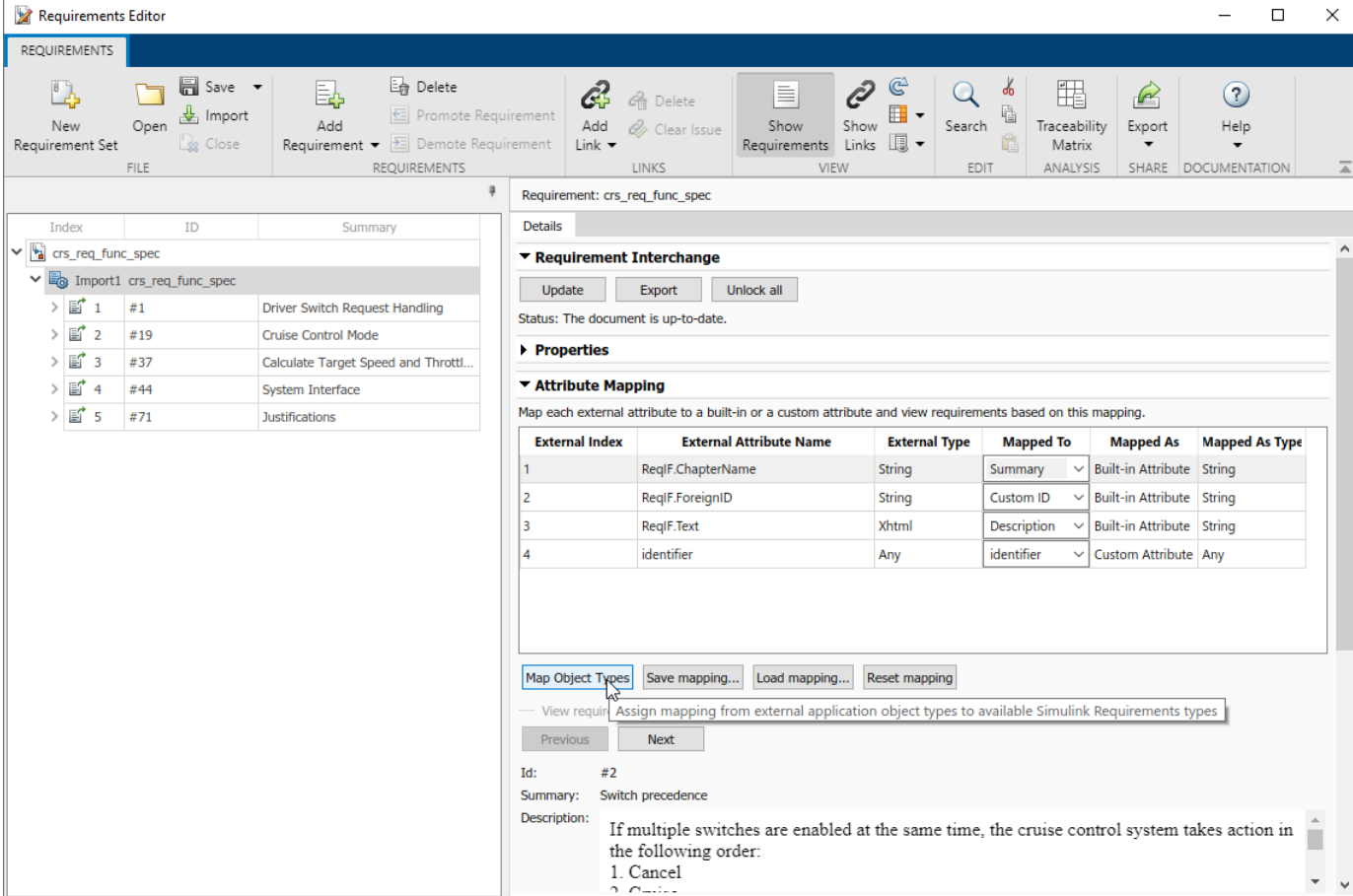
After you import the requirements, you can edit the `SpecObjectType` object attribute mapping.

Select the Import node, which is denoted by , or the top-level requirement, depending on how you imported the requirements. In the right pane, under **Attribute Mapping**, you can edit the attribute mappings. You can save the current mapping by clicking **Save mapping**. You can load a saved mapping by clicking **Load mapping**. For more information, see “Edit the Attribute Mapping for Imported Requirements” on page 1-110.

Map SpecObjectTypes to Requirement Types

After you import the requirements, you can map the `SpecObjectType` objects to requirement types in Requirements Toolbox.

- 1 In the **Requirements Editor**, select the Import node, which is denoted by , or the top-level requirement, depending on if you imported the ReqIF requirements as referenced requirements or requirements.
- 2 In the right pane, under **Attribute Mapping**, click **Map Object Types**.



Requirement: crs_req_func_spec

Details

▼ Requirement Interchange

Update Export Unlock all

Status: The document is up-to-date.

► Properties

▼ Attribute Mapping

Map each external attribute to a built-in or a custom attribute and view requirements based on this mapping.

External Index	External Attribute Name	External Type	Mapped To	Mapped As	Mapped As Type
1	ReqIF.ChapterName	String	Summary	Built-in Attribute	String
2	ReqIF.ForeignID	String	Custom ID	Built-in Attribute	String
3	ReqIF.Text	Xhtml	Description	Built-in Attribute	String
4	identifier	Any	identifier	Custom Attribute	Any

Map Object Types Save mapping... Load mapping... Reset mapping

View requirement Assign mapping from external application object types to available Simulink Requirements types

Previous Next

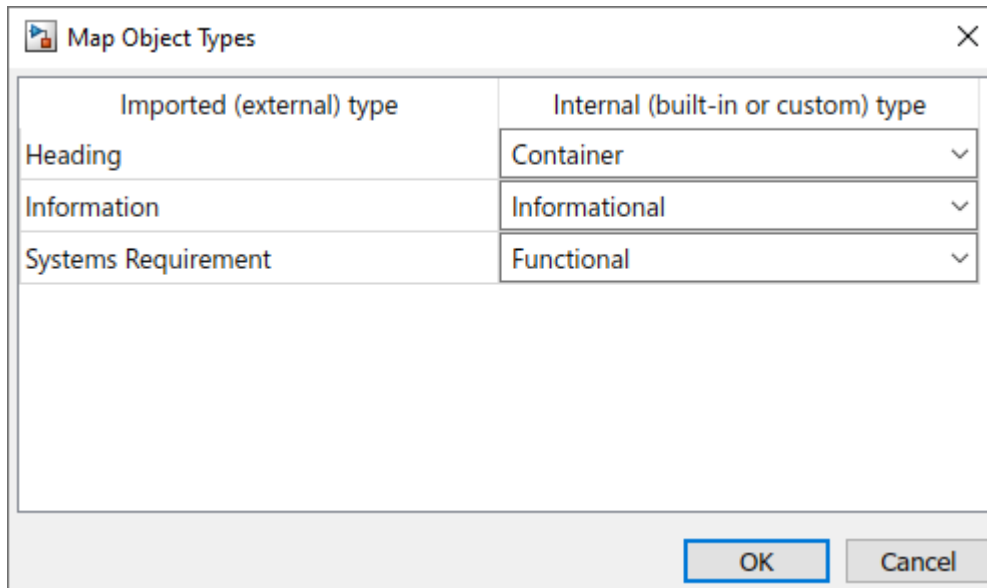
Id: #2

Summary: Switch precedence

Description: If multiple switches are enabled at the same time, the cruise control system takes action in the following order:

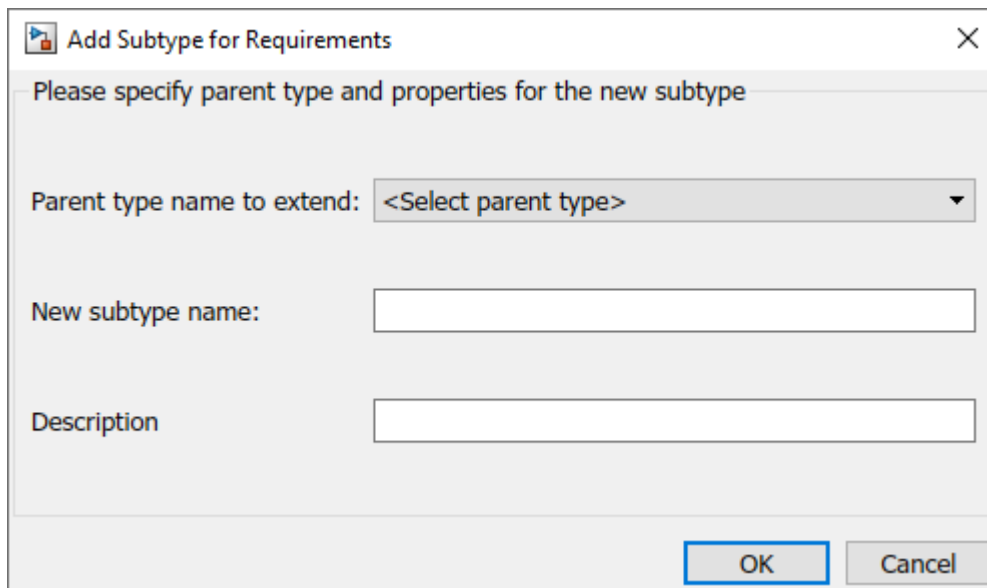
1. Cancel
2. Cruise

- 3 The Map Object Types dialog appears. **Imported (external) type** lists the SpecObjectType objects and **Internal (built-in or custom) type** lists the available Requirements Toolbox requirement types. Map each SpecObjectType object by selecting a requirement type from the list. For more information about requirement types, see “Requirement Types” on page 1-6. You can also select <Add custom subtype> to add a custom requirement type that is a subtype of a built-in type. For more information about custom requirement types, see “Define Custom Requirement and Link Types by Using sl_customization Files” on page 3-42.



To add a custom requirement type:

- 1 In the Add Subtype for Requirements dialog, set **Parent type name to extend** to the built-in requirement type that you want the custom requirement type to inherit from.
- 2 Next to **New subtype name**, enter the name for your new custom requirement type.
- 3 Next to **Description**, enter a description for your new custom requirement type.
- 4 Click **OK** to create the custom requirement type.



- 4 Click **OK** to map the SpecObjectType objects to requirement types. A dialog lists the number of items updated.

Navigate from Referenced Requirements to Requirements in Third-Party Applications

To navigate from a referenced requirement to the original requirement in the third-party application, create a navigation callback function and register the function in MATLAB.

- 1 Create a navigation callback function. Optionally, you can use the Requirements Toolbox template to develop your navigation callback function. To generate the template:
 - 1 Import your requirements to Requirements Toolbox.
 - 2 In the **Requirements Editor**, select a referenced requirement and, in the right pane, under **Properties**, click **Show in document**.
 - 3 In the Requirement Navigation Error dialog box, click **Go to Editor**. The generated template opens in the MATLAB Editor and is saved in the current folder. Add your navigation callback function to the template.
- 2 Register the navigation callback function by using `slreq.registerNavigationFcn`. Enter the name of the application that generated the ReqIF file, which is specified in the Domain property of the Import node. Use `slreq.getNavigationFcn` to confirm that you registered the callback.

To navigate from a referenced requirement in Requirements Toolbox to the original requirement in the third-party application, in the **Requirements Editor**, select a referenced requirement and, in the right pane, under **Properties**, click **Show in document**.

See Also

Requirements Editor | `slreq.import` | `slreq.registerNavigationFcn` | `slreq.getNavigationFcn`

More About

- “Use Stereotypes when Importing from ReqIF Files” on page 1-28
- “Export Requirements to ReqIF Files” on page 1-61
- “Round-Trip Importing and Exporting for ReqIF Files” on page 1-99
- “Import Requirements from Third-Party Applications” on page 1-8
- “Create and Edit Attribute Mappings” on page 1-110

Use Stereotypes when Importing from ReqIF Files

When you import requirements and links from ReqIF files, you can map the ReqIF requirement and link types and attributes to Requirements Toolbox stereotypes and stereotype properties. For more information about importing requirements and links using ReqIF files, see “Import Requirements from ReqIF Files” on page 1-17.

To map the requirements and link types, enable the **Requirements Editor** to use profiles during import, then create a new profile. The **Requirements Editor** creates a profile with a stereotype for each requirement type and link type in the ReqIF file. Each stereotype has a property for the attributes defined by the requirement and link types. Alternatively, you can use a profile that you created during a previous import or by using the **Profile Editor**. For more information about profiles and stereotypes, see “Customize Requirements and Links by Using Stereotypes” on page 1-71.

Create Profile During Import

To import requirements and links and create a new profile:

- 1 Enable the **Requirements Editor** to use profiles during import by entering this command at the MATLAB command line:

```
rmipref(ReqIfImportUseProfile=true)
```

Note This setting persists between MATLAB sessions. To disable using profiles during import, enter `rmipref(ReqIfImportUseProfile=false)`.

- 2 Open the **Requirements Editor**. For more information, see “Open the Requirements Editor App”.
- 3 In the **File**, click **Import**.
- 4 In the Importing Requirements dialog, set **Document type** to ReqIF file (*.reqif or *.reqifz).
- 5 Next to **Document location**, click **Browse** and select the ReqIF file.
- 6 In the **Profile Mapping** section, enter a name for the new profile in the **Profile** field.

Importing Requirements

Source document

Document type: ReqIF file (*.reqif or *.reqifz) Use current

Document location: C:\Users\jdoe\ReqIF\CruiseControlDesignr Browse

Source specifications

Import a single specification: Requirements Document

Combine all specifications into one Requirement Set

Import each specification into a separate Requirement Set

Source links

Import links

Profile Mapping

Browse an existing profile then map types, or enter new profile name and create profile

Profile: CruiseProfile Browse

Create Profile Map Types ...

Destination(s)

Requirement Set: ControlDesign\CruiseControlDesignSpec.slreqx Browse

Allow updates from external source

Callbacks

Import Cancel Help

- 7 Click **Create Profile**.
- 8 Select whether to allow updates to the imported requirements. If you want to continue to manage your imported requirements in the third-party tool, select **Allow updates from external source**, which imports the requirements as referenced requirements. If you want to migrate your requirements to Requirements Toolbox, clear **Allow updates from external source**. For more information about import options, see “Select an Import Mode” on page 1-8.
- 9 Click **Import** to import the requirements.

For more information about importing requirements and links from ReqIF files, see “Import Requirements from ReqIF Files” on page 1-17.

When you create a new profile when you import requirements and links, Requirements Toolbox:

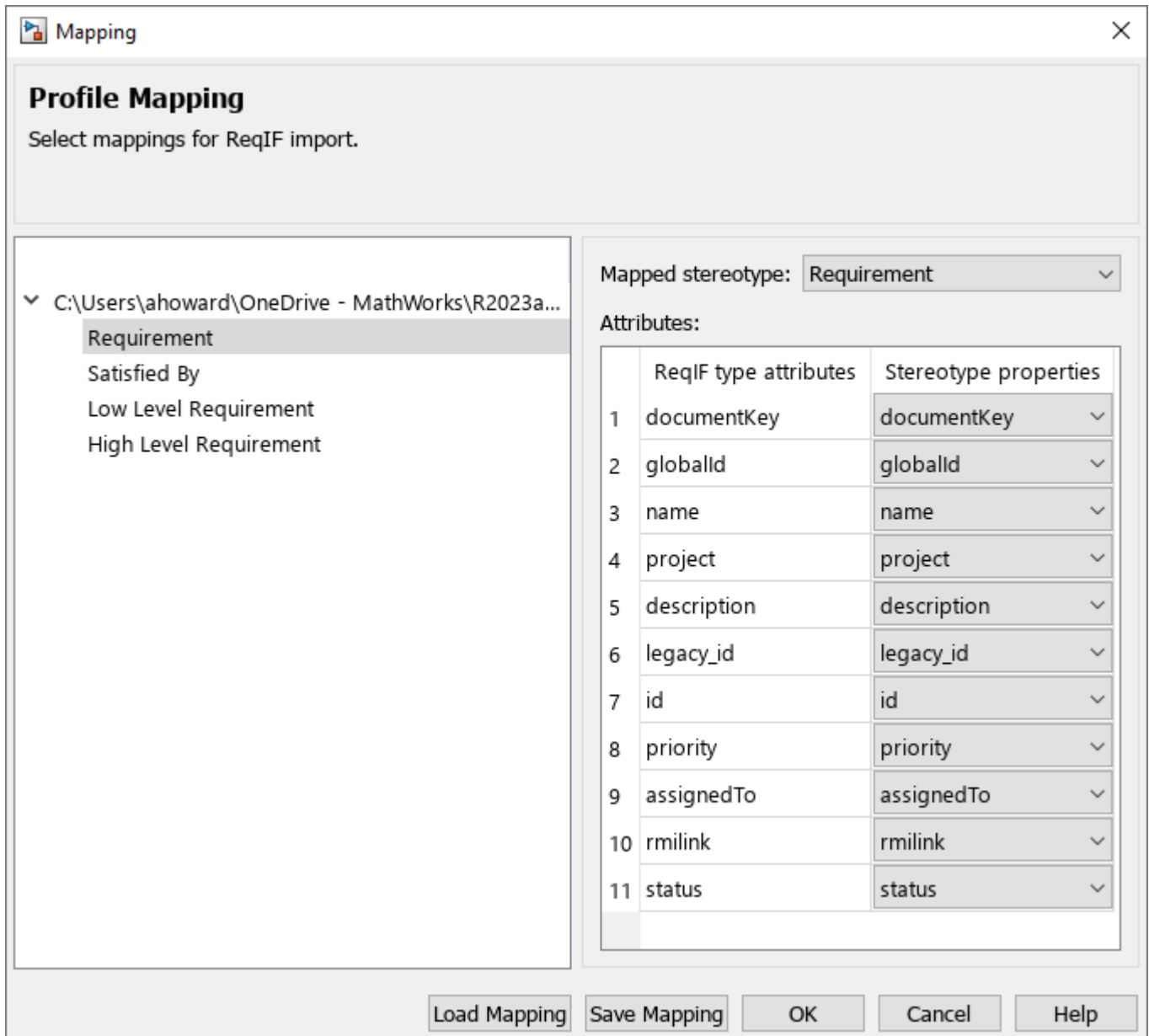
- Assigns the profile to the imported requirement set and link set
- Creates stereotypes for each ReqIF requirement type and link type
- Creates stereotype properties in the stereotypes for each ReqIF requirement attribute and link attribute
- Maps the stereotypes to the ReqIF requirement types and link types and assigns the stereotypes to the imported requirements and links
- Maps the stereotype properties in the stereotypes to the ReqIF requirement and link attributes and imports the attribute values as stereotype attribute values

For more information about stereotypes, see “Customize Requirements and Links by Using Stereotypes” on page 1-71.

Use Existing Profile During Import

To use an existing profile when you import requirements and links:

- 1 In the Importing Requirements dialog, in the **Profile Mapping** section, click **Browse** and select a profile.
- 2 Map the ReqIF requirement and link types and attributes to stereotypes and stereotype properties in the existing profile by clicking **Map Types**.
- 3 In the Mapping dialog box, in the left pane, select a ReqIF requirement type or link type.

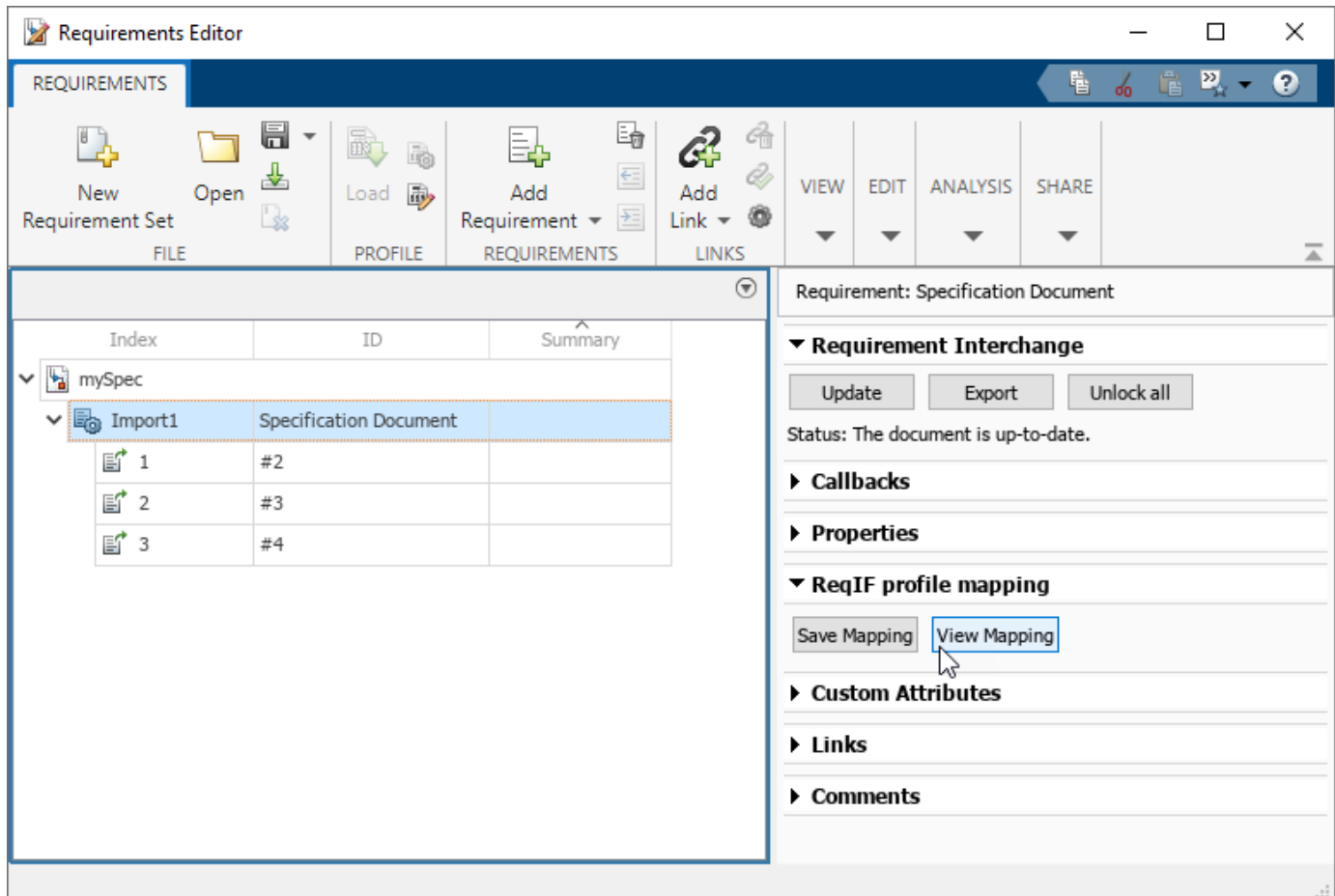


- 4 In the right pane, set **Mapped stereotype** to the stereotype that you want to map the ReqIF requirement or link type to. To prevent Requirements Toolbox from importing requirements or links of the selected type, set **Mapped stereotype** to [Ignored].
- 5 Map each ReqIF requirement or link attribute to a built-in property or a stereotype property by setting **Stereotype properties** to the desired Requirements Toolbox property or stereotype property. To prevent Requirements Toolbox from importing requirement or link attribute values, set **Stereotype properties** to [Ignored] for that attribute.
- 6 Click **OK**.
- 7 In the Importing Requirements dialog box, click **Import**.

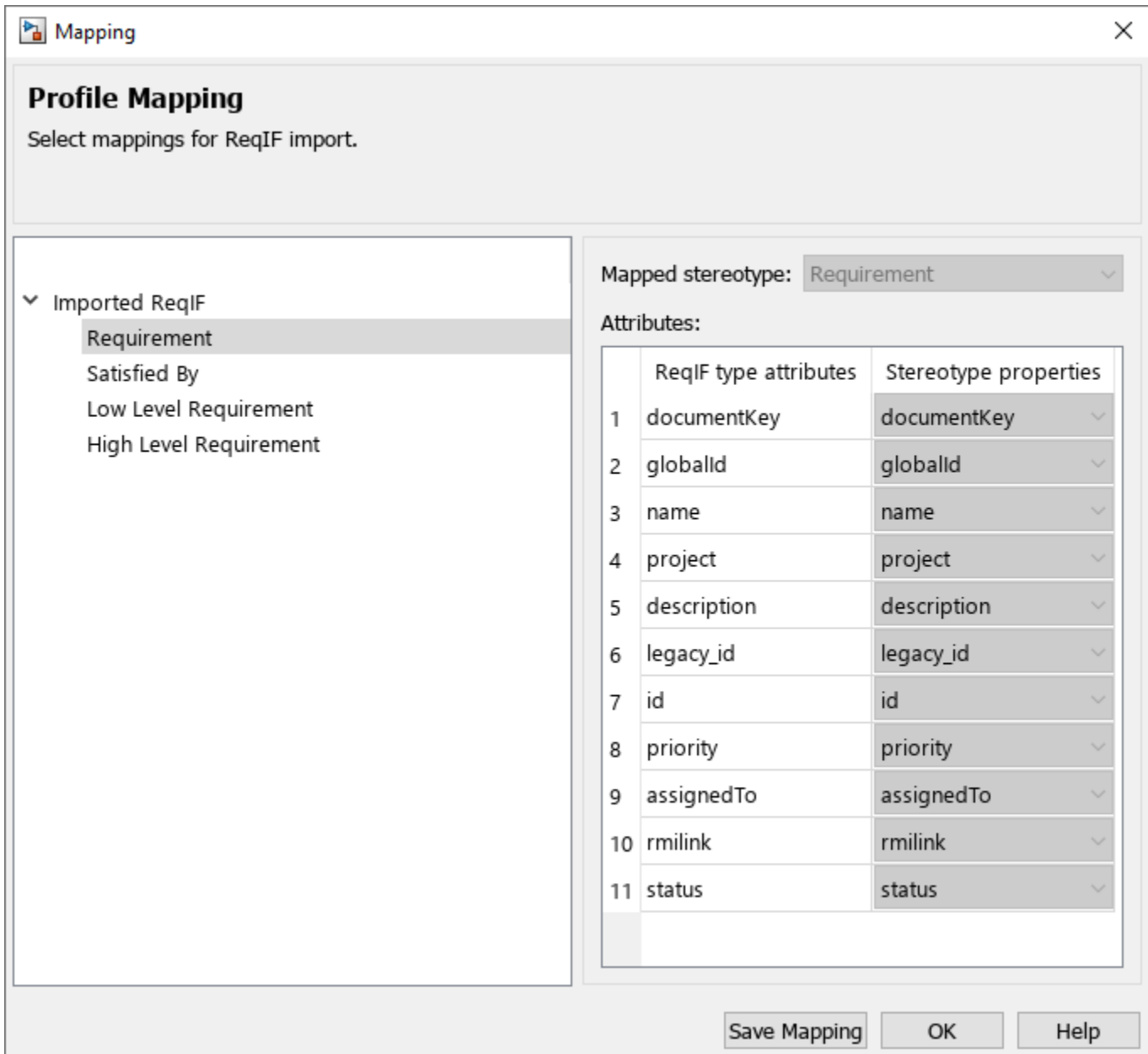
To save the mapping to an XML file so that you can re-use it, in the Mapping dialog box, click **Save Mapping**. To load an existing stereotype mapping, click **Load Mapping**.

Adjust Stereotype Mapping

You can view the mapping between ReqIF requirement and link data to Requirements Toolbox stereotypes in the **Requirements Editor** by selecting the import node and, in the right pane, under **ReqIF profile mapping**, clicking **View Mapping**.



The mapping in the Mapping dialog box is read-only.



To adjust the stereotype mapping, note the profile that is applied to the requirement set. Close the requirement set and re-import it using the same profile. Map the stereotypes as described in “Use Existing Profile During Import” on page 1-30.

See Also

Apps
Requirements Editor | Profile Editor

More About

- “Import Requirements from ReqIF Files” on page 1-17
- “Customize Requirements and Links by Using Stereotypes” on page 1-71
- “Import Requirements from Third-Party Applications” on page 1-8

Import Requirements from IBM DOORS Next

You can manage requirements in IBM DOORS Next and import either the entire requirements module or requirements that match a query into Requirements Toolbox. Requirements Toolbox imports the requirements as `slreq.Reference` objects, which are also called referenced requirements. After you establish links with the imported referenced requirements, you can use the **Requirements Editor** or Requirements Perspective to navigate from the imported referenced requirements to the original requirement in DOORS Next. You can also measure how the requirements contribute to the implementation status, verification status, and change tracking. For more information, see:

- “Review Requirements Implementation Status” on page 4-2
- “Review Requirements Verification Status” on page 4-6
- “Track Changes to Requirement Links” on page 5-3

Configure IBM DOORS Next Session

To interface with IBM DOORS Next, you must configure MATLAB every session.

- 1 At the MATLAB command prompt, enter:

```
slreq.dngConfigure
```

- 2 In the DOORS Server dialog box, provide the DOORS Next server address, port number, and service root as they appear in the web browser when accessing DOORS Next. Click **OK**.

If you do not see a port number in the web browser, enter the default value of 443.

- 3 In the Server Login Name and Server Login Password dialog box, enter your user name, then click **OK**. Enter your password, then press **Enter**.
- 4 In the DOORS Project dialog box, select the project and, if applicable, the configuration context. If your configuration context is not listed in the **Select configuration stream or changeset** list, load additional configurations by selecting **<more>**. To display global configurations in the configurations list, at the MATLAB command line, enter:

```
rmipref("OslcUseGlobalConfig",true);
```

For more information, see `rmipref` and “Specifying and Updating the IBM DOORS Next Configuration” on page 8-12.

MATLAB then tests the connection by opening a window in your browser. If the connection is successful, the MATLAB Connector Test dialog box appears with a confirmation message. Click **OK**. If the dialog does not appear or if an error appears after you enter `slreq.dngConfigure`, see the Tips for using `slreq.dngConfigure`.

Note If you plan to create direct links to requirements in IBM DOORS Next, you must leave the test connection browser window open, because this instance of the web browser is authenticated to communicate with MATLAB. You can re-open the test connection browser window by copying and pasting this address in the browser address bar: `https://localhost:31515/matlab/oslc/inboundTest`. Use this browser to select requirements in your IBM DOORS Next project to create direct links.

Import DOORS Next Requirements

You can import requirements by selecting a DOORS Next module or by creating a query. Because Requirements Toolbox imports requirements as `slreq.Reference` objects, you must import the requirements into a new requirement set. You cannot import images in requirements from DOORS to Requirements Toolbox.

When you import requirements from IBM DOORS Next, Requirements Toolbox sets the Type property based on the external type:

IBM DOORS Next Requirement Type	Requirements Toolbox Requirement Type
Heading	Container
Information	Informational
Diagrams and sketches	Informational
All other types	Functional

For more information, see “Requirement Types” on page 1-6.

Additionally, Requirements Toolbox disables index numbering for requirements that do not have the external type `Heading`. For more information, see “Customize Requirement Index Numbering” on page 1-85.

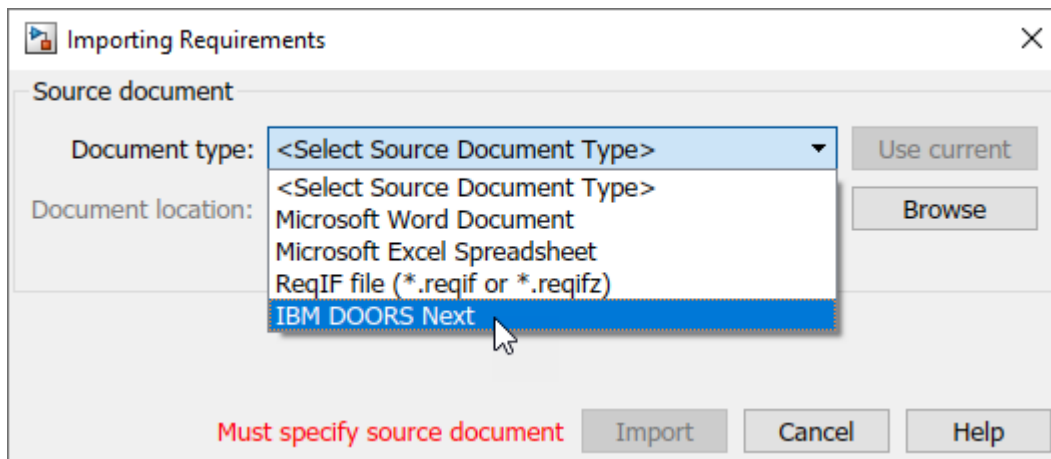
Note When you import requirements from a DOORS Next project that has configuration management enabled, you must select the configuration context for your MATLAB session before importing. The content of the imported requirements corresponds to the configuration. For more information about enabling configuration management in DOORS Next, see Configuration management in the RM application on the IBM website. For more information about configuring your MATLAB session, see `slreq.dngConfigure`.

Importing a Requirements Module

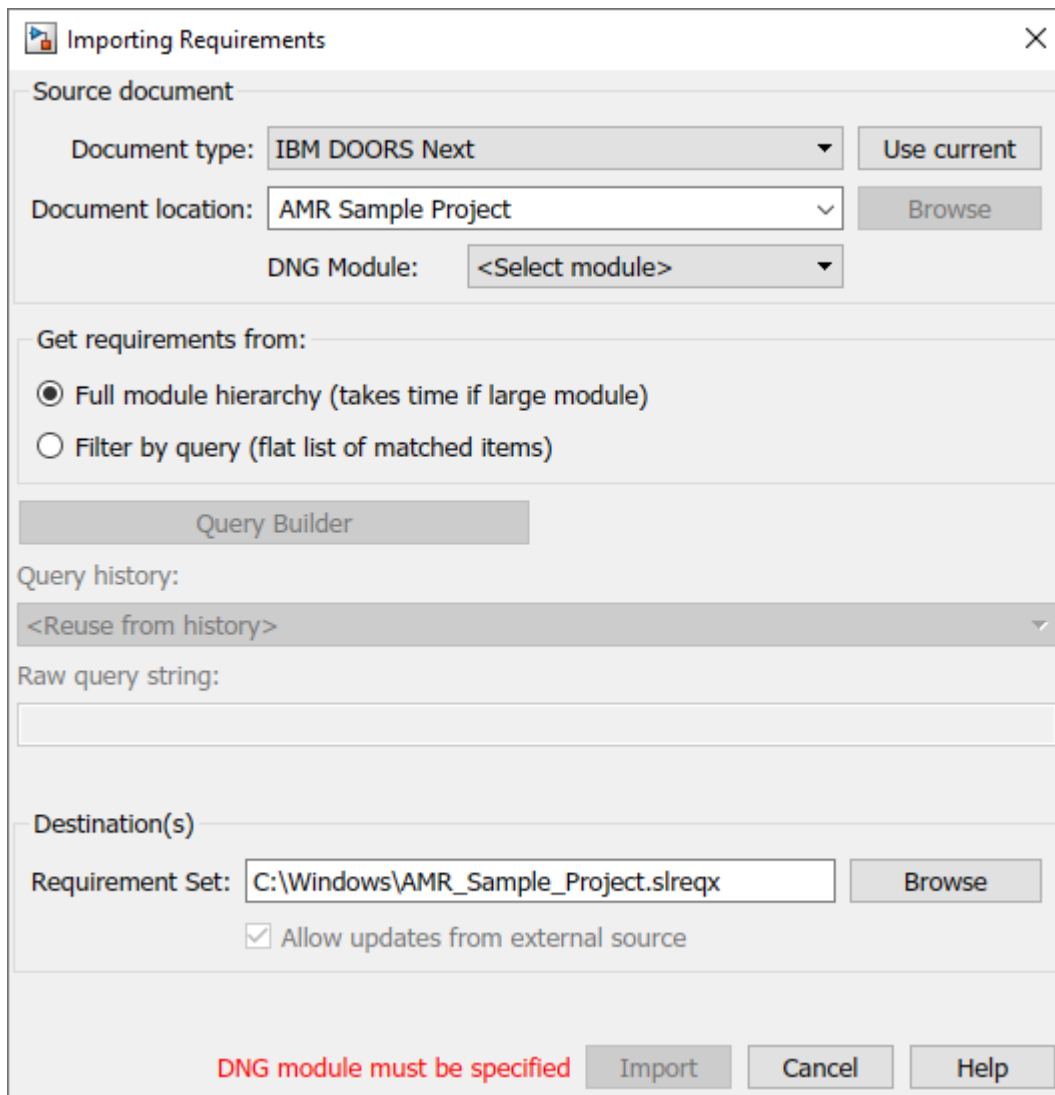
When you import requirements from a DOORS Next module, the Requirements Toolbox imports the entire module.

- 1 Open the **Requirements Editor**:
`slreq.editor`
- 2 In the **Requirements Editor**, click **Import**.

In the Importing Requirements dialog box, set **Document type** to `IBM DOORS Next`.



- 3 Set **Document location** to the project that you want to work with.
- 4 Under **Get requirements from**, select **Full module hierarchy (takes time if large module)**. Wait for the **DNG Module** list to populate.

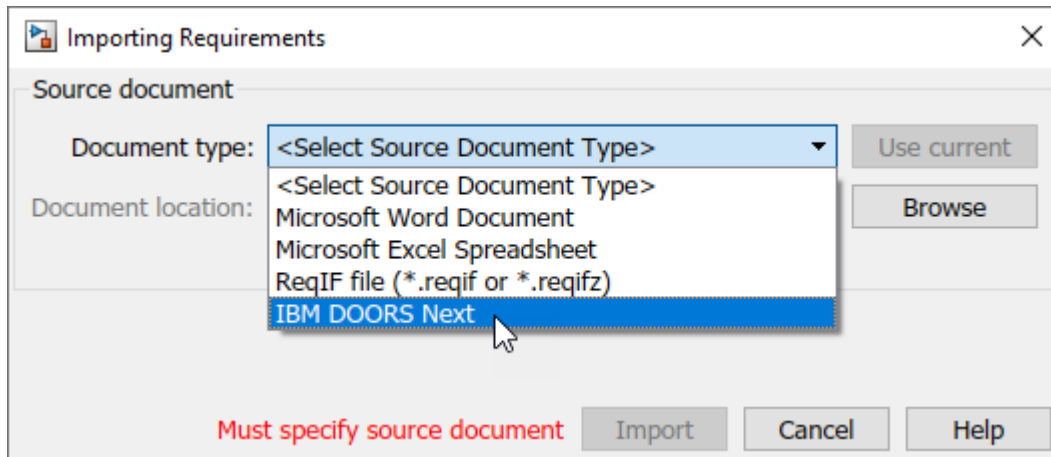


- 5 Select the desired module from the **DNG Module** list.
- 6 Enter the name and file path for the **Requirement Set**. You can click **Browse** to browse for a save location.
- 7 Click **Import** and wait for the process to import the data from the server into Requirements Toolbox. When the import completes, the **Requirements Editor** displays the hierarchy of the imported items.

Importing Requirements by Using Queries

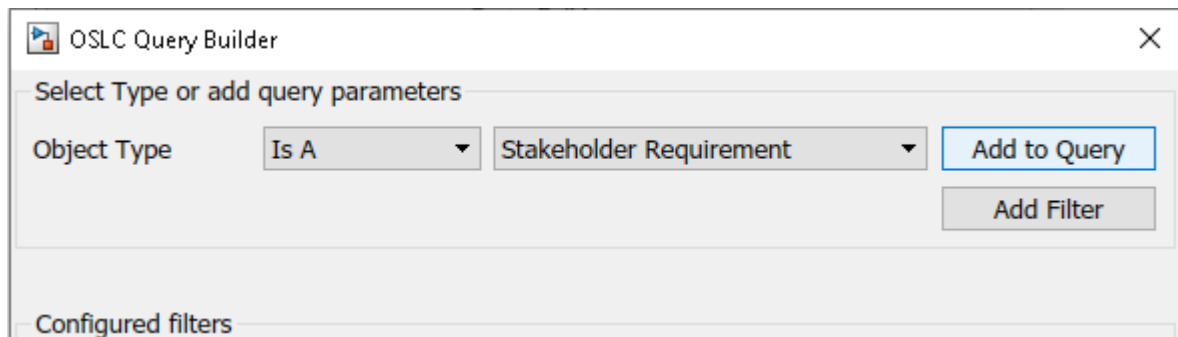
- 1 Open the **Requirements Editor** by entering:
slreq.editor
- 2 In the **Requirements Editor**, click **Import**.

In the Importing Requirements dialog box, set **Document type** to IBM D00RS Next.



- 3 Set **Document location** to the project that you want to work with.
- 4 Under **Get requirements from**, select **Filter by query (flat list of matched items)**.
- 5 Click **Query Builder** to open the OSLC Query Builder dialog box and specify your query. Use the drop-down menus next to **Object Type** to choose the object type to import.

For example, you can import only stakeholder requirements by setting the drop-down menus to Is A and Stakeholder Requirement.



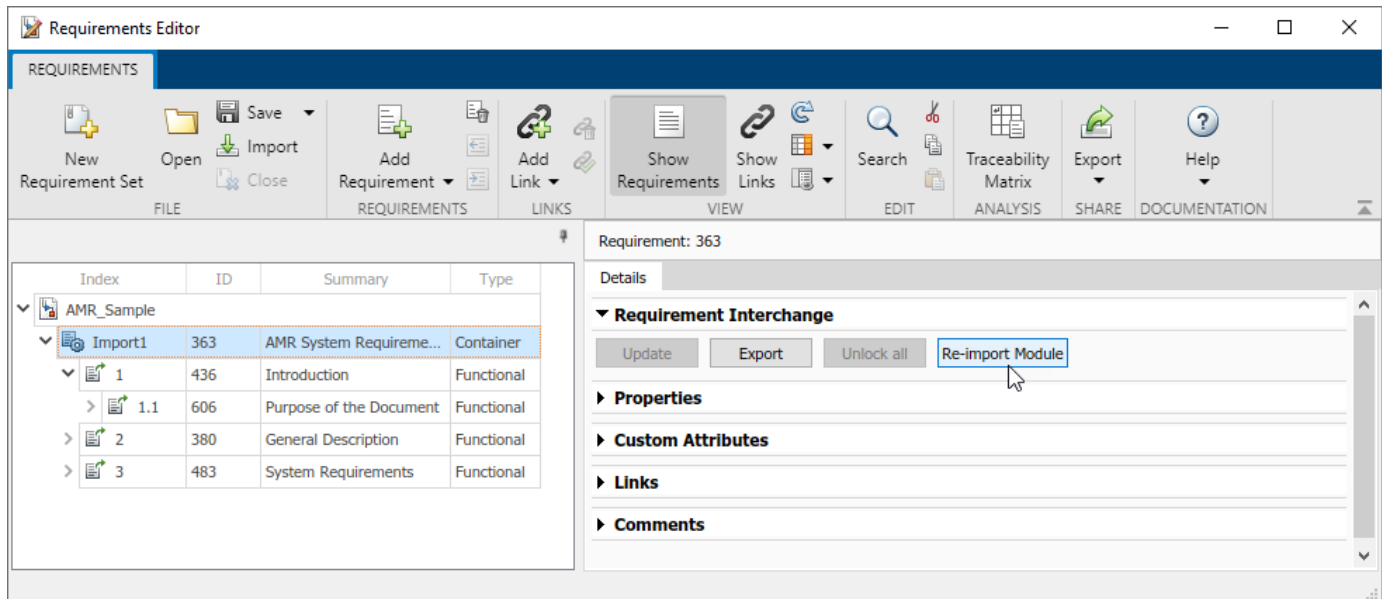
- 6 To create a query with other query parameters, click **Add Filter**.

Note If an attribute is only defined for a given type of object, you must set **Object Type** to that object type before you can filter by that attribute.

- 7 When you are done building your query, click **Add to Query**, then click **OK**. In the Importing Requirements dialog, the **Raw query string** is populated.
- 8 Enter the name and file path for the **Requirement Set**. You can click **Browse** to browse for a save location.
- 9 Click **Import** and wait for the process to import the data from the server into Requirements Toolbox.

Update Referenced Requirements

If you update the requirements in DOORS Next after importing them to Requirements Toolbox, you can update the requirement set to reflect the changes. In the **Requirements Editor**, select the top import node and, in the right pane, under **Requirement Interchange**, click **Re-import Module** or **Re-run Query**, depending on the type of import you originally did.



If the update changes or removes a referenced requirement that has links, the links have a change issue. For more information, see “Track Changes to Requirement Links” on page 5-3.

Tip Re-importing a large module might take some time. If you know which requirement has changed on the DOORS Next server, you can select that referenced requirement in the **Requirements Editor** and in the right pane, under **Properties**, click **Update from Server** to update that individual requirement.

If your DOORS Next project has configuration management enabled and you selected a configuration context when you configured your MATLAB session, then the DOORS Next requirement updates from the configuration context.

Navigate from Referenced Requirements to Requirements in DOORS Next

You can use the **Requirements Editor** to navigate from the referenced requirement to the original requirement in DOORS Next. Select the referenced requirement in the **Requirements Editor**. In the right pane, under **Properties**, click **Show in document**.

You can also use the Requirements Perspective to navigate from the imported referenced requirement to the original requirement. In a Simulink model, navigate to the **Apps** tab and select **Requirements Manager**. Ensure that **Layout > Requirements Browser** is selected. In the Requirements pane, in the **View** drop-down, select Requirements. Select a requirement. In the Property Inspector, in the **Details** tab, under **Properties**, click **Show in document**.

If your DOORS Next project has configuration management enabled and you selected a configuration context when you configured your MATLAB session, then the DOORS Next requirement opens in the configuration context.

Linking with Referenced Requirements

After importing requirements from DOORS Next to Requirements Toolbox, you can link these referenced requirements the same way you link other `slreq.Reference` objects. For more information, see “Create and Store Links” on page 3-31.

You can also add backlinks in your DOORS Next project, which allow you to navigate from DOORS Next requirements to items that are linked to the corresponding referenced requirement in Requirements Toolbox. For more information, see “Inserting Backlinks in DOORS Next” on page 8-5.

See Also

Requirements Editor | `slreq.dngConfigure`

More About

- “Link and Trace Requirements with IBM DOORS Next” on page 8-4
- “Import Requirements from Third-Party Applications” on page 1-8
- “Import Requirements from ReqIF Files” on page 1-17

Import Requirements from IBM Rational DOORS

You can import either the entire requirements module or a subset of requirements from an IBM Rational DOORS module. Requirements Toolbox imports the requirements as `slreq.Reference` objects, which are also called referenced requirements.

After you establish links with the imported referenced requirements, you can use the **Requirements Editor** or Requirements Perspective to navigate from the imported referenced requirements to the original requirement in IBM Rational DOORS.

Configure IBM Rational DOORS Session

To interface with IBM Rational DOORS, you must configure MATLAB after you install or update MATLAB or IBM Rational DOORS. Close open instances of IBM Rational DOORS. Then, at the MATLAB command prompt, enter:

```
rmi setup doors
```

For more information, see “Configure Requirements Toolbox for IBM Rational DOORS” on page 6-2.

Note Depending on the permissions required by your machine, you might need to run MATLAB and/or IBM Rational DOORS as an administrator.

Import an Entire Requirements Module

You can import requirements from IBM Rational DOORS from the **Requirements Editor** or the MATLAB command line. To import requirements programmatically, see “Import Requirements from IBM Rational DOORS by Using the API” on page 1-119.

To import an IBM Rational DOORS requirements module in the **Requirements Editor**, first open the **Requirements Editor** by using one of these approaches:

- At the MATLAB command line, enter:

```
slreq.editor
```
- In the MATLAB **Apps** tab, under **Verification, Validation, and Test**, click the **Requirements Editor** app.
- In the Simulink **Apps** tab, under **Model Verification, Validation, and Test**, click the **Requirements Editor** app.

Open the project in IBM Rational DOORS that contains the requirements modules that you want to import. Then, in the **Requirements Editor**:

- 1** Click **Import**.
- 2** In the Importing Requirements dialog box, set **Document Type** to IBM Rational DOORS Module.
- 3** Next to **Document Location**, click **Use current** to select the active requirements module, or click **Browse** to open the Browse dialog in IBM Rational DOORS. In the Browse dialog, select the module that you want to import, then click **OK**.

- 4 In the Importing Requirements dialog, under **Content**, select **Text only (better performance)** to import requirements as text or **Include graphics and layout** to import images, graphics, and text formatting.
- 5 Under **Row filter**, the dialog shows the currently applied filter in your selected IBM Rational DOORS requirements module. If you do not see the currently applied filter, click **Refresh**. Requirements Toolbox imports only the requirements that match the currently applied filter. For more information, see “Import a Subset of Requirements from a Module” on page 1-44.

- 6 Under **Attributes to import**, click **Map Attributes** to select the attributes to import from your requirements module. Requirements Toolbox maps some default attributes automatically to requirements properties. In the DOORS Module dialog box, you can map additional attributes to remaining unmapped requirements properties or to custom attributes. You can omit an attribute during import by selecting <Ignore>.

You can also edit the attribute mapping after you import. For more information, see “Create and Edit Attribute Mappings” on page 1-110.

- 7 Under **Destination(s)**, enter the name and file path for the requirement set. Click **Browse** to select a save location.

- 8 Click **Import**. When the import completes, the **Requirements Editor** displays the requirements hierarchy.

Requirements Toolbox imports the requirements as referenced requirements in a new requirement set. If you make changes to the requirements module in IBM Rational DOORS, you can update the referenced requirements. For more information, see “Update Imported Requirements” on page 1-89.

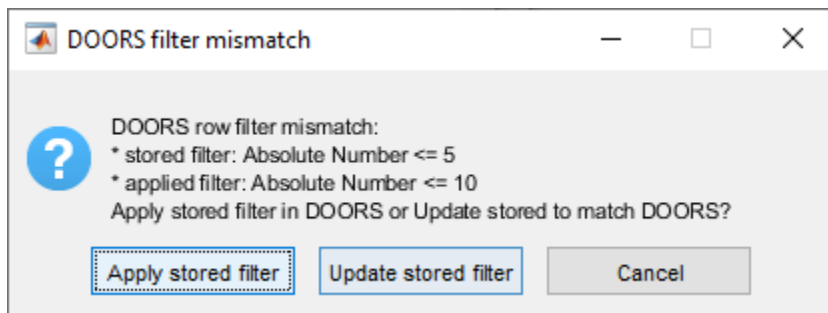
Import a Subset of Requirements from a Module

You can import a subset of requirements from an IBM Rational DOORS requirements module by applying a filter to the module. For more information about applying a filter to a requirements module, see Defining filters on the IBM website.

When you import requirements that have an applied filter in the **Requirements Editor**, the Importing Requirements dialog displays the filter. Only the requirements that match the filter import to Requirements Toolbox. For more information, see “Import an Entire Requirements Module” on page 1-42.

Update the Filtered Requirement Set

When you import the requirements, you can choose to store the filter by selecting **Store current row filter to apply automatically in future updates** in the Importing Requirements dialog. You can use this filter if you update the requirement set. If you stored the filter, but update the requirement set with a different filter, the DOORS filter mismatch dialog box appears.



You can then:

- Update the requirement set with the filter that was stored during import by clicking **Apply stored filter**. The import process updates the requirements in the requirement set to reflect any changes made in your requirements module.
- Update the requirement set by using the currently applied filter in your requirements module by clicking **Update stored filter**. This action replaces the currently stored filter with the new filter. Requirements Toolbox adds the requirements to or removes them from the requirement set to reflect the currently applied filter in your requirements module and updates the existing requirements to reflect the changes in DOORS.

If you choose not to store the filter during import and then update the requirement set, Requirements Toolbox adds to or removes requirements from the requirement set to reflect the currently applied filter in your requirements module and updates existing requirements in the requirement set to reflect the changes made in the requirements module.

Update the Requirement Set

After you import requirements from an IBM Rational DOORS requirements module, you can update the requirement set. For more information, see “Update Imported Requirements” on page 1-89.

Navigate Between Referenced Requirements and Requirements in IBM Rational DOORS

You can navigate from a referenced requirement to the original requirement in an IBM Rational DOORS requirements module, or from the original requirement to a referenced requirement in MATLAB.

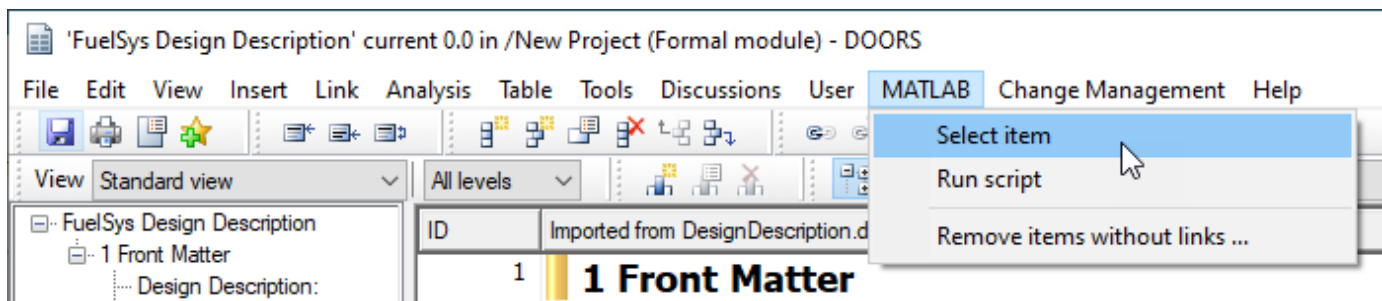
Navigate from MATLAB to IBM Rational DOORS

To navigate from the **Requirements Editor** to the original requirement in IBM Rational DOORS, select the referenced requirement in the **Requirements Editor**. In the right pane, under **Properties**, click **Show in document**.

You can also use the Requirements Perspective to navigate to the original requirement. In a Simulink model, open the **Apps** tab and select **Requirements Manager**. Ensure that **Layout > Requirements Browser** is selected. In the Requirements pane, in the **View** drop-down, select **Requirements**, then select a requirement. In the Property Inspector, in the **Details** tab, under **Properties**, click **Show in document**.

Navigate from IBM Rational DOORS to MATLAB

To navigate from a requirement in an IBM Rational DOORS requirements module to the corresponding referenced requirement in Requirements Toolbox, select the requirement, then click **MATLAB > Select item**. The referenced requirement opens in the **Requirements Editor**.



See Also

Requirements Editor | `slreq.import`

Related Examples

- “Import Requirements from IBM Rational DOORS by Using the API” on page 1-119
- “Working with IBM Rational DOORS 9 Requirements” on page 8-30

More About

- “Import Requirements from Third-Party Applications” on page 1-8
- “Import Requirements from IBM DOORS Next” on page 1-35
- “Import Requirements from ReqIF Files” on page 1-17
- “Update Imported Requirements” on page 1-89
- “Create and Edit Attribute Mappings” on page 1-110
- “Manage Navigation Backlinks in External Requirements Documents” on page 3-51

Define Custom Document Interface for Importing Requirements

You can define custom document interfaces to import requirements from third-party applications that Requirements Toolbox does not support. To define a custom document interface, create a function that specifies the interface properties and callback functions to use during the import process. You can also create a custom version of a built-in Requirements Toolbox document interface for a supported products, such as Microsoft Word. Your custom version of a built-in document interface can have different properties or callback functions than the built-in type.

Alternatively, you can import requirements from third-party applications that Requirements Toolbox does not support by importing ReqIF files. For more information, see “Import Requirements from ReqIF Files” on page 1-17.

Define Custom Document Interface

To define a custom document interface for importing requirements:

- 1 Author a function that defines and returns an instance of the document interface object, `ReqMgr.LinkType`. Your function must take this form:

```
function docDomain = myDomainType
    docDomain = ReqMgr.LinkType;
end
```

- 2 Define these properties for the document interface object:

Name	Description	Value Types	Example Value
Registration	Requirement document interface name, specified as a string scalar or character vector. The value of this property must be the same as the document interface definition function name.	<ul style="list-style-type: none"> • String • Character vector 	mfilename
Label	Label for the custom document type in the Importing Requirements Dialog, specified as a string scalar or character vector.	<ul style="list-style-type: none"> • String • Character vector 	"My Custom Document Interface"

Name	Description	Value Types	Example Value
IsFile	Indicator for the file-based requirement documents, specified as a logical 1 (true) or 0 (false). Set the value to 1 if your requirement document is file-based, or 0 if your requirement document is not file based.	<ul style="list-style-type: none">• Logical 1 (true) or 0 (false)	1
Extensions	Allowed file type extensions for file based requirement documents, specified as a cell array of character vectors. Leave empty for requirement documents that are not file based.	Cell array of character vectors	{'.doc', '.docx'}

Name	Description	Value Types	Example Value
LocDelimiters	<p>Characters that are used to specify the supported types of location identifiers in the requirement document, specified as a string scalar or character array that is a combination of one or more of these characters:</p> <ul style="list-style-type: none"> • ?: Used to search for text in the document • @: Used as a named item such as a requirement ID • #: Page number or item number • >: Item number • \$: Cell address in spreadsheet <p>This property is optional.</p>	Character array containing one or more of these characters: ?, @, #, >, \$	'?@#'

3 Determine which callback functions your document interface object needs:

Callback	Description	Required or Optional?	Function Syntax and Description
NavigateFcn	Function to navigate from imported requirement to original requirement in external document using Show In Document in the Requirements Editor	Required	NavigateFcn(document, id) opens the external requirement document specified by document to the requirement specified by id.

Callback	Description	Required or Optional?	Function Syntax and Description
ContentsFcn	Function to get the table of contents of the external requirement document or server	Required	[labels, depths, ids] = ContentsFcn(document) returns requirements information as cell arrays for the external document specified by document. labels contains requirements text, ids contains requirements identifiers, and depths contains the level of the requirement in the hierarchy.
BrowseFcn	Function to execute when you click the Browse button in the Importing Requirements dialog box for requirement documents that are not file-based. This function is not required for file-based requirements domains.	Optional	BrowseFcn opens the browser for requirement documents in the non-file-based requirements application.
SummaryFcn	Function to get text for the "Summary" property of the imported requirements	Optional, but if you do not define this function, you have to manually map an imported attribute to the Summary property. For more information, see "Create and Edit Attribute Mappings" on page 1-110.	summary = SummaryFcn(document, id) returns the summary for the requirement specified by the identifier id in the external requirement document specified by document.

Callback	Description	Required or Optional?	Function Syntax and Description
HtmlViewFcn	Function to get the HTML view of the requirements content for rich text import and assign it to the "Description" property of the imported requirements	Optional, but if you do not define this function or TextViewFcn, you have to manually map an imported attribute to the Description property. For more information, see "Create and Edit Attribute Mappings" on page 1-110.	html = HtmlViewFcn(document, id) returns the HTML content of the requirement specified by the identifier id in the external requirement document specified by document.
TextViewFcn	Function to get the plain text view of the requirements content for plain text import	Optional, but if you do not define this function or HtmlViewFcn, you have to manually map an imported attribute to the Description property. For more information, see "Create and Edit Attribute Mappings" on page 1-110.	text = TextViewFcn(document, id) returns the plain text content of the requirement specified by the identifier id in the external requirement document specified by document.

Callback	Description	Required or Optional?	Function Syntax and Description
AttributeNamesFcn	Function to get the names of the requirement attributes to import	Optional, but if you do not define this function, your imported requirements do not have custom attributes or properties except for those imported by SummaryFcn, HtmlViewFcn, or TextViewFcn. For more information about custom attributes, see "Create and Edit Attribute Mappings" on page 1-110.	<p>[attributes, datatype] = AttributeNamesFcn(document, id) returns the attribute names, attributes, and details about the attribute data type, datatype, for the requirement specified by the identifier id in the external requirement document specified by document.</p> <ul style="list-style-type: none"> • If the attribute type is a Boolean, the datatype output is the default Boolean value, true or false. • If the attribute type is numeric, the datatype output is the default numeric value. • If the attribute is an enumeration, the datatype output is a cell array of character vectors of the possible enumeration values.

Callback	Description	Required or Optional?	Function Syntax and Description
GetAttributeFcn	Function to get value of the requirement attributes to import	Required if you define a callback function for AttributeNamesFcn	value = GetAttributeFcn(document, id, attrName) returns the attribute value of the attribute attrName for the requirement specified by the identifier id in the external requirement document specified by document.
CreateURLFcn	Function to navigate from requirements report to requirement in external document	Optional	url = CreateURLFcn(document, documentURL, id) returns the URL for the requirement specified by the identifier id in the external requirement document specified by document or the document URL documentURL.
UrlLabelFcn	Function to get the hyperlink text for the requirement navigation URL in requirements reports	Required if you define a callback for CreateURLFcn	label = UrlLabelFcn(document, id) returns a label for the requirement specified by the identifier id in the external requirement document specified by document.
IsValidDocFcn	Function to check if the external requirement document exists and is valid for this document interface	Optional. If you define this callback, you can use the Model Advisor to check for consistency.	tf = IsValidDocFcn(document, ~) returns true if the external requirement document specified by document exists and is valid for the document interface.

Callback	Description	Required or Optional?	Function Syntax and Description
IsValidIdFcn	Function to check if the requirement ID exists in external requirement document	Optional. If you define this callback, you can use the Model Advisor to check for consistency.	<pre>tf = IsValidIdFcn(document, id)</pre> returns true if the requirement specified by the identifier <code>id</code> exists in the external requirement document specified by <code>document</code> .
IsValidDescFcn	Function to check if the imported requirement description matches the string found at given requirement ID in external requirement document	Optional. If you define this callback, you can use the Model Advisor to check for consistency.	<pre>[tf, newDesc] = IsValidDescFcn(document, id, currDesc)</pre> checks if the current value of the <code>Description</code> property of the imported requirement, <code>currDesc</code> , matches the text found at the location of the external requirement specified by <code>id</code> , in the external requirement document specified by <code>document</code> .

Callback	Description	Required or Optional?	Function Syntax and Description
BacklinkCheckFcn	Function to check if backlink exists in external requirement documents. For more information about backlinks, see "Manage Navigation Backlinks in External Requirements Documents" on page 3-51.	Optional	<pre>[linkExists, newLinkURL] = BacklinkCheckFcn(mwArtifactName, mw ItemId, extDoc, ext Req) checks whether the external requirement extReq in the external requirement document extDoc has a backlink to the linkable item in MATLAB or Simulink specified by mwItemId in the artifact mwArtifactName.</pre> <ul style="list-style-type: none"> • If the backlink exists, linkExists returns true and newLinkURL returns empty. • If the backlink does not exist, linkExists returns false and newLinkURL contains the URL to insert as a backlink.

Callback	Description	Required or Optional?	Function Syntax and Description
BacklinkInsertFcn	Function to insert backlinks in external requirement documents. For more information about backlinks, see "Manage Navigation Backlinks in External Requirements Documents" on page 3-51.	Required if you define a callback function for BacklinkCheckFcn	[navCmd, dispText] = BacklinkInsertFcn(extDoc, extReq, mwSourceArtifact, mwItemId, mwDomain) inserts a backlink from the external requirement specified by extReq in the external requirement document extDoc to the linkable item in MATLAB or Simulink specified by mwItemId in the artifact specified by mwSourceArtifact with the artifact document interface mwDomain. The function returns the navigation command that the backlink uses, navCmd, and the displayed hyperlink text for the backlink, dispText.

Callback	Description	Required or Optional?	Function Syntax and Description
BacklinksCleanupFcn	Function to check for stale backlinks in external requirement documents. For more information about backlinks, see "Manage Navigation Backlinks in External Requirements Documents" on page 3-51.	Optional, but if this function is not defined, the Update Backlinks context menu option in the Requirements Editor cannot check for and delete stale backlinks	<ul style="list-style-type: none"> <li data-bbox="1203 331 1469 1150">• [countRemoved, countChecked] = BacklinksCleanupFcn(extDoc, mwSourceArtifact, mwLinksDataMap) checks the backlinks in the external requirement document extDoc that point to linkable items in the MATLAB or Simulink artifact mwSourceArtifact and deletes backlinks that do not have a corresponding forward link in the Requirements Toolbox link map specified by mwLinksDataMap. <li data-bbox="1203 1161 1469 1570">• [countRemoved, countChecked] = BacklinksCleanupFcn(extDoc, mwSourceArtifact, mwLinksDataMap, saveBeforeCleanup) saves the requirement document before deleting the backlinks.

Callback	Description	Required or Optional?	Function Syntax and Description
BacklinkDeleteFcn	Function to delete backlinks in external requirement document. For more information about backlinks, see "Manage Navigation Backlinks in External Requirements Documents" on page 3-51.	Optional	success = BacklinkDeleteFcn(extDoc,extReq,mwSourceArtifact,mwItemId) deletes the backlink from the external requirement specified by extReq in the external requirement document extDoc that points to the linkable item in MATLAB or Simulink specified by mwItemId in the artifact specified by mwSourceArtifact.

- 4 Author the callback functions that your document interface needs as local functions in the main function.
- 5 Assign the callback function handles to the corresponding properties of the ReqMgr.LinkType object.

For an example of a custom document interface, see the custom_jira function.

```
open([matlabroot '\toolbox\slrequirements\linktype_examples\custom_jira.m'])
```

Define Custom Version of Built-In Document Interface

To define a custom version of a built-in document interface:

- 1 Author a function that defines and returns an instance of one of the built-in document interfaces:
 - Microsoft Word: linktypes.linktype_rmi_word
 - Microsoft Excel: linktypes.linktype_rmi_excel
 - IBM Rational DOORS: linktypes.linktype_rmi_doors

Your function must take this form:

```
function docDomain = custom_word
    docDomain = linktypes.linktype_rmi_word;
end
```

- 2 Overwrite the Registration and Label properties of the ReqMgr.LinkType object by setting them to custom values. You can also overwrite other properties. For more information, see "Define Custom Document Interface" on page 1-47.
- 3 Determine which callback functions to overwrite. For more information, see "Define Custom Document Interface" on page 1-47.
- 4 Author new callback functions as local functions in the main function.

- 5 Assign the callback function handles to the corresponding properties of the `ReqMgr.LinkType` object.

For example, this code sets the `NavigateFcn` property of the document interface object to a local function called `myNavigateFcn`.

```
function docDomain = custom_word
    docDomain = linktypes.linktype_rmi_word;
    docDomain.NavigateFcn = @myNavigateFcn;
end

function myNavigateFcn(document,id)

end
```

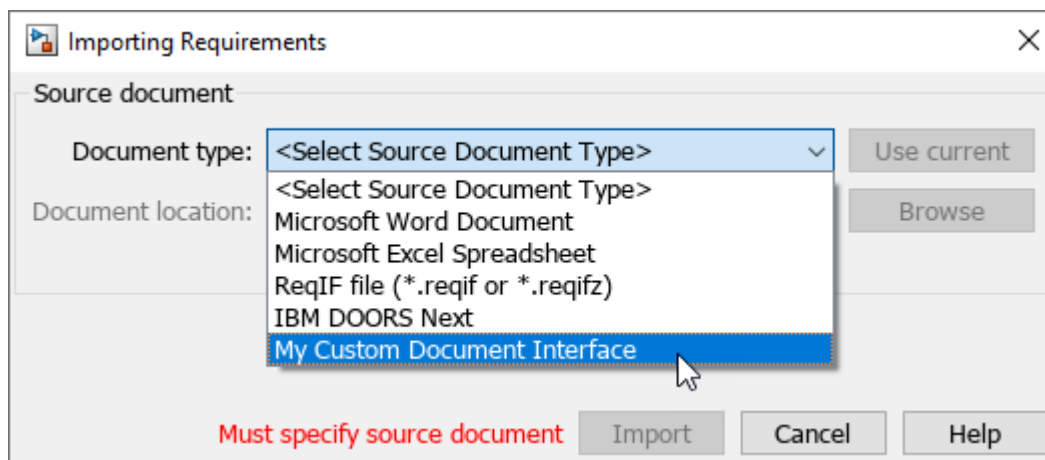
For an example of a custom version of a built-in document interface, see the `custom_word` function.

```
open([matlabroot '\toolbox\slrequirements\linktype_examples\custom_word.m'])
```

Use Custom Document Interface During Import

To use the custom document interface during import:

- 1 Register the custom document interface.
`rmi register myDomainType`
- 2 Open the **Requirements Editor**. For more information, see “Open the Requirements Editor App”.
- 3 Click **Import**.
- 4 In the Importing Requirements dialog, set **Document type** to the custom document interface.



For more information about importing requirements, see “Import Requirements from Third-Party Applications” on page 1-8.

The custom domain interface registration persists between MATLAB sessions. To unregister the document interface, enter:

```
rmi unregister myDomainType
```

See Also

Apps
Requirements Editor

More About

- “Import Requirements from Third-Party Applications” on page 1-8
- “Import Requirements from ReqIF Files” on page 1-17

Export Requirements to ReqIF Files

Many third-party requirements management tools support data exchange using the Requirements Interchange Format, also known as ReqIF. You can export requirements in Requirements Toolbox to a ReqIF file.

Choosing an Export Mapping

ReqIF represents requirements as `SpecObject` objects and links as `SpecRelation` objects between `SpecObject` objects. Each `SpecObject` object specifies the associated `SpecObjectType` object and the `SpecRelationType` objects classify each `SpecRelation` object. The `SpecObjectType` and `SpecRelationType` objects define attributes to store requirements and link information. The `SpecObject` and `SpecRelation` objects contain values for these attributes.

When you export requirements and links to a ReqIF file, the export process maps the Requirements Toolbox objects to `SpecObject` and `SpecRelation` objects. The exported value of the `SpecObjectType` and `SpecRelationType` objects depends on the export mapping that you choose.

For more information about ReqIF data organization, see the Exchange Document Content section in Requirements Interchange Format (ReqIF) Version 1.2.

Requirements Toolbox provides built-in export mappings for some third-party applications that use ReqIF:

- IBM Rational DOORS
- IBM DOORS Next
- Polarion
- PREEvision
- Jama

You can also use a generic mapping.

A ReqIF round-trip is when you import requirements from a ReqIF file, edit the requirements, and export them back to a ReqIF file. When you import requirements during a ReqIF round-trip, avoid unexpected behavior by using either:

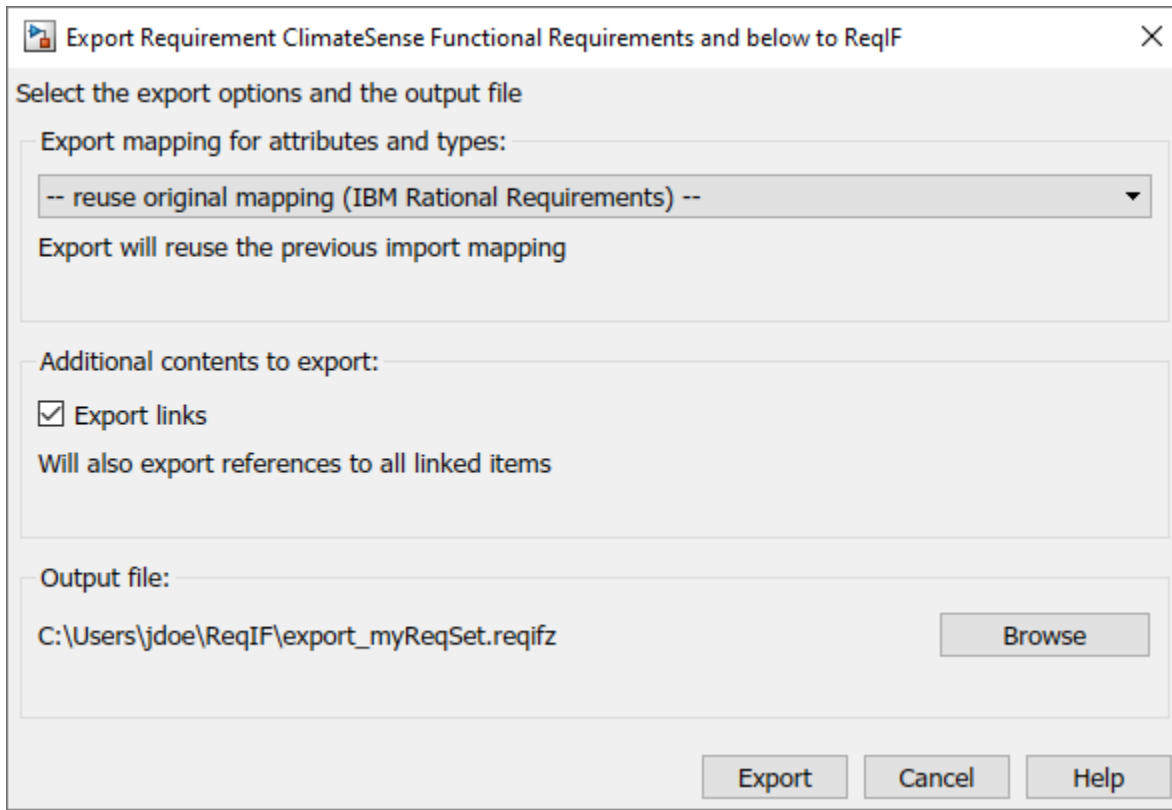
- A generic mapping
- The same mapping for import and export

For more information about ReqIF round-trips, see “Round-Trip Importing and Exporting for ReqIF Files” on page 1-99.

When you export requirements authored in Requirements Toolbox, use a generic mapping.

Reusing the Import Mapping During Export

If you import requirements from a ReqIF file, you can change the requirement types manually or by mapping the `SpecObjectType` object values to requirement types in Requirements Toolbox. For more information, see “Map SpecObjectTypes to Requirement Types” on page 1-24. If you export requirements during a round-trip with the same attribute mapping used for the import, the exported `SpecObjectType` object values revert to the original imported values regardless of changes that you made to the requirement type after importing.



Similarly, if you import links from a ReqIF file, you can change the link types manually. If you export links during a round-trip and use the same attribute mapping used for the import, the exported `SpecRelationType` object values revert to the original imported values.

Using the Generic Mapping During Export

When you export requirements content to a ReqIF file by using a generic attribute mapping, the requirements and referenced requirements that use the built-in requirement types on page 1-6 and the justifications export as `SpecObject` objects with the associated `SpecObjectType` object `LongName` attribute set to `Requirement`. However, you can specify what `LongName` is set to by setting the Requirements Toolbox requirement type to a custom requirement type. For more information about creating custom requirement types, see “Define Custom Requirement and Link Types by Using `sl_customization` Files” on page 3-42.

When you export links to a ReqIF file by using the generic mapping, Requirements Toolbox exports the links as `SpecRelation` objects with the associated `SpecRelationType` object `LongName` attribute set to the same value as the link type in Requirements Toolbox. For more information about link types, see “Link Types” on page 3-34.

Exporting Requirement Attributes

The `SpecObjectType` object defines requirement attributes. Each `SpecObject` object specifies the associated `SpecObjectType` object. The `SpecObject` object also contains the requirement attribute values. For more information, see the table in “Choosing an Import Mapping” on page 1-17.

If your ReqIF file contains `SpecObjectType` objects that have requirement attributes and you export the requirements to a ReqIF during a round-trip, the exported `SpecObject` object attribute values

revert to the original imported values regardless of the export mapping chosen. The values revert even if you mapped the attributes to requirement properties after the import. For more information about editing attribute mappings for requirements after import, see “Mapping ReqIF Attributes in Requirements Toolbox” on page 1-24

When you author requirements in Requirements Toolbox and export them to a ReqIF file, the export process only exports the requirement ID, summary, and custom attributes.

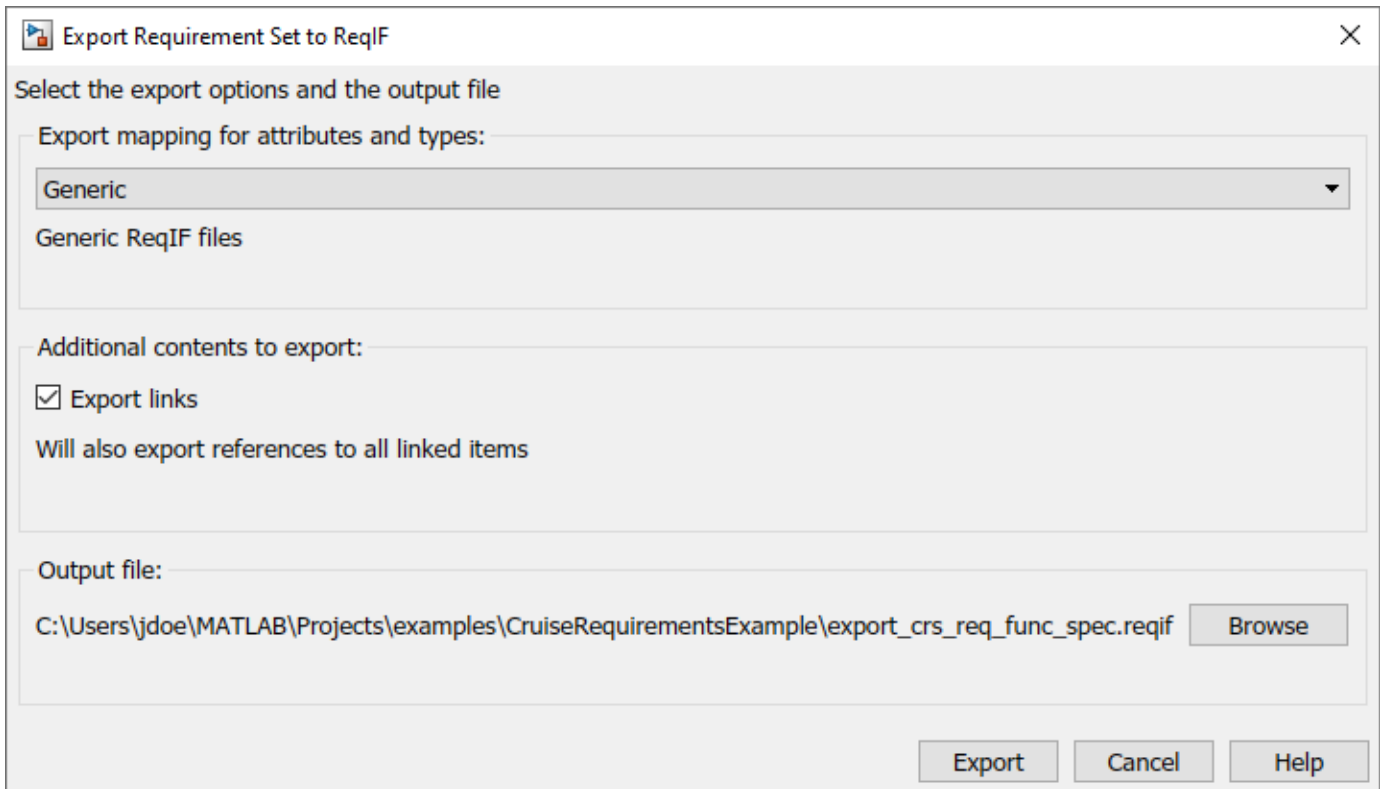
Exporting Requirements

You can export a single requirement set, a single Import node, which is denoted by , or a single parent requirement and all of its children to a ReqIF file.

If you export a single parent requirement, the export process also exports the requirements above the parent requirement up to the top-level requirement. You can only export a single parent requirement if it was authored in Requirements Toolbox.

To export requirements content:

- 1** In the **Requirements Editor**, select the requirement set, Import node, or requirement that you want to export.
- 2** Click **Export > ReqIF**.
- 3** The Export Requirement Set to ReqIF dialog appears. In the dialog, set **Export mapping for attributes and types** to the attribute mapping that aligns with your third-party tool, or set it to **Generic**. For more information, see “Choosing an Export Mapping” on page 1-61.
- 4** Under **Additional contents to export**, select **Export links** to include links in the exported ReqIF, or clear the selection to omit links.
- 5** **Output file** shows the default file path and name for the exported ReqIF file. To edit the file path or name, click **Browse** and save the file path and name by clicking **Save**.
- 6** Export the ReqIF file by clicking **Export**.



Exporting Links

If your requirements have links, you can export the links along with the requirements to a ReqIF file. For more information, see “Exporting Requirements” on page 1-63.

ReqIF represents links as `SpecRelation` objects between `SpecObject` objects. When you export links to a ReqIF file, the exported `SpecRelationType` depends on the export mapping that you use. For more information, see “Choosing an Export Mapping” on page 1-61.

If you link a requirement in Requirements Toolbox to an item that is not contained in the requirement set, such as a Simulink block or a requirement in a different requirement set, and then export the requirement and associated links to a ReqIF file, the export process inserts a `SpecObject` object into the ReqIF file that serves as a proxy object for the linked item.

If the linked item is one of the supported types, then the `SpecObjectType` object associated with the proxy `SpecObject` has a `SpecObjectType longName` value that describes the linked object type:

Linked Item	SpecObjectType longName Value
<ul style="list-style-type: none"> • Simulink model element • Stateflow® model element • System Composer™ model element 	Simulink Object

Linked Item	SpecObjectType longName Value
Simulink Test™: <ul style="list-style-type: none"> • Test file • Test suite • Test case • Iteration • Assessment 	Simulink Test Object
MATLAB code	MATLAB Code Range
Web browser URL	External Resource
Simulink data dictionary entry	Simulink DDEntry
<ul style="list-style-type: none"> • Requirement • Referenced requirement 	Requirements Toolbox Object

For other items, the proxy `SpecObject` has an associated `SpecObjectType` object with `longName` set to `Requirement`.

Note The exported proxy `SpecObject` objects include persistent IDs that can be used by the third-party tool to avoid duplicating the proxy objects. Duplication may occur if different ReqIF files contain links from the same MATLAB or Simulink object.

If you re-import a ReqIF file generated by the Requirements Toolbox export process, the software reconstructs the links that relate the proxy `SpecObject` objects and requirements for proxy objects of the supported types. Links that relate the requirements and proxy objects that have `SpecObjectType longName` value set to `Requirement` cannot be reconstructed. For more information, see “Importing Links from a ReqIF File Generated by Requirements Toolbox” on page 1-24.

See Also

Requirements Editor





More About

- “Import Requirements from ReqIF Files” on page 1-17
- “Round-Trip Importing and Exporting for ReqIF Files” on page 1-99
- “Import Requirements from Third-Party Applications” on page 1-8

Define Requirements Hierarchy

Using Requirements Toolbox, you can derive lower-level requirements from higher-level requirements to establish and manage parent-child relationships.

The requirement set is the top level of hierarchy for all requirements. All requirements in Requirements Toolbox are contained in requirement sets. Every top-level parent requirement in a requirement set is the first-level hierarchy for that set. Referenced requirements (`slreq.Reference` objects) and requirements (`slreq.Requirement` objects) cannot share a parent requirement.

Within a requirement set, you can change the level of individual requirements by using  **Promote Requirement** or  **Demote Requirement** in the **Requirements Editor**, or the   icons on the **Requirements Browser** toolbar. When you promote or demote a requirement with children, the parent-child hierarchical relationship is preserved. You can also move requirements up and down the same level of hierarchy by right-clicking the requirement and selecting **Move up** or **Move down**.

The Implementation and Verification Status metrics for a requirement set are cumulatively aggregated over all the requirements in the set. Each parent requirement in a requirement set derives its metrics from all its child requirements. For more information on the Implementation and Verification Status metrics, see “Review Requirements Implementation Status” on page 4-2 and “Review Requirements Verification Status” on page 4-6.

Requirement Sets

You can create requirement sets from the **Requirements Editor** and from the **Requirements Browser**. Requirement set files (`.slreqx`) are not inherently associated with your Simulink models.

Requirement sets have built-in properties such as the Filepath and the Revision number associated with them as metadata. Except for the **Description**, properties of the requirement set are read-only and are updated as you work with the requirement set.

Custom Attributes of Requirement Sets

Define custom attributes for your requirement sets that apply to the requirements they contain. Custom attributes extend the set of properties associated with your requirements. Define custom attributes for a requirement set from the **Custom Attribute Registries** pane of the **Requirements Editor**.

To define custom attributes:

- 1 Open the **Requirements Editor**. In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Requirements Editor**.
- 2 Select the requirement set and click **Add** in the **Custom Attribute Registries** pane.
- 3 The Custom Attribute Registration dialog box opens. Select the type of custom attribute you want to set for your requirements by using the **Type** drop-down list. You can specify custom attributes as text fields, check boxes, and combo boxes and date time entries.

To view the custom attributes for your requirements in the spreadsheet, right-click the requirement set and click **Select Attributes**.

When you define a custom attribute as a combobox, the first entry is preset to **Unset** and it cannot be renamed or deleted. Custom attributes that are imported as referenced requirements from an

external document become read-only custom attributes after they are imported. The custom attributes of a requirement set are associated with every individual requirement in the set and removing the custom attributes for a requirement set removes it from all the requirements in the set.

See “Add Custom Attributes to Requirements” on page 1-81 for more information about creating custom attributes for requirements.

See Also

Requirements Editor

More About

- “Add Custom Attributes to Requirements” on page 1-81
- “Review Requirements Implementation Status” on page 4-2
- “Review Requirements Verification Status” on page 4-6

Create Requirement Set Hierarchies by Using the Requirements Toolbox API

This example shows how to use the Requirements Toolbox™ API to create a requirement set with a custom hierarchy and custom requirement types. You create a requirement set as an `.slreqx` file.

Requirement Set Hierarchy

The requirement set that you create in this example contains two top-level parent requirements and parent justifications for implementation and verification. The requirement set follows this hierarchical structure.

Index	ID	Summary
my_New_Req_Set		
1	R1	System Requirements
1.1	R1.1	
1.1.1	R1.1.1	
1.1.2	R1.1.2	
1.1.3	R1.1.3	
1.1.3.1	R1.1.3.1	
1.2	R1.2	
2	R2	Safety Requirements
2.1	R2.1	
2.2	R2.2	
2.2.1	R2.2.1	
2.2.2	R2.2.2	
2.2.3	R2.2.3	
3	#17	Justifications
3.1	J	Requirement Justifications
3.1.1	J1	Implementation Justifications
3.1.2	J2	Verification Justifications

Create Requirement Set

Navigate to the folder where you want to create the requirement set. Create a requirement set `my_New_Req_Set` with handle `myReqSet` by using the `slreq.new()` function.

```
myReqSet = slreq.new('my_New_Req_Set');
```

Add System Requirements to the Requirement Set

Add a top-level Container requirement for System Requirements to the requirement set.

```
myParentReq1 = add(myReqSet, 'Id', 'R1', ...
    'Summary', 'System Requirements', ...
    'Type', 'Container');
```

Create child requirements for R1.


```
childReqR11 = add(myParentReq1, 'Id', 'R1.1');
childReqR12 = add(myParentReq1, 'Id', 'R1.2');
```

Create child requirements for R1.1.

```
childReqR111 = add(childReqR11, 'Id', 'R1.1.1');
childReqR112 = add(childReqR11, 'Id', 'R1.1.2');
childReqR113 = add(childReqR11, 'Id', 'R1.1.3');
```

Create a child requirement for R1.1.3.

```
childReqR1131 = add(childReqR113, 'Id', 'R1.1.3.1');
```

Add Safety Requirements to the Requirement Set

Add a top-level Safety requirement to the requirement set. Safety requirements are informational and do not contribute to the Implementation and Verification status summaries. In this example, you define a custom requirement type that extends the Informational requirement type by using the `sl_customization.m` file.

Refresh customizations to add the Safety requirement type to the list of requirement types.

```
slreq.refreshCustomizations;
```

Create the parent safety requirement.

```
myParentReq2 = add(myReqSet, 'Id', 'R2', ...
    'Summary', 'Safety Requirements', ...
    'Type', 'Safety');
```

Create child requirements for R2.

```
childReqR21 = add(myParentReq2, 'Id', 'R2.1');
childReqR22 = add(myParentReq2, 'Id', 'R2.2');
```

Create child requirements for R2.2.

```
childReqR221 = add(childReqR22, 'Id', 'R2.2.1');
childReqR222 = add(childReqR22, 'Id', 'R2.2.2');
childReqR223 = add(childReqR22, 'Id', 'R2.2.3');
```

Add Justifications to the Requirement Set

Create the parent justification.

```
myParentJustification = addJustification(myReqSet, 'Id', 'J', ...
    'Summary', 'Requirement Justifications');
```

Add child justifications to the parent justification J to justify requirements for Implementation.

```
childJust1 = add(myParentJustification, 'Id', 'J1', ...
    'Summary', 'Implementation Justifications');
```

Add child justifications to the parent justification J to justify requirements for Verification.

```
childJust2 = add(myParentJustification, 'Id', 'J2', ...
    'Summary', 'Verification Justifications');
```

Save the Requirement Set

```
save(myReqSet);
```

Cleanup

Close any open requirement sets.

```
slreq.clear;
```

Customize Requirements and Links by Using Stereotypes

You can use stereotypes to define custom requirement or link types with custom properties. You can use stereotypes to apply properties to a subset of requirements or links in a requirement set or link set. To define a stereotype, create a profile to contain the stereotype, assign the profile to a requirement set or link set, then use the stereotypes with the requirements or links. You can reuse stereotypes by assigning profiles to multiple requirement sets or link sets.


You can create custom requirement and link types and properties by using `sl_customization` files and custom attributes. For more information, see “Define Custom Requirement and Link Types and Properties” on page 1-78.

Create a Profile and Define Stereotypes

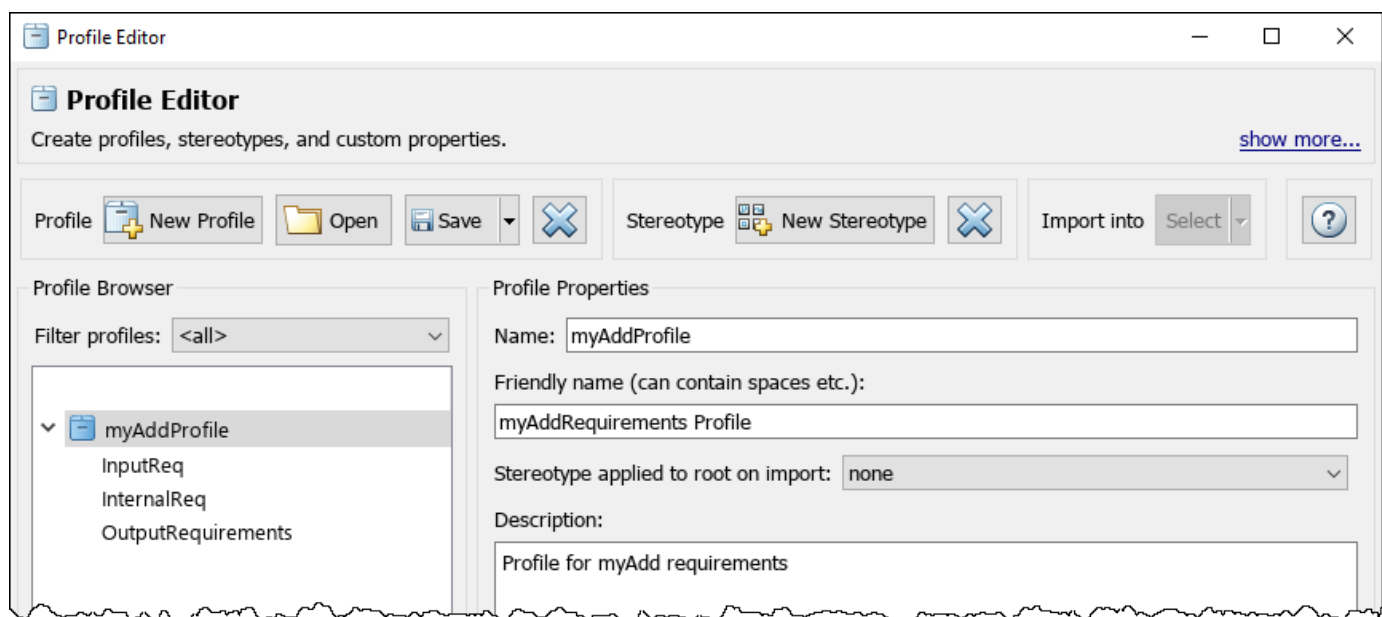
Stereotypes define domain-specific metadata. A profile is a package of stereotypes. In Requirements Toolbox, you can use stereotypes to define custom requirement or link types with custom properties by using the **Profile Editor**.

Create Profiles

Before you can define stereotypes, you must create a profile.

- 1 Open the **Requirements Editor** by entering this command at the MATLAB command line:
`slreq.editor`
- 2 In the **Profile** section, click **Profile Editor** .
- 3 In the **Profile Editor**, click **New Profile**. Select the new profile.
- 4 In the right pane, name the profile and add a description. Click **Save**.

The editor saves the profile as an XML file.

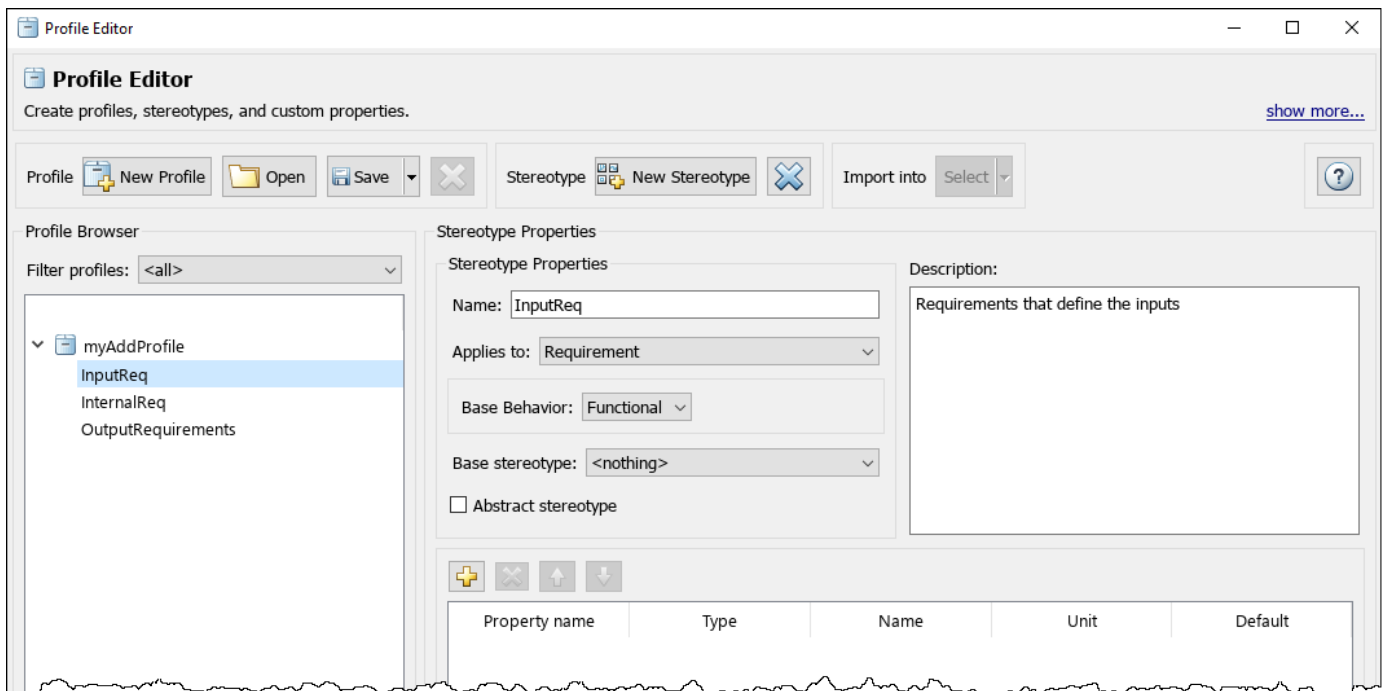


Define Stereotypes

To define a stereotype:

- 1 In the **Profile Editor**, select the profile to contain the stereotype and click **New Stereotype**. Select the new stereotype.
- 2 In the right pane, enter a name and description for the stereotype.
- 3 Set **Applies to** to Requirement or Link.
- 4 Set **Base Behavior** to one of the built-in requirement types or link types.

The base behavior affects how the requirement or link contributes to the implementation and verification status. Requirement and link stereotypes inherit functionality from the built-in requirement and link types the same way as custom requirement and link types. For more information, see “Choose a Built-in Type as a Base Behavior” on page 1-80.



- 5 If you are creating a stereotype for links, set the forward and backward name for the stereotype by using the **Forward Name** and **Backward Name** properties.

The **Requirements Editor** uses the forward name and backward name to describe the relationship between two linked items. For example, if you create a stereotype named **Satisfy**, you might choose the forward name **Satisfies** and the backward name **Satisfied by**. For more information, see the table in “Link Types” on page 3-34.

Tip To inherit properties from another stereotype, set **Base stereotype** to a stereotype from an open profile. The base stereotype and the child stereotype must both have **Applies to** set to the same value. However, the base and child stereotypes can have different **Base Behavior** values.

You can only use abstract stereotypes as base stereotypes and cannot apply them directly to requirements or links. To create an abstract stereotype, select **Abstract stereotype**.

Add Properties to Stereotypes

You can add properties to stereotypes to provide additional information about requirements and links. Stereotype properties only apply to the requirements or links that you apply the stereotype to. This behavior is different from custom attributes, which apply to all of the requirements or links in the requirement set or link set. For more information, see “Define Custom Requirement and Link Types and Properties” on page 1-78.

To add properties to a stereotype, in the **Profile Editor**, select a stereotype, then click the Add a new property button . Enter a name for the property and select a type. For numeric types, you can set the unit.

To use an enumeration as the type for a stereotype property, set **Type** to **Enumeration**. Enter the name of an existing enumeration class in the Enumeration Definition dialog, then click **OK**. If an enumeration class does not exist, you can create one by entering the name for a new enumeration class, then clicking **OK**. In the Enumeration class not found dialog, click **Create** to create an enumeration class file from a template. Edit the class file to contain your desired enumerations. For more information, see “Use Enumerated Data in Simulink Models” (Simulink).

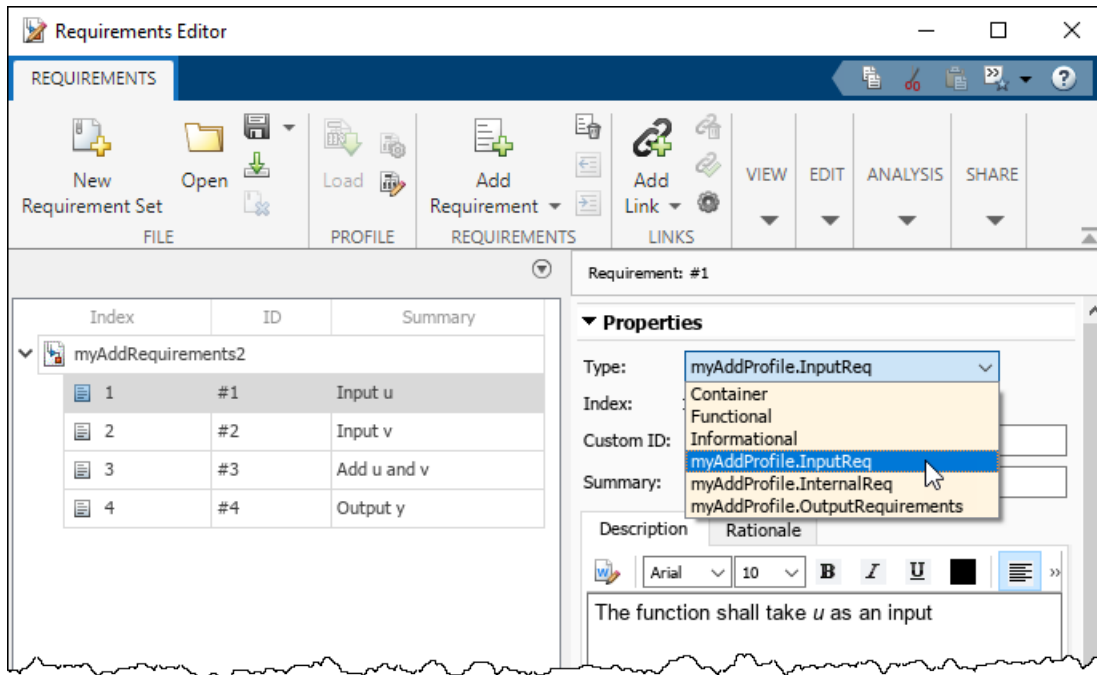
Assign Profiles and Use Stereotypes

To use stereotypes with requirements and links, assign the profile to the requirement set or link set:

- 1 Load a requirement set or link set or create a new one. For more information, see the “Examples” section of **Requirements Editor**.
- 2 In the **Requirements Editor**, click **Show Requirements** and select a requirement set, or click **Show Links** and select a link set.
- 3 In the **Profile** section, click **Load**.
- 4 Select the profile, then click **Open**.

Alternatively, you can assign profiles programmatically by using `slreq.ReqSet.importProfile` or `slreq.LinkSet.importProfile`.

To use a stereotype with a requirement or a link, in the **Requirements Editor**, select the requirement or link. In the right pane, under **Properties**, set the **Type** to the stereotype.

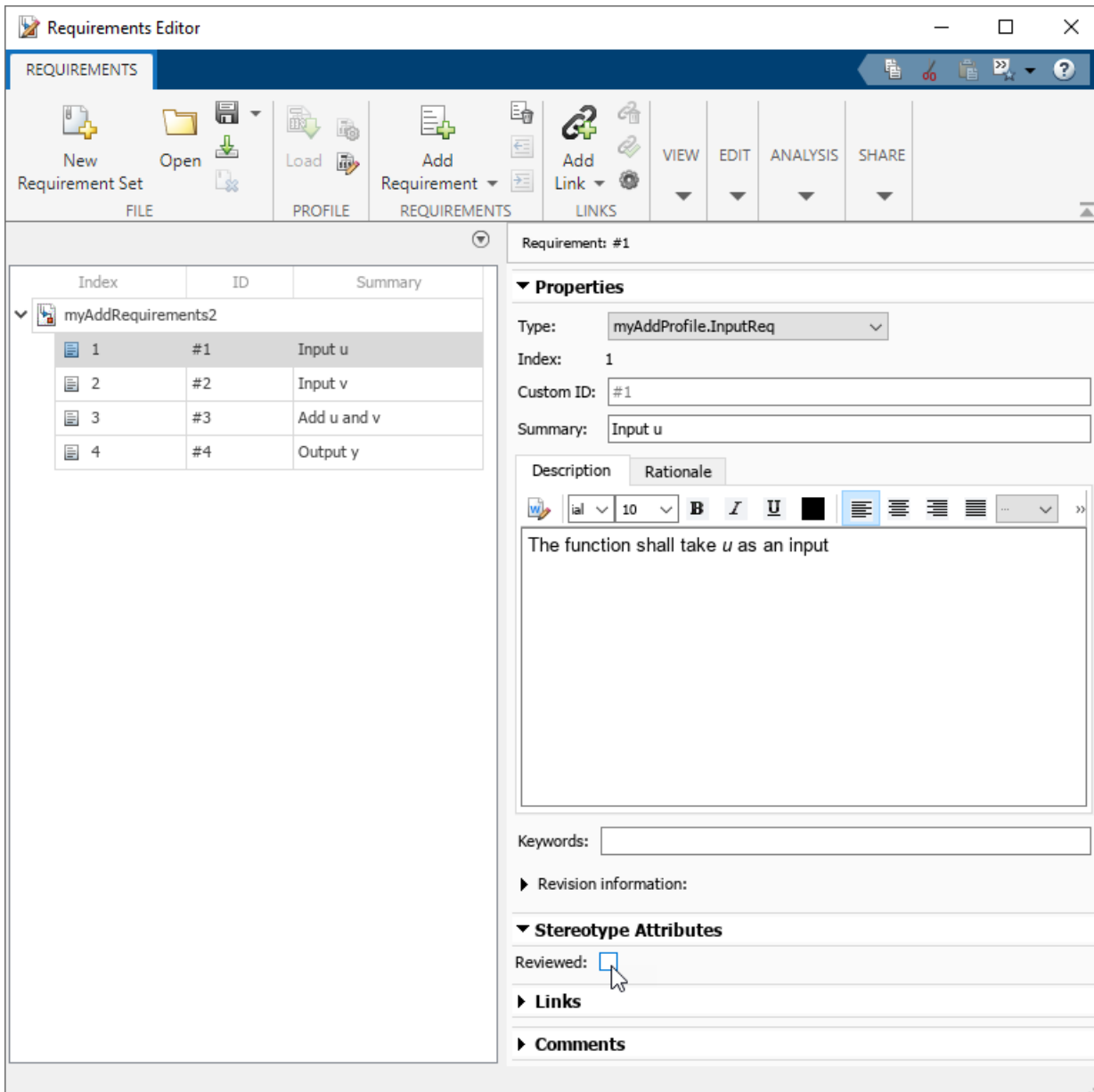


Alternatively, you can programmatically set the Type property of the requirement or link to the fully qualified name of the stereotype. For example:

```
myReq.Type = "myProfile.ReqStereotype";
```

Set Stereotype Property Values

To set stereotype property values, in the **Requirements Editor**, select the requirement or link. In the right pane, under **Stereotype Attributes**, set the value.



To set stereotype property values programmatically, use `slreq.Requirement.setAttribute` or `slreq.Link.setAttribute`. To view stereotype property values programmatically, use `slreq.Requirement.getAttribute` or `slreq.Link.getAttribute`.

Edit Profiles and Stereotypes

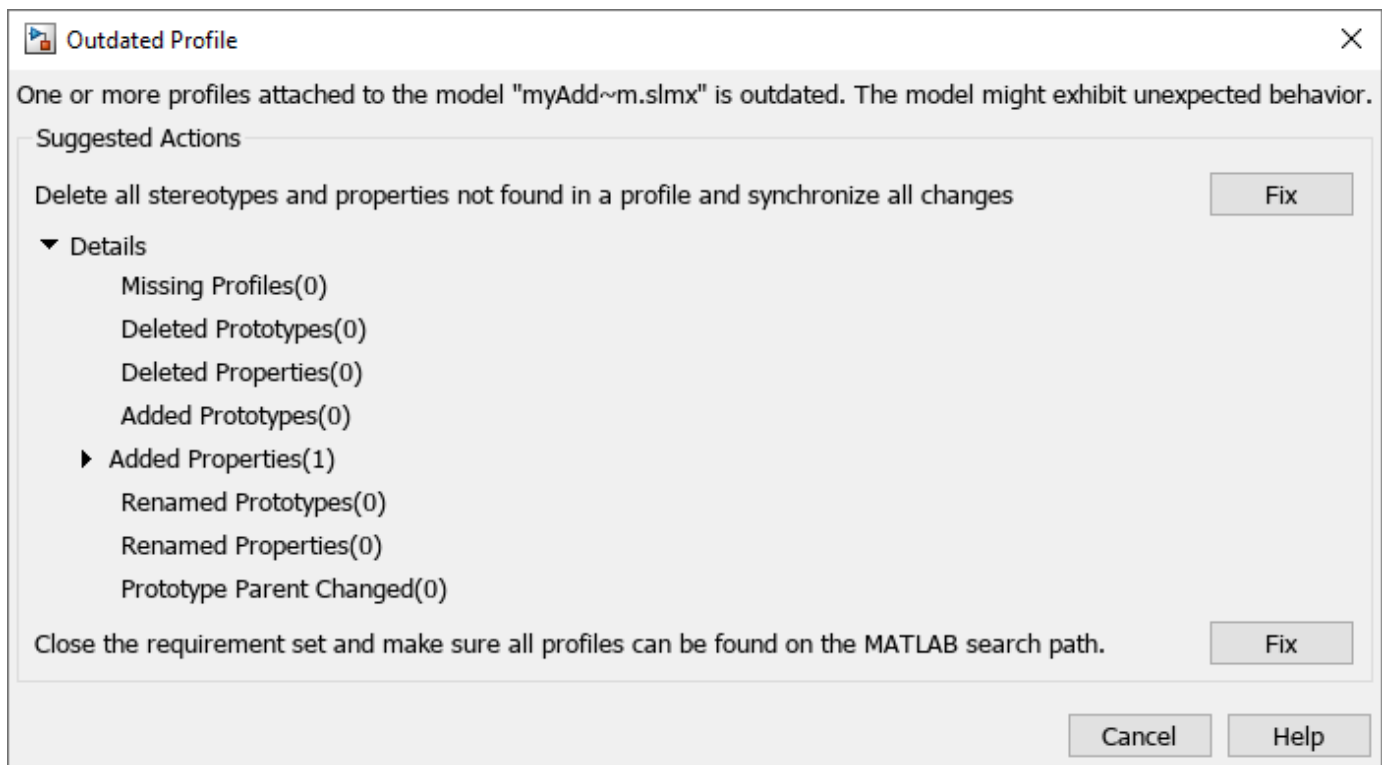
Before you can edit profiles or stereotypes, close the requirement sets and link sets that the profile is assigned to. In the **Requirements Editor**, select the requirement set or link set, then click **Close**. Alternatively, you can use `close` or `slreq.clear`.

To edit a profile or stereotype, open the **Profile Editor**. Click **Open** to open the profile. You can edit properties of the profile and its stereotypes, including the name, description, base behavior, base stereotype, and stereotype properties.

Fix Outdated Profiles

If you make changes to a profile and then load a requirement set or link set that uses that profile, Requirements Toolbox indicates that the profile assigned to the requirement set or link set is outdated.

If you load a requirement set with an outdated profile, the Outdated Profile dialog appears. You can view the changes made to the profile since the requirement set was last loaded by expanding the **Details** section. To apply the changes to the requirement set, next to **Delete all stereotypes and properties not found in a profile and synchronize all changes**, click **Fix**. If you do not want to apply the changes to your requirement set, you can close the requirement set by, next to **Close the requirement set and make sure all profiles can be found on the MATLAB search path**, clicking **Fix**.



Alternatively, you can load a requirement set and fix the outdated profile by using `slreq.load` with the `forceResolve` argument set to `true`.

If you load a link set with an outdated profile, the MATLAB command window displays a warning that a profile assigned to the link set is outdated. To open the Outdated Profile dialog, in the **Requirements Editor**, click **Show Links**. Select the link set and, in the right pane, under **Properties**, click **Details**.

Note You cannot load the link set and fix the outdated profile by using the API.

Remove Stereotypes or Properties

If you remove a stereotype from a profile and apply the profile changes to a requirement set or link set, Requirements Toolbox reverts the requirement type to `Functional` and the link type to `Relate` for requirements or links that used that stereotype.

If you remove a stereotype property from a stereotype and apply the profile changes to a requirement set or link set, Requirements Toolbox removes that property and the property values from the requirements or links that used that stereotype.

Manage and Remove Profiles

To view the profiles applied to a requirement set or link set, in the **Requirements Editor**, select the requirement set or link set and click **Manage**. The Linked profiles dialog shows the profiles assigned to the selected requirement set or link set.

To remove a profile from a requirement set or link set, select the profile in the Linked profiles dialog and click **Remove**. If you remove a profile, Requirements Toolbox applies these changes to requirements and links that used the profile:

- Sets the requirement type to `Functional`
- Sets the link type to `Relate`
- Removes the stereotype properties and deletes the stereotype property values

Note Requirements Toolbox also applies these changes if you delete or rename the profile XML file.

See Also

More About

- “Define Profiles and Stereotypes” (System Composer)
- “Define Custom Requirement and Link Types and Properties” on page 1-78
- “Use Stereotypes when Importing from ReqIF Files” on page 1-28

Define Custom Requirement and Link Types and Properties

Requirements Toolbox includes built-in requirement and link types that you can use to indicate the role of your requirements and how linked items relate to each other. You can define your own custom requirement or link types that extend the built-in types by using stereotypes or an `sl_customization` file.

Additionally, Requirements Toolbox includes properties that you can use to provide information about your requirements and links. You can define your own custom requirement or link properties that extend the built-in properties by using stereotype properties or custom attributes.

Define Custom Requirement and Link Types

To define custom requirements or link types, you can define a stereotype or create an `sl_customization` file.

This table summarizes the differences between the two approaches:

Stereotypes	<code>sl_customization</code> Files
You can use stereotypes only with the requirement set or link set that you assigned the profile to.	You can use the custom types with all open requirements or links. However, you must execute <code>slreq.refreshCustomizations</code> during each MATLAB session.
You can reuse stereotypes across projects.	You can reuse <code>sl_customization</code> files across projects.
You can create multiple profiles in a single project and assign multiple profiles to a single requirement set or link set.	You can have only one <code>sl_customization</code> file on the MATLAB path.
To share requirement or link sets with other users, you must share the requirement or link set and the profile XML files	To share requirement or link sets with other users: <ol style="list-style-type: none"> 1 You must share the requirement or link set and the <code>sl_customization</code> file. 2 They must execute <code>slreq.refreshCustomizations</code> for Requirements Toolbox to recognize the custom types.
When you make changes to profile or stereotype and then load the requirement or link set, you must fix the outdated profile.	When you make changes to the <code>sl_customization</code> file, you must execute <code>slreq.refreshCustomizations</code> to update the custom types.

Define Custom Types by Using Stereotypes

To define custom types by using stereotypes, you must create a profile to contain the stereotype, then define the stereotype. To use the stereotype, assign the profile to the requirement set or link set. Set the requirement or link type to the stereotype in the **Requirements Editor** or at the MATLAB command line. For more information, see “Customize Requirements and Links by Using Stereotypes” on page 1-71.

Define Custom Types by Using `sl_customization` Files

To define custom types by using `sl_customization` files, you must create a MATLAB code file that defines a function called `sl_customization`, then add definitions for the custom types to the file. Register the custom types by using `slreq.refreshCustomizations`. For more information, see “Define Custom Requirement and Link Types by Using `sl_customization` Files” on page 3-42.

Set the requirement or link type to the custom type in the **Requirements Editor** or at the MATLAB command line.

Define Custom Properties for Requirements and Links

To define custom properties for requirements or links, you can define a stereotype that has stereotype properties or define custom attributes for a requirement set or link set.

This table summarizes the differences between the two approaches:

Custom Properties Defined by Stereotype Properties	Custom Properties Defined by Custom Attributes
You must apply the profile to a requirement set or link set before you can use the stereotype properties.	You must define custom attributes for each requirement set or link set.
You can use stereotype properties only with requirements or links that use that stereotype.	You can use custom attributes only with requirements or links in the requirement set or link set that you defined custom attributes for.
You can reuse stereotype properties across projects by reusing profiles.	You cannot reuse custom attributes across projects unless you reuse the entire requirement set.
When you share requirement sets or link sets and profile XML files with other users, Requirements Toolbox recognizes stereotype properties.	When you share requirement sets or link sets with other users, Requirements Toolbox recognizes custom attributes. You do not need to share additional files.
If you make changes to the profile or stereotype and then load the requirement set or link set, you must fix outdated profile.	If you make changes to the custom attributes, Requirements Toolbox automatically applies the changes, but might lose some data. For more information, see the section Edit Custom Attributes in “Add Custom Attributes to Requirements” on page 1-81 and “Add Custom Attributes to Links” on page 3-44.

Define Custom Properties by Using Stereotypes

To define custom properties by using stereotypes, you must create a profile, define a stereotype, then add properties to the stereotype. After you set the requirement or link type to the stereotype, you can set the values of your custom property in the **Requirements Editor** or at the MATLAB command line. For more information, see “Customize Requirements and Links by Using Stereotypes” on page 1-71.

Define Custom Properties by Using Custom Attributes

To define custom properties by using custom attributes, you must define each property as a custom attribute for each requirement set or link set. You can define the custom attributes in the

Requirements Editor or at the MATLAB command line. For more information, see “Add Custom Attributes to Requirements” on page 1-81 and “Add Custom Attributes to Links” on page 3-44.

Choose a Built-in Type as a Base Behavior

Because all custom types must be based on one of the built-in types, custom types contribute to the implementation and verification status in the same way as the built-in types. Additionally, custom link types inherit their impact direction from the built-in link types.

- To create a custom requirement type that you can implement and verify by linking to design elements and tests, use the **Functional** built-in type as the base behavior. **Functional** requirements contribute to the implementation and verification status. For more information, see:
 - “Requirement Types” on page 1-6
 - “Review Requirements Implementation Status” on page 4-2
 - “Review Requirements Verification Status” on page 4-6
- To create a custom link type that contributes to the implementation status, use the **Implement** link type as the base behavior.
- To create a custom link type that contributes to the verification status, use the **Verify** link type as the base behavior.
- To create a custom link type that verifies requirements by linking to external test results, use the **Confirm** link type as the base behavior. For more information, see the “Link Types” on page 3-34 section of “Create and Store Links” on page 3-31.
- To create a link type with forward and backward names that align with the impact direction, use the same link direction as the built-in types. For more information, see the table under “Link Types” on page 3-34.

Note Impact direction describes how changes propagate between nodes in a traceability diagram. For more information, see the “Impact Direction” on page 3-20 section of “Visualize Links with Traceability Diagrams” on page 3-17.

See Also

More About

- “Customize Requirements and Links by Using Stereotypes” on page 1-71
- “Define Custom Requirement and Link Types by Using `sl_customization` Files” on page 3-42
- “Add Custom Attributes to Requirements” on page 1-81
- “Add Custom Attributes to Links” on page 3-44

Add Custom Attributes to Requirements

In this section...

“Define Custom Attributes for Requirement Sets” on page 1-81

“Set Custom Attribute Values for Requirements” on page 1-82

“Edit Custom Attributes” on page 1-83

“Custom Attributes for Referenced Requirements” on page 1-83

“Import Custom Attributes” on page 1-84

When you create a requirement set using Requirements Toolbox, you can create custom attributes that apply to the requirements contained in the requirement set. Custom attributes extend the set of properties associated with your requirements.

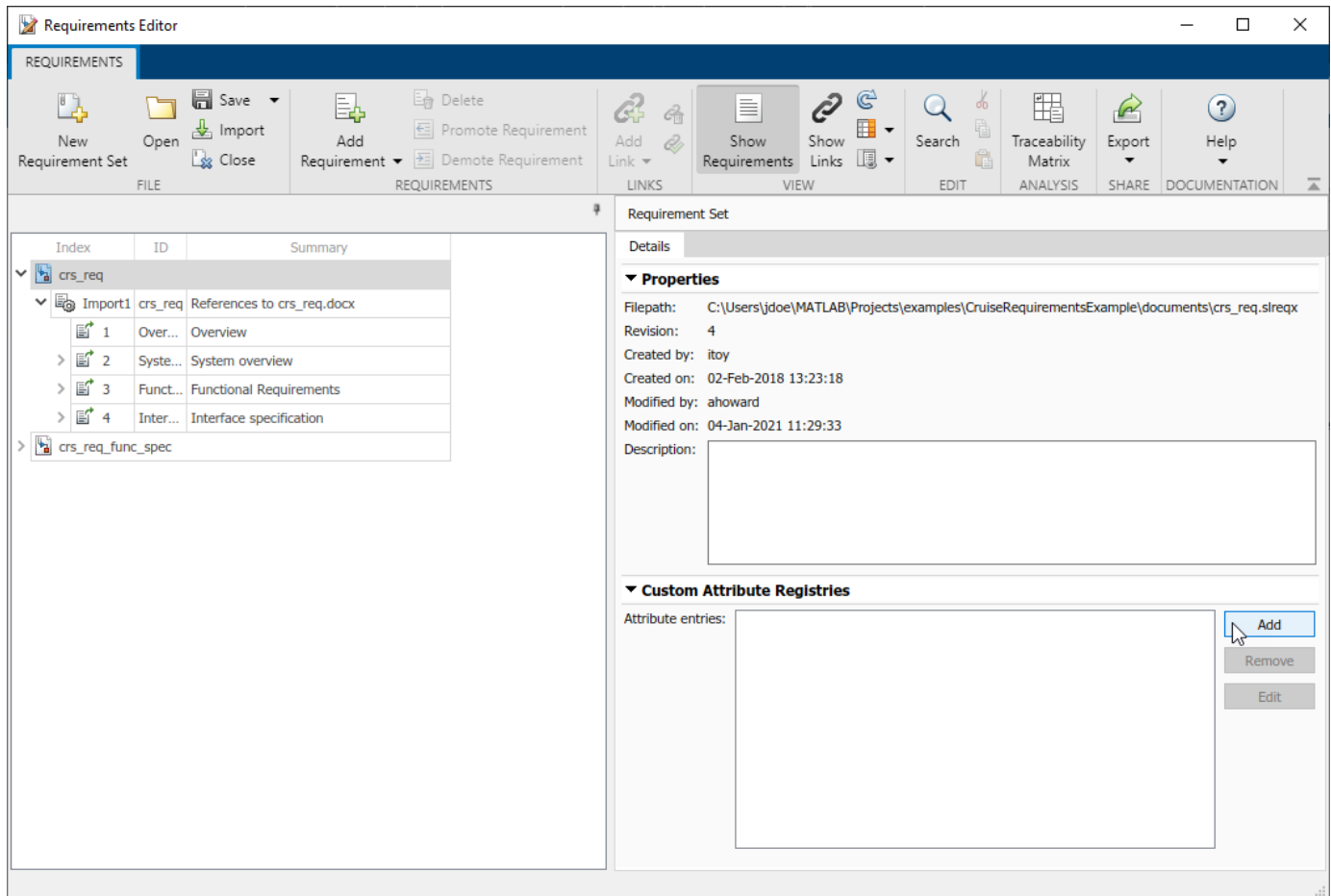
Alternatively, you can customize your requirements by creating stereotypes that define custom requirement types and properties. For more information, see “Define Custom Requirement and Link Types and Properties” on page 1-78.

Define Custom Attributes for Requirement Sets

To define a custom attribute for a requirement set:

- 1 Open the **Requirements Editor**. At the MATLAB command prompt, enter:

```
slreq.editor
```
- 2 Click **Show Requirements**.
- 3 Open an existing requirement set, or create a new one.
- 4 Select the requirement set.
- 5 In the right pane, under **Custom Attribute Registries**, click **Add** to add a custom attribute to the requirement set.
- 6 The **Custom Attribute Registration** dialog box appears. Enter the name of your custom attribute in the **Name** field. Select the type from the **Type** drop-down menu. Enter a description of the custom attribute in the **Description** field.



Alternatively, you can programmatically add a custom attribute to a requirement set by using `addAttribute`.

Custom Attribute Types

There are four custom attribute types:

- **Edit**: Text box that accepts a character array. There is no default value.
- **Checkbox**: Single check box that can be either checked or unchecked. The default value is unchecked.
- **Combobox**: Drop-down menu with user-defined options. Unset is always the first option in the drop-down menu and the default attribute value.
- **DateTime**: Text box that only accepts a `datetime` array. There is no default value. See `datetime` for more information on `datetime` arrays.

Set Custom Attribute Values for Requirements

After you define custom attributes for a requirement set, you can set the custom attribute value for each requirement. Select the requirement in the **Requirements Editor**. In the right pane, under **Custom Attributes**, enter the desired value in the field.

You can also set the custom attribute value for a requirement programmatically by using `setAttribute`.

If you do not define a value for **Checkbox** or **Combobox** type custom attributes for a requirement, the value will be set to the default. For **Checkbox** custom attributes, the default value is defined in the **Custom Attribute Registries** pane for the requirement set. For **Combobox** custom attributes, the default value is **Unset**.

Note If you copy a requirement and paste it within the same requirement set, the copied requirement retains the same custom attribute values as the original. If the requirement is pasted into a different requirement set, the copied requirement does not retain the custom attribute values.

Edit Custom Attributes

After you define a custom attribute for a requirement set, you can make limited changes to the custom attribute. To make changes, select the requirement set in the **Requirements Editor**. In the right pane, under **Custom Attribute Registries**, select the custom attribute you want to edit and click **Edit**.

Note You can only set the custom attribute value for one requirement at a time.

Alternatively, you can edit custom attributes programmatically by using `updateAttribute`.

For custom attributes of any type, you can edit the name and description. For **Combobox** custom attributes, you can also edit the value of each option in the drop-down menu, or add and remove options. If you remove an option, the custom attribute value for requirements that previously had that value reset to the default value, **Unset**. Editing the value of multiple options at once resets the custom attribute values of that attribute for all requirements in the requirement set. However, you can edit the value of a single option without resetting the custom attribute values. Additionally, if you reorder the existing options without changing the value of any of the options, the requirements' custom attribute values do not reset.

After you set the custom attribute value for a requirement, you can change the value by selecting the requirement in the **Requirements Editor** and setting the updated value in the **Custom Attributes** pane.

Custom Attributes for Referenced Requirements

When importing requirements from an external file into Requirements Toolbox, if you select **Allow updates from external source**, the requirements are imported as referenced requirements (s1req.Reference objects). For more information, see "Select an Import Mode" on page 1-8.

Referenced requirements are read-only by default. Although you can add custom attributes to a requirement set that includes referenced requirements, you must unlock the requirement to add a custom attribute value. Select the referenced requirement and, in the right pane, under **Properties**, click **Unlock**. Alternatively, you can unlock the referenced requirements by selecting the top import node and, in the right pane, under **Requirement Interchange**, clicking **Unlock all**.

If you click **Update** in the **Requirement Interchange** pane, changes to your requirement set such as new custom attributes or new custom attribute values will be lost. Save or export your

requirement set files before using **Update**. You can use **Export** in the **Requirement Interchange** pane to export a ReqIF file with new custom attributes.

Import Custom Attributes

When importing requirements from an external source, you can also import custom attributes that exist in the external source.

Import Custom Attributes from ReqIF

When importing requirements from a ReqIF file, you can map information to built-in properties and custom attributes. For more information, see “Mapping ReqIF Attributes in Requirements Toolbox” on page 1-24.

Import Custom Attributes with Direct Import from IBM DOORS Next

When importing requirements using direct import from IBM DOORS Next®, custom attributes that are defined in DOORS Next are automatically imported to Requirements Toolbox. For information about importing requirements from IBM DOORS Next using direct import, see “Link and Trace Requirements with IBM DOORS Next” on page 8-4.

Import Custom Attributes from Microsoft Excel

When importing requirements from a Microsoft Excel file, you can map predefined headers or a row of cells to built-in properties and custom attributes. See “Import Options for Microsoft Excel Spreadsheets” on page 1-14.

See Also

Apps

Requirements Editor

Classes

slreq.ReqSet | slreq.Requirement

Related Examples

- “Manage Custom Attributes for Requirements by Using the Requirements Toolbox API” on page 1-105

More About

- “Add Custom Attributes to Links” on page 3-44
- “Define Custom Requirement and Link Types and Properties” on page 1-78
- “Customize Requirements and Links by Using Stereotypes” on page 1-71

Customize Requirement Index Numbering

You can disable the index for an individual requirement, referenced requirement, or justification or set the index to a specified value. To customize the index numbering, use the **Requirements Editor** or set the object properties at the MATLAB command line.

Disable Index Numbering

Requirement sets in Requirements Toolbox contain a hierarchy of requirements. Requirements Toolbox assigns each requirement an index number that identifies what level of the hierarchy the requirement is in and where it is within the level.

Index	ID	Summary
crs_req_func_spec		
1	#1	Driver Switch Request Handling
1.1	#2	Switch precedence
1.2	#3	Avoid repeating commands
1.3	#4	Long Switch recognition
1.3.1	#5	Waiting state for Long Increment swi...
1.3.2	#6	Waiting state for Long Decrement sw...
1.4	#7	Cancel Switch Detection

You can disable index numbering for an individual requirement, referenced requirement, or justification in the hierarchy. Disabling numbering also disables index numbering for all descendant requirements. However, the requirements, referenced requirements, and justifications remain in the same place in the hierarchy.

To disable index numbering, in the **Requirements Editor**, right-click the requirement and select **Disable index numbering**.

Index	ID	Summary
crs_req_func_spec		
1	#1	Driver Switch Request Handling
1.1	#2	Switch precedence
1.2	#3	Avoid repeating commands
	#4	Long Switch recognition
	#5	Waiting state for Long Increment swi...
	#6	Waiting state for Long Decrement sw...
1.3	#7	Cancel Switch Detection
1.4	#8	Set Switch Detection

To re-enable index numbering, right-click the requirement and select **Enable index numbering**. However, index numbering for descendant requirements remains disabled. You must individually re-enable index numbering for each descendant requirement.

Tip You can also re-enable requirement index numbering at the MATLAB command line. For more information, see “Programmatically Customize Index Numbering” on page 1-86.

Index	ID	Summary
crs_req_func_spec		
1	#1	Driver Switch Request Handling
1.1	#2	Switch precedence
1.2	#3	Avoid repeating commands
1.3	#4	Long Switch recognition
	#5	Waiting state for Long Increment swi...
	#6	Waiting state for Long Decrement sw...
1.4	#7	Cancel Switch Detection

Set the Index to a Specified Value

To set the index of a requirement, referenced requirement, or justification to a specified value:

- 1 In the **Requirements Editor**, right-click the requirement and select **Set index numbering**.
- 2 In the Index number dialog box, clear **Use default auto numbering**. In the **Restart numbering from** field, enter the desired requirement index value. You can only enter an integer value.
- 3 Click **OK**.

Setting the requirement index to a specific value changes the index numbering for the requirements that come after the requirement and any descendant requirements.

Index	ID	Summary
crs_req_func_spec		
1	#1	Driver Switch Request Handling
1.1	#2	Switch precedence
1.2	#3	Avoid repeating commands
1.4	#4	Long Switch recognition
1.4.1	#5	Waiting state for Long Increment swi...
1.4.2	#6	Waiting state for Long Decrement sw...
1.5	#7	Cancel Switch Detection

To set the index number back to the default value, right-click the requirement and select **Set index numbering**. In the Index number dialog box, select **Use default auto numbering**.

Note You cannot move a requirement to another level in the hierarchy by specifying the index value. Instead, use **Promote Requirement** or **Demote Requirement** in the **Requirements Editor** toolstrip. You cannot promote or demote referenced requirements.

Programmatically Customize Index Numbering

You can customize the index numbering for `s1req.Requirement`, `s1req.Reference`, and `s1req.Justification` objects at the MATLAB command line. To get a handle to an object, use:

- `s1req.find` to search all loaded Requirements Toolbox objects
- `s1req.ReqSet.find` to search within a requirement set

- `slreq.getCurrentObject` to get a handle to the selected object in the **Requirements Editor**

Disable Index Numbering Programmatically

To disable index numbering programmatically, set the `IndexEnabled` property of the requirement object to `false`.

```
myReq = slreq.find(Type="Requirement", Summary="My Requirement 1");
myReq.IndexEnabled = false;
```

Disabling index numbering also disables index requirement numbering for all descendant requirements.

To re-enable index numbering programmatically, set the `IndexEnable` property of the requirement object to `true`.

```
myReq.IndexEnabled = true;
```

Index numbering for descendant requirements remains disabled. You must individually re-enable index numbering for each descendant requirement.

Specify the Index Value Programmatically

To specify the requirement index value programmatically, set the `IndexNumber` property of the requirement object to an integer number.

```
myReq = slreq.find(Type="Requirement", Summary="My Requirement 1");
myReq.IndexNumber = 101;
```

Note You cannot move the requirement to another level in the hierarchy by specifying the index value. Instead, use:

- `slreq.Requirement.move` for requirements
- `slreq.Justification.move` for justifications

You cannot move referenced requirements.

Setting the requirement index to a specific value changes the index numbering for the requirements that come after the requirement and any descendant requirements.

To reset the requirement index to the default value, set the `IndexNumber` to an empty array.

```
myReq.IndexNumber = [];
```

Reset Index Numbering for Multiple Requirements Programmatically

You can re-enable index numbering and reset the requirement index to the default value for multiple requirements programmatically.

For example, this script:

- Finds a loaded requirement set called `myReqSet`
- Finds the `slreq.Reference` objects in the requirement set
- Enables index numbering for all referenced requirements in the requirement set

- Resets the index value to the default value for all referenced requirements in the requirement set

```
rs = slreq.find(Type="ReqSet",Name="myReqSet");
refs = find(rs,Type="Reference");
for i = 1:numel(refs)
    refs(i).IndexEnabled = true;
    refs(i).IndexNumber = [];
end
```

Alternatively, use this function to reset index numbering for all descendant requirements of a given requirement. The function enables index numbering and resets the index to the default numbering. It accepts `slreq.Requirement`, `slreq.Reference`, and `slreq.Justification` objects as inputs.

```
function resetDescendantIndex(ref)
    childRefs = children(ref);
    for i = 1:numel(childRefs)
        childRefs(i).IndexEnabled = true;
        childRefs(i).IndexNumber = [];
        resetDescendantIndex(childRefs(i));
    end
end
```


See Also

Requirements Editor | `slreq.Requirement` | `slreq.Reference` | `slreq.Justification`

More About

- “Define Requirements Hierarchy” on page 1-66

Update Imported Requirements

You can import referenced requirements from external requirements source documents, then update them when changes are made to the source document. To import referenced requirements, open the **Requirements Editor** click **Import**, choose the source document and check the option to **Allow updates from external source**. When you import requirements as referenced requirements from external requirement documents, they retain a reference to the source document. Check if you have an updated version of the source document by refreshing an import node. The top import node icon changes to  when an updated source document is available, indicating that the timestamp of the source document is more recent than the last imported or updated timestamp.

Select the updated version of the source document during the Update operation. Alternatively, you can update the file name and location of the source requirements document by right-clicking the top node of the requirement set and selecting **Update source document name or location**.

Update a Requirement Set

To update your requirements in the requirement set, select the Import node  in the **Requirements Editor**. In the right pane, under **Requirements Interchange** click **Update**.

Updating requirements:

- Matches the previously imported requirements to the updated source requirements and updates the requirements in the new version of the document. This includes overwriting any local changes you made to unlocked requirements.
- Generates comments about the differences between the document versions. You can view comments when you select the top Import node in the requirement set in the right pane under **Comments**.
- Updates the **modifiedOn** value for the updated requirements and the **updatedOn** value for the top Import node of the requirement set.
- Marks the requirement set as dirty, even if the requirements data did not change because its **updatedOn** value changed.
- Preserves links to updated requirements.
- Preserves requirement SIDs.
- Preserves comments on requirements.
- Preserves local custom attributes you create within Requirements Toolbox. See “Add Custom Attributes to Requirements” on page 1-81 for more information about creating custom attributes for requirements.

Updating requirements does not change the links to updated requirements, the requirement SIDs, the comments on requirements, or local custom attributes you create. If attributes in the requirement set and the external source document use the same name, the updated requirements use the attribute values defined in the external source document.

Update Requirements with Change Tracking Enabled

If you have change tracking enabled, and there are changes to a requirement with links, updating requirements might trigger change issues that you might have to resolve:

- **Match:** No changes were detected between document versions. When you import different versions of the same document, the Update operation might detect only whitespace differences, such as carriage returns, linefeeds, and nonbreaking spaces. In this scenario, the Update operation does not update the rich text fields such as the **Description** and the **Rationale**.
- **Insertion:** A new requirement was inserted in the requirement set.
- **Deletion:** A previously imported requirement was deleted from the requirement set.
- **Update:** The built-in or custom attribute values of a previously updated requirement were changed.
- **Move:** A requirement was moved in the requirement hierarchy.
- **Reorder:** A requirement was reordered with respect to its sibling requirements.

Before importing requirements into Requirements Toolbox, make sure that your requirements in the requirements document have persistent and unique custom IDs that do not change across document versions. The Update operation otherwise matches unrelated requirements and displays more differences between document versions than actually exist.

Considerations for Microsoft Word Documents

Follow these guidelines when importing requirements from Microsoft Word documents:

- Use bookmarks for requirement custom IDs. You can then add content to the document while maintaining requirement references. If you use section headings as requirement custom IDs, changing the document can result in unresolved links when updating requirements.
- If you import requirements into a requirement set on one computer and update your requirements on a different computer with a different set of fonts or styles installed, additional changes to the requirement descriptions may be tracked. These changes occur because the font or style is embedded in the HTML descriptions of the requirements.
- Before you execute update requirements, convert documents that you created in an older version of Microsoft Word to the current version. This conversion prevents Microsoft Word from inserting spurious whitespaces in your requirements document.
- In Microsoft Word, resolve issues related to the Trust Center or pending updates if you encounter any errors during the Import or Update operations. These issues might cause Microsoft Word to block incoming connections from MATLAB .

See Also

Requirements Editor

More About


- “Import Requirements from Third-Party Applications” on page 1-8
- “Import Requirements from Microsoft Office Documents” on page 1-12
- “Import Requirements from ReqIF Files” on page 1-17

Filter Requirements and Links in the Requirements Editor

You can filter requirements and links in the **Requirements Editor** and Requirements Perspective by creating a view that filters based on requirement and link metadata. You can share views with other users by sharing a MAT-file or by a requirement set that has views stored internally.

Create Views

To create a view:

- 1 Open the Filtered View Editor:
 - In the **Requirements Editor**, select **Filter View**  > **Manage Views**.
 - In the Requirements Perspective, click the **Select View** menu and select **Manage Views**.
- 2 In the Filtered View Editor, click **New**. Enter a name in the **View name** field.
- 3 To store the view in the preferences folder, set **Storage type** to **User**. For more information, see “Where MATLAB Stores Preferences”. To store the view in a requirement set, select the requirement set from the list next to **Storage type**. You cannot store a view in a link set.

Note When you store a view in a the requirement set, changes to the view do not cause the requirement set to have unsaved changes.

- 4 In the **Requirements Filter** and **Links Filter** tabs, define your filter criteria, then click **OK**. If you stored a view in the requirement set, save the requirement set.

Alternatively, you can save the current column layout and applied filters as a new view:

- In the **Requirements Editor**, select **Filter View**  > **Save View**.
- In the Requirements Perspective, click the **Select View** menu and select **Save View**.

In the Save View dialog, enter a name for the view and click **OK**. You can also overwrite an existing view by selecting it from the list.

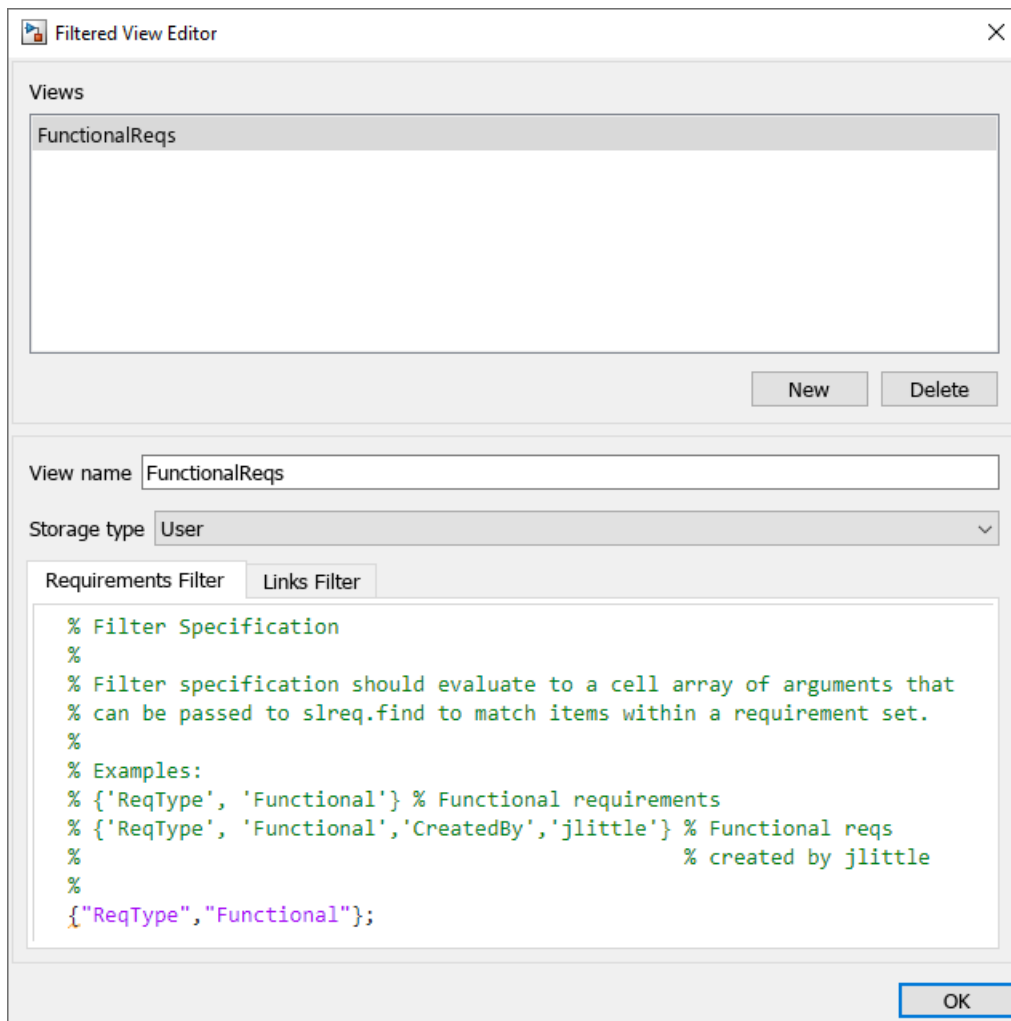
Create Views Programmatically

To create a view programmatically, use `create`.

Define Requirement and Link Filters

To define filter criteria in the Filtered View Editor, enter a cell array of comma-separated name-value arguments with the same input syntax as `slreq.find`. For example, to display only the requirements with `Type` set to `Functional`, use this filter:

```
{"ReqType", "Functional"};
```



Requirement filters apply to all loaded requirements, referenced requirements, and justifications. Link filters apply to all loaded links.

Define Filters Programmatically

To define filters programmatically, set the “ReqFilter” and “LinkFilter” properties of the `slreq.View` object. Enter the filter as a string scalar or character array that contains a cell array of comma-separated name-value arguments with the same input syntax as `slreq.find`.

For example, consider an `slreq.View` object that is assigned to a variable called `myView`. To display only the requirements with `Type` set to `Functional`, enter:

```
myView.ReqFilter = {'ReqType','Functional'};
```

Apply Views

To apply a view in the **Requirements Editor**, click **Filter View** , then select a view from the list. To reset the view to the default view, select **Default view**.

To apply a view in the Requirements Perspective, click the **Select View** menu and select a view from the list.

Apply Views Programmatically

To apply a view programmatically, use `activate`. To reset the view to the default view, use `activateDefaultView`.

Manage Views

To edit or delete a view, open the Filtered View Editor:

- In the **Requirements Editor**, select **Filter View**  > **Manage Views**.
- In the Requirements Perspective, click the **Select View** menu and select **Manage Views**.

To edit a view, select the view and edit the name or filter criteria. You cannot edit the storage location.

To delete a view, select the view and click **Delete**.

Manage Views Programmatically

To edit a view programmatically, edit the “Name”, “ReqFilter”, or “LinkFilter” properties of the `slreq.View` object.

To delete a view programmatically, use `delete`.

Share Views

To share views with other users, you can share a:

- Requirement set that has views stored in the set
- User preferences MAT-file
- Requirements Toolbox view settings MAT-file

To share a view stored in a requirement set, share the SLREQX file.

To share your views stored in your preferences folder, identify the location of your preferences folder by using `prefdir`. For more information, see “Where MATLAB Stores Preferences”. Share the MAT-file called `slreqViewSettings_v2.mat`. Other users can place this file in their preferences folder to use the views.

To share a Requirements Toolbox view settings as a MAT-file, use `slreq.exportViewSettings` to export the currently loaded views. To import the MAT-file, use `slreq.importViewSettings`. The views are saved where they were originally stored.

See Also

Apps
Requirements Editor

Objects

slreq.View

Functions

create | slreq.exportViewSettings | slreq.importViewSettings

More About

- “Access Frequently Used Features and Commands from the Requirements Editor”
- “Where MATLAB Stores Preferences”

Import and Edit Requirements from a Microsoft Word Document

This example shows you how to import and update requirements from a Microsoft® Word requirements document. After you import the requirements, you can modify the requirements in either the **Requirements Editor** or the Word document. This functionality is available only on Microsoft Windows® platforms.

Open the Project and Import the Requirements

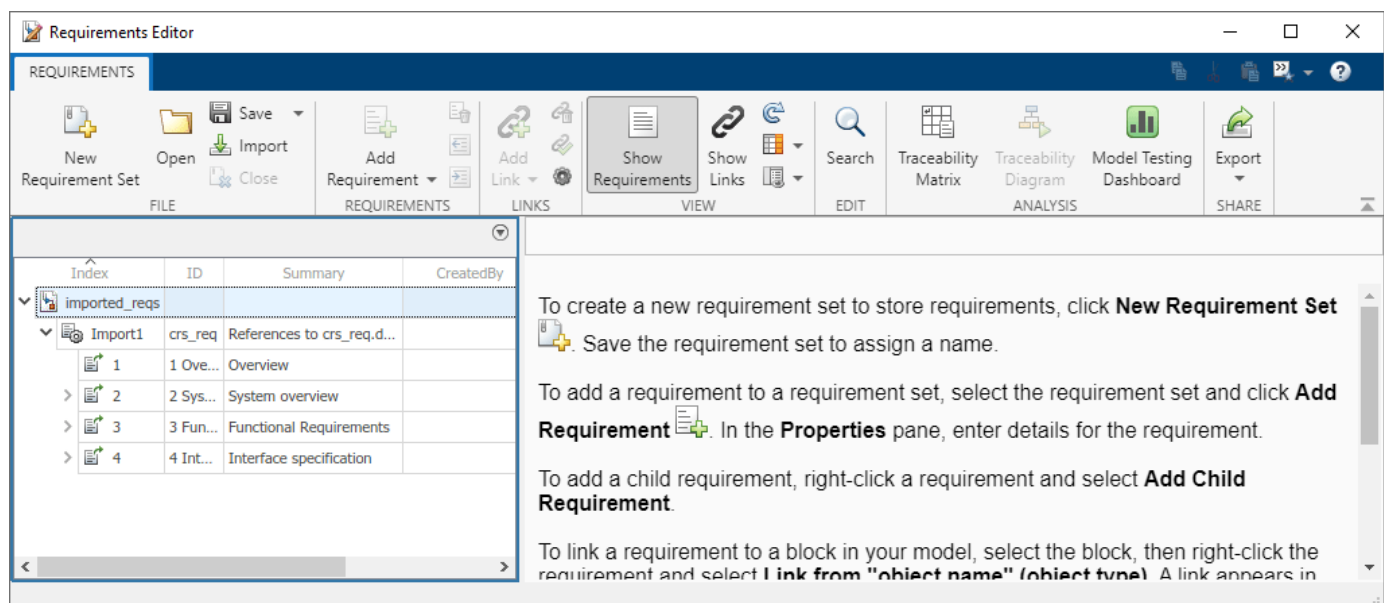
Open the project that includes the model and supporting files. At the MATLAB® command prompt, enter:

```
slreqCCProjectStart
```

Next, import the requirements into the **Requirements Editor**.

- 1 Open the **Requirements Editor**. In the **Apps** tab, click **Requirements Editor**.
- 2 Import the requirement set. Click **Import**.
- 3 In the Importing Requirements window, set **Document type** to Microsoft Word Document.
- 4 Click **Browse**, open the documents folder, and select `crs_req.docx`.
- 5 In the **Content** section, select **Plain text**. Under **Requirement Identification**, select **Use bookmarks to identify items and serve as custom IDs** and **Allow updates from external source**. For more information on import options, see “Import Options for Microsoft Word Documents” on page 1-12.
- 6 In the **Destination(s)** section, set **Requirement Set** to a folder on the path and change the file name to `imported_reqs.slreqx`.

The **Requirements Editor** opens the imported requirement set and displays the requirements under the top-level node, `Import1`.



The requirements import based on their assigned heading numbers in the Word document.

Edit the Requirements

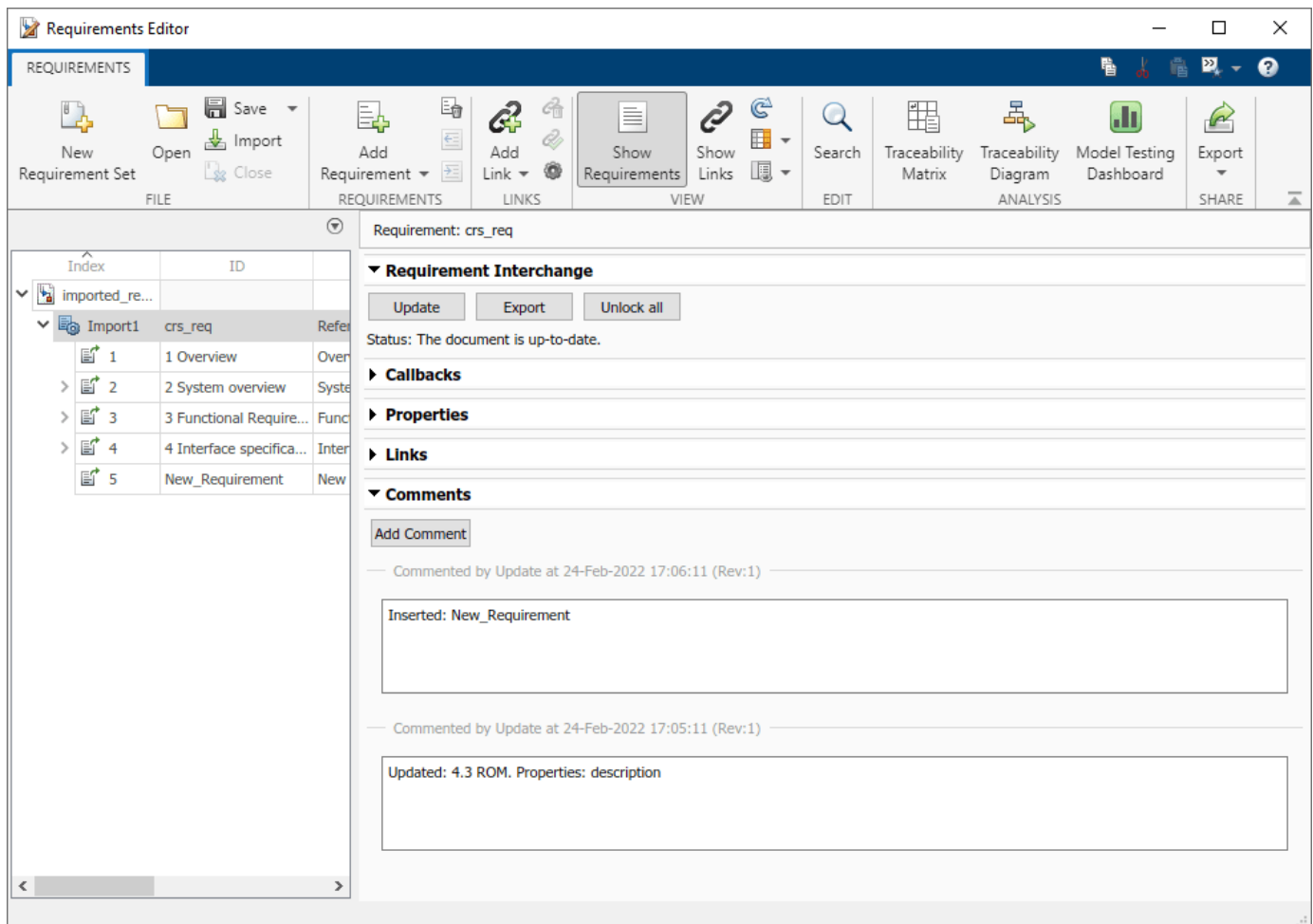
After you import the requirements, you can edit the requirements in the Word document and then update the imported requirements, or edit the imported requirements directly in the **Requirements Editor**.

Change the Requirements in the Word Document

To add a requirement to the Word document:

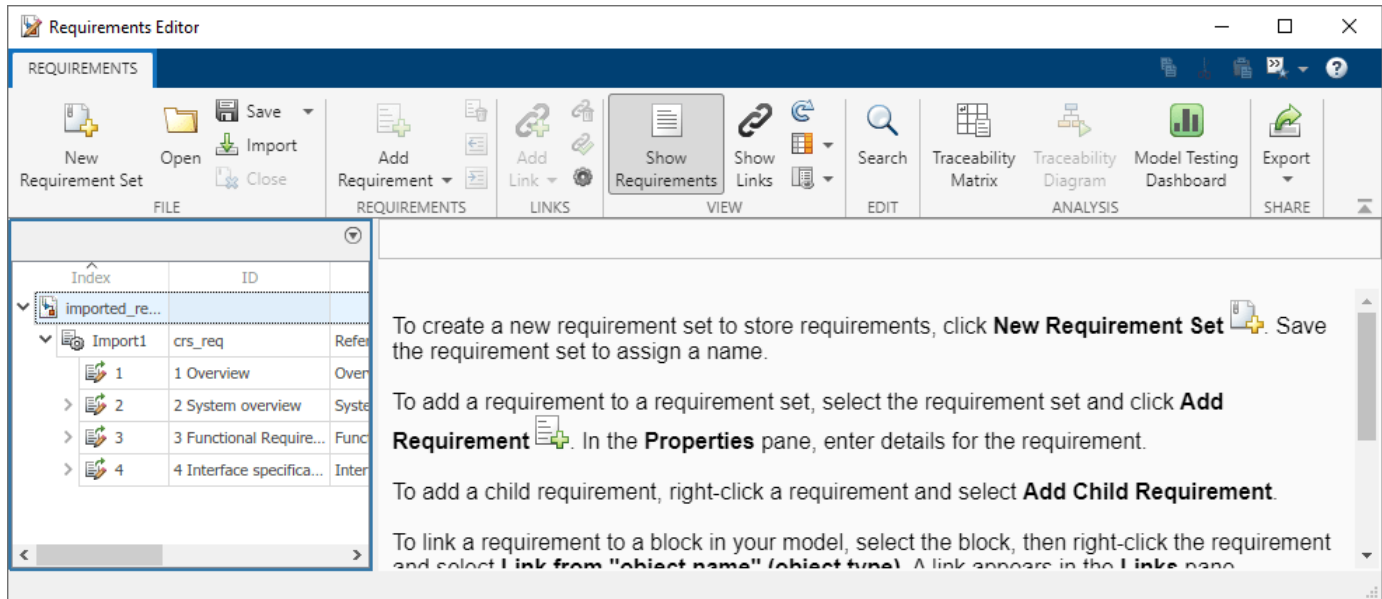
- 1 Open the Word document and add a new requirement. Maintain the heading hierarchy used throughout the document.
- 2 Add a bookmark to the requirement. To add a bookmark to your Microsoft Word document, see [Add or delete bookmarks in a Word document or Outlook message on the Microsoft website](#).
- 3 Save the Word document.
- 4 In the **Requirements Editor**, select the top-level node of the requirement set. In the right pane under **Requirement Interchange**, click **Update** to update the referenced requirements.

To view the changes, select **Import1**. In the right pane, click **Comments** to see the changes to the file.



Change the Referenced Requirements in the Requirements Editor

You can also edit a referenced requirement in the **Requirements Editor**. In the **Requirements Editor**, select a referenced requirement. In the right pane, under **Properties**, click **Unlock**. You can unlock all referenced requirements in the requirement set by selecting the Import node and, in the right pane, under **Requirement Interchange**, clicking **Unlock All**.



You can revert the requirement set to the requirements in the Word file by updating the top-level Import node. Click **Import1** and, in the right pane, under **Requirement Interchange**, click **Update**.

See Also

slreq.ReqSet | **Requirements Editor**

More About

- “Import Requirements from Microsoft Office Documents” on page 1-12
- “Link to Requirements in Microsoft Word Documents” on page 7-2
- “Import Requirements from Third-Party Applications” on page 1-8
- “Define Requirements Hierarchy” on page 1-66

Export Requirement Sets and Link Sets to Previous Versions of Requirements Toolbox

You can export requirement sets and link sets to files that are compatible with previous versions of Requirements Toolbox and MATLAB. You can export requirement sets and link sets to R2017b and later.

Export Requirement Sets

You can export requirement sets from the **Requirements Editor**. Before you export a requirement set, ensure that it is not open in the Requirements Perspective in a Simulink model.

Open the **Requirements Editor** using one of these approaches:

- At the MATLAB command line, enter:

```
slreq.editor
```
- In the MATLAB **Apps** tab, under **Verification, Validation, and Test**, click the **Requirements Editor** app.
- In the Simulink **Apps** tab, under **Model Verification, Validation, and Test**, click the **Requirements Editor** app.

In the **Requirements Editor**, select **Open** to open a requirement set. Select **Show Requirements** to view the requirement set. Click the requirement set, then select **Save > Export to Previous**. In the dialog, enter your desired name for the requirement set. Select the MATLAB version that you want to export to from the **Save as type** list.

If you export a requirement set that has outgoing links to a previous version, Requirements Toolbox also exports a link set file that is compatible with the selected version.

You can also use `slreq.ReqSet.exportToVersion` to export a requirement set and associated link sets to a previous version.

Export Link Sets

You can export link sets to previous versions. The method that you use depends on the associated artifact. If a link set is associated with a requirement set, exporting the requirement set also exports the link set. For more information, see “Export Requirement Sets” on page 1-98.

If a link set is associated with a Simulink model, exporting the model also exports the link set. For more information, see “Export Model to Previous Version of Simulink” (Simulink).

You can also directly export the link set to a previous version by using `slreq.LinkSet.exportToVersion`.

See Also

Requirements Editor | `slreq.ReqSet.exportToVersion` | `slreq.LinkSet.exportToVersion`

Round-Trip Importing and Exporting for ReqIF Files

Many third-party requirements management applications can export and import requirements using the ReqIF format. If you manage your requirements in a third-party tool, you can import the requirements to Requirements Toolbox, edit the requirements, and export the requirements back to your third-party tool with ReqIF files. This procedure is referred to as a ReqIF round trip.

ReqIF represents requirements as `SpecObject` objects and links as `SpecRelation` objects between `SpecObject` objects. Each `SpecObjectType` object specifies the associated `SpecObject` object and the `SpecRelationType` objects classify each `SpecRelation` object. The `SpecObjectType` and `SpecRelationType` objects define attributes to store requirements and link information. The `SpecObject` and `SpecRelation` object contain values for these attributes.

For more information about ReqIF data organization, see *Exchange Document Content in Requirements Interchange Format (ReqIF) Version 1.2*.

Considerations for Importing Requirements

You can import requirements from ReqIF files in the **Requirements Editor**. For more information, see “Import Requirements from ReqIF Files” on page 1-17.

Import Mapping Considerations

When you import requirements from ReqIF files, you can choose which import mapping to use. For more information, see “Choosing an Import Mapping” on page 1-17.

If you use a generic mapping during import, you must use a generic mapping during export. The export mapping affects the content exported to the ReqIF file. For more information, see “Considerations for Exporting Requirements” on page 1-101.

Alternatively, you can map ReqIF data to stereotypes in Requirements Toolbox. For more information, see “Use Stereotypes when Importing from ReqIF Files” on page 1-28.

Considerations for ReqIF Files with Multiple Specifications

When you import requirements from ReqIF files with multiple specifications, you can:

- Select a single ReqIF source specification to import into a requirement set
- Combine ReqIF source specifications into one requirement set
- Import each ReqIF source specification into a separate requirement set

For more information, see “Importing Requirements from a ReqIF File with Multiple Specifications” on page 1-21.

When you export requirements to a ReqIF file, you can only export one requirement set at a time. Consequently, if you plan to perform a ReqIF round trip with a ReqIF file with multiple source specifications, consider which of the three import methods in “Importing Requirements from a ReqIF File with Multiple Specifications” on page 1-21 allows you to export your requirements with your preferred number of ReqIF files.

Edit Imported Content

Edit imported requirements content by using the **Requirements Editor**. Depending on the import mode that you use, the requirements import as referenced requirements or requirements, which are

slreq.Reference or slreq.Requirement objects, respectively. For more information, see “Select an Import Mode” on page 1-8.

Edit the Attribute Mapping

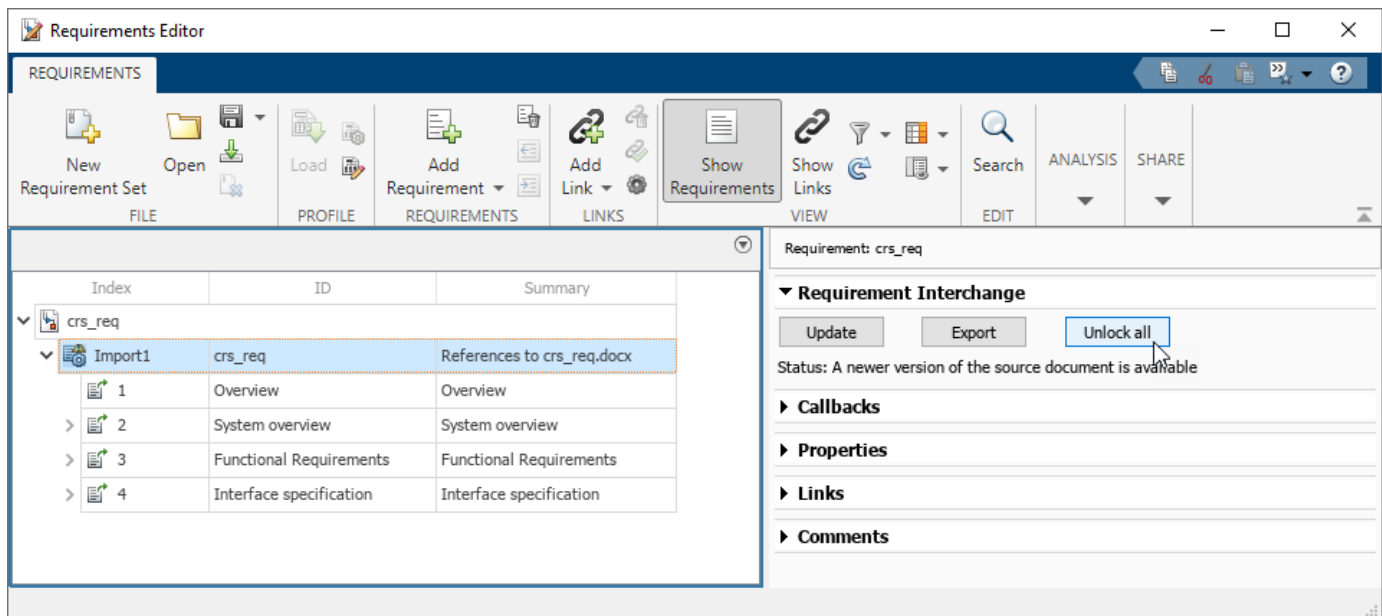
When you import requirements from a ReqIF file, the import process maps SpecObjectType object attributes to built-in requirement properties or requirement custom attributes. For more information about SpecObjectType object attributes, see “Choosing an Import Mapping” on page 1-17.

After you import the requirements, you can map the SpecObjectType objects to requirement types. You can also edit the SpecObjectType object attribute mappings to requirement properties. See “Mapping ReqIF Attributes in Requirements Toolbox” on page 1-24.

Edit Imported Requirements

You can edit a requirement or referenced requirement and change the requirement properties such as the **Summary** or **Description**. You can also define custom attributes for the requirement set and set values for those custom attributes. For more information, see “Add Custom Attributes to Requirements” on page 1-81.

Before you edit an imported referenced requirement, you must unlock it. To unlock all requirements in the requirement set, select the top-level Import node of the requirement set and, in the right pane, under **Requirement Interchange**, click **Unlock all**.



To unlock individual requirements, navigate to the requirement and, in the right pane, under **Properties**, click **Unlock**.

To add, remove, and edit custom attributes associated with the requirement set, select the requirement set and use the interface in the right pane under **Custom Attribute Registries**. For more information about managing custom attributes for requirements, see “Add Custom Attributes to Requirements” on page 1-81. Select an individual referenced requirement and unlock it to set custom attribute values.

Update Imported Requirements Content

If you select **Allow updates from external source** during the import operation, you can update your imported requirement sets with data from the updated ReqIF file. Select the Import node of the requirement set and, in the right pane, under **Requirement Interchange**, click **Update**. The update operation overwrites all local modifications, such as edits to unlocked referenced requirements. The update operation preserves comments and local attributes. For more information, see “Manage Imported Requirements with External Applications” on page 1-10.

Link Requirements to Items in MATLAB and Simulink

When you link a requirement to an item in MATLAB or Simulink and then export the requirements to a ReqIF file, Requirements Toolbox creates a proxy object for that object in the exported file. If the linked item is one of the supported types, the proxy object has a type value that describes the linked object type. For more information, see “Exporting Links” on page 1-64.

When you re-import the ReqIF file generated by the Requirements Toolbox, the software reconstructs the links between requirements and the items represented by the proxy objects of the supported types. For more information, see “Importing Links from a ReqIF File Generated by Requirements Toolbox” on page 1-24.

Considerations for Exporting Requirements

You can export a requirement set, an Import node, or a parent requirement and its children to a ReqIF file. When you export requirements, you can also export links associated with the requirements. For more information, see “Export Requirements to ReqIF Files” on page 1-61.

When you export requirements and links, you can choose which export mapping to use. You can either reuse the same mapping that you used during import, or use a generic mapping. For more information, see “Choosing an Export Mapping” on page 1-61.

The export mapping that you use affects the content that is exported to the ReqIF file:

- SpecObjectType object values
- SpecObject object attributes
- SpecRelationType object values

You can export the requirement type when you define and use custom requirement types and export using the generic mapping. For more information, see “Using the Generic Mapping During Export” on page 1-62.

For more information about how the export mapping affects the exported content, see “Choosing an Export Mapping” on page 1-61.

See Also

Requirements Editor | `slreq.import`

More About

- “Import Requirements from ReqIF Files” on page 1-17
- “Export Requirements to ReqIF Files” on page 1-61

- “Create and Edit Attribute Mappings” on page 1-110
- “Import Requirements from Third-Party Applications” on page 1-8
- “Update Imported Requirements” on page 1-89
- “Best Practices and Guidelines for ReqIF Round-Trip Workflows” on page 1-103

Best Practices and Guidelines for ReqIF Round-Trip Workflows

Managing Requirement Custom IDs

- When you import requirements as referenced requirements, Requirements Toolbox attempts to map a requirement identifier generated by the third-party requirements management application to the Custom ID attribute of the requirement. Verify that the intended attribute mapping between the Custom ID and the requirement identifier is selected.
- Do not modify the requirement custom ID attribute to maintain traceability.

Guidelines for Updating Referenced Requirements Content

- The Update operation overwrites local modifications such as edits to unlocked referenced requirements with values from the ReqIF source file. Save, check-in, or export your requirement set files before attempting the Update operation.
- The Update operation preserves comments and attributes. Do not delete imported custom attributes as they will be restored when you update the requirement set. For a complete ReqIF round trip workflow, include all previously imported attributes.

Guidelines for Editing Referenced Requirements Content

- Rich text attributes like the **Description** and **Rationale** may lose some formatting, particularly tables, during the round trip workflow. To preserve formatting, edit these attributes in the same application. Plain text attributes can be edited in multiple applications.
- Rename imported attributes through the Attribute Mapping pane of the **Requirements Editor** to maintain the connection to the corresponding attribute in the external requirements document during the Export operation.

Guidelines for Adding Details to Imported Requirements

You can add additional details to imported requirements by:

- Adding additional attributes
- Authoring new requirements and linking imported requirements to them

You cannot insert locally authored requirements as children of imported requirements. To associate newly authored requirements with imported requirements, add them to a separate requirement set and link related requirements.

Guidelines for Exporting Requirements to ReqIF Files

- Do not import requirements from multiple ReqIF files into the same requirement set. Each ReqIF file can contain multiple specifications which get imported under separate top Import nodes in the requirement set. Every Import node has a Custom ID that matches the name of the specification.
- Do not import referenced requirements into a requirement set that contains locally authored requirements. For round trip workflows, reuse the previous import settings to requirements that were previously imported.
- You cannot update requirements you author within Requirements Toolbox if you export them to ReqIF. Import the exported file as referenced requirements into a new requirement set that you

can update in the future. Links created to authored requirements will not be preserved when you re-import them. Export and re-import the locally authored requirements before you create links.

See Also

More About

- “Import Requirements from Third-Party Applications” on page 1-8
- “Round-Trip Importing and Exporting for ReqIF Files” on page 1-99
- “Update Imported Requirements” on page 1-89

Manage Custom Attributes for Requirements by Using the Requirements Toolbox API

This example shows how to use the Requirements Toolbox™ API to create custom attributes for requirement sets and set custom attribute values for requirements.

Establish Requirement Set

Load the requirement file `crs_req_func_spec`, which describes a cruise control system, and assign it to a variable.

```
rs = slreq.load('crs_req_func_spec');
```

Add a Custom Attribute of Each Type

Add a custom attribute of each type to the requirement set. Create an `Edit` custom attribute with a description.

```
addAttribute(rs, 'MyEditAttribute', 'Edit', 'Description', ...
    'You can enter text as the custom attribute value.')
```

Create a `Checkbox` type attribute and set its `DefaultValue` property to `true`.

```
addAttribute(rs, 'MyCheckboxAttribute', 'Checkbox', 'DefaultValue', true)
```

Create a `Combobox` custom attribute. Because the first option must be `'Unset'`, add the options `'Unset'`, `'A'`, `'B'`, and `'C'`.

```
addAttribute(rs, 'MyComboboxAttribute', 'Combobox', 'List', {'Unset', 'A', 'B', 'C'})
```

Create a `DateTime` custom attribute.

```
addAttribute(rs, 'MyDateTimeAttribute', 'DateTime')
```

Check the defined custom attributes for the requirement set. Get information about `MyComboboxAttribute` to see the options you added.

```
rs.CustomAttributeName
```

```
ans = 1x4 cell
    {'MyCheckboxAttribute'}    {'MyComboboxAttribute'}    {'MyDateTimeAttribute'}    {'MyEditAtt
```

```
atrb = inspectAttribute(rs, 'MyComboboxAttribute')
```

```
atrb = struct with fields:
    name: 'MyComboboxAttribute'
    type: Combobox
    description: ''
    list: {'Unset' 'A' 'B' 'C'}
```

Set a Custom Attribute Value for a Requirement

Find a requirement in the requirement set, and set the custom attribute value for all four custom attributes that you created.

```
req = find(rs, 'Type', 'Requirement', 'SID', 3);
setAttribute(req, 'MyEditAttribute', 'Value for edit attribute. ');
setAttribute(req, 'MyCheckboxAttribute', false);
setAttribute(req, 'MyComboboxAttribute', 'B');
```

Set `MyDateTimeAttribute` with the desired locale to ensure that the date and time is set in the correct format on systems in other locales. See “Locale” for more information.

```
localDateTimeStr = datestr(datetime('15-Jul-2018 11:00:00', 'Locale', 'en_US'), 'Local');
setAttribute(req, 'MyDateTimeAttribute', localDateTimeStr);
```

View the attribute values.

```
getAttribute(req, 'MyEditAttribute')
```

```
ans =
'Value for edit attribute.'
```

```
getAttribute(req, 'MyCheckboxAttribute')
```

```
ans = logical
      0
```

```
getAttribute(req, 'MyComboboxAttribute')
```

```
ans =
'B'
```

```
getAttribute(req, 'MyDateTimeAttribute')
```

```
ans = datetime
      15-Jul-2018 11:00:00
```

Edit Custom Attributes

After you define a custom attribute for a link set, you can make limited changes to the custom attribute.

Add a description to `MyCheckboxAttribute` and `MyComboboxAttribute`, and change the list of options for `MyComboboxAttribute`. Because you cannot update the default value of `Checkbox` attributes, you can only update the description of `MyCheckboxAttribute`. View the changes.

```
updateAttribute(rs, 'MyCheckboxAttribute', 'Description', ...
    'The checkbox value can be true or false. ');
updateAttribute(rs, 'MyComboboxAttribute', 'Description', ...
    'Choose an option from the list.', 'List', {'Unset', '1', '2', '3'});
atrb2 = inspectAttribute(rs, 'MyCheckboxAttribute')
```

```
atrb2 = struct with fields:
    name: 'MyCheckboxAttribute'
    type: Checkbox
    description: 'The checkbox value can be true or false.'
    default: 1
```

```
atrb3 = inspectAttribute(rs, 'MyComboboxAttribute')
```

```
atrb3 = struct with fields:
    name: 'MyComboboxAttribute'
```

```

    type: Combobox
    description: 'Choose an option from the list.'
    list: {'Unset' '1' '2' '3'}

```

Find Requirements That Match Custom Attribute Value

Search the requirement set for all requirements where 'MyEditAttribute' is set to 'Value for edit attribute.'

```
req2 = find(rs, 'Type', 'Requirement', 'MyEditAttribute', 'Value for edit attribute.')
```

```
req2 =
```

```
Requirement with properties:
```

```

    Type: 'Functional'
    Id: '#3'
    Summary: 'Avoid repeating commands'
    Description: '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-ht
    Keywords: {}
    Rationale: ''
    CreatedOn: 27-Feb-2017 10:15:38
    CreatedBy: 'itoy'
    ModifiedBy: 'batserve'
    IndexEnabled: 1
    IndexNumber: []
    SID: 3
    FileRevision: 46
    ModifiedOn: 04-Mar-2023 01:16:00
    Dirty: 1
    Comments: [0x0 struct]
    Index: '1.2'

```

Search the requirement set for all requirements where 'MyCheckboxAttribute' is set to true.

```
reqsArray = find(rs, 'Type', 'Requirement', 'MyCheckboxAttribute', true)
```

```
reqsArray=1x69 object
```

```
1x69 Requirement array with properties:
```

```

Type
Id
Summary
Description
Keywords
Rationale
CreatedOn
CreatedBy
ModifiedBy
IndexEnabled
IndexNumber
SID
FileRevision
ModifiedOn
Dirty
Comments
Index

```

Search the requirement set for all requirements where 'MyComboboxAttribute' is set to 'Unset'.

```
reqsArray2 = find(rs, 'Type', 'Requirement', 'MyComboboxAttribute', 'Unset')
```

```
reqsArray2=1x70 object
```

```
1x70 Requirement array with properties:
```

```
Type  
Id  
Summary  
Description  
Keywords  
Rationale  
CreatedOn  
CreatedBy  
ModifiedBy  
IndexEnabled  
IndexNumber  
SID  
FileRevision  
ModifiedOn  
Dirty  
Comments  
Index
```

Delete Custom Attributes

You can use `deleteAttribute` to delete attributes. However, because the custom attributes created in this example are assigned to requirements, you must set 'Force' to `true` to delete the attributes. Delete 'MyEditAttribute' and confirm the change.

```
deleteAttribute(rs, 'MyEditAttribute', 'Force', true);  
rs.CustomAttributeNames
```

```
ans = 1x3 cell  
    {'MyCheckboxAttribute'}    {'MyComboboxAttribute'}    {'MyDateTimeAttribute'}
```

Add a new custom attribute, but don't set any requirement custom attribute values for requirements.

```
addAttribute(rs, 'NewEditAttribute', 'Edit');  
rs.CustomAttributeNames
```

```
ans = 1x4 cell  
    {'MyCheckboxAttribute'}    {'MyComboboxAttribute'}    {'MyDateTimeAttribute'}    {'NewEditAttribute'}
```

Because 'NewEditAttribute' is not used by any requirements, you can delete it with `deleteAttribute` by setting 'Force' to `false`. Confirm the change.

```
deleteAttribute(rs, 'NewEditAttribute', 'Force', false);  
rs.CustomAttributeNames
```

```
ans = 1x3 cell  
    {'MyCheckboxAttribute'}    {'MyComboboxAttribute'}    {'MyDateTimeAttribute'}
```


Cleanup

Clear the open requirement sets without saving changes and close the open models without saving changes.

```
slreq.clear;  
bdclose all;
```

See Also

slreq.ReqSet | addAttribute | deleteAttribute | updateAttribute | inspectAttribute | getAttribute | setAttribute

Related Examples

- “Manage Custom Attributes for Links by Using the Requirements Toolbox API” on page 3-73

More About

- “Add Custom Attributes to Requirements” on page 1-81

Create and Edit Attribute Mappings

The ReqIF format represents a requirement as a `SpecObject`. The `SpecObject` has a `SpecObjectType`, which defines attributes to store requirements information. The `SpecObjects` contains values for these attributes.

After you import requirements from a ReqIF file, you can customize how attributes from ReqIF requirements map to Requirements Toolbox requirement properties and custom attributes. For more information, see “Add Custom Attributes to Requirements” on page 1-81. You can also save this mapping for reuse.

Edit the Attribute Mapping for Imported Requirements

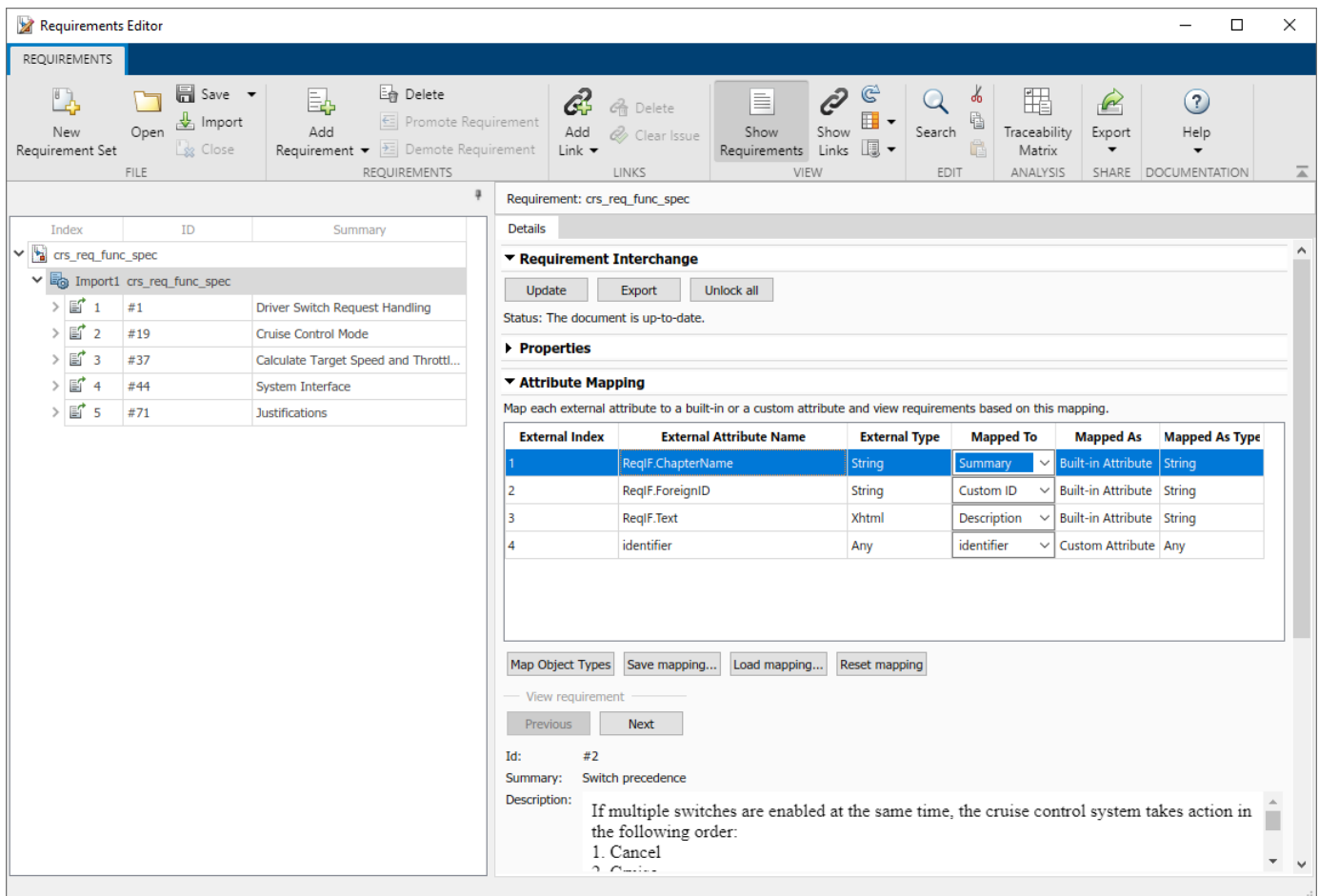
You can import requirements from ReqIF files in the **Requirements Editor**. For more information, see “Import Requirements from ReqIF Files” on page 1-17.

When you import requirements from ReqIF files, you must choose an import mapping to use. The imported requirement type, properties, and imported link type depend on the import mapping that you choose. For more information, see “Choosing an Import Mapping” on page 1-17.

After you import requirements from a ReqIF file, you can edit the attribute mapping for the imported requirements:

- 1** Open the **Requirements Editor** and import the ReqIF file. For more information, see “Import Requirements from ReqIF Files” on page 1-17.
- 2** Select the Import node or the top-level requirement, depending on if you imported referenced requirements or requirements. For more information, see “Select an Import Mode” on page 1-8.

You can see the attribute mappings in the right pane, under **Attribute Mapping**.



- 3 Edit the mapping by selecting a property or attribute from the drop-down in the **Mapped To** column.

Note When editing the attribute mapping, you can only map an attribute to a built-in requirement type. You cannot select a custom attribute from the drop-down in the **Mapped To** column.

You can save the current attribute mapping by clicking **Save mapping**. The mapping saves as an XML file. You can load a saved mapping by clicking **Load mapping**.

To change the name or description of the attribute mapping, open the XML file that you created in a text editor and modify the values of the <name> and <description> tags.

To have Requirements Toolbox select the import attribute mapping based on the tool that originally created the ReqIF file:

- 1 In a text editor, open the attribute mapping and the ReqIF file.
- 2 Find the value of the <REQ-IF-TOOL-ID> tag in the ReqIF file.
- 3 Change the value of the <name> tag in the attribute mapping file to match the value of the <REQ-IF-TOOL-ID> tag.

Specify Default ReqIF Requirement Type

Some external requirements management tools, such as Polarion, support multiple types of requirements. In this case, modify the attribute mapping file to specify the default ReqIF requirement type to use when exporting to ReqIF. For example:

```
<thisType>SpecObject</thisType>  
<thisSubType>System Requirement</thisSubType>
```

The value of the `<thisSubType>` tag indicates that each exported `SpecObject` will have the `SpecObject` type as `System Requirement`.

Specify ReqIF Template

Some external requirements management tools, such as Polarion and IBM Rational DOORS, require a specific set of ReqIF data type, attribute, and `SpecObject` type definitions. They may also require that the ReqIF specification be of a certain type. You can supply these definitions by specifying the name of a template ReqIF file in the mapping file produced by the external requirements management tool. During ReqIF export, Requirements Toolbox imports the template file and uses it to generate and export a ReqIF file with a format that is compatible with the external tool.

Save the template files in the same folder as the attribute mapping file, `matlabroot/toolbox/slrequirements/attribute_maps`. To specify a template file in the attribute mapping, open the attribute mapping file that corresponds to the external requirements management tool in a text editor. Modify the value of the `<templateFile>` tag to match the name of the template file. You might need to restart MATLAB to be able to select the mapping file in the Importing Requirements dialog.

See Also

Requirements Editor

More About

- “Import Requirements from ReqIF Files” on page 1-17
- “Export Requirements to ReqIF Files” on page 1-61
- “Round-Trip Importing and Exporting for ReqIF Files” on page 1-99
- “Best Practices and Guidelines for ReqIF Round-Trip Workflows” on page 1-103

Use Callbacks to Customize Requirement Import Behavior

You can use callbacks to execute code when you import requirements from third-party tools or when you update requirements. The `PreImportFcn` callback executes before you import requirements and the `PostImportFcn` callback executes after you import the requirements. You can use these callbacks to customize import behavior.

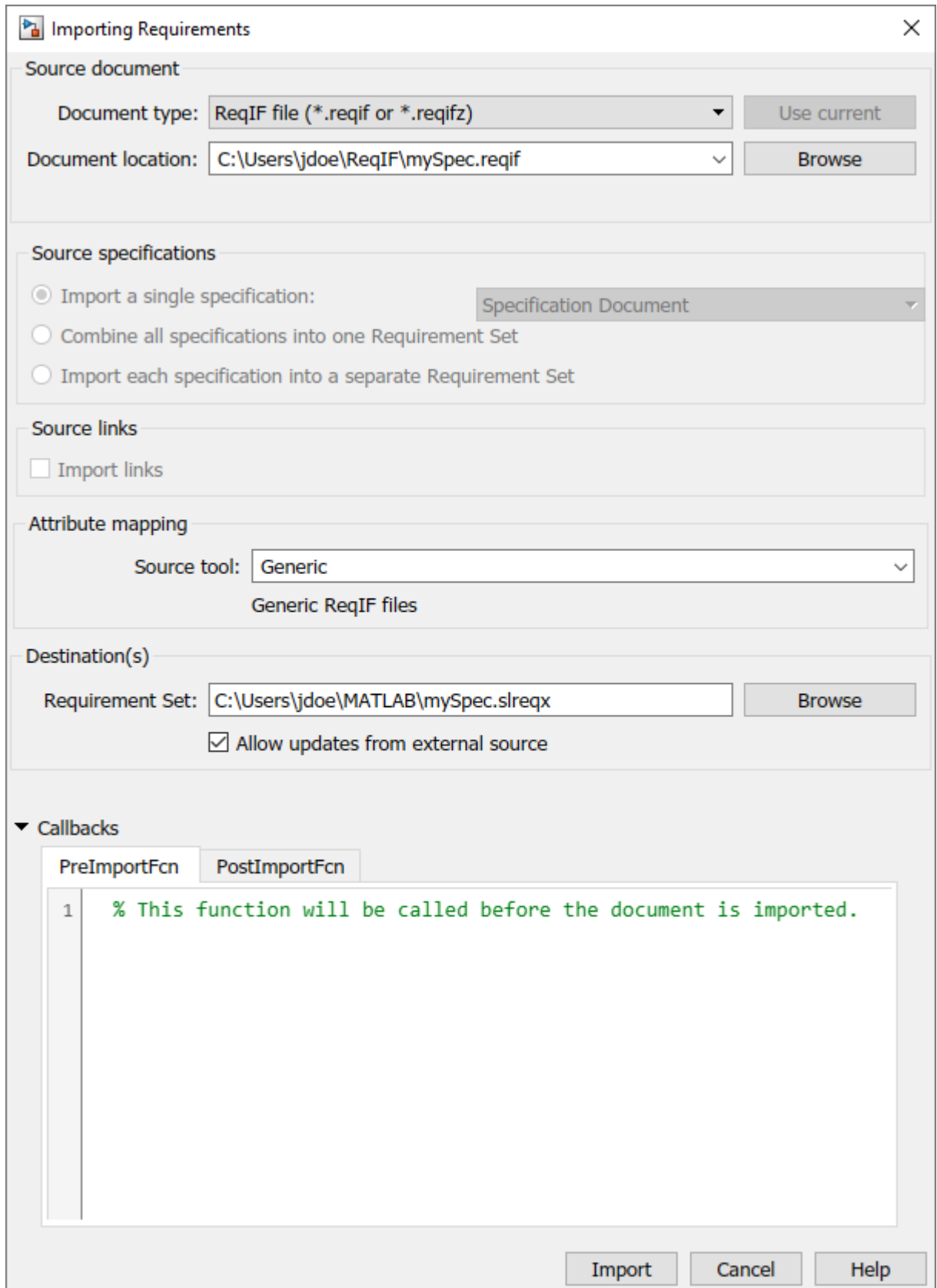
Assign Code to Callbacks

You can assign code to the `PreImportFcn` and `PostImportFcn` callbacks by using the **Requirements Editor** or at the MATLAB command line.

Assign Code to Callbacks by Using the Requirements Editor

To use the **Requirements Editor** to assign code to the `PreImportFcn` and `PostImportFcn` callbacks:

- 1** In the **Requirements Editor**, click **Import**.
- 2** In the Importing Requirements dialog, set **Document type** to the third-party tool that you are importing requirements from.
- 3** Next to **Document location**, click **Browse** and select the requirement file, document, module, or other requirement container.
- 4** Apply your desired import settings. For more information, see “Import Requirements from Third-Party Applications” on page 1-8.
- 5** In the Importing Requirements dialog box, expand the **Callbacks** section.
- 6** Select the **PreImportFcn** or **PostImportFcn** tab.



- 7 Enter your code in the box. You can enter code or the name of a MATLAB script that contains your code.
- 8 Click **Import**.

After you import the requirements, you can view and edit the callbacks in the **Requirements Editor**. Select the import node and, in the right pane, under **Callbacks**, select the **PreImportFcn** or **PostImportFcn** tab.

Assign Code to Callbacks Programmatically

To assign code to callbacks programmatically:

- 1 In MATLAB, select the **Home** tab, then click **New Script**.
- 2 In the script, enter the code that you want the callback to execute.
- 3 Select the **Editor** tab, then click **Save**. Enter a name for the script, then click **Save**.
- 4 At the MATLAB command line, use `slreq.import` to import your requirements. Use the `preImportFcn` and `postImportFcn` arguments to assign your scripts to the callbacks.

After you import the requirements, you can get the code registered to the callbacks by using `getPreImportFcn` and `getPostImportFcn`. You can change the code assigned to the callbacks by using `setPreImportFcn` and `setPostImportFcn`.

Customize Requirement Import Behavior

You can use the code assigned to callbacks to execute commands before and after you import requirements. For example, you can use the `PreImportFcn` callback to customize your import options or use the `PostImportFcn` callback to specify property values of imported requirements.

Customize the Pre-Import Behavior

You can use the `PreImportFcn` callback to specify how Requirements Toolbox imports requirements from different third-party tools. Use `slreq.getCurrentImportOptions` in the callback to return one of these objects, depending on the third-party tool:

- `slreq.callback.CustomImportOptions`
- `slreq.callback.DOORSImportOptions`
- `slreq.callback.MSExcelImportOptions`
- `slreq.callback.MSWordImportOptions`
- `slreq.callback.ReqIFImportOptions`

You can modify this object in the callback code to change how Requirements Toolbox imports the requirements.

For example, you can specify the mapping file to use when you import ReqIF files. Modify this example code to specify the full file path of your mapping file and assign the code to the `PreImportFcn` callback:

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "C:\Users\jdoe\Documents\myMappingFile.xml";
```

Customize the Post-Import Behavior

You can use the `PostImportFcn` callback to execute code that modifies requirements after the import completes.

For example, to specify property values of the imported requirements in the `PostImportFcn` callback, use `slreq.getCurrentObject` to get a handle to the Import node, then use `slreq.Reference` methods to get handles to the imported requirements. Then, use dot notation to set the property values. For more information, see “Property Access Syntax”. This example code shows how to get handles to the child referenced requirements under the current import node.

```
topRef = slreq.getCurrentObject;  
refs = find(topRef);
```

You can also use the “`IndexEnabled`” and “`IndexNumber`” properties to customize requirement index numbering. For more information, see “Customize Requirement Index Numbering” on page 1-85. For an example, see “Import Requirements from a Microsoft Excel Document” on page 1-125.

To move a referenced requirement, use `setParent`, `moveUp`, or `moveDown` in the `PostImportFcn` callback.

See Also

`slreq.getCurrentObject` | `slreq.getCurrentImportOptions` | `setParent`

Related Examples

- “Import Requirements from a Microsoft Excel Document” on page 1-125

More About

- “Execute Code When Loading and Saving Requirement Sets” on page 1-117
- “Customize Requirement Index Numbering” on page 1-85

Execute Code When Loading and Saving Requirement Sets

You can use Requirements Toolbox callbacks to execute code when you load and save requirement sets. The `PostLoadFcn` callback executes when you load the requirement set and the `PreSaveFcn` callback executes when you save the requirement set. You can assign code to the callbacks to customize the requirement set load and save behavior.

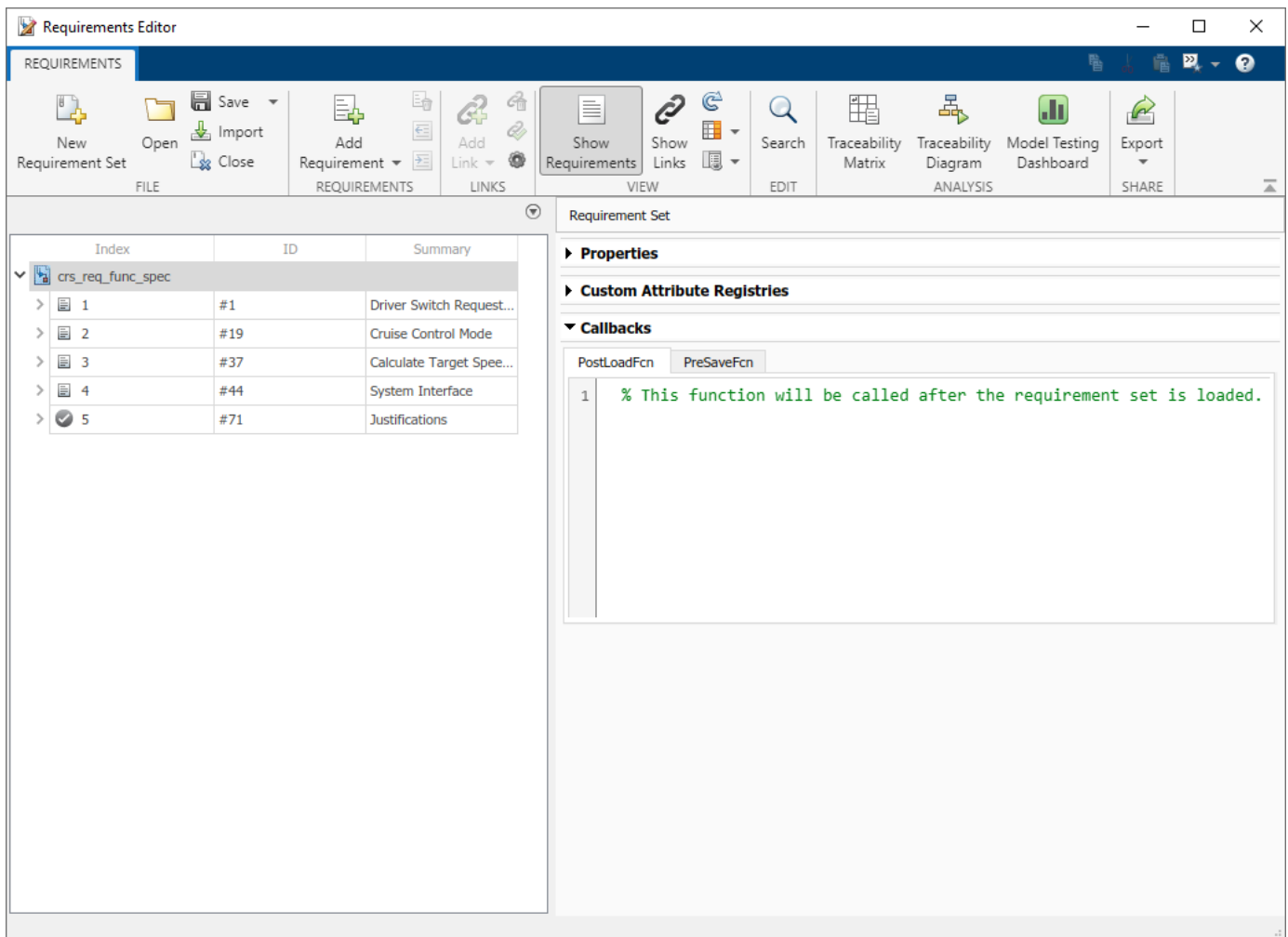
Assign Code to Callbacks

You can assign code to the callbacks in the **Requirements Editor** or at the MATLAB command line.

Assign Code to Callbacks in the Requirements Editor

To assign code to the `PostLoadFcn` and `PreSaveFcn` callbacks in the **Requirements Editor**:

- 1 Open your requirement set by clicking **Open**. Select the SLREQX file and click **Open**.
- 2 Select the requirement set in the **Requirements Editor**.
- 3 In the right pane, under **Callbacks**, select the **PostLoadFcn** or **PreSaveFcn** tab.



- 4 Enter your code in the box.

Alternatively, you can enter your code in a script. In the **PostLoadFcn** or **PreSaveFcn** tab, enter the name of the script. The script must be on the MATLAB path.

Assign Code to Callbacks Programmatically

To assign code to the `PostLoadFcn` or `PreSaveFcn` callback at the MATLAB command line:

- 1 In MATLAB, select the **Home** tab, then click **New Script**.
- 2 In the script, enter the code that you want the callback to execute.
- 3 Select the **Editor** tab, then click **Save**. Enter a name for the script, then click **Save**.
- 4 Load your requirement set and return the `slreq.ReqSet` object by using `slreq.load` or `slreq.open`.
- 5 Assign the script as the `PostLoadFcn` or `PreSaveFcn` callback by using `setPostLoadFcn` or `setPreSaveFcn`.

You can view the code assigned to the `PostLoadFcn` and `PreSaveFcn` callbacks by using `getPostLoadFcn` and `getPreSaveFcn`.

Customize Requirement Set Load and Save Behavior

You can use the code assigned to callback to customize the requirement set load and save behavior. For example, you can use the `PostLoadFcn` to:

- Load the **Requirements Editor** view settings from a MAT-file by using `slreq.importViewSettings`.
- Open a design artifact, such as a Simulink model or MATLAB script, by using `open`.
- Run linked tests by using `runTests`.

You can also, for example, use `PreSaveFcn` to export the current **Requirements Editor** view settings to a MAT-file by using `slreq.exportViewSettings`.

You can use `slreq.getCurrentObject` in the `PostLoadFcn` and `PreSaveFcn` callbacks to get a handle to the requirement set within the callbacks. For more information, see the Tips section of `slreq.getCurrentObject`.

See Also

`slreq.ReqSet` | `slreq.getCurrentObject` | `setPostLoadFcn` | `setPreSaveFcn`

More About

- “Use Callbacks to Customize Requirement Import Behavior” on page 1-113

Import Requirements from IBM Rational DOORS by Using the API

This example shows you how to import requirements from an IBM® Rational® DOORS® module by using the Requirements Toolbox™ API.

Configure IBM Rational DOORS

To interface with IBM Rational DOORS, configure MATLAB®. At the MATLAB command prompt, enter:

```
rmi setup doors
```

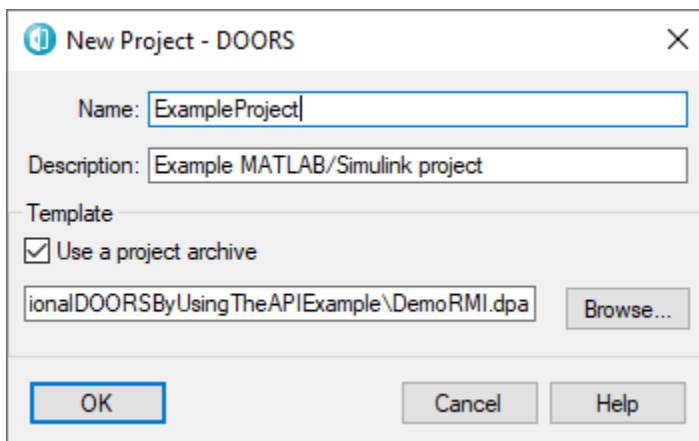
For more information, see “Configure IBM Rational DOORS Session” on page 1-42.

Open the DOORS Project

In this example, you will use the `DemoRMI.dpa` project in IBM Rational DOORS, which contains requirements modules that describe a fault-tolerant control system.

In IBM Rational DOORS, create a new project:

- 1 Select **File > New > Project**.
- 2 In the New Project dialog, enter `ExampleProject` in the **Name** field.
- 3 In the **Description** field, enter `Example MATLAB/Simulink project`.
- 4 Select **Use a project archive**.
- 5 Click **Browse** and select `DemoRMI.dpa`.



Import a Requirements Module

In this example, you will import all of the requirements from the `FuelSys Requirements Specification` module.

In IBM Rational DOORS, open the `FuelSys Requirements Specification` module and find the module ID. For more information, see [How to identify the unique ID for an item in DOORS database explorer on the IBM website](#).

Use `slreq.import` to import the module. Enter the name of the requirement set file, specify that the requirements are referenced requirements and should use Rich Text Formatting, name the requirement set `fuelSysReqSpec`, and enter the module ID. The function returns the number of imported referenced requirements, the requirement set file path, and the requirement set object.

```
[refCount1, reqSetFilePath1, myReqSet1] = slreq.import("linktype_rmi_doors", ...  
    AsReference=true, RichText=true, ReqSet="fuelSysReqSpec", DocID="000001c1");
```

```
Importing from 000001c1 of type linktype_rmi_doors ..  
.. done.
```

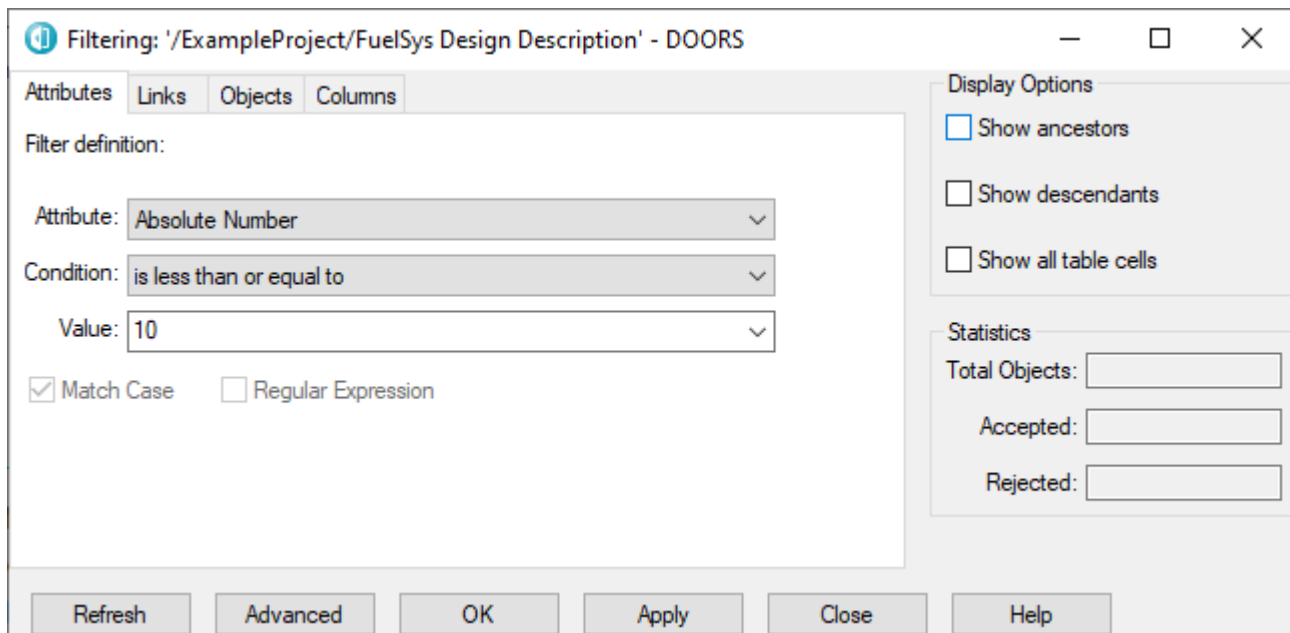
Import a Subset of Requirements from a Module

You can import a subset of requirements from the `FuelSys Design Description` module by applying a filter. Open the `FuelSys Design Description` module in IBM Rational DOORS.

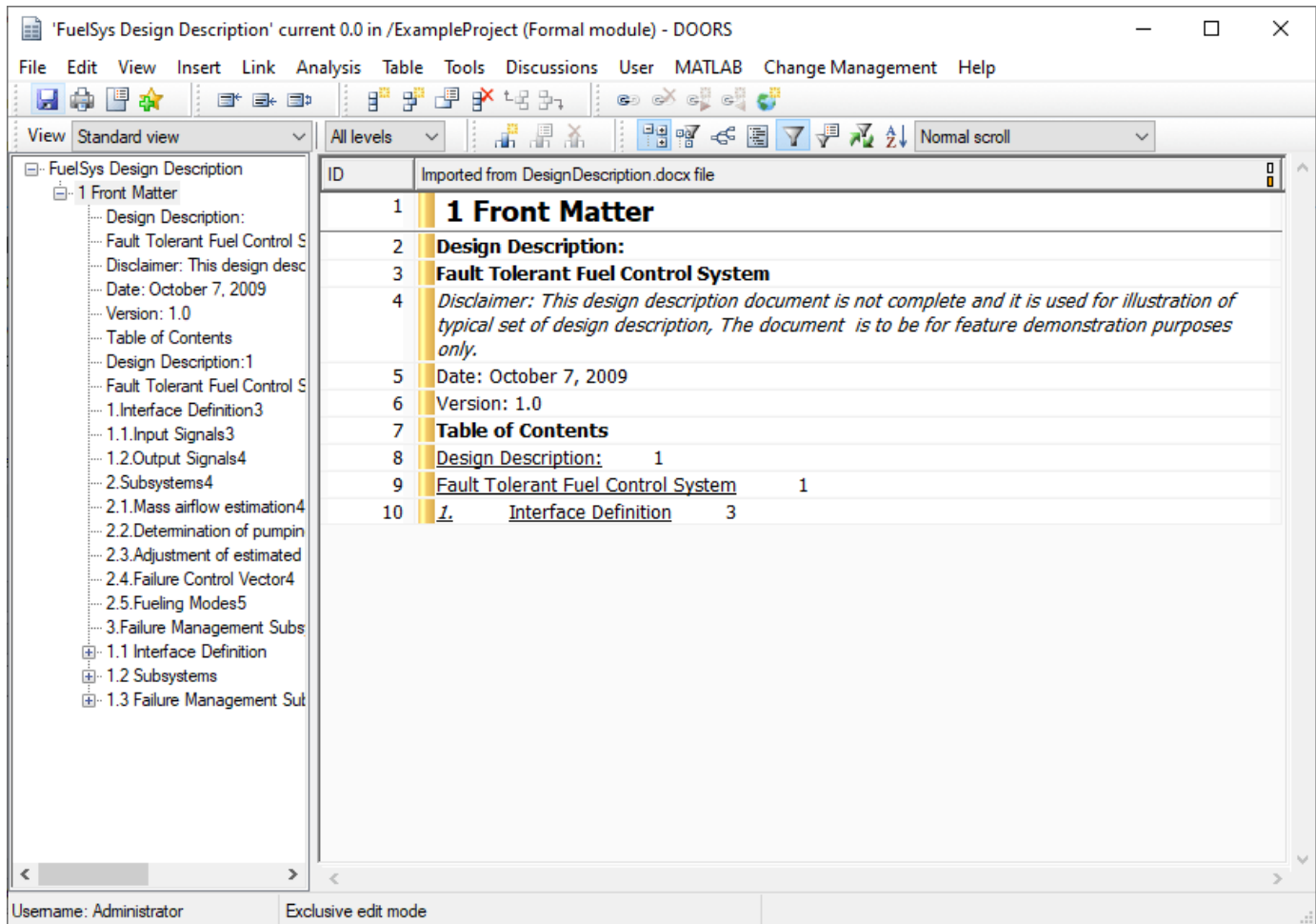
Filter the Requirements Module

Apply a filter to the module. For more information on applying a filter to a requirements module, see [Defining filters on the IBM website](#). In the `Filtering` dialog:

- 1 Set **Attribute** to `Absolute Number`.
- 2 Set **Condition** to `is less than or equal to`.
- 3 Next to **Value**, enter `10`.



The module displays only requirements that match the filter.



When you apply a filter to your DOORS module and import the module to Requirements Toolbox, the process imports only the requirements that match the filter. When you import requirements by using the API, Requirements Toolbox does not store the filter for future use.

Import the Filtered Requirements Module

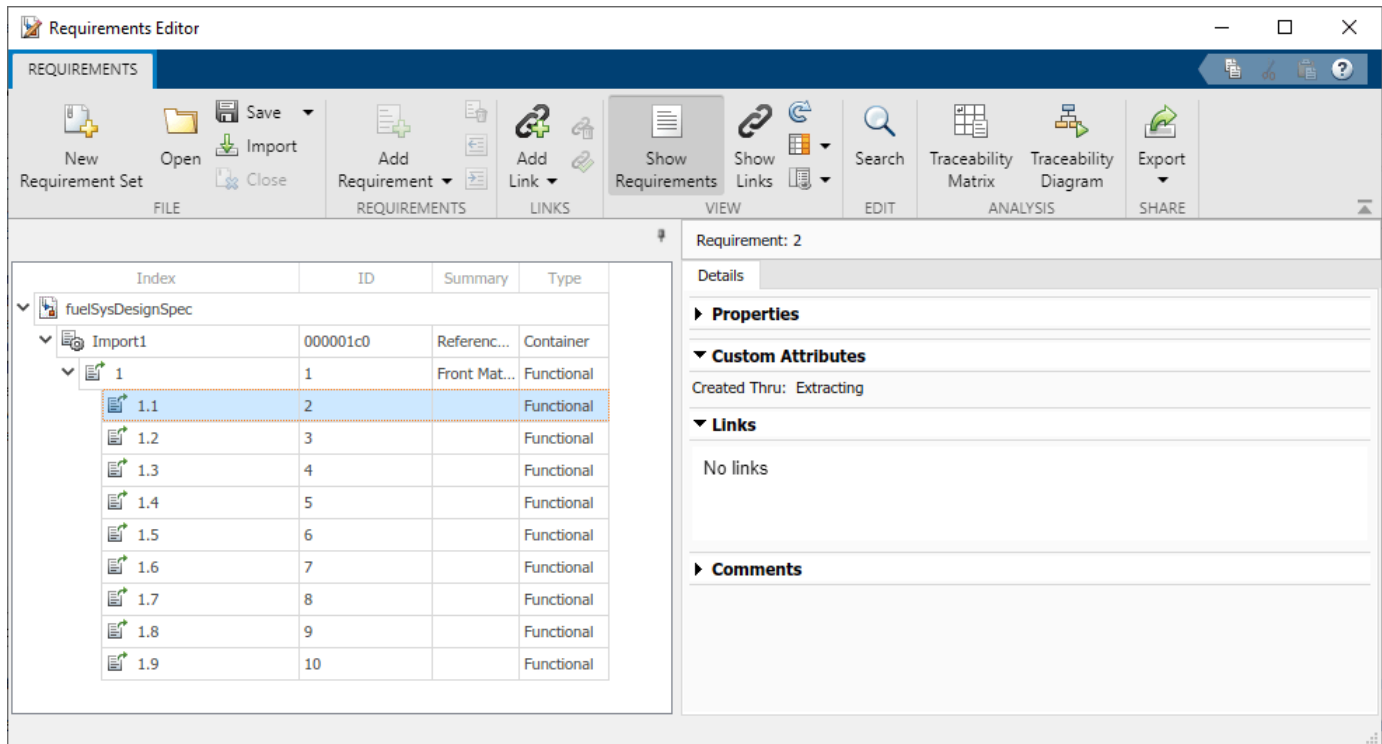
To import the filtered requirements module, use `slreq.import`. Enter the name of the requirement set file, specify that the requirements are referenced requirements and should use Rich Text Formatting, name the requirement set `fuelSysDesignSpec`, but don't enter the module ID. If you don't specify the module ID, the `slreq.import` function imports the active requirements module.

The module contains a requirements attribute called `Created Thru`. Import the attribute along with the requirements as a custom attribute. The function returns the number of imported referenced requirements, the requirement set file path, and the requirement set object.

```
[refCount2,reqSetFilePath2,myReqSet2] = slreq.import("linktype_rmi_doors",ReqSet="fuelSysDesignSpec")
```

```
Importing from FuelSys Requirements Specification of type linktype_rmi_doors ..  
.. done.
```

Requirements Toolbox imports only the first 10 requirements from the module and maps the `Created Thru` attribute to a new custom attribute in the requirement set.



If you have custom attributes that you want to import as the built-in requirement properties “Rationale” or “Keywords”, you can use:

```
slreq.import("linktype_rmi_doors",keywords="Keyword DOORS Attribute",rationale="Rationale DOORS /
```

For more information about custom attributes, see “Add Custom Attributes to Requirements” on page 1-81.

Update the Filtered Requirement Set

After you import the requirement set, you can update it. For more information, see “Update Imported Requirements” on page 1-89.

In DOORS, change the applied filter in the FuelSys Design Description module.

In the Filtering dialog:

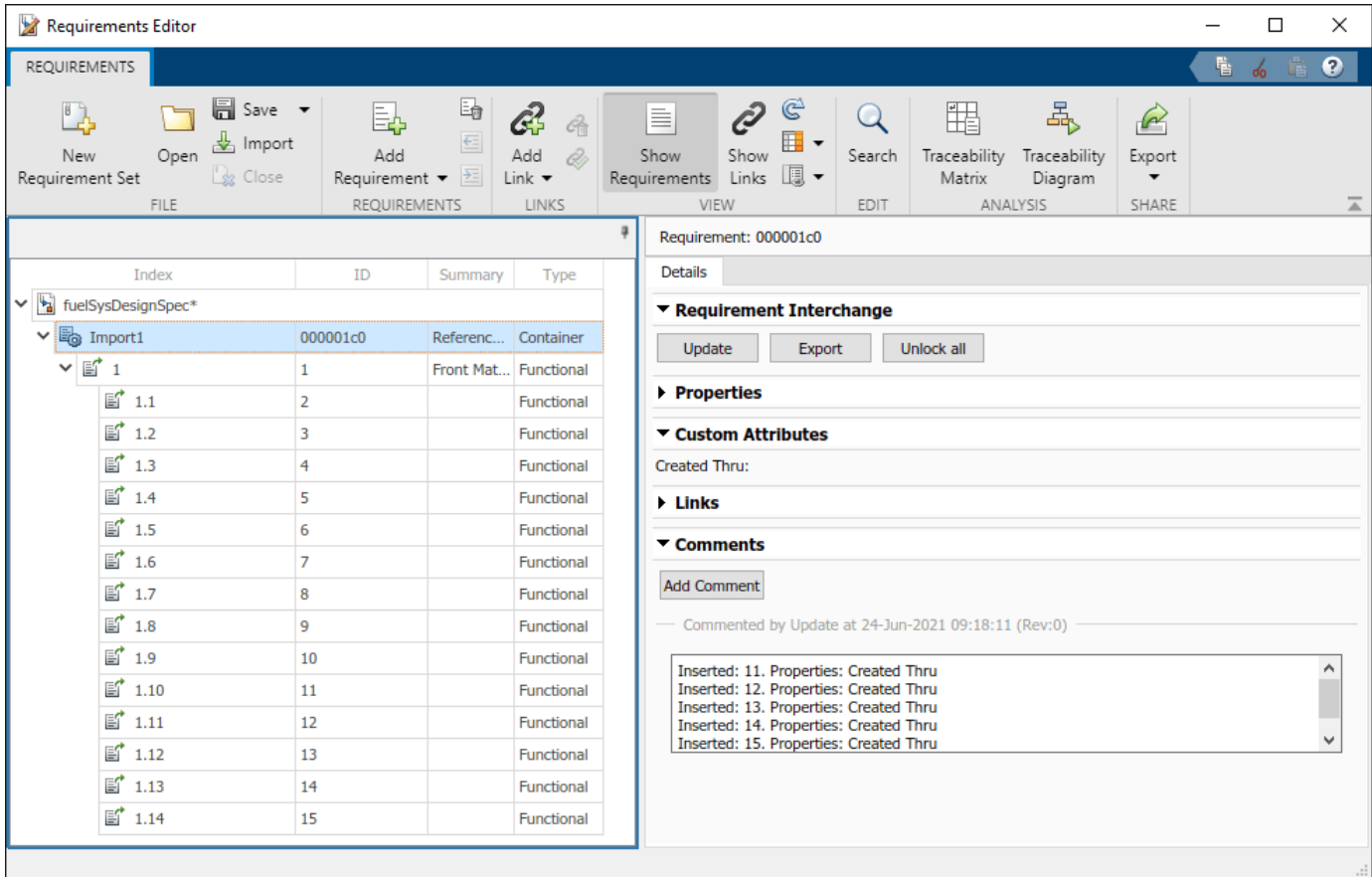
- 1 Set **Attribute** to Absolute Number.
- 2 Set **Condition** to is less than or equal to.
- 3 Next to **Value**, enter 15.

Find the Import node from the requirement set myReqSet2. Update the requirement set.

```
importNode = find(myReqSet2,Index="Import1");
status = updateFromDocument(importNode);
```

```
Importing from FuelSys Requirements Specification of type linktype_rmi_doors ..
.. done.
```

Requirements Toolbox amends the requirement set to contain the first 15 requirements.

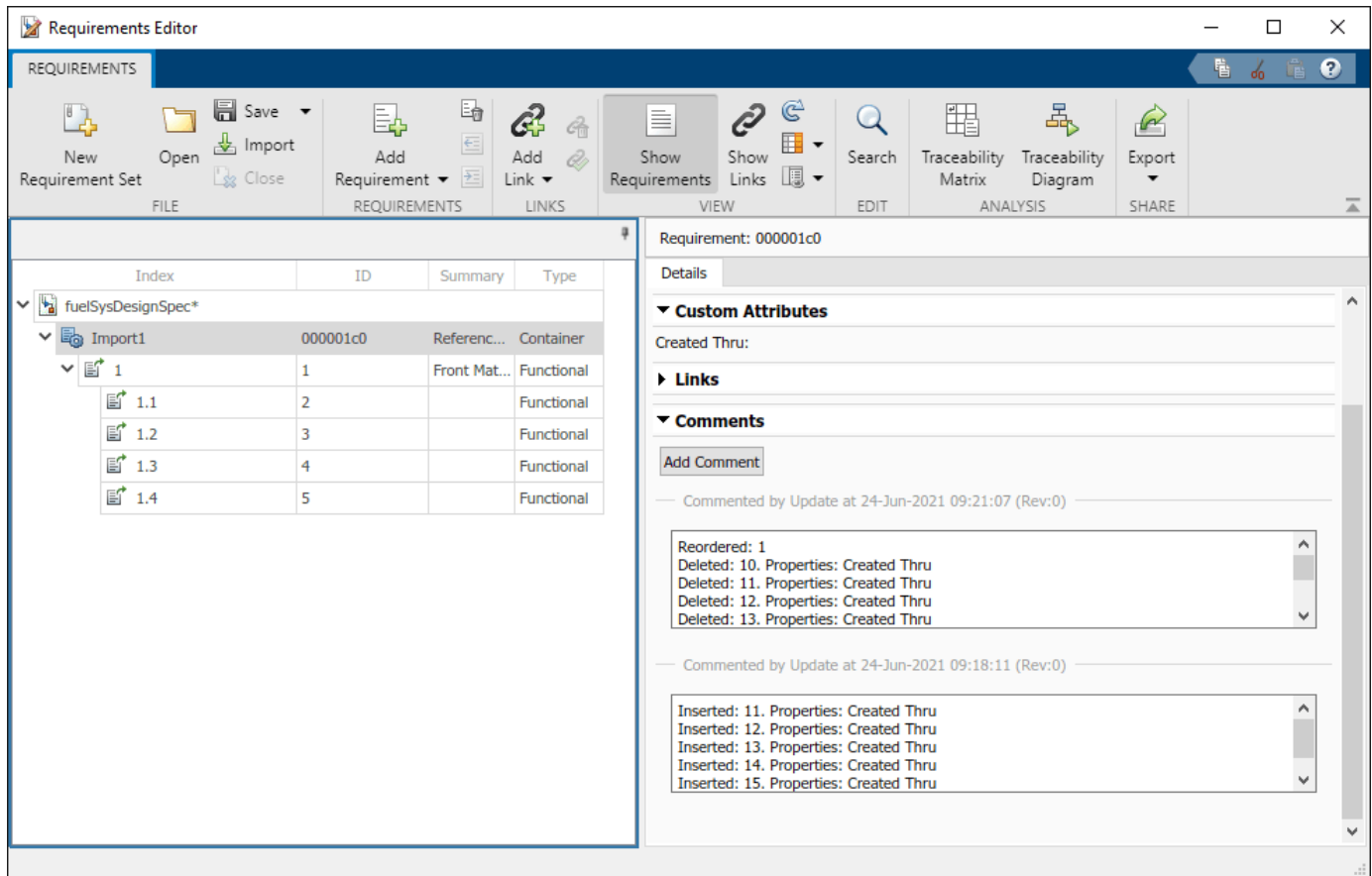


In your DOORS requirements module, update the filter again. For **Value**, enter 5. Find the Import node from the requirement set myReqSet2. Update the requirement set.

```
importNode = find(myReqSet2, Index="Import1");
status = updateFromDocument(importNode);
```

Importing from FuelSys Requirements Specification of type linktype_rmi_doors ..
.. done.

Requirements Toolbox truncates the requirement set to only contain the first 5 requirements.



See Also

`updateFromDocument | slreq.import`

More About

- "Import Requirements from IBM Rational DOORS" on page 1-42
- "Working with IBM Rational DOORS 9 Requirements" on page 8-30
- "Import Requirements from Third-Party Applications" on page 1-8

Import Requirements from a Microsoft Excel Document

This example shows how to import requirements from a Microsoft® Excel® document. You can map columns from the Excel spreadsheet to certain requirements properties and custom attributes. You can also assign specific values to the imported requirements, such as the requirement type or index, by using a callback that executes after the requirements import.

This example uses the `ExampleRequirements` Excel file. To open the file, enter:

```
winopen("ExampleRequirements.xlsx")
```

Author a PostImportFcn Callback Script

You can author a script and register it as the `PostImportFcn` callback to process imported requirements in Requirements Toolbox™ immediately after the import finishes. This example uses the `slreqExamplePostImportXls` script as the `PostImportFcn` callback. View the script by entering:

```
edit("slreqExamplePostImportXls.m")
```

Requirements import from Excel with the `Functional` type by default. The first `for` loop in the script iterates over each imported requirement to assign the requirement type. The loop checks the original requirement type from the Microsoft Excel file and then sets the imported requirement type to the built-in Requirements Toolbox requirement type that aligns with the imported type. If the requirement in Excel does not have a requirement type, the script leaves the requirement as a `Functional` type requirement.

```
idToRef = containers.Map('KeyType', 'char', 'ValueType', 'Any');
topRefs = slreq.getCurrentObject;
reqSet = topRefs(1).parent;

refs = reqSet.find('type', 'Reference');
for i = 1:numel(refs)
    ref = refs(i);
    switch ref.getAttribute('orig_Type')
        case 'Heading'
            ref.Type = 'Container';
        case 'Note'
            ref.Type = 'Informational';
        otherwise
            % leave AS IS
    end
    % build the Map of IDs for the next step
    idToRef(ref.Id) = ref;
end
```

The second `for` loop iterates over each imported requirement and applies the requirements hierarchy defined in Excel.

```
sortedIds = sort(keys(idToRef));
for i = 2:numel(sortedIds)
    thisId = sortedIds{i};
    % Find the parent ID by truncating this ID at the last '.'
    parentId = '';
    dotCharIdx = find(thisId == '.');
    if ~isempty(dotCharIdx)
```

```

        parentId = thisId(1:dotCharIdx(end)-1);
    end
    if ~isempty(parentId)
        ref = idToRef(thisId);
        parentObj = idToRef(parentId);
        ref.overrideLocked('parent', parentObj.SID);
    end
end
end

```

The Excel file uses a period (.) in the requirement ID to indicate different levels of the requirement hierarchy. The script uses the period to identify the hierarchy levels and re-create the hierarchy.

	A	B	C	D
1				
2		Unique ID	Type	Title
3		01	Heading	Overview
4		01.01	Note	
5		01.02	Note	
6		02	Heading	System Description
7		02.01	Heading	System inputs

Configure Import Options

Create a structure that contains the options to use during import. List the range of columns and rows of the Excel file that contain requirements.

```
importOptions = struct("columns",[2 7],"rows",[3 56]);
```

Add information to map the requirements data from requirements in Excel to Requirements Toolbox. Map columns 2, 4, and 5 to the built-in `slreq.Reference` properties ID, Summary, and Description.

```
importOptions.idColumn = 2;
importOptions.summaryColumn = 4;
importOptions.descriptionColumn = 5;
```

Columns 3, 6, and 7 cannot be directly mapped to `slreq.Reference` properties. Map these columns to custom attributes called `orig_Type`, `Remark`, and `Status`. Note that the requirement type in Excel maps to a custom attribute, instead of to the `Type` property of the `slreq.Reference` objects.

```
importOptions.attributes = {'orig_Type','Remark','Status'};
importOptions.attributeColumn = [3 6 7];
```

Register the `slreqExamplePostImportXls` script as the `PostImportFcn` callback.

```
importOptions.postImportFcn = "slreqExamplePostImportXls";
```

Import the Requirements

Import the requirements from the Requirements sheet in the `ExampleRequirements` Excel file into a new requirement set called `ImportedFromExcel` by using `slreq.import`. Import the requirements as plain text referenced requirements with additional import options specified from the

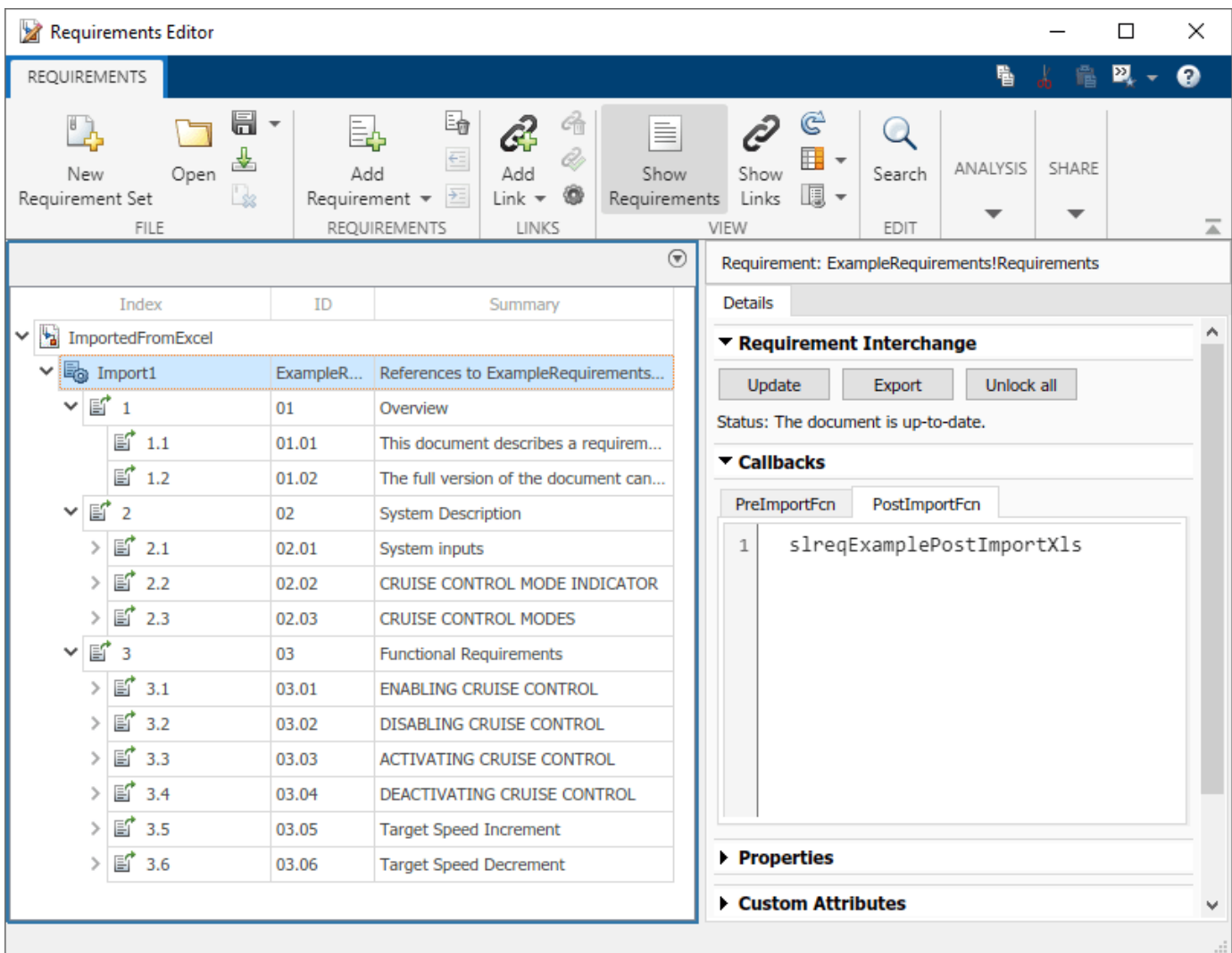
options structure. Return the number of imported referenced requirements, the requirement set file path, and a handle to the requirement set.

```
[count, reqSetFilePath, reqSet] = slreq.import("ExampleRequirements.xlsx", ...
    ReqSet="ImportedFromExcel", AsReference=true, RichText=false, ...
    subDoc="Requirements", options=importOptions);
```

Save the imported requirement set, then examine it in the **Requirements Editor**.

```
save(reqSet);
explore(reqSet);
```

The `slreqExamplePostImportXls` script is registered as the `PostImportFcn` callback function for the Import node. The callback also executes if you update the requirement set.



Examine the Imported Requirement Types

Examine the imported requirements and compare the requirement types to the requirements in Excel. The callback function assigned types to the imported requirements by using the Requirements

Toolbox built-in types. For example, requirement 1 had the type `Heading` in the Excel file and imports with the type `Container`.

```
req1 = slreq.find(Type="Reference",Index="1");  
originalType1 = getAttribute(req1,"orig_Type")
```

```
originalType1 =  
'Heading'
```

```
importedType1 = getAttribute(req1,"Type")
```

```
importedType1 =  
'Container'
```

Requirement 1.1 had the type `Note` in Excel and imports with the type `Informational`.

```
req2 = slreq.find(Type="Reference",Index="1.1");  
originalType2 = getAttribute(req2,"orig_Type")
```

```
originalType2 =  
'Note'
```

```
importedType2 = getAttribute(req2,"Type")
```

```
importedType2 =  
'Informational'
```

Requirement 2.1.1 had no indicated type in Excel and imports with the type `Functional`.

```
req3 = slreq.find(Type="Reference",Index="2.1.1");  
originalType3 = getAttribute(req3,"orig_Type")
```

```
originalType3 =
```

```
    0×0 empty char array
```

```
importedType3 = getAttribute(req3,"Type")
```

```
importedType3 =  
'Functional'
```

Examine the Imported Requirement Hierarchy

Examine the hierarchy of the imported requirements and compare the hierarchy to the Excel file. The callback function created the hierarchy of the imported requirements based on the hierarchy from the Excel file. For example, requirement 2.1.1 in Excel has 6 child requirements.

	Unique ID	Type	Title	
1				
2				
3	01	Heading	Overview	
6	02	Heading	System Description	
7	02.01	Heading	System inputs	
8	02.01.01		Cruise control buttons	Five butt
9	02.01.01.01		Cruise	A button
10	02.01.01.02		Cancel	A button
11	02.01.01.03		Set	A button system i:
12	02.01.01.04		Res	A button
13	02.01.01.05		Inc	A button increase the buttc amount i
14	02.01.01.06		Dec	A button decrease the buttc amount i
15	02.01.02	Heading	Other inputs	

Get the child requirements for the imported requirement 2.1.1.

```
childReqs = children(req3)
```

```
childReqs=1x6 object
```

```
1x6 Reference array with properties:
```

```

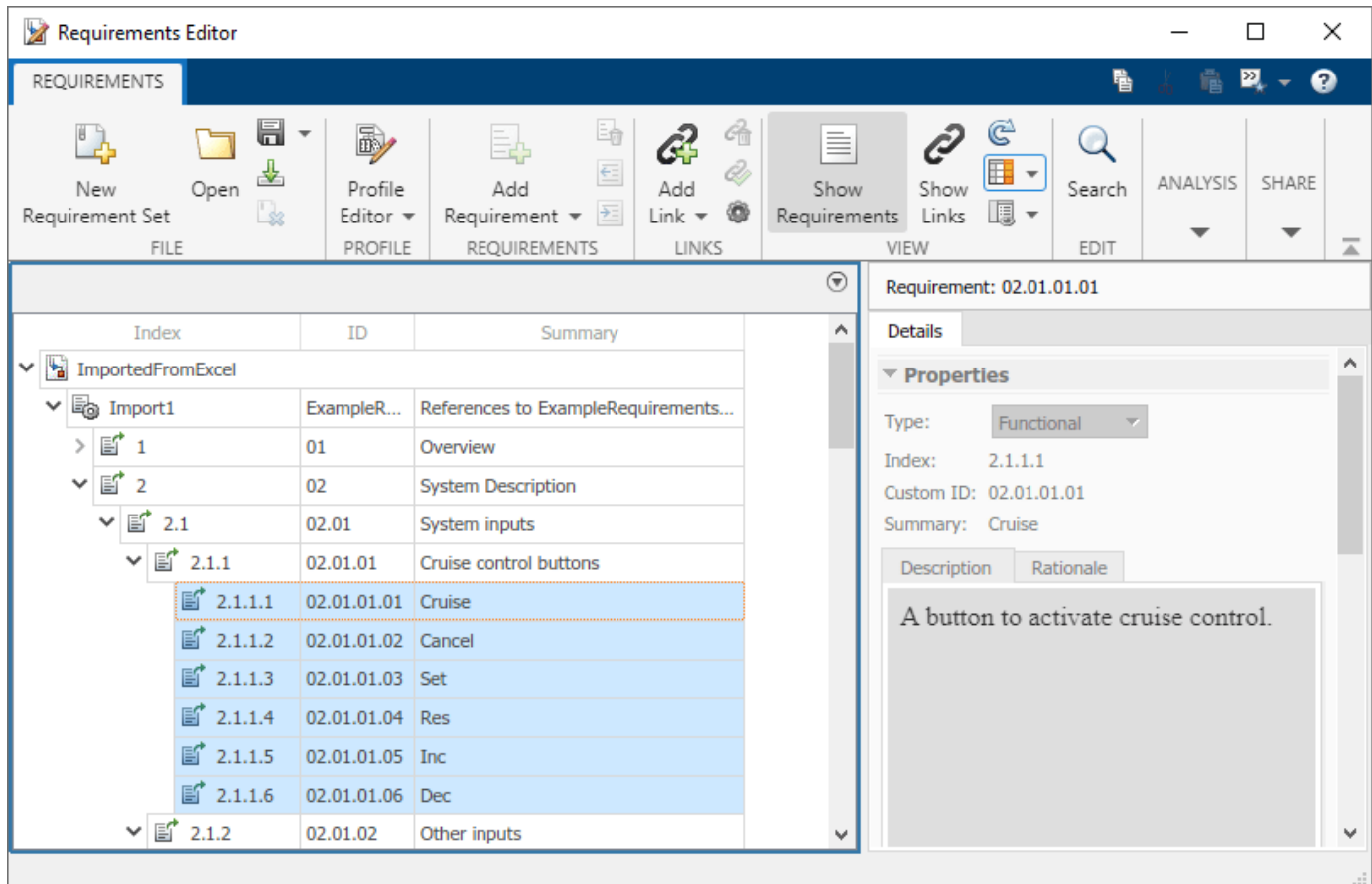
Id
CustomId
Artifact
ArtifactId
Domain
UpdatedOn
CreatedOn
CreatedBy
ModifiedBy
IsLocked
Summary
Description
Rationale
Keywords
Type
IndexEnabled
IndexNumber
SID
FileRevision
ModifiedOn
Dirty
Comments
Index
    
```

In Excel, the first child requirement of 2.1.1 has the title `Cruise`. Get the summary for the first child requirement in the array.

```
childSummary = childReqs(1).Summary
```

```
childSummary =  
'Cruise'
```

You can also check that the imported requirements hierarchy matches the hierarchy from Excel in the **Requirements Editor**.



See Also

`slreq.import`

More About

- “Use Callbacks to Customize Requirement Import Behavior” on page 1-113
- “Update Imported Requirements” on page 1-89

Export Requirement and Link Information to Excel

This example shows how to use Requirements Toolbox™ to export requirement and link information programmatically. The example constructs a custom function that you can use on a requirement set and saves the information as an Excel® spreadsheet.

Inspect the Export Function

Open the attached MATLAB® program file, `reqXlsExport.m`, to view the function code. The function uses programmatic commands to export the information to an Excel file.

The function takes three input arguments in order:

- 1 The requirement set that you want to export, specified as a character vector. The argument must include the `.slreqx` extension.
- 2 The name of the Excel file that you want to save the requirements to, specified as a character vector. The name must include the `.xls` extension.
- 3 The properties that you want to export, specified as a cell array of character vectors. You can also include properties of the links associated with the requirements. Specify at least one requirement property.

`reqXlsExport` can retrieve this built-in requirement information, which corresponds to the respective `slreq.Requirement` class properties:

- `Id`
- `Index`
- `Summary`
- `Description`
- `Rationale`
- `Keywords`
- `CreatedOn`
- `Dirty`
- `ModifiedOn`
- `ModifiedBy`
- `SID`

You specify one or more of the properties in the property cell array argument.

The `Description` and `Rationale` properties normally return text that includes formatting commands. Exporting this text obscures the text in Excel. To prevent this, the `reqXlsExport` code uses the `getDescriptionAsText` and `getRationaleAsText` methods to return the description and rationale as plain text.

`reqXlsExport` can also retrieve the associated link information of the requirements. The function code uses the `slreq.getIncomingLinkTypeLabel` and `slreq.getOutgoingLinkTypeLabel` to establish the headers in the table for each link direction and identifies if the label corresponds to the requirement by using the `getOutgoingTypeLabel`, `getIncomingTypeLabel`, `getSourceLabel`, and `getDestinationLabel` methods. To retrieve the link information, you must load the links associated with the requirements before executing the function.

You export the link information by using these syntaxes:

- '`<links-in>`': Exports the incoming link labels associated with the requirements and groups them by type.
- '`<links-out>`': Exports the incoming link labels associated with the requirements and groups them by type.

You can also specify the link type associated with the requirement by entering a colon and the link type after `links-out` or `links-in`. For example, to populate the table with associated outgoing links of the verify type, include '`<links-out:verify>`' in the cell array.

You can also specify custom attributes from stereotypes and custom link types. For more information on defining custom attributes and link types, see “Add Custom Attributes to Requirements” on page 1-81 and “Define Custom Requirement and Link Types and Properties” on page 1-78.

After retrieving the requirement and link information, the function consolidates the information as an array and uses the `writecell` function to export the array to Excel.

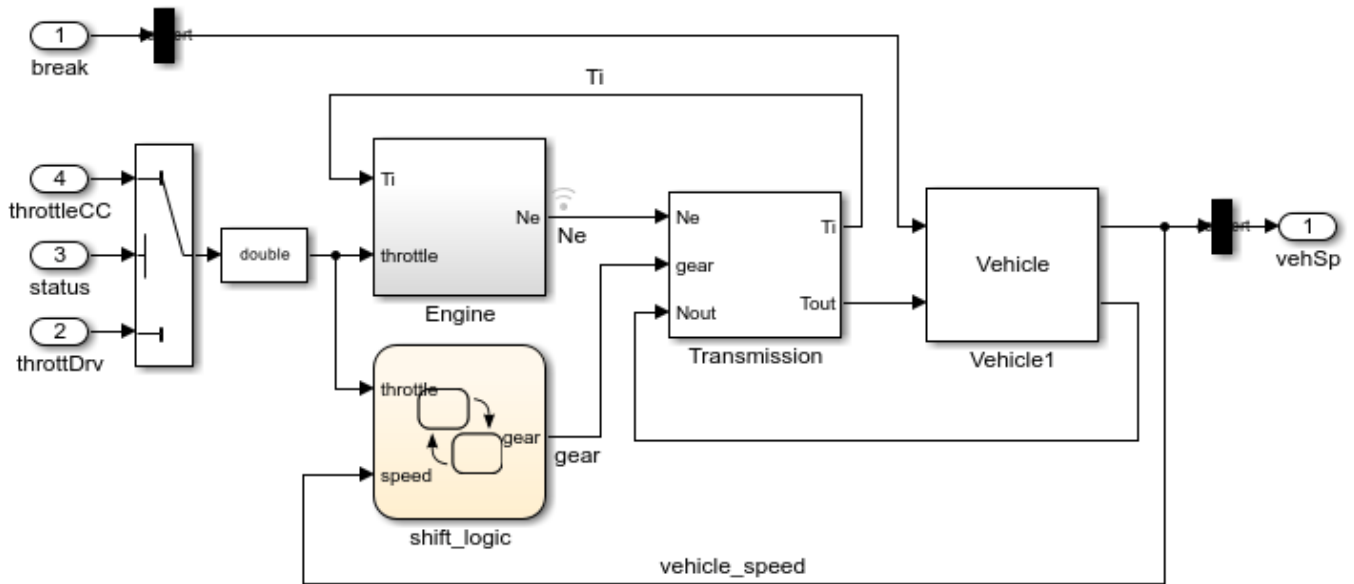
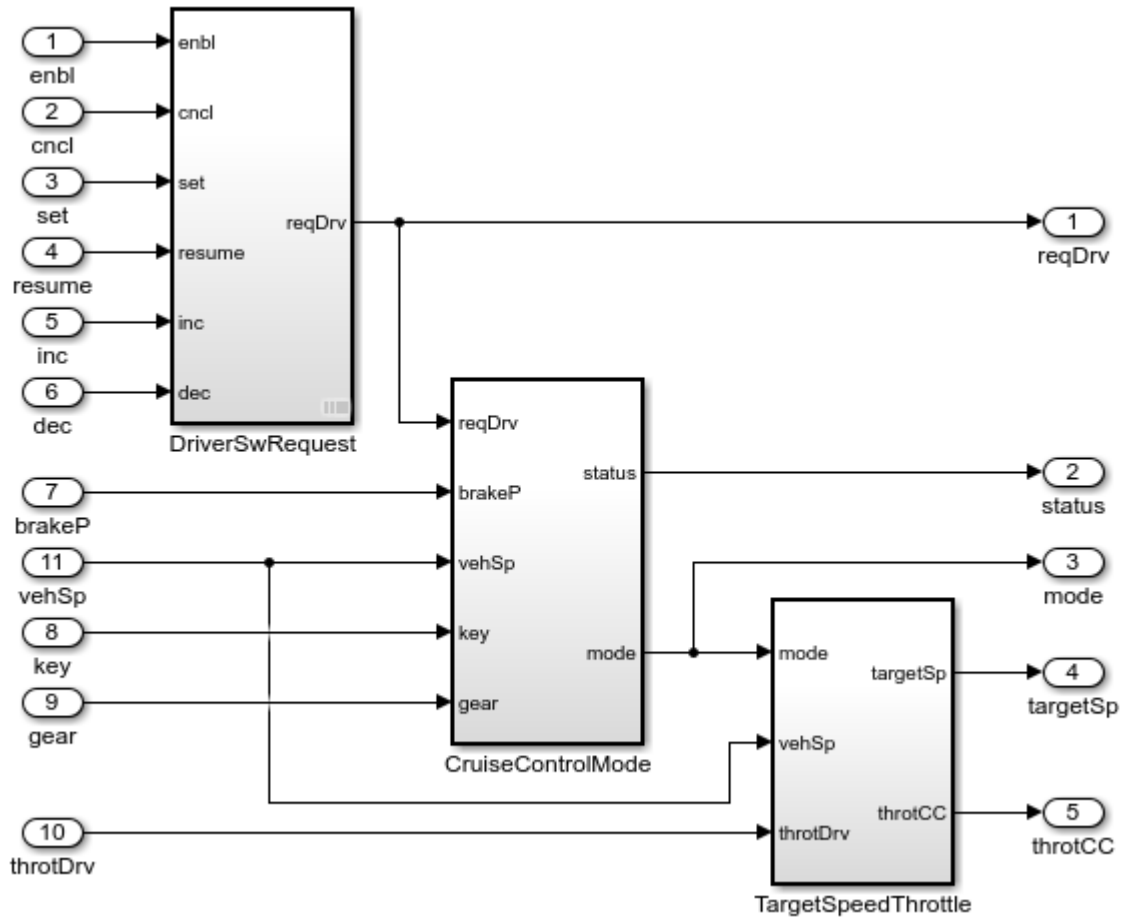
Export Requirements from Project

In this example, you export requirement information in a MATLAB project. Open the attached example project, `CruiseRequirementsExample`.

```
addpath(pwd)
openProject("CruiseRequirementsExample");
```

Next, open the requirement set that you want to export and the files that contain associated information, such as link sets, models, and tests.

```
sysReqSeqSet = slreq.open("crs_req.slreqx");
funcReqSet = slreq.open("crs_req_func_spec.slreqx");
sltest.testmanager.load("crs_controller_tests.mldatx");
sltest.testmanager.load("DriverSwRequest_Tests.mldatx");
open_system("crs_controller");
open_system("crs_plant");
open("crs_controllerdic.sldd");
```

After loading the requirement set and the supporting files, use `reqXlsExport`. For example, to export the system requirements together with linking details, enter this code:

```
outFile = 'sys_req_export.xls';
columnConfig = {'Index', 'Id', 'Type', 'Summary', ...
    'Description', '<links-out:Derive>'};
reqXlsExport(sysReqSeqSet, outFile, columnConfig)
```

The Excel spreadsheet is configured similarly to the **Link Details** pane in the **Requirements Editor**.

Index	Id	Type	Summary	Description	Derives
10	3	Functional	Functional Requirements	Functional Requirements Enabling cruise control Cruise control is enabled when the following conditions are met: Vehicle speed is within the target speed range (40km/h–100km/h). Key position is ON. Gear position is Drive. Cruise button is pushed while the cruise control mode is disabled. Dashboard image	Derives
11	3.1	Functional	Enabling cruise control	Enabling cruise control Dashboard image	Enable Switch Detection
12	3.2	Functional	Disabling cruise control	Disabling cruise control Cruise control is disabled when one or more of the following are met: Key position is set to any other position than ON. When the vehicle is started. Cruise button is pushed while the cruise control mode is enabled or activated. Gear position is not Drive Dashboard image	Cancel Switch Detection
13	3.3	Functional	Activating cruise control	Activating cruise control Cruise control is activated when the following conditions are met: Cruise control mode is enabled. Set button is pushed.	Set Switch Detection
14	3.3.1	Functional	Cruise control mode is enabled.	Cruise control mode is enabled.	
15	3.3.2	Functional	Set button is pushed.	Set button is pushed.	
16	3.3.3	Functional	Vehicle speed is in the target s...	Vehicle speed is in the target speed range (40-100 km/h)	
17	3.3.4	Functional	Gear position is Drive.	Gear position is Drive.	
18	3.3.5	Functional	The target speed is set to a cur...	The target speed is set to a current vehicle speed when the Set button is pushed while the cruise control mode is enabled.	

You can specify as few or as many of the properties that you want to retrieve. For example, to export the creation and modification information for the functional requirement set, `crs_req_func_spec.s\lreqx`, enter:

```
outFile = 'full_func_req_export.xls';
columnConfig = {'Index', 'Id', 'Type', 'Summary', 'Description', 'Rationale', ...
    '<links-in:Implement>', '<links-in:Verify>', 'Keywords', ...
    'CreatedOn', 'CreatedBy', 'ModifiedOn', 'ModifiedBy', 'SID'};
reqXlsExport(funcReqSet, outFile, columnConfig)
```

Index	Id	Type	Summary	Description	Rationale	Implemented by	Verified by	Keywords	CreatedOn	CreatedBy	ModifiedOn	ModifiedBy	SID
1				system to operate upon.		DriverSwRequest			2/27/2017 10:15	itoy	8/2/2017 13:49	asriram	1
2	1.1	#1	Functional	Driver Switch Request Handling	In the following order: 1. Cancel 2. Cruise 3. Set 4. Resume 5. Increment 6. Decrement				2/27/2017 10:15	itoy	8/2/2017 14:09	asriram	2
3	1.1	#2	Functional	Switch precedence	If the following switch operations are repeated, the system should output NoRequest the second time and after as long as the same switch is enabled: Cancel Cruise Set Resume				2/27/2017 10:15	itoy	8/2/2017 14:09	asriram	2
4	1.2	#3	Functional	Avoid repeating commands	When the Increment or Decrement switches are pressed for more than 500ms, they are recognized as LongInc and LongDec respectively.	doNot Repeat			2/27/2017 10:15	itoy	8/2/2017 14:53	asriram	3
5	1.3	#4	Functional	Long Switch recognition	When the Increment switch is enabled repeatedly but it does not reach the threshold to be determined as a Long Switch operation, it outputs Inc_Middle so that a downstream function can recognize that the transition can occur.	Enumerated Constant5			2/27/2017 10:15	itoy	8/3/2017 11:21	asriram	4
6	1.3.1	#5	Functional	Waiting state for Long Increment switch detection	When the Decrement switch is enabled repeatedly but it does not reach the threshold to be determined as a Long Switch operation, it outputs Dec_Middle so that a downstream function can recognize that the transition can occur.	counter Enumerated Constant7 Switch			2/27/2017 10:15	itoy	8/2/2017 15:11	asriram	5
7	1.3.2	#6	Functional	Waiting state for Long Decrement switch detection	If the Cancel switch is pressed, the value of reqDrv should be set to reqMode.Cancel.	Enumerated Constant Switch2	Cancel button		2/27/2017 10:15	itoy	8/3/2017 14:38	asriram	7
8	1.4	#7	Functional	Cancel Switch Detection	If the Set switch is pressed, the value of reqDrv should be set to reqMode.Set.	Enumerated Constant2 Switch1			2/27/2017 10:15	itoy	2/2/2018 14:34	itoy	8
9	1.5	#8	Functional	Set Switch Detection	If the Enable switch is pressed, the value of reqDrv should be set to reqMode.Cruise.	Enumerated Constant1 Switch3	Enable button		2/27/2017 10:15	itoy	8/3/2017 14:39	asriram	9
10	1.6	#9	Functional	Enable Switch Detection	If the Resume switch is pressed, the value of reqDrv should be set to reqMode.Resume.	Enumerated Constant3			2/27/2017 10:15	itoy	8/3/2017 14:40	asriram	10
11	1.7	#10	Functional	Resume Switch Detection									

Clean Up

Clear the open requirement sets and links without saving changes and close the open models and tests without saving changes.

```
slreq.clear;
bdclose all;
```

See Also

Functions

```
slreq.export | slreq.getOutgoingLinkTypeLabel | slreq.getIncomingLinkTypeLabel |
writecell | getIncomingTypeLabel | getOutgoingTypeLabel | getSourceLabel |
getDestinationLabel | getDescriptionAsText | getRationaleAsText
```

Classes

```
slreq.Link | slreq.LinkSet | slreq.Requirement | slreq.ReqSet
```

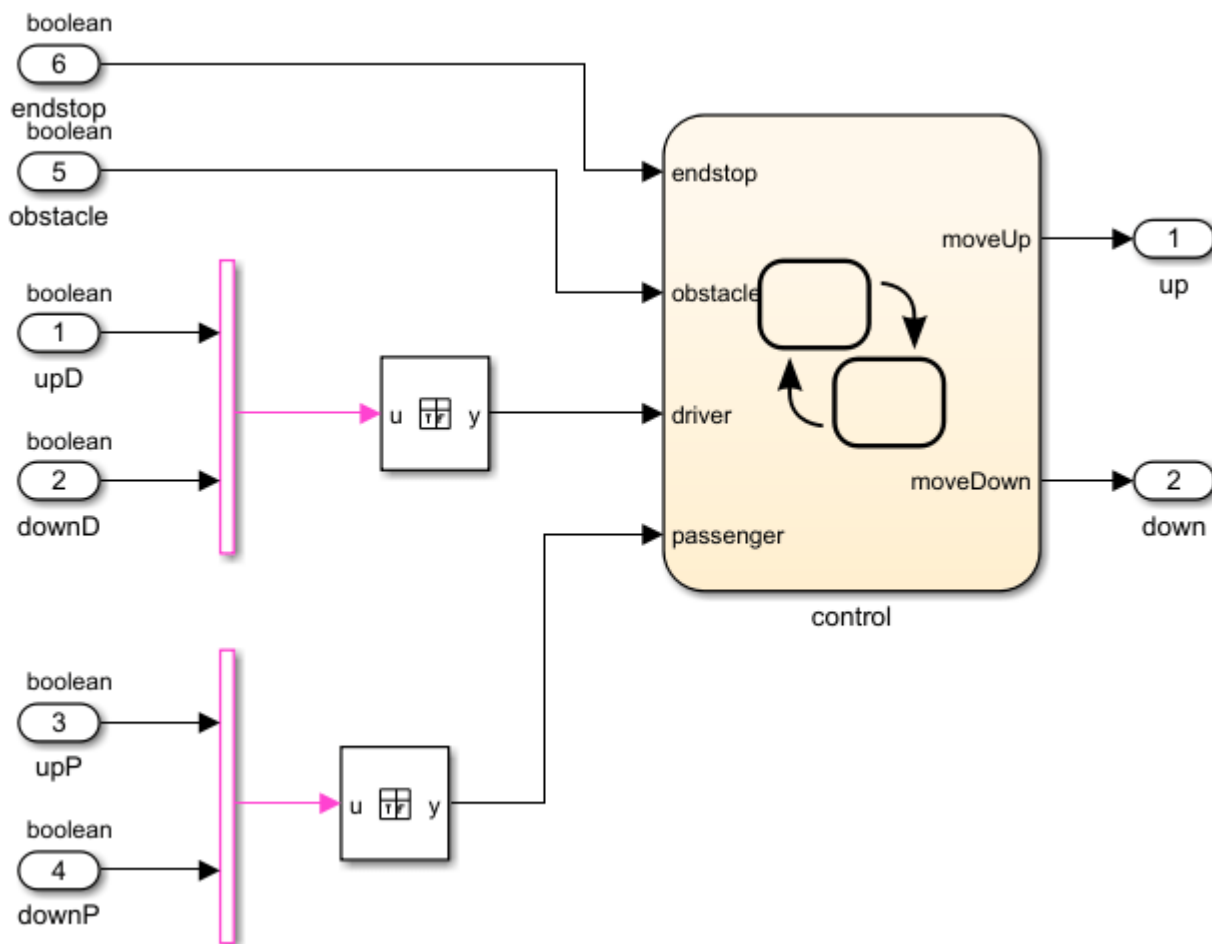
Related Examples

- “Export Requirements to ReqIF Files” on page 1-61

Use Command-Line API to Document Simulink Model in Requirements Editor

You can use the Requirements Toolbox™ programmatic interface to import and link requirements in MATLAB® or Simulink®. In this tutorial, you follow these steps to create, import, and resolve requirements for a model of a power window controller system for a car:

- 1 “Capture Model as Requirements and Create Links” on page 1-138: Capture the Simulink model design as requirements in the **Requirements Editor**, then link the requirements and model elements.
- 2 “Import Requirements and Reuse Existing Links” on page 1-143: Import multiple requirement sets from Microsoft® Word and reuse existing links when you migrate the source of truth from one requirement set to another requirement set.
- 3 “Programmatically Repair Broken Links” on page 1-145: Resolve broken links when the ID of the link source no longer exists.



See Also

Apps

Requirements Editor

Functions

add | slreq.createLink | slreq.import

Related Examples

- “Use Command-Line API to Update or Repair Requirements Links” on page 3-61

More About

- “Create and Store Links” on page 3-31
- “Import Requirements from Microsoft Office Documents” on page 1-12
- “Load and Resolve Links” on page 3-39

Capture Model as Requirements and Create Links

In this step, you use the Requirements Toolbox™ programmatic interface to capture elements from your Simulink® model into a set of proxy `slreq.Requirement` objects that have the same hierarchy as your model. You can then use the **Description** and **Rationale** properties of the proxy requirement objects to document each element of the design. The example also shows how to create links between the model elements and the proxy requirement objects for traceability.

Open the Models

Open the `slvndemo_powerwindow_vs` and `slvndemo_powerwindowController` models. The `slvndemo_powerwindow_vs` model is a specification model that verifies the functional properties of the design in `slvndemo_powerwindowController`.

```
open_system("slvndemo_powerwindow_vs");
open_system("slvndemo_powerwindowController");
```

Capture Model as Proxy Requirements

Use the Requirements Toolbox APIs to capture the Simulink models as proxy `slreq.Requirement` objects in the **Requirements Editor** for the `slvndemo_powerwindow_vs` and `slvndemo_powerwindowController` models.

First, create an array with the model names to make it easy to make changes to both models.

```
models = ["slvndemo_powerwindow_vs", "slvndemo_powerwindowController"];
```

For each model:

- Create a requirement set with the same name as the model by using `slreq.new`.
- Add an `slreq.Requirement` object that corresponds to the top-level model by using `add`.
- Open the model.
- Use `rmi` to get properties for each model element, including the handle, the index of the parent of the model element, an indicator that shows if the element belongs to a Stateflow® chart, and the SID.

```
for modelIndex = 1:numel(models)
    modelName = models(modelIndex);
    reqSetName = modelName+".slreqx";
    myReqSet = slreq.new(reqSetName);
    topProxy = add(myReqSet, Id=modelName, Summary=modelName+" Description");
    open_system(modelName);
    [elementHandles, parentIndex, isSf, SID] = rmi("getObjectsInModel", modelName);
```

For each element in the models, get the name and type of the element by using the `getNameAndType` function. To view the code for this function, see [Helper Functions](#) on page 1-141. Add an `slreq.Requirement` object under the `slreq.Requirement` object of the parent of the model element in the model hierarchy. Add a summary to the `slreq.Requirement` object based on the name and type.

```
createdProxies(1) = topProxy;
for elementIndex = 2:numel(elementHandles)
    [name, type] = getNameAndType(isSf(elementIndex), elementHandles(elementIndex), modelName, S
    if ~isempty(name) && ~isempty(type)
```

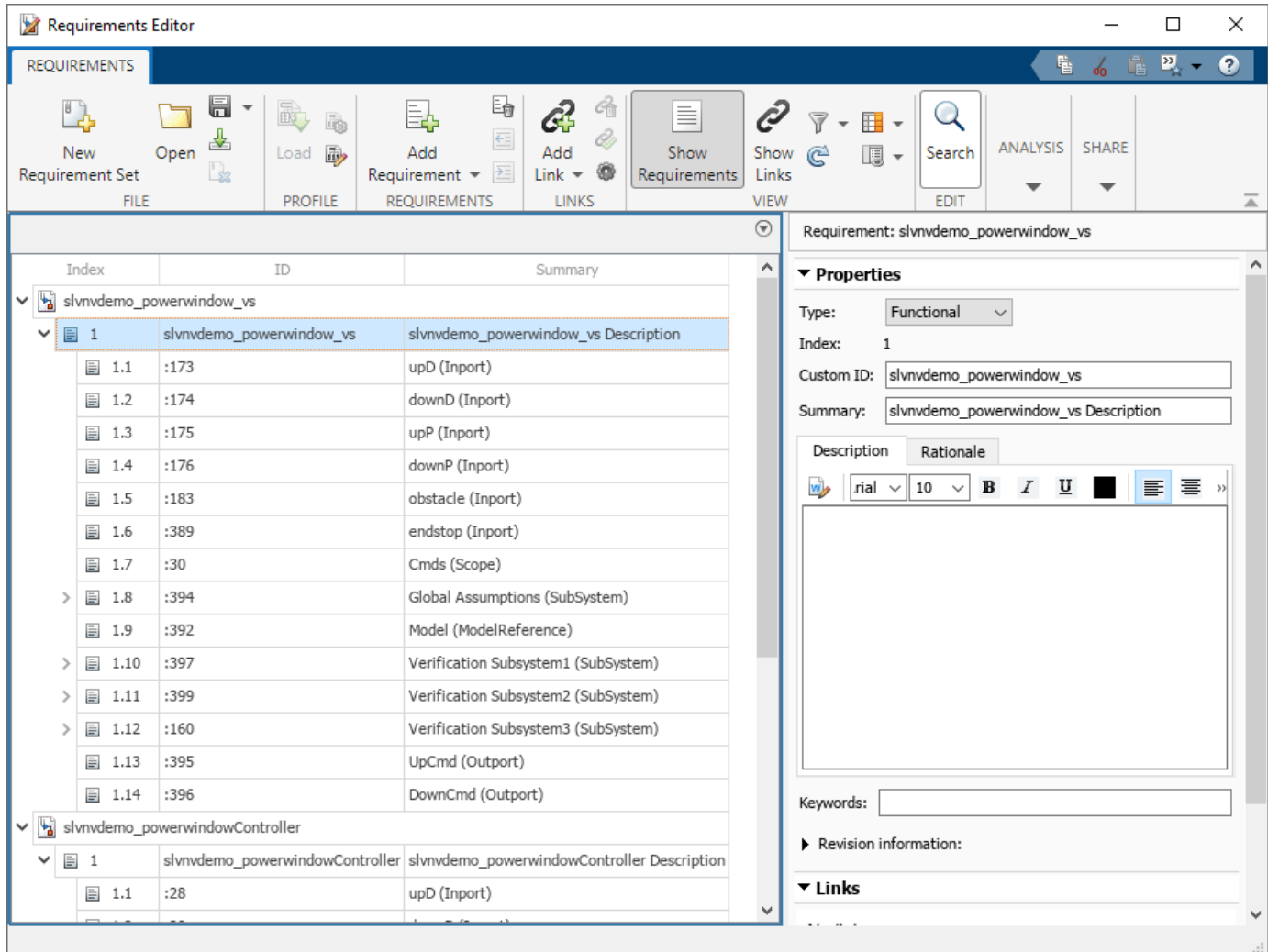
```

        parentProxy = createdProxies(parentIndex(elementIndex));
        createdProxies(elementIndex) = add(parentProxy, Id=SID{elementIndex}, Summary=name+" (
    end
end
save(myReqSet);
end

```

Open the **Requirements Editor** to view the requirements.

slreq.editor



Link Requirements and Model Elements

Create links in both the `slvndemo_powerwindow_vs` and `slvndemo_powerwindowController` requirement sets to navigate between the `slreq.Requirement` object and the corresponding model element.

For each model and its associated requirement set, create an array that contains the requirements.

```

for modelIndex = 1:numel(models)
    modelName = models(modelIndex);

```

```
reqSetName = slreq.find(Type="ReqSet",Name=modelName);  
proxyRequirements = find(reqSetName,Type="Requirement");
```

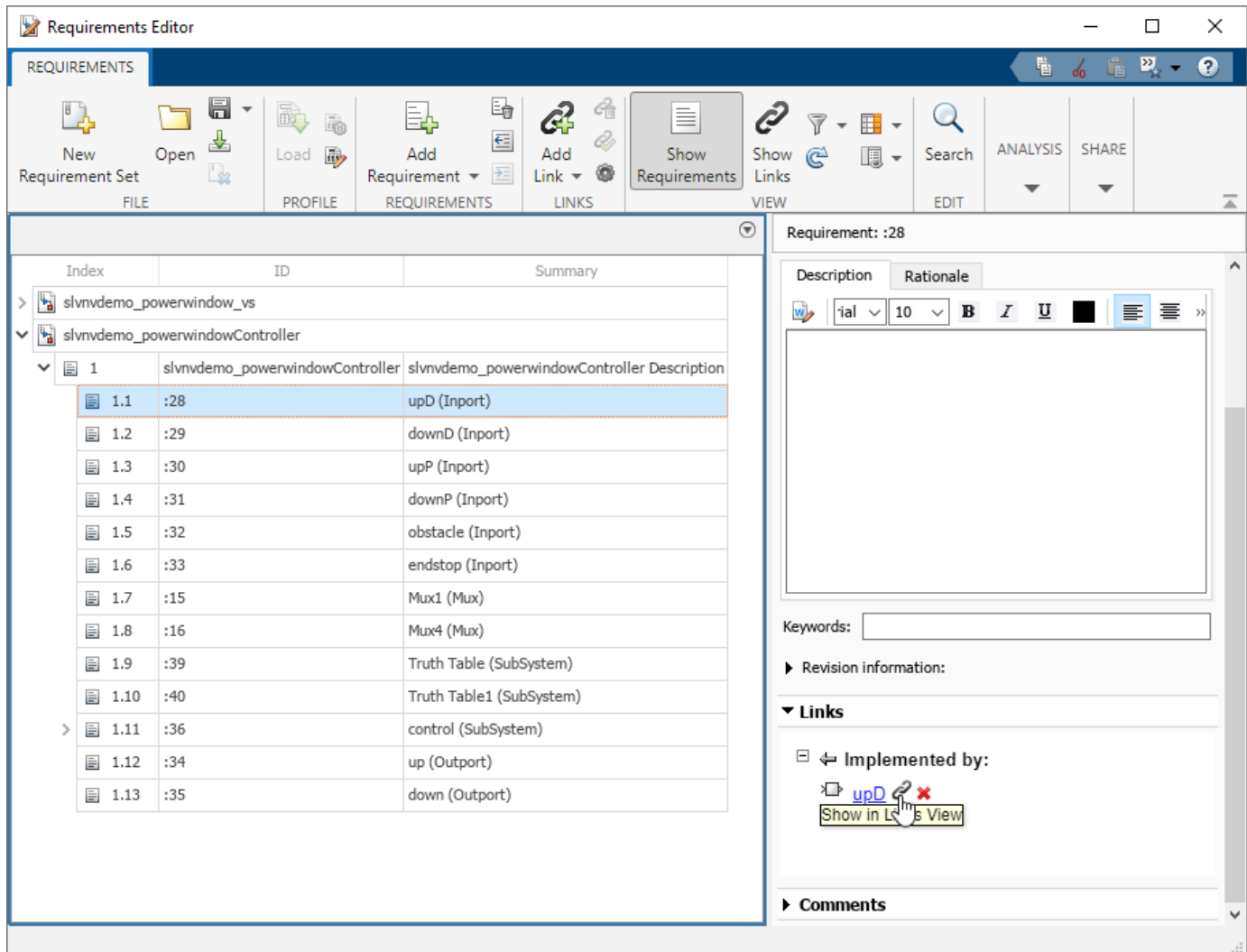
For each requirement in the requirement set other than the top-level requirement:

- Get the SID of the associated model element and then get a handle to the model element.
- Create a link between the model element and the requirement.
- Add a description to the link that includes the index of the linked requirement.
- Record the number of created links.

```
counter = 0;  
for reqIdx = 1:numel(proxyRequirements)  
    myProxy = proxyRequirements(reqIdx);  
    if ~contains(myProxy.Summary,modelName)  
        sid = strcat(modelName,myProxy.Id);  
        linkSrc = Simulink.ID.getHandle(sid);  
        newLink = slreq.createLink(linkSrc,myProxy);  
        newLink.Description = strcat("Link between requirement ",myProxy.Index," and model element ");  
        counter = counter + 1;  
    end  
end  
disp(strcat("Created ",num2str(counter)," links for ",modelName));  
end
```

```
Created 116 links for slvndemo_powerwindow_vs  
Created 66 links for slvndemo_powerwindowController
```

View the new links in the **Requirements Editor** by clicking a requirement and, in the right pane, scrolling to the **Links** section.



Save the link sets.

```
lsArray = slreq.find(Type="LinkSet");
save(lsArray(1));
save(lsArray(2));
```

Helper Functions

This function returns the model element name and type. This function skips Stateflow object types other than Stateflow.State and Stateflow.Transition.

```
function [name,type] = getNameAndType(isSf,elementHandle,modelName,SID)

if ~isSf
    name = get_param(elementHandle,"Name");
    type = get_param(elementHandle,"BlockType");
else
    sfObj = Simulink.ID.getHandle(strcat(modelName,SID));
    sfType = class(sfObj);
    switch sfType
        case 'Stateflow.State'
```

```
        name = sf('get',elementHandle, '.name');
        type = strrep(class(sfObj), "Stateflow.", "");
    case 'Stateflow.Transition'
        name = sf('get',elementHandle, '.labelString');
        type = strrep(class(sfObj), "Stateflow.", "");
    otherwise
        warning('Stateflow object of type %s was skipped.', class(sfObj));
        name = "";
        type = '';
    end
end
end
```

See Also

Apps

Requirements Editor

Functions

add | slreq.createLink

More About

- “Create and Store Links” on page 3-31

Import Requirements and Reuse Existing Links

In this step, you use the Requirements Toolbox™ programmatic interface to import requirements from a Microsoft® Word document and create links to the Simulink® model. Then, you import another set of requirements and reuse the existing links by migrating the link destinations to the most recently imported requirements.

Import the Requirements

Open the `slvndemo_powerwindowController` model, which also opens its associated link set.

```
open_system("slvndemo_powerwindowController.slx")
```

The `PowerWindowSpecification` Word document contains requirements for the power window controller design. Import the requirements as referenced requirements so that you can continue to manage them in the Microsoft Word document and update the imported requirement set when you make changes in the Word document.

```
[~,~,refReqSet] = slreq.import("PowerWindowSpecification.docx",ReqSet="ReadOnlyImport.slreqx");
```

Find the referenced requirement with the `Index` property set to 1.3.5.

```
ref = find(refReqSet,Index="1.3.5");
```

Create a link from the `Truth Table` block to the referenced requirement.

```
designModel = "slvndemo_powerwindowController";
truthTable = designModel+ "/Truth Table";
link1 = slreq.createLink(truthTable,ref);
```

Create a link from the `Truth Table1` block to the referenced requirement.

```
truthTable1 = designModel+"/Truth Table1";
link2 = slreq.createLink(truthTable1,ref);
```

Navigate to the source of the first link in the Simulink model.

```
slreq.show(link1.source);
```

Navigate to the destination of the first link in the **Requirements Editor**.

```
slreq.show(link1.destination);
```

Re-Import Requirements and Reuse Existing Links

Suppose that you decide to migrate the source of truth of the requirements from Microsoft Word to Requirements Toolbox. Re-import requirements from the Microsoft Word document as editable requirements.

```
[~,~,editableReqSet] = slreq.import("PowerWindowSpecification.docx",ReqSet="EditableImport.slreqx");
```

Get a handle to the link set that contains links between the Simulink model and the referenced requirements. Get the links from the link set.

```
ls = slreq.find(Type="LinkSet");
links = getLinks(ls);
```

For each link in the link set, check if the link destination is a requirement in the referenced requirement set, then get a handle to the referenced requirement by using `slreq.structToObj`. Get the ID of the referenced requirement, then find a requirement in the editable requirement set with the same ID. Use `setDestination` to redirect the link destination to point to the editable requirement.

```
updatedLinks = 0;
for linkIdx = 1:numel(links)
    link = links(linkIdx);
    if strcmp(link.destination.reqSet, strcat(refReqSet.Name, ".slreqx"))
        ref = slreq.structToObj(link.destination);
        ID = ref.Id;
        req = find(editableReqSet, Id=ID);
        link.setDestination(req);
        updatedLinks = updatedLinks + 1;
    end
end
```

Display the number of updated links.

```
disp("Updated "+num2str(updatedLinks)+" links from "+designModel);
```

```
Updated 2 links from slvndemo_powerwindowController
```

See Also

Apps

Requirements Editor

Functions

`slreq.createLink` | `slreq.import` | `setDestination`

More About

- “Create and Store Links” on page 3-31
- “Import Requirements from Microsoft Office Documents” on page 1-12
- “Load and Resolve Links” on page 3-39

Programmatically Repair Broken Links

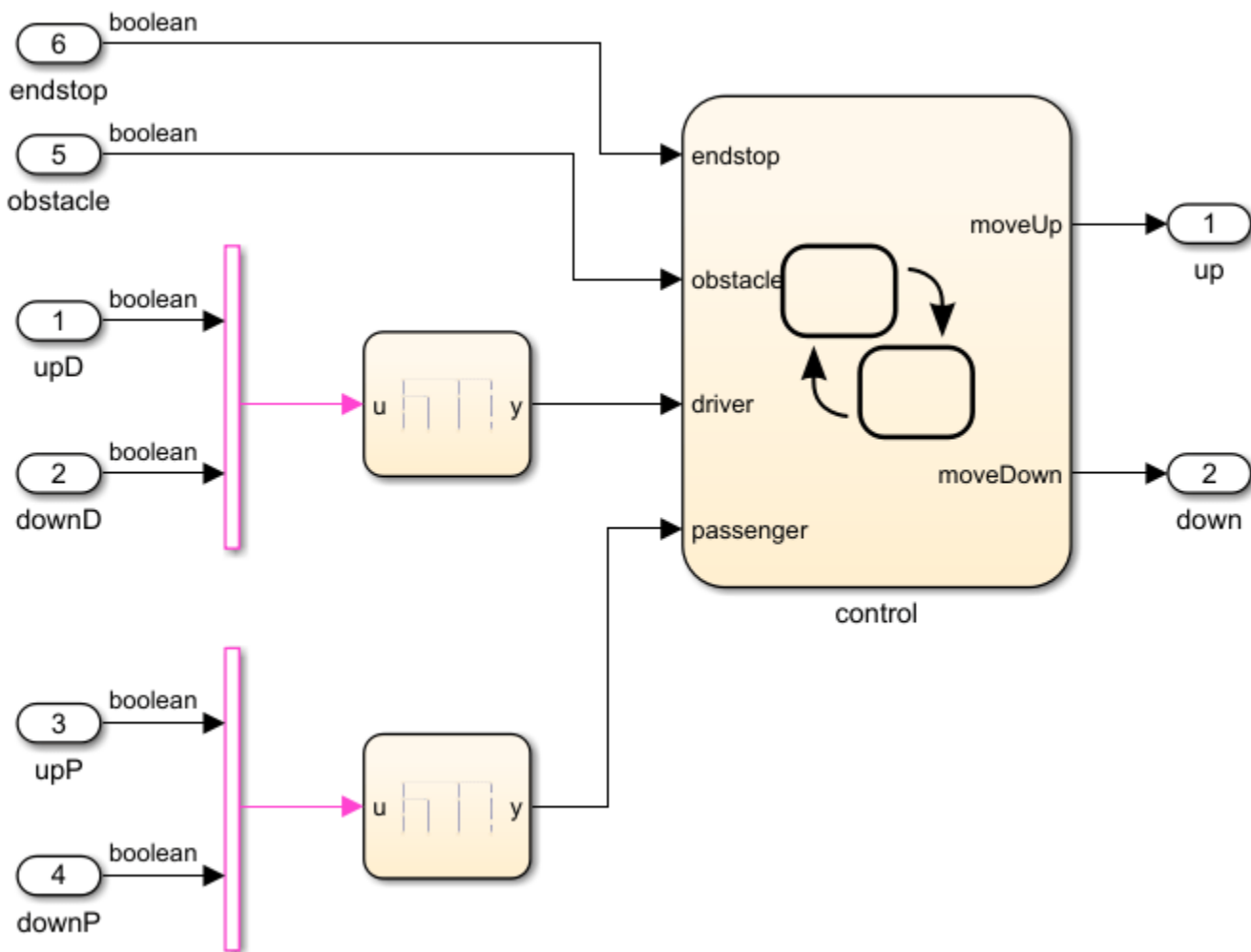
In this step, you use the Requirements Toolbox™ programmatic interface to repair broken links when the specified ID of the link sources no longer exists.

Examine Broken Links

Open the `slvndemo_powerwindowController_stateflow` model, which also loads its associated link set.

```
model = "slvndemo_powerwindowController_stateflow";
open_system(model)
```

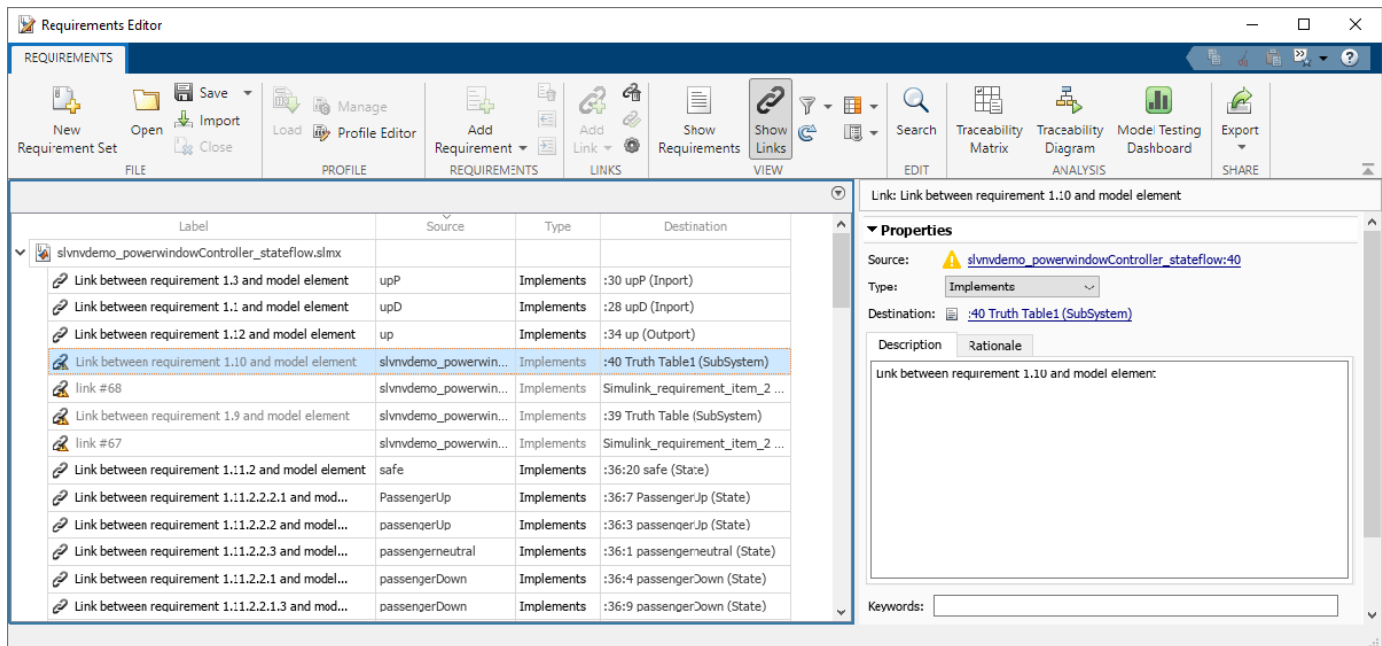
The model is similar to the `slvndemo_powerwindowController` model, but it implements the Truth Table and Truth Table1 blocks by using Stateflow charts that have logic.



Open the **Requirements Editor**.

```
slreq.editor
```

Click **Show Links** to view the links. Because the Truth Table blocks are missing, the four links in the link set that point to those blocks are broken.



Repair Broken Links

Get a handle to the `slvndemo_powerwindowController_stateflow` link set. Assign the links from the link set to an array.

```
ls = slreq.find(Type="LinkSet");
linksArray = getLinks(ls);
```

Check if the link source is resolved for each link in the link set by creating an array that contains the broken links. If the link source is unresolved, add it to the `brokenLinks` array.

```
j = 1;
for i = 1:numel(linksArray)
    link = linksArray(i);
    if ~link.isResolvedSource
        brokenLinks(j) = link;
        j = j+1;
    end
end
```

Open the `slvndemo_powerwindowController` model.

```
oldModel = "slvndemo_powerwindowController";
open_system(oldModel)
```

For each broken link, use the link source ID to get the model element name from the `slvndemo_powerwindowController` model.

```
repairedLinks = 0;
for i = 1:numel(brokenLinks)
    link = brokenLinks(i);
```

```

elementSID = link.source.id;
elementHandle = Simulink.ID.getHandle(strcat(oldModel,elementSID));
elementPath = getfullname(elementHandle);
elementName = strrep(convertCharsToStrings(elementPath),oldModel+"/","");

```

In the `slvndemo_powerwindowController_stateflow` model, the `Truth Table` block was replaced with the Stateflow chart `Chart`, while the `Truth Table1` block was replaced with the Stateflow chart `Chart1`. Use `setSource` to change the source to the corresponding Stateflow chart.

```

switch elementName
    case "Truth Table"
        newPath = model+"/Chart";
    case "Truth Table1"
        newPath = model+"/Chart1";
end
newHandle = getSimulinkBlockHandle(newPath);
setSource(link,newHandle);
repairedLinks = repairedLinks + 1;
end

```

Display the number of repaired links.

```
disp("Fixed "+num2str(repairedLinks)+" links in "+model+".slmx");
```

```
Fixed 4 links in slvndemo_powerwindowController_stateflow.slmx
```

See Also

Apps

Requirements Editor

Functions

`isResolvedSource` | `setSource`

More About

- “Load and Resolve Links” on page 3-39

Requirements Formalization

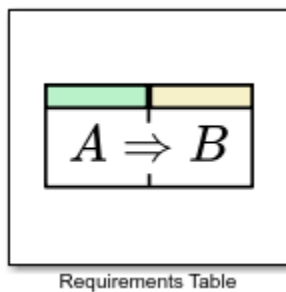
Use a Requirements Table Block to Create Formal Requirements

Use the Requirements Table block to model formal requirements. You establish formal requirements by using Boolean formulas that check whether the required model behavior occurs during simulation. When you create a requirement in a Requirements Table block, you also create an equivalent requirement in the **Requirements Editor**. See “Configure Properties of Formal Requirements” on page 2-27.

Define Formal Requirements

Formal requirements are a set of unambiguous specifications that you express mathematically and execute through simulation. Each formal requirement executes if at least one condition is true for a specified duration. When the requirement executes, the block verifies specified simulation behavior and produces additional simulation data.

To define formal requirements, add a Requirements Table block to your model.



Opening a new Requirements Table block displays a blank requirement. By default, the block opens on the **Requirements** tab. Each row of the table represents a requirement. The columns specify information used to evaluate each requirement. The columns are:

- **Index** — A unique identifier for each requirement. This column is read-only.
- **Summary** — A text-based summary of the requirement. Entries in this column do not affect the block behavior. This column is optional.
- **Precondition** — A Boolean expression that must be true for a specified duration before evaluating the rest of the requirement.
- **Duration** — The time, in seconds, during which the precondition must be true before evaluating the rest of the requirement. This column is optional. If left blank, the requirement does not have a duration. You must specify a precondition to use this column. See “Using the Duration Column” on page 2-33.
- **Postcondition** — A Boolean expression that must be true if the associated precondition is true for the specified duration. If you do not include a precondition, the block checks the postcondition at each time step.
- **Action** — An action performed by the block if the associated precondition is true for the specified duration. You can use actions to define a block output or call a function. If you do not include a precondition, the block executes the action at each time step.

Requirements		Assumptions			
Index	Summary	Precondition	Duration	Postcondition	Action
1	Requirement 1				

You can create multiple requirements, preconditions, postconditions, and actions. To manage columns and requirements, use options available in the **Table** tab. You can also access additional options with the context menu by right-clicking the requirement index or column header cell.

You can also specify assumptions that define physical constraints of the system in the **Assumptions** tab. See “Add Assumptions to Requirements” on page 2-10.

Create Expressions for Formal Requirements

You define preconditions, postconditions, and actions with expressions. Use Boolean expressions for preconditions and postconditions, and use a single equal sign = to define actions. The expressions in each precondition and postcondition cell must evaluate to a scalar logical.

Use data in your expressions to manage information, such as signal values, workspace variables, or constant values. Expressions can also include numerical values and functions. For example, the precondition $u \geq 0$ checks if the data u is greater than or equal to 0 , and the action $y = 0$ sets the data y equal to 0 .

You can define preconditions, postconditions, and actions by explicitly listing each full expression in the requirement cells or by listing the left-hand side of the expression in the column header and defining the right-hand side of the expression in the requirement. To write Boolean expressions that use headers in **Precondition** and **Postcondition** columns, use the Boolean relational operator of interest, leave the left side blank, and enter the value of interest. To write actions that use headers, enter the value of the action into the cell.

For example, in this table the first postcondition specifies the data $y1$ in the header, the second specifies $y2$ in the requirement cell, and the action sets $y3$ equal to a vector of values.

Requirements		Assumptions			
Index	Summary	Precondition	Postcondition		Action
			$y1$		$y3$
1	Requirement 1	$u > 0$	≥ 0	$y2 \geq 0$	[1,2,3]

You can also check if a scalar header equals a single value, multiple values, or a range of values in the **Precondition** and **Postcondition** columns. Use these syntaxes to specify the checked values:

Syntax	Description
value	Checks if the header data equals value.
value1,value2,value3,...,valueN	Checks if the header data equals one of the comma-separated values.
[value1,value2]	Checks if the header data is greater than or equal to value1 and is less than or equal to value2.
(value1,value2)	Checks if the header data is greater than value1 and is less than value2.
[value1,value2)	Checks if the header data is greater than or equal to value1 and is less than value2.
(value1,value2]	Checks if the header data is greater than value1 and is less than or equal to value2.

In this table, the precondition checks if u is equal to 1, 2, or 3. The postcondition evaluates the data y1 and y2 by using the same Boolean expression but with different notation. The first postcondition uses header data with interval notation and the second postcondition uses explicit Boolean operators without header data.

Requirements		Assumptions		
Index	Summary	Precondition	Postcondition	
		u	y1	
1	Requirement 1	1,2,3	[0.3, 0.5]	y2 >= 0.3 && y2 <= 0.5

To evaluate header data against multiple intervals, combine them with the logical OR operator. For example, [0.3, 0.5] || (0.1, 0.2) is valid. In addition to real numbers, you can also use Inf in an interval to evaluate a bound that goes to infinity. Intervals including Inf must be preceded or followed by a parenthesis. Intervals including Inf can be simplified if the cell contains only one interval. For example, (0.3, Inf) is equivalent to > 0.3.

To specify that the header data should not equal an expression, use the logical NOT operator by entering ~= or ~ to the left of the cell values. You can apply this operator to single values, comma-separated values, or intervals. For example, if you did not want the header data u to fall within the interval [0.3, 0.5], enter ~[0.3, 0.5] or ~= [0.3, 0.5]. For comma-separated values, use only ~.

If your requirement cells contain a lot of content, the cell size can obstruct the expression. To view all of the content, you can adjust the size of the cell or add a new line. To add a new line, finish the first line with ... and then press **Alt+Enter**.

Use the X Placeholder

When you enter expressions in a header, do not reuse the header expression in the cells of that column. For example, if a header contains the data u, do not enter the expression u >= 0.5 && u <= 0.3 in an associated column cell. Instead, use X as a placeholder for the header. For example, the expression X >= 0.5 && X <= 0.3 in a precondition with u in the header is equivalent to u >= 0.5 && u <= 0.3 without u in the header.

Requirements		Assumptions		
Index	Summary	Precondition		Action
		u		
1	Requirement 1	X >= 0.5 && X <= 0.3		

Using the X placeholder can improve readability when the header is long.

Check Array Data in Preconditions and Postconditions

If you use array data in preconditions and postconditions, the block checks if each element of the data satisfies the logical expression and produces a logical array. To check if a precondition or postcondition satisfies an array of values, use functions that evaluate to a scalar logical, such as the `isequal` or `all` functions. For example, if `y1` is a 1-by-4 array, and you want to check that it equals `[1 2 3 4]`, enter `isequal(y1,[1 2 3 4])` or `all(y1 == [1 2 3 4])` into the cell.

Use Functions in Requirements Table Blocks

You can use functions and operators in Requirements Table blocks in the requirement cells or column headers. Enter data as arguments to the function or operator like you do with variables in MATLAB. In this table, the cell `u1 + u2 == 2` and the cell below the header `u3 + u4` both check if the sum of the two input data equals 2.

Requirements		Assumptions		
Index	Summary	Precondition		Action
			u3 + u4	
1	Requirement 1	u1 + u2 == 2	2	

The Requirements Table block does not support functions in headers that write to global variables or an internal state. For example, you cannot use the `rand` function in a header because `rand` writes to an internal state each time it generates a number.

Define Block Data

In order to use data in the block, you must enter them in your requirements and define them in the **Symbols** pane. To open the **Symbols** pane, open a Requirements Table block. In the **Modeling** tab, in the **Design Data** section, click **Symbols Pane**.

In the **Symbols** pane, the icon in the **Type** column indicates the scope of your data. You can set the data scope to be **Input**, **Output**, **Local**, **Constant**, or **Parameter**. You can also change the name, the initial data value, and the port number. To modify additional properties, right-click the data name and select **Inspect** to open the Property Inspector. For more information, see “Define Data in Requirements Table Blocks” on page 2-53.

TYPE	NAME	VALUE	PORT
	u		1
	y		1

As you edit your requirements, the Requirements Table block detects undefined data and marks them in the **Symbols** pane with the Undefined symbol icon . Unresolved data can cause errors during compilation. To resolve data, click the Resolve undefined symbols button .

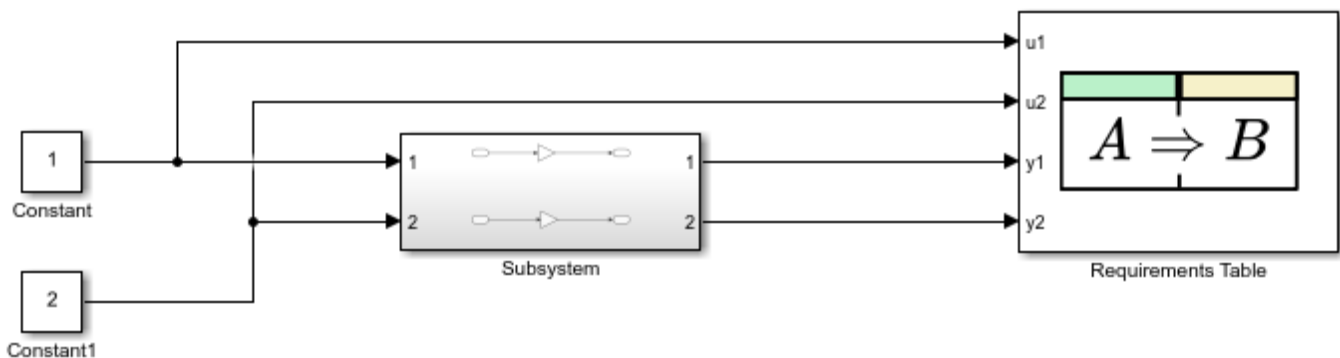
Observe or Generate Model Behavior

You can use the Requirements Table block to check model requirements by observing the model behavior, or by generating behavior that can be compared to the model behavior.

Observe Model Behavior

You can use the Requirements Table block to observe the model behavior without executing actions. If the model behavior does not satisfy a precondition, the block does not check the associated postcondition. If the model behavior does not satisfy the postcondition for a requirement when the precondition is satisfied, the Requirements Table block produces a warning in the Diagnostic Viewer. To use the Requirements Table block to observe requirements, use the preconditions to specify model input behavior, and use the postconditions to specify model output behavior. To differentiate between the model inputs and outputs, enable the **Treat as design model output for analysis** property in the Property Inspector for the data specified in the postconditions.

This model contains a Requirements Table block that tests a basic subsystem that has two inputs and outputs two values. The Requirements Table block checks whether the inputs and outputs of the subsystem meet the specified requirements by using logical definitions for the preconditions and postconditions.



Copyright 2021 The MathWorks, Inc.

Open the Requirements Table block to see the requirements specified for the inputs and outputs of the subsystem. The requirements specify logical constraints on the subsystem outputs for each subsystem input. The Requirements Table block captures the subsystem inputs with the data `u1` and `u2`, and captures the subsystem outputs with the data `y1` and `y2`. The block defines the data as input data, which allows the block to capture the subsystem signals as block inputs.

Requirements		Assumptions	
Index	Summary	Precondition	Postcondition
1	Requirement 1	$u1 > 0$	$y1 > 0$
2	Requirement 2	$u2 > 0$	$y2 > 0$

When you run the model, the Requirements Table block checks if the Subsystem block inputs satisfy the preconditions. If the preconditions are met, the Requirements Table block checks if the Subsystem block outputs satisfy the postconditions.

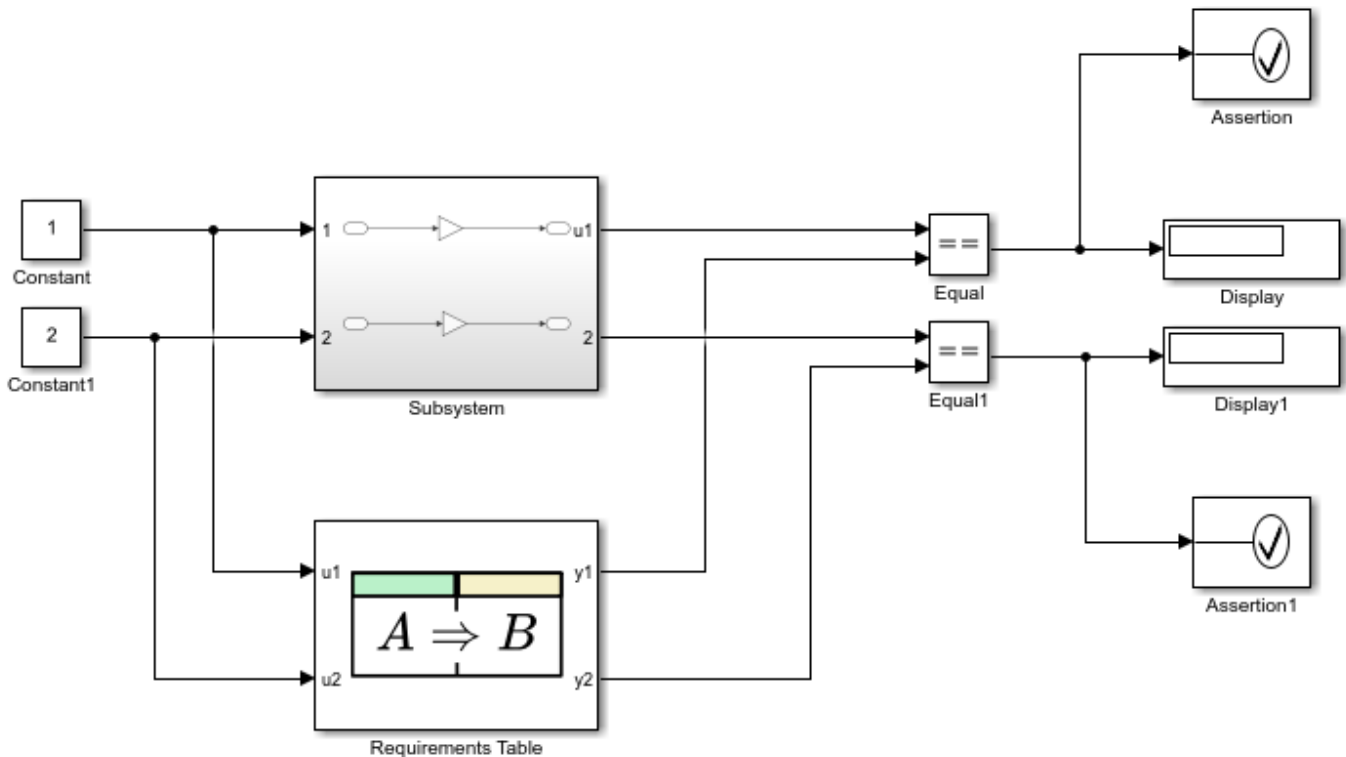
This example Subsystem block satisfies the preconditions and postconditions. To generate a warning, set the postcondition of the first requirement to $y1 < 0$ and run the simulation again. The **Diagnostic Viewer** displays a warning message.

```
Test verification failed at t = 0.
Requirement '1': {
  reqtable_postcondition((y1 < 0), 'R:1');
}
Component: Requirements Table | Category: Runtime warning
```

Generate Model Behavior

You can also use the Requirements Table block to execute actions when the model meets corresponding preconditions. You specify the action for each requirement by using the **Action** column.

This model contains a Requirements Table block that tests a subsystem that takes two inputs and creates two outputs. The Requirements Table block takes the subsystem inputs and outputs a set of values if the preconditions are met. Each Requirements Table block output corresponds to one of the outputs from the subsystem. If the output of the subsystem does not equal the corresponding output signal from the Requirements Table block, the Assertion blocks stop the simulation and produce an error.



Copyright 2021 The MathWorks, Inc.

Open the Requirements Table block to see the two requirements specified for the inputs of the subsystem. The requirements specify logical constraints on the inputs. The block defines the data $u1$ and $u2$ as input data, which allows the block to capture the subsystem inputs as block inputs. The block defines the data $y1$ and $y2$ as output data.

Requirements		Assumptions	
Index	Summary	Precondition	Action
1	Requirement 1	$u1 > 0$	$y1 = 2*u1$
2	Requirement 2	$u2 > 0$	$y2 = 0.5*u2$

When you run the model, the Requirements Table block checks whether the Subsystem block inputs meets the preconditions, and assigns the output signals to the values specified in the **Action** column. The model then compares the outputs from the Subsystem block with the output signals from the Requirements Table block. In this example, the outputs are equal and the assertions pass.

See Also

Requirements Table

Related Examples

- “Specify Requirements Table Block Properties” on page 2-50
- “Add Assumptions to Requirements” on page 2-10
- “Define Data in Requirements Table Blocks” on page 2-53
- “Set Data Types in Requirements Table Blocks” on page 2-59
- “Establish Hierarchy in Requirements Table Blocks” on page 2-45
- “Identify Inconsistent and Incomplete Formal Requirement Sets” on page 2-17

Add Assumptions to Requirements

While specifying formal requirements with a Requirements Table block, you may have a model where physical or mathematical limitations prevent data from being certain values. You can constrain the values of the input data checked in your requirements by creating assumptions in the **Assumptions** tab.

The Requirements Table block evaluates assumptions before requirements. Assumptions include a **Precondition** and **Postcondition** column to define preconditions and postconditions. New Requirements Table blocks contain one empty assumption.

Requirements		Assumptions	
Index	Summary	Precondition	Postcondition
1	Assumption 1		

If the precondition of an assumption is met, the postcondition enforces a constraint on data used in the requirements. If the precondition of an assumption is not met, the block ignores the constraint in the postcondition. Unlike postconditions in the **Requirements** tab, the postconditions in the **Assumptions** tab must be satisfied, otherwise the block returns an error. If you want to enforce an assumption regardless of input values, fill in the postcondition and leave the precondition blank.

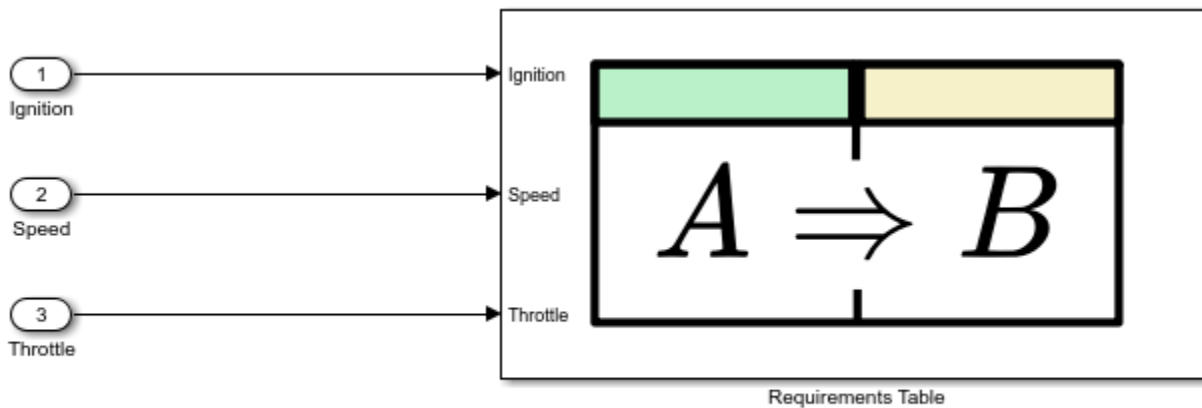
In the table, you can use the context menu to do the same actions as in the **Requirements** tab, as well as add, modify, or delete assumptions.

- Add assumptions by right-clicking a cell in the **Index** column and clicking **Add Assumption After**.
- Delete assumptions by right-clicking the row index and selecting **Delete Assumption**.
- Clear the assumptions by right-clicking the row index and selecting **Clear Assumption**.

You define assumptions by using the same syntax used for requirement preconditions and postconditions. See “Create Expressions for Formal Requirements” on page 2-3. Assumptions do not have column header cells, and each assumption has only one precondition and postcondition.

Use Assumptions in an Example Model

This example uses assumptions to evaluate the requirements for a simple car ignition system to establish the physical limitations of the model. By including assumptions, the block constrains the possible values of the data used in the requirements.



Copyright 2021 The MathWorks, Inc.

Open the Requirements Table block. The block captures the signals with three input ports, **Throttle**, **Ignition**, and **Speed**. The block confirms that the **Speed** signal satisfies conditions for a provided range of the **Throttle** and **Ignition** values by defining data with the same names.

Requirements		Assumptions		
Index	Summary	Precondition		Postcondition
		Ignition	Throttle	Speed
1	When ignition is OFF and there is no throttle then speed shall be zero	false	0	0
2	When ignition is on then the speed is determined as follows:	true		
2.1	When there is throttle then the speed is non zero		> 0	> 0
2.2	When there is no throttle then the speed is non negative		0	>= 0

Click the **Assumptions** tab. The table has two assumptions that constrain the input **Throttle** based on the physical limitations of the system:

- 1 If the car ignition is off, then the throttle must also be off.
- 2 The throttle cannot be negative.

The first assumption enforces the first limitation, and the second assumption enforces the second limitation.

Requirements		Assumptions	
Index	Summary	Precondition	Postcondition
1	Switch is off then throttle is zero	Ignition == false	Throttle == 0
2	Throttle is engaged		Throttle >= 0

See Also

Requirements Table

Related Examples

- “Use a Requirements Table Block to Create Formal Requirements” on page 2-2
- “Identify Inconsistent and Incomplete Formal Requirement Sets” on page 2-17
- “Establish Hierarchy in Requirements Table Blocks” on page 2-45

Create Requirements Table Blocks Programmatically

In this section...

“Create a Requirements Table Block” on page 2-13

“Specify Requirements Table Block Name and Column Headers” on page 2-13

“Define Data” on page 2-14

“Add and Modify Requirements” on page 2-14

“Manage Requirements as slreq.Requirement Objects” on page 2-15

You can create and manage Requirements Table blocks programmatically. In this example, you programmatically create a Requirements Table block in a new model, define block data, add requirements, and access requirement properties. For information on how to create and manage Requirements Table blocks with the graphical interface, see “Use a Requirements Table Block to Create Formal Requirements” on page 2-2.

Create a Requirements Table Block

Each Requirements Table block you add has an associated `RequirementsTable` object. Create a Requirements Table block in a new model named `myNewModel` by using the `slreq.modeling.create` function, and assign the block to a `RequirementsTable` object named `table`.

```
table = slreq.modeling.create("myNewModel");
```

If you have an existing model that you want to add a Requirements Table block to, use the `add_block` function and retrieve the `RequirementsTable` object by using `slreq.modeling.find` with the name of the block.

```
add_block("reqmanage/Requirements Table", ...
strcat(gcs, "/Requirements Table"));
table = slreq.modeling.find(gcs);
```

Specify Requirements Table Block Name and Column Headers

After you add a Requirements Table block to a model, use dot notation to modify the `Name` and `RequirementHeader` properties. When you create a new Requirements Table block, the block uses the default name. Rename the block by modifying the `Name` property to `My Requirements`:

```
table.Name = "My Requirements";
```

You can add **Precondition**, **Postcondition**, and **Action** columns to the block by adjusting the `RequirementHeader` property. In this example, specify that the block contains two precondition columns and one postcondition column:

```
table.RequirementHeaders.Preconditions = ["input1", "input2"];
table.RequirementHeaders.Postconditions = "output";
```

You can hide column types that you do not use in the **Requirements** tab and the **Assumptions** tab with the `hideRequirementColumn` and `hideAssumptionColumn` functions. This example does not use actions or durations. Hide the **Action** and **Duration** columns with the `hideRequirementColumn` function:

```
hideRequirementColumn(table, "Actions");
hideRequirementColumn(table, "Duration");
```

Note To delete columns, use the Requirements Table block graphical interface.

Define Data

In addition to adding data to the preconditions and postcondition, you must resolve the data by defining it. Currently, you have not defined `input1`, `input2`, and `output`. To define the data programmatically, use the `addSymbol` function on the `RequirementsTable` object. `addSymbol` allows you to adjust some of the properties of the data, including the data name, the scope, data type, and size. If you want to set additional properties, use the **Symbols** pane and Property Inspector in the graphical interface. For more information, see “Define Data in Requirements Table Blocks” on page 2-53.

Define data named `input1`, `input2`, and `output` and specify them as input data. Because `output` is the only data used in the postconditions, you must enable the `isDesignOutput` property for it. See “`isDesignOutput`” and “Treat as design model output for analysis” on page 2-56. Specify `output` as a design model output.

```
addSymbol(table, Name="input1", Scope="Input");
addSymbol(table, Name="input2", Scope="Input");
addSymbol(table, Name="output", Scope="Input", isDesignOutput=1);
```

The Requirements Table block appears with one requirement, three columns, and the data in the column headers.

Requirements		Assumptions		
Index	Summary	Precondition		Postcondition
		input1	input2	output
1	Requirement 1			

To modify the data, retrieve the `Symbol` object array with the `findSymbol` function and use dot notation. See “Retrieve Data and Change It”.

Add and Modify Requirements

To add requirements to the block, use the `addRequirementRow` function. To add assumptions, use the `addAssumptionRow` function. Each function adds one row. To add multiple rows, you can either repeat the entry or use a `for` loop and adjust the properties after creating them.

Add a requirement to the block and specify the summary, preconditions, and postconditions.

```
addRequirementRow(table, Summary="Requirement 2", ...
Preconditions={'> 0', ''});
```

The first requirement is empty. Use dot notation to specify the properties of an existing requirement. Retrieve the requirements as RequirementRow objects from the table and modify the first requirement.

```
rrow = getRequirementRows(table);
rrow(1).Summary = "Requirement 1";
rrow(1).Preconditions = {'0', ''};
rrow(1).Postconditions = {'> 2'};
```

You can also add children to requirements or assumptions by using the addChild function. Add two children to the second requirement that specify the preconditions and postconditions.

```
addChild(rrow(2),Preconditions={'','> 0'},...
Postconditions={'> 5'});
addChild(rrow(2),Preconditions={'','<= 0'},...
Postconditions={'<= -1'});
```

To complete the requirement set, add a default row by using the addRequirementRow function and specifying the rowType as "default".

```
addRequirementRow(table,rowType="default",...
Postconditions={'< 1'});
```

After you enter the above commands, the **Requirements** tab lists the new requirements:

Requirements		Assumptions		
Index	Summary	Precondition		Postcondition
		input1	input2	output
1	Requirement 1	0		> 2
2	Requirement 2	> 0		
2.1			> 0	> 5
2.2			<= 0	<= -1
3	D	Else		< 1

If you close out of the block, you can reopen it with the open_system function:

```
open_system("myNewModel/My Requirements");
```

Manage Requirements as slreq.Requirement Objects

Adding a Requirements Table block to a model creates a slreq.ReqSet object for each block, and creating requirements creates corresponding slreq.Requirement objects. To retrieve the objects from the table discussed in this example, load the requirement set into Requirements Toolbox and retrieve the slreq.ReqSet object with the slreq.load function:

```
[~, reqSet] = slreq.load("myNewModel.slx");
```

You can also use `slreq.open` on the model, which loads the requirement set and opens the requirement set in the **Requirements Editor**.

After retrieving the `slreq.ReqSet` object, you can retrieve the properties like other requirement sets. Retrieve the `slreq.Requirement` objects with `find`.

```
requirements = find(reqSet, "Type", "Requirement");
```

If you want to retrieve only the requirements at the highest hierarchy level, use the `children` function on the `slreq.ReqSet` object. You can then access the child requirements of each requirement by using the `children` function on each `slreq.Requirement` object.

If the Requirements Table block is open, you can retrieve `slreq.Requirement` objects by using the `slreq.getCurrentObject` function. To retrieve the `slreq.Requirement` object from the interface, select the requirement in the block or the **Requirements Editor** and enter `slreq.getCurrentObject`.

You can only adjust requirement hierarchy in the graphical interface. To add or remove requirements, use `addRequirementRow` and `removeRow`.

See Also

[RequirementsTable](#) | Requirements Table

Related Examples

- “Use a Requirements Table Block to Create Formal Requirements” on page 2-2
- “Configure Properties of Formal Requirements” on page 2-27
- “Establish Hierarchy in Requirements Table Blocks” on page 2-45

Identify Inconsistent and Incomplete Formal Requirement Sets

When using a Requirements Table block, you can identify inconsistent and incomplete requirement sets, and find read-before-write errors. These problems can depend on semantic errors in individual requirements or can be caused by issues in the requirement set.


Analyze the Block



Formal requirements must uniquely define the data used in the preconditions, postconditions, and actions at each simulation time step. Three scenarios can violate this principle:

- If the block can execute a scenario where at least one data is not assigned a value, the requirement set is incomplete.
- If at least one data defined in the requirement set can equal more than one value during simulation, the requirement set is inconsistent.
- If you define requirements that use data before the data is written, the requirement set can cause a read-before-write error.

Typically, these scenarios generate an error during simulation, but they can be difficult to detect manually. If you have Simulink Design Verifier, you can detect the causes of the scenarios. To analyze the requirements, open the Requirements Table block. In the **Table** tab, in the **Analyze** section, click **Analyze Table**.

Inconsistency Issues

Inconsistency issues occur when the block can perform different behaviors for a single combination of input values. For example, the table below illustrates inconsistent requirements. When analyzed, it finds two inconsistency issues, which are highlighted red and include an alert icon .

Requirements		Assumptions			
Index	Summary	Precondition		Action	
		u1	u2	y1	y2
1	Requirement 1	<0			
1.1	child 1		0	1	1
1.2	child 2		<0	2	2
1.3	child 3		>0	1	1
2	Requirement 2	>=0			
2.1	child 1		0	1	1
2.2	child 2 		<0	2	2
2.3	child 3 		<0	1	1
2.4	child 4		>0	1	1

The **Analysis Results** pane displays additional details on the inconsistency issues.

Inconsistency Issues

Inconsistency 1: y1 is inconsistent at time 0.6 in requirements 2.2 and 2.3 for the following inputs:

Time	0	0.2-0.4	0.6
Step	1	2-3	4
u1	5.9566	-1	1.6229
u2	1.9184	0	-4.7853

Inconsistency 2: y2 is inconsistent at time 0.6 in requirements 2.2 and 2.3 for the following inputs:

Time	0	0.2-0.4	0.6
Step	1	2-3	4
u1	5.9566	-1	1.6229
u2	1.9184	0	-4.7853

Incompleteness Issues

Requirements have incompleteness issues when the block does not comprehensively specify outputs for possible input values. For example, in the model used in the “Generate Model Behavior” on page 2-7 example, the table includes two incompleteness issues.

Incompleteness Issues

Incompleteness 1: 'y1' is not specified at time 0.2 for the following inputs:

Time	0	0.2
Step	1	2
u1	5.9566	0
u2	1.9184	0

Incompleteness 2: 'y2' is not specified at time 0.2 for the following inputs:

Time	0	0.2
Step	1	2
u1	5.9566	0
u2	1.9184	0

You can fix these issues by introducing requirements that define when $u1$ and $u2$ are less than or equal to 0, or by creating an assumption that constrains $u1$ and $u2$ to be greater than or equal to 0. For more information about how to write assumptions, see “Add Assumptions to Requirements” on page 2-10.

Read-Before-Write Issues

Read-before-write issues occur when a requirement attempts to read data that has not yet been defined. In most cases, you can minimize the potential for read-before-write issues by not calling output data in preconditions and postconditions, or by specifying the requirements in order. For more information, see “Establish Hierarchy in Requirements Table Blocks” on page 2-45. For example, in this table, the first requirement reads the output data $y2$ before $y2$ has an assigned value.

Requirements		Assumptions		
Index	Summary	Precondition	Action	
			y1	y2
1	Requirement 1	$u \geq 0$	y2	3
2	Requirement 2	$u < 0$	2	1

When you analyze the requirements, the block detects a read-before-write issue.

Read-Before-Write Issues

Read-Before-Write 1: Requirement 1 reads 'y2' at time 0 before being specified for the following inputs:

Time	0
Step	1
u	5.9566

You can create requirement preconditions that rely on output data if you list the requirements in order and enable the **Enable outputs in preconditions** property in the block. See “Enable outputs in preconditions” on page 2-51. For example, the model used in “Example Using Requirement Order” on page 2-43 does not produce requirement issues because the requirement order specifies the local data before the other requirements need it. If you change the order of the requirements so that Requirements 3 and 4 are listed first, then analyze the requirements, the analysis finds two read-before-write issues.

Note If you write more than two requirements that read the same data before the data is written, the analysis detects the issue in only the first listed requirement with the issue. After resolving the issue, rerun the analysis to detect the next read-before-write issue.

Include the Entire Model in Analysis

By default, the Requirements Table block assumes that input data is independently generated. If the input data is not independent, you may find that you must over specify your requirements to prevent issues. You can prevent this problem by doing the following:

- Configure the block to identify the source of the data in the context of the entire model. In the **Table** tab, in the **Analyze** section, expand the **Analyze Table** menu and enable **Include entire model**.
- Specify assumptions based on how physical or mathematical limitations prevent data from being certain values. See “Add Assumptions to Requirements” on page 2-10.

Analyze Rigid and Flexible Requirements

Depending on how you define the relationships between data, you can establish two kinds of requirements: rigid and flexible. In addition to individual requirements, sets of requirements in the block can also be rigid or flexible.

Rigid Requirements

If a requirement postcondition specifies an exact value, or if the requirement specifies only an action, the requirement is rigid. You express these requirement postconditions with the double equals sign $==$. For example, a requirement with a postcondition $y == 0$ is rigid. If each requirement in the requirement set is rigid, then the requirement set is rigid.

Flexible Requirements

If a requirement postcondition can satisfy a range of values, then the requirement is flexible. For example, a requirement with a postcondition $y >= 0$ or $y >= 0 \ \&\& \ y < -2$ is flexible. Additionally,

postconditions that specify multiple values also create flexible requirements. For example, a requirement with a postcondition $u == 3 \ || \ u == 4$ is flexible. If at least one requirement in the requirement set is flexible, then the entire requirement set is flexible.

Analysis Limitations

You can analyze the Requirements Table block with some features only if the requirement set is rigid. These features include:

- Persistent variables.
- Using arrays as Requirements Table block outputs, or inputs with the **Treat as design model output for analysis** property enabled.
- Some functions. Incompatible functions will cause an error when analyzing the table. The error message identifies the responsible function.

See Also

Requirements Table

Related Examples

- “Use a Requirements Table Block to Create Formal Requirements” on page 2-2
- “Control Requirement Execution by Using Temporal Logic” on page 2-33
- “Establish Hierarchy in Requirements Table Blocks” on page 2-45
- “Add Assumptions to Requirements” on page 2-10


Debug Requirements Table Blocks


You can debug formal requirements in a Requirements Table block. The Requirements Table block includes some of the same debugging tools available in the MATLAB editor. To use the debugger, set a breakpoint on at least one the requirements and run the simulation. The Requirements Table block also checks for erroneous table cells as you create and edit requirements.

Add a Breakpoint to a Requirement Set

This example uses the model in “Model Dice Game Rules Using a Requirements Table Block” on page 2-80. To debug the Requirements Table block requirements:

- 1 Open the Requirements Table block, `Dice Game Rules`, in the example model, `ReqTableDiceGame`:


```
openExample('slrequirements/DiceGameReqTableBlockExample')
open_system("ReqTableDiceGame/Dice Game Rules");
```
- 2 Right-click the requirement with the index 1 and click **Set Breakpoint**. The requirement displays the breakpoint icon  next to the index.

Requirements		Assumptions
Index	Summary	
1		Conditions for the first roll
1.1		If the player rolls a 7 or a 11 on the first roll
1.2		If the player rolls a 2, 3, or 12 on the first roll
1.3		Otherwise, establish the point value

- 3 Select when the simulation pauses during the requirement evaluation. Click the breakpoint to expand the options:
 - **Before Checking Precondition** — The simulation stops before the block checks the precondition.
 - **After Precondition is Valid** — The simulation stops only if the precondition is true.

If you clear both of these options, the table removes the breakpoint. If you select both options, the block creates two breakpoints.

In this example, select **Before Checking Precondition** and clear **After Precondition is Valid**.

- 4 Simulate the model. The simulation pauses when execution reaches the breakpoint. The block highlights the precondition that the simulation stops at in green.

Requirements		Assumptions			
Index	Summary	Precondition		Action	
			dice1 + dice2	point	gameStop
1	Conditions for the first roll	isStartup			

- 5 When the simulation pauses at a breakpoint, you can use buttons in the **Debug** tab to continue execution in other ways:

Button	Description
Continue	Continue execution to the next breakpoint.
Step Over	Advance to the next step in the requirement set execution. Possible steps include: <ul style="list-style-type: none"> Evaluate a precondition or postcondition Execute an action. Each action cell executes from left to right as you step over. If an action cell contains multiple lines, execute each line of the action as you step over.
Step In	From a precondition, postcondition, or action that calls a function, advance to the first executable statement in the function. Otherwise, advance to the next step in the requirement set execution. Each action cell executes from left to right as you step in. If an action cell contains multiple lines, execute each line of the action as you step in.
Step Out	From a function call, return to the requirement precondition, postcondition, or action calling the function. Otherwise, continue execution to the next breakpoint.
Stop	Exit debug mode and interrupt the execution.

In this example model, clicking **Continue** advances to the next time step or ends the simulation.

Using Multiple Breakpoints

You can add multiple breakpoints to a Requirements Table block. The block evaluates the listed requirement first and works downward. See “Establish Hierarchy in Requirements Table Blocks” on page 2-45.


In the Dice Game Rules block, add a breakpoint to the requirement with index 2, clear **Before Checking Precondition**, select **After Precondition is Valid**, and simulate the model. Click **Continue** to advance to the next breakpoint or the next time step. At the first time step, the requirement with index 2 does not pause the simulation. However, if the player sets the point value, the simulation pauses at both breakpoints at the following time step.

Requirements		Assumptions			
Index	Summary	Precondition		Action	
			dice1 + dice2	point	gameStop
1	Conditions for the first roll	isStartup			
1.1	If the player rolls a 7 or a 11 on the first roll		7,11	0	natural
1.2	If the player rolls a 2, 3, or 12 on the first roll		2,3,12	0	craps
1.3	Otherwise, establish the point value	Else		dice1 + dice2	roll_again
2	Following rolls	~isStartup			

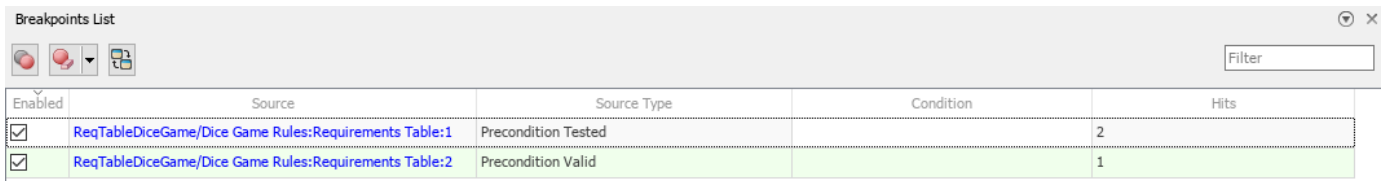
Run the simulation again, but click **Step Over** or **Step In** instead. The debugger pauses at each step of the block execution. Depending on the roll, the debugger stops at only one of the actions, but evaluates several preconditions.

View the Simulation Status at the Breakpoints

You can view additional information about the breakpoints in the **Breakpoints List** pane. To open the pane, click a breakpoint and click **Breakpoints List**. The **Breakpoints List** pane displays five columns:

Column Header	Description
Enabled	Specifies whether the breakpoint is enabled. When you disable Before Checking Precondition and After Precondition is Valid on a requirement, the breakpoint icon is grey  .
Source	Specifies the location of the breakpoint.
Source Type	Specifies whether the block checks the requirement before checking the precondition or after the precondition is valid.
Condition	Specifies a logical expression that must be true before the simulation pauses. Specify an expression by using Requirements Table block data and literal values. The expression must evaluate to a logical <code>true</code> (1) or <code>false</code> (0).
Hits	Specifies the number of times that the simulation pauses at the breakpoint.

In the **Dice Game Rules** block, the **Breakpoints List** pane lists two breakpoints. Run the model to see how many times each breakpoint pauses the simulation as you run the debugger. The pane highlights the breakpoint that the simulation pauses at.



Enabled	Source	Source Type	Condition	Hits
<input checked="" type="checkbox"/>	ReqTableDiceGame/Dice Game Rules:Requirements Table:1	Precondition Tested		2
<input checked="" type="checkbox"/>	ReqTableDiceGame/Dice Game Rules:Requirements Table:2	Precondition Valid		1

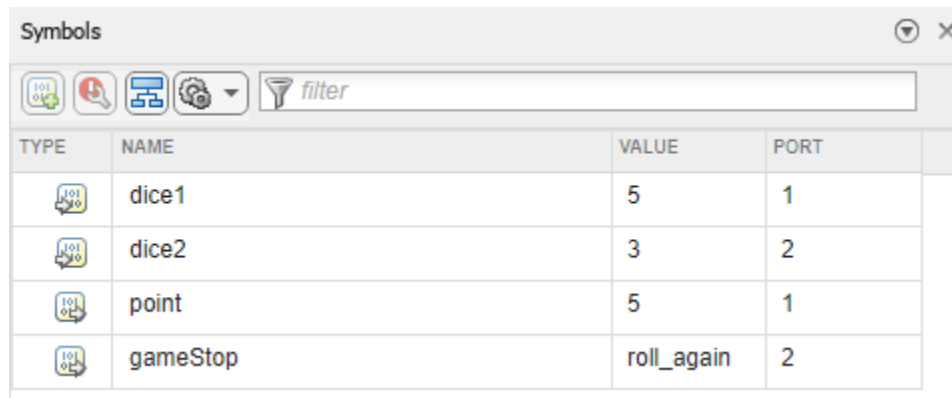
Watch Data Values During Simulation

You can track the data values in the block while you simulate by using the **Symbols** pane or the Command Window.

Watch Data Values in the Symbols Pane

To view the value of the data in the **Symbols** pane in a Requirements Table block during simulation:

- 1 Open the **Symbols** pane. In the **Modeling** tab, in the **Design Data** section, click **Symbols Pane**.
- 2 Add a breakpoint to the a requirement.
- 3 Run the model.
- 4 When the simulation pauses, the **Symbols** pane displays the values at the breakpoint. If the data is an array, the data column displays the dimension and data type of the data. In this example, the data are scalar.



TYPE	NAME	VALUE	PORT
	dice1	5	1
	dice2	3	2
	point	5	1
	gameStop	roll_again	2

Watch Block Data with the Command Line Debugger

You can view the values for block data in the MATLAB Command Window. When you reach a breakpoint, enter the name of a data at the `debug>>` prompt to see its value and data type.


```
debug>> dice1
    int8
     5
debug>>
```

You can also enter these commands in the Command Window while debugging:

Command	Description
dbcont	Continue execution to next breakpoint.
dbstep [in out]	Advance to next step of execution after encountering a breakpoint.
dbquit	Quit debugging and terminate the simulation.
help	Display help for command line debugging.
print <data>	Display the value of the data <code>data</code> at the current time step and breakpoint. If <code>data</code> is a vector or matrix, you can also index into <code>data</code> . For example, <code>data(1,2)</code> .
save	Save data at the current time step and breakpoint to the specified file. The command uses the syntax of the MATLAB <code>save</code> command. To retrieve data from the MATLAB base workspace, use <code>load</code> function after simulation completes.
<data>	Equivalent to <code>print <data></code> if the data is defined in the block.
who	Display the data defined in the block.
whos	Display the size and type of the data defined in the block.

To issue a command in the MATLAB base workspace at the `debug>>` prompt, use the `evalin` command with the first argument "base" followed by the second argument command, for example, `evalin("base", "whos")`. You cannot define or change data at the `debug>>` prompt.

Resolve Cell Errors

Table cell errors appear when the block identifies syntax errors or data that has not been defined in the **Symbols** pane. Point to or click the alert icon  next to the cell to read the error message. If you run the simulation without addressing these problems, the **Diagnostic Viewer** displays details about the error.

The block does not detect all issues while editing the table cells. For instance, if the duration for a requirement evaluates to a vector or matrix, a compilation error occurs. These warnings and errors appear in the **Diagnostic Viewer**.

See Also

Requirements Table

Related Examples

- "Use a Requirements Table Block to Create Formal Requirements" on page 2-2
- "Identify Inconsistent and Incomplete Formal Requirement Sets" on page 2-17

Configure Properties of Formal Requirements

In this section...

“Open the Requirements in the Requirements Editor” on page 2-27

“Manage Requirements In the Requirements Editor and Property Inspector” on page 2-28

“Link Requirements” on page 2-30

“Create a Traceability Matrix and Traceability Diagram” on page 2-31

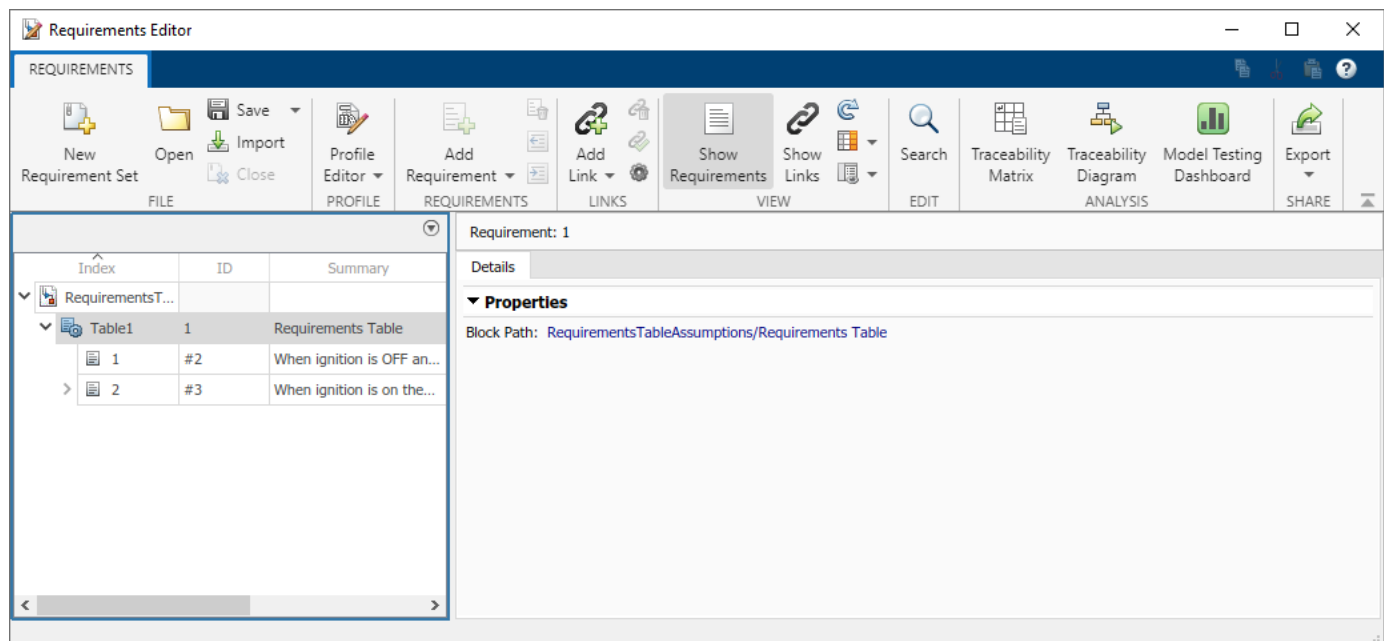
When you create formal requirements with a Requirements Table block, the block also creates requirements that you can configure in the **Requirements Editor** or the Property Inspector. You can also use the **Requirements Editor** to link the requirements in Requirements Table blocks to model elements. After you configure your requirements, you can analyze the requirements by using a traceability matrix or traceability diagram.

Open the Requirements in the Requirements Editor

When you load a Simulink model that contains at least one Requirements Table block, the **Requirements Editor** loads the requirements associated with the block. To open the **Requirements Editor**, in the **Apps** tab, click **Requirements Editor**. Alternatively, enter `slreq.editor` at the MATLAB command line. You can load one or multiple models.

If the models that you load contain at least one Requirements Table block, the **Requirements Editor** loads each model as a requirement set. The **Requirements Editor** lists the Requirements Table blocks contained in each model as Import nodes that you can expand to view the requirements. The **Summary** column of the node is the name of each block.

For example, load the model used in “Use Assumptions in an Example Model” on page 2-10 and expand the Import node **Table1** to see the requirements.



Simulink stores the requirement set of each block in the model. When you load the model, Requirements Toolbox creates a copy of the requirement set file in memory. However, Requirements Toolbox does not create a separate requirement set file.

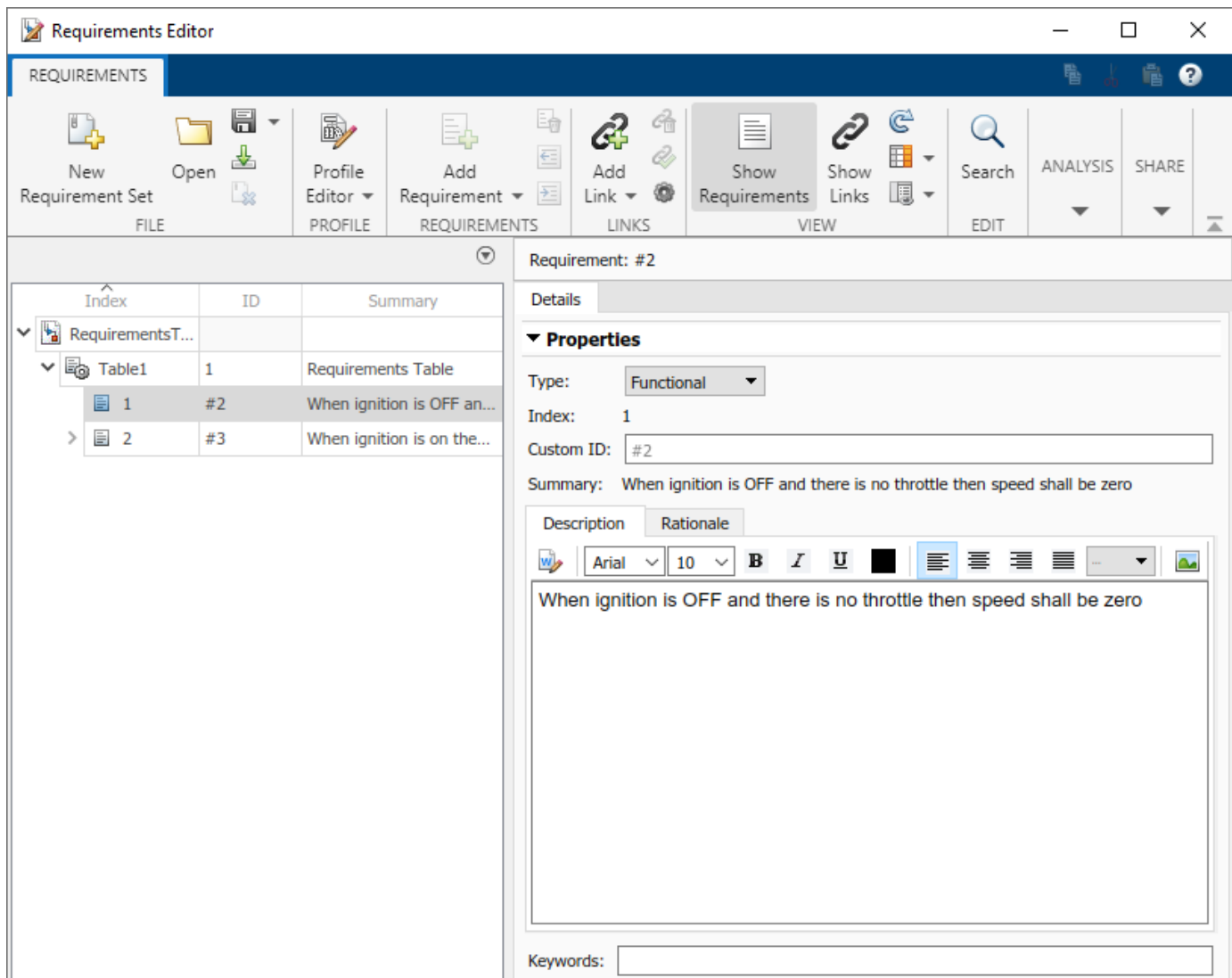
If you load a model that contains more than one Requirements Table block, the **Requirements Editor** loads the requirement sets of each Requirements Table block. You cannot specify individual Requirements Table blocks to load from a given model.

Manage Requirements In the Requirements Editor and Property Inspector

After loading the model, you can use the **Requirements Editor** and the Property Inspector to modify the requirement type, custom ID, description, rationale, or keywords for each requirement.

Use the Requirements Editor

In the **Requirements Editor**, expand the Import node to see the requirements. Click a requirement to modify its properties.



You can then modify the **Type**, **Custom ID**, **Description**, **Rationale**, **Keywords**, and **Comments** properties. Use the **Type** property to specify the role of the requirement. See “Requirement Types” on page 1-6. Use the **Custom ID**, **Description**, **Rationale**, **Keywords**, and **Comments** properties to add text-based details to requirements that can help you locate requirements and convey useful information.

When you add or remove requirements in a block, the requirements update in the editor. If you change the name of the summaries of each requirement in the block, the **Summary** property automatically updates. When you first create a requirement, the **Description** field has the same value as the **Summary** property.

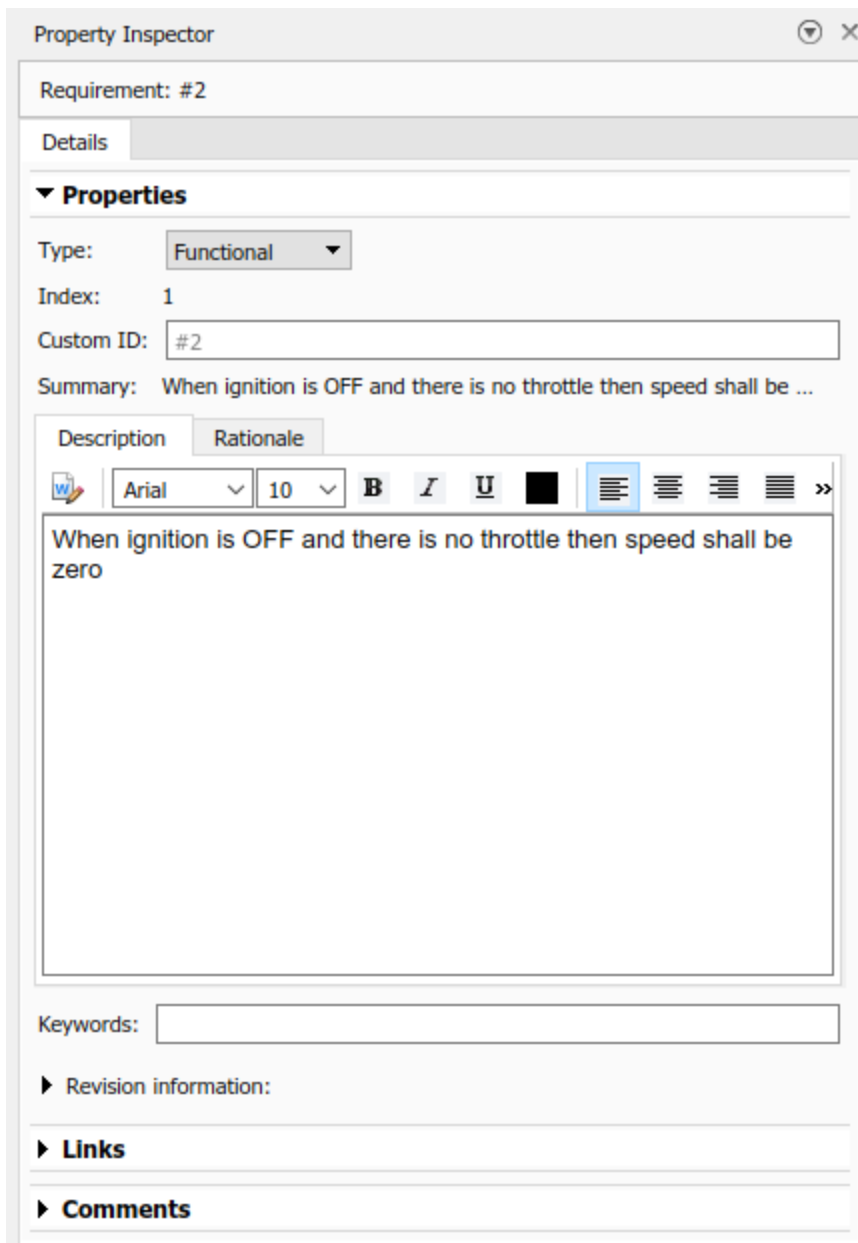
Use the Property Inspector

To use the Property Inspector:

- 1 Open the model that contains the Requirements Table block.
- 2 Open the Requirements Table block.

- 3 Open the Property Inspector. In the **Modeling** tab, in the **Design Data** section, click **Property Inspector**.
- 4 In the **Requirements** tab of the block, click the requirement.

The Property Inspector shows the same properties that appear in the right pane of the **Requirements Editor**.



Link Requirements

You can use the **Requirements Editor** to add incoming or outgoing links to requirements in the Requirements Table block. To create incoming links from the block requirements to another element in a model:

- 1 In the model, select a model element.
- 2 In the **Requirements Editor**, select a requirement.
- 3 Click **Add Link > Link from Selection in Simulink**.

You can also add outgoing links to the block requirements. See “Outgoing Links Editor” on page 11-6.

If you save the model with a new name, the software creates a copy of the link set file that has the same name of the model file. However, if the requirements have any incoming links, a dialog box asks whether to copy or clear the links when you save the model. Click **Update incoming links** to copy the links or **Disconnect incoming links** to clear the new link set.

When you link requirements in a Requirements Table block, the software creates a separate link set file with the same name as the model by default. If you do not want to create a separate link set file, you can also store the links internally as part of the model:

- 1 In the **Apps** tab, click **Requirements Manager**.
- 2 In the **Requirements** pane, set **View** to **Links**.
- 3 In the **Requirements** tab, in the **Settings** section, click **Link Settings > Default Link Storage**.
- 4 In the Requirements Settings window, in the **Storage** tab, select **Store internally (embedded in Simulink diagram file)**.

For more information, see “Requirements Link Storage” on page 6-4.

Create a Traceability Matrix and Traceability Diagram

After linking the requirements, you can create a traceability matrix or a traceability diagram.

Create a Traceability Matrix

To access the Traceability Matrix window, use one of these approaches:

- In the **Requirements Editor**, click **Traceability Matrix**.
- At the MATLAB command line, enter:

```
slreq.generateTraceabilityMatrix
```

To create a traceability matrix, select the model file and the SLREQX file of the same name. For more information, see “Track Requirement Links with a Traceability Matrix” on page 3-5.

Create a Traceability Diagram

To create a traceability diagram, use one of these approaches:

- In the **Requirements Editor**, select the requirement or Import node and, in the **Analysis** section, click **Traceability Diagram**.
- In the **Requirements Editor**, right-click the requirement set or block node and select **View Traceability Diagram**.
- At the MATLAB command line, enter:

```
slreq.generateTraceabilityDiagram
```

For more information, see “Visualize Links with Traceability Diagrams” on page 3-17.

See Also

Requirements Table

Related Examples

- “Author Requirements in MATLAB or Simulink” on page 1-2
- “Use a Requirements Table Block to Create Formal Requirements” on page 2-2
- “Create Requirements Table Blocks Programmatically” on page 2-13

Control Requirement Execution by Using Temporal Logic

In this section...

“Using the Duration Column” on page 2-33

“Use Temporal Logic Operators” on page 2-35

You can control the evaluation and execution of requirements in Requirements Table blocks in Simulink models by using temporal logic. Because the **Duration** column and temporal operators rely on simulation time or data values at different time steps, you can use them to execute temporal logic.

Using the Duration Column

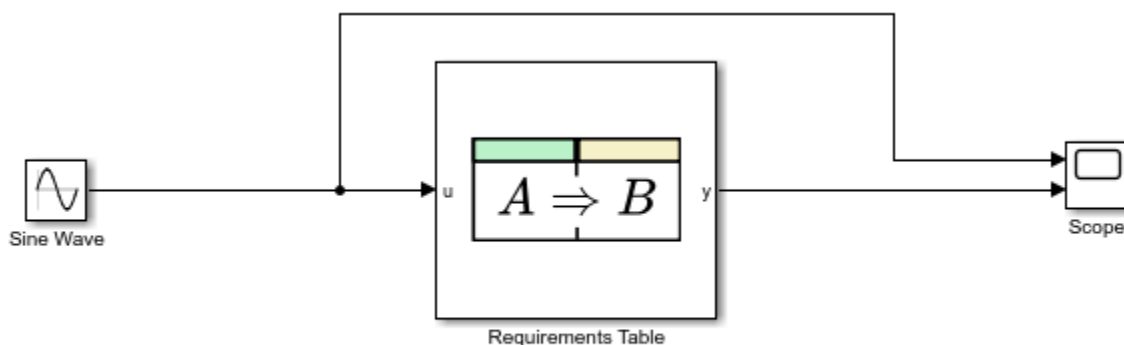
The **Duration** column specifies how long a requirement precondition must be valid before the block checks the postconditions or executes actions. If the duration is not met, the block does not check the requirement postconditions or execute the requirement actions. You must specify a precondition to use the **Duration** column.

You can view or hide the **Duration** column by right-clicking the column and selecting **Toggle Column > Toggle Duration Column**. You can also view or hide the column in the **Table** tab, in the **Columns** section, by clicking **Show Columns > Duration**. The column appears in the **Requirements** tab only.

To use the **Duration** column, define a requirement and enter the amount of time that the precondition must be true for the precondition to be satisfied. You must specify the duration in seconds and the duration must evaluate to a positive scalar.

Example Using Duration Column

The example uses a Requirements Table block to check that an input signal satisfies temporally dependent requirements. The block uses the **Duration** column to specify how long a requirement must be valid before executing outputs.



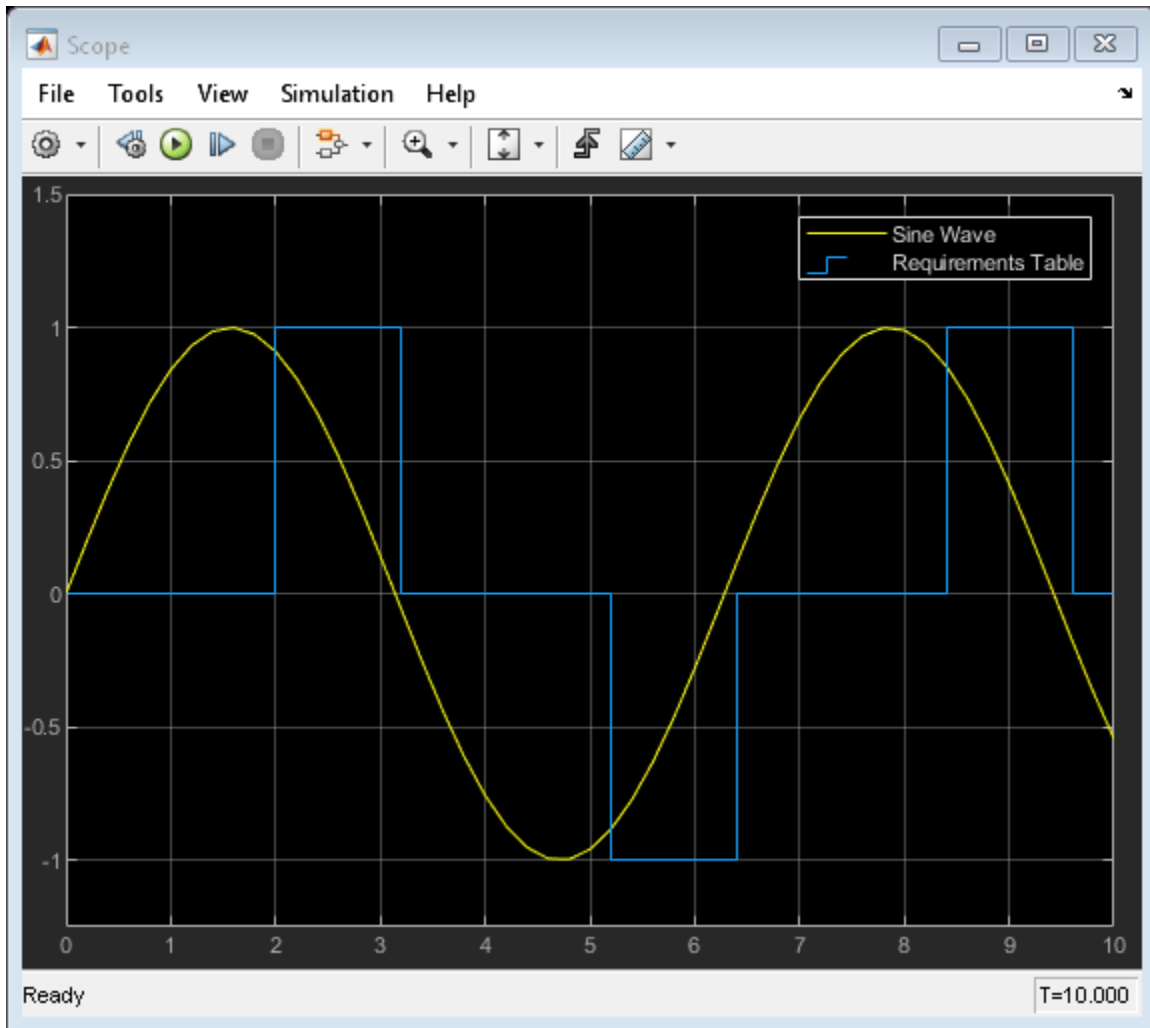
Copyright 2021 The MathWorks, Inc.

Open the Requirements Table block. The block uses three requirements to check the input data u and to specify the output data y :

- If u is greater than or equal to θ for more than 2 seconds, the block sets y to 1.
- If u is less than θ for more than 2 seconds, the block sets y to -1.
- If neither requirement is met, the block sets y to 0.

Requirements		Assumptions		
Index	Summary	Precondition	Duration	Action
1	Requirement 1	$u \geq 0$	2	$y = 1$
2	Requirement 2	$u < 0$	2	$y = -1$
3	Requirement 3 D	Else		$y = 0$

Run the simulation to view the output in the Scope block.



For an additional example, see “Define Formal Requirements with a Duration” on page 2-89.

Use Temporal Logic Operators

You can also add temporal logic to a Requirements Table block by using operators. These operators can appear in preconditions, postconditions, or actions in either the **Requirements** or **Actions** tabs.

Operator	Syntax	Description
duration	duration(C)	Returns the length of time in seconds that has elapsed since the conditional expression C became true.
isStartup	isStartup isStartup()	Returns true if the simulation time equals 0 and returns false at the other simulation times.

Operator	Syntax	Description
getPrevious	prev(data_name) getPrevious(data_name)	Returns the value of the data at the previous time step.
t	t	Returns the simulation time in seconds.

Considerations When Using Temporal Logic Operators

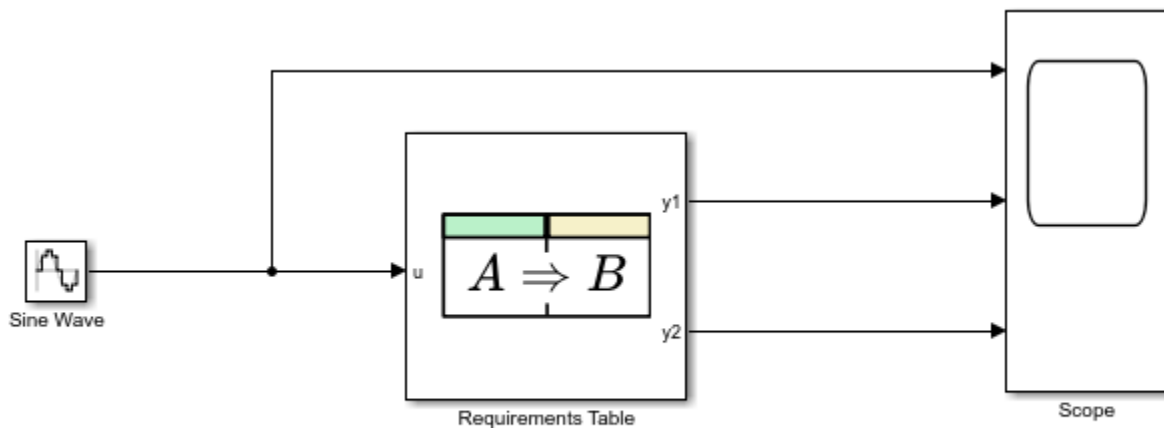
If `getPrevious` attempts to return the value of the data at a time step when it was not defined, it returns an undefined value. For example, data is not defined before the simulation time is 0.

Use the `isStartup` operator in the preconditions to define additional requirements at a simulation time of 0, and `~isStartup` at the other time steps. Avoid using `prev` in preconditions unless `~isStartup` is used in the parent requirement. See “Add Child Rows” on page 2-45 and “Check Previous Data Values”.

If you use a Requirements Table block containing `isStartup` in an Enabled Subsystem block, the `isStartup` operator returns `true` at the time step you reenable the Enabled Subsystem block. However, the `t` operator captures the time value used in the top model, and therefore does not reset in the same circumstances.

Example Using Temporal Logic Operators

This example shows a Requirements Table block that uses temporal logic operators to check an input signal. The Requirements Table block checks the value of the input signal and the simulation time to execute different values for the two outputs.



Copyright 2021 The MathWorks, Inc.

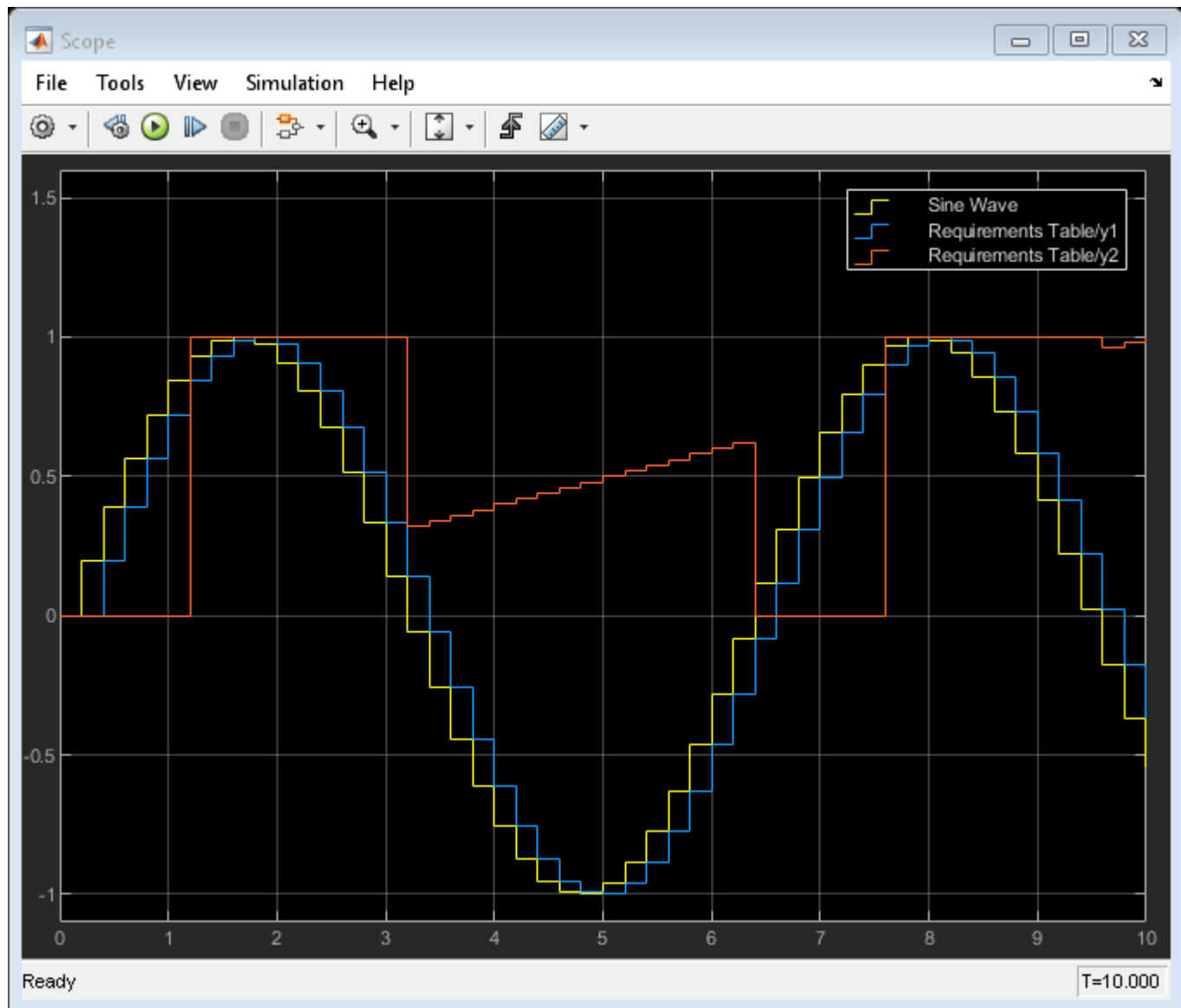
Open the Requirements Table block. The block uses the four top-level requirements and the two child requirements of the third requirement to check the value of the input data u:

- If the simulation has just started, the block sets the output data `y1` to 0. Otherwise, the block sets `y1` to the value of `u` at the previous time step.

- If u is greater than or equal to 0, the block checks if the condition is true for more than 1 second. If the condition is satisfied for more than 1 second, the block sets $y2$ to 1. Otherwise, the block sets $y2$ to 0.
- If u is less than 0, the block sets $y2$ equal to the simulation time divided by 10.

Requirements		Assumptions	
Index	Summary	Precondition	Action
1	Requirement 1	<code>isStartup</code>	<code>y1 = 0</code>
2	Requirement 2	<code>~isStartup</code>	<code>y1 = prev(u)</code>
3	Requirement 3	<code>u >= 0</code>	
3.1	Requirement 3.1	<code>duration(u >= 0) > 1</code>	<code>y2 = 1</code>
3.2	Requirement 3.2	Else	<code>y2 = 0</code>
4	Requirement 4	<code>u < 0</code>	<code>y2 = t/10</code>

Run the simulation to view the outputs in the Scope block.



See Also

Requirements Table

Related Examples

- "Detect Data Changes by Using Requirements Table Blocks" on page 2-39
- "Define Formal Requirements with a Duration" on page 2-89
- "Identify Inconsistent and Incomplete Formal Requirement Sets" on page 2-17
- "Define Data in Requirements Table Blocks" on page 2-53

Detect Data Changes by Using Requirements Table Blocks

In this section...
“Change Detection Operators” on page 2-39
“Example of Requirements Table Block with Change Detection” on page 2-40

Requirements Table blocks can detect changes in the values of data between time steps. You can use change detection operators to determine when data changes to or from a value.

Change Detection Operators

To detect changes in data, use the operators listed in this table.

Operator	Syntax	Description
hasChanged	tf = hasChanged(data_name)	Returns 1 (true) if the value of data_name at the beginning of the current time step is different from the value of data_name at the beginning of the previous time step. Otherwise, the operator returns 0 (false).
hasChangedFrom	tf = hasChangedFrom(data_name, value)	Returns 1 (true) if the value of data_name was equal to the specified value at the beginning of the previous time step and is a different value at the beginning of the current time step. Otherwise, the operator returns 0 (false).
hasChangedTo	tf = hasChangedTo(data_name, v alue)	Returns 1 (true) if the value of data_name was not equal to the specified value at the beginning of the previous time step and is equal to value at the beginning of the current time step. Otherwise, the operator returns 0 (false).

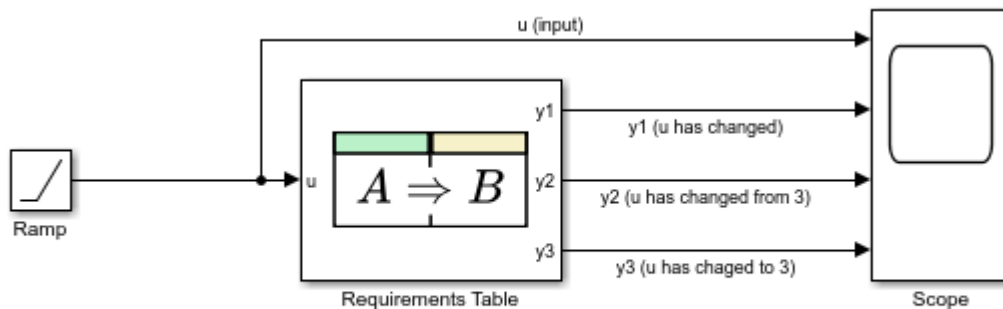
The input argument data_name is data defined in the Requirements Table block, specified as a:

- Scalar
- Matrix or an element of a matrix
- Structure or a field in a structure
- Valid combination of structure fields or matrix elements

For the hasChangedFrom and hasChangedTo operators, the argument value must be an expression that resolves to a value that is comparable with data_name. For example, if data_name is a matrix, then value must resolve to a matrix value with the same dimensions as data_name.

Example of Requirements Table Block with Change Detection

This example shows how the operators `hasChanged`, `hasChangedFrom`, and `hasChangedTo` detect specific changes in an input signal. In this example, a Ramp (Simulink) block sends a discrete, increasing time signal to a Requirements Table block.



Copyright 2021 The MathWorks, Inc.

The model uses a fixed-step solver with a step size of 1. The signal increments by 1 every time step. The block checks the input `u` for these changes:

- Changes from the previous time step
- A change from the value 3
- A change to the value 3

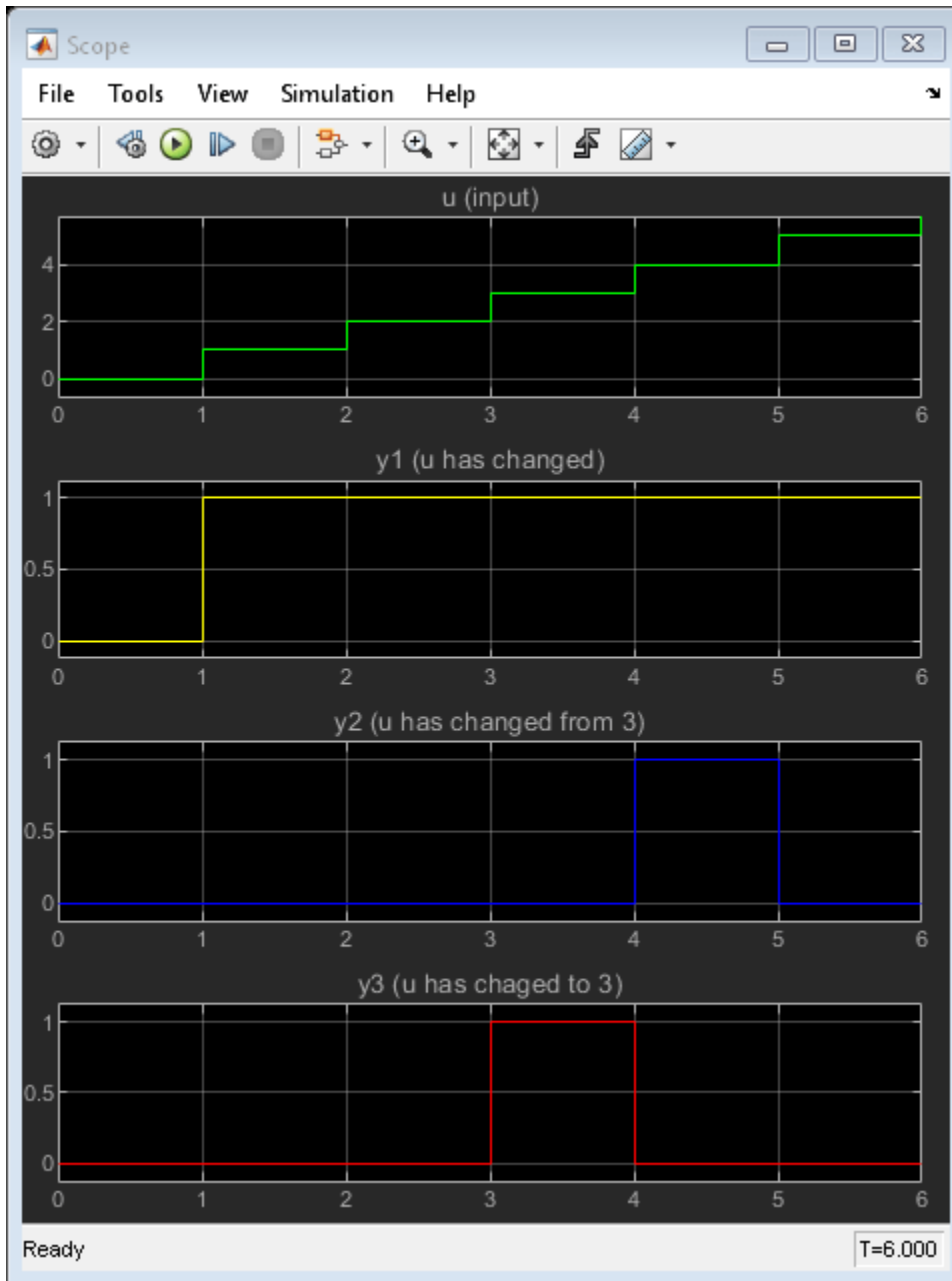
To check the signal, the block calls three change detection operators and specifies six requirements. Each change detection operator determines the value of the output data `y1`, `y2`, and `y3`.

- If `hasChanged(u)` is `true`, `y1` equals 1. Otherwise, `y1` equals 0.
- If `hasChangedFrom(u, 3)` is `true`, `y2` equals 1. Otherwise, `y2` equals 0.
- If `hasChangedTo(u, 3)` is `true`, `y3` equals 1. Otherwise, `y3` equals 0.

Requirements		Assumptions	
Index	Summary	Precondition	Action
1	Requirement 1	<code>hasChanged(u)</code>	$y1 = 1$
2	Requirement 2	<code>~hasChanged(u)</code>	$y1 = 0$
3	Requirement 3	<code>hasChangedFrom(u,3)</code>	$y2 = 1$
4	Requirement 4	<code>~hasChangedFrom(u,3)</code>	$y2 = 0$
5	Requirement 5	<code>hasChangedTo(u,3)</code>	$y3 = 1$
6	Requirement 6	<code>~hasChangedTo(u,3)</code>	$y3 = 0$

During simulation, the Scope (Simulink) block shows the input and output signals for the block.

- The value of u increases by 1 every time step.
- The value of $y1$ changes from 0 to 1 at time $t = 1$. The value of $y1$ remains 1 because u continues to change at each subsequent time step.
- The value of $y2$ changes from 0 to 1 at time $t = 4$ when the value of u changes from 3 to 4. The value of $y2$ returns to 0 after one time step.
- The value of $y3$ changes from 0 to 1 at time $t = 3$ when the value of u changes from 2 to 3. The value of $y3$ returns to 0 after one time step.



See Also

Requirements Table

Related Examples

- "Specify Requirements Table Block Properties" on page 2-50
- "Define Data in Requirements Table Blocks" on page 2-53
- "Set Data Types in Requirements Table Blocks" on page 2-59
- "Control Requirement Execution by Using Temporal Logic" on page 2-33

Leverage Evaluation Order of Formal Requirements

The Requirements Table block evaluates requirements by starting at the first requirement and working downward. You can leverage this behavior by assigning values to data and then using the data in requirements listed later in the requirement set.

For example, if you want to reference local data in the preconditions of other requirements:

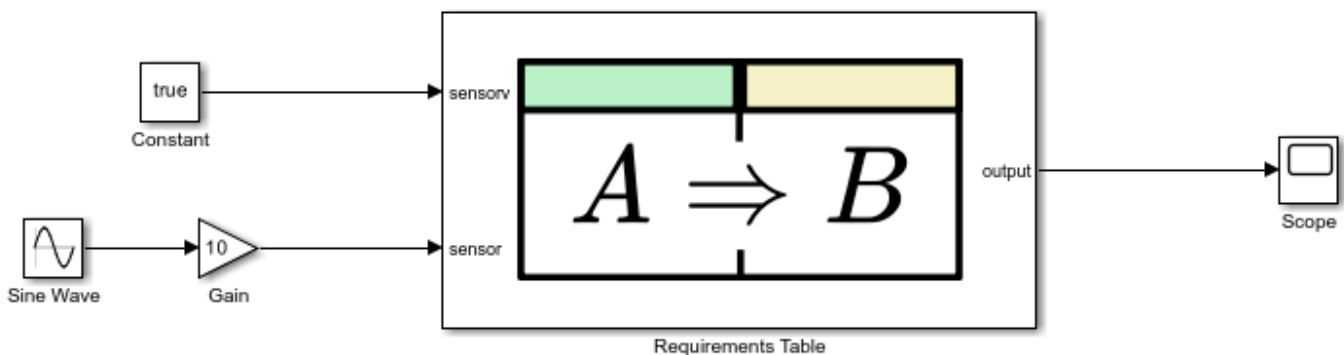
- 1 List the requirements that define the local data values first in the set.
- 2 List the requirements that require the local data in preconditions later.
- 3 Verify that your requirements do not read data before being written.
 - If you have Simulink Design Verifier, you can analyze the requirements to detect instances where data is read before it is written. In the **Table** tab, in the **Analyze** section, click **Analyze Table**. See “Identify Inconsistent and Incomplete Formal Requirement Sets” on page 2-17.

For more information on how to define requirements, see “Use a Requirements Table Block to Create Formal Requirements” on page 2-2.

You can reference output data in preconditions if the **Enable outputs in preconditions** property is enabled or if the data is an argument in the `getPrevious` operator. To enable this property, open the Requirements Table block. In the **Modeling** tab, in the **Design Data** section, click **Property Inspector**. In the **Properties** tab, enable the **Enable outputs in preconditions** property. See “Enable outputs in preconditions” on page 2-51.

Example Using Requirement Order

This example shows how requirement order affects data in Requirements Table blocks.



Copyright 2021, The MathWorks, Inc.

Examine the Requirements

Open the Requirements Table block. The block uses the input data `sensorv` and `sensor` to determine the value of the local data `value`. The requirements then use `value` to determine the value of the output data `output`.

Requirements		Assumptions		
Index	Summary	Precondition		Action
		sensorv		
1	Requirement 1	true		value = sensor
2	Requirement 2	false		value = sensor/2
3	Requirement 4		value > 4	output = 1
4	Requirement 3		value <= 4	output = 0

If you define Requirements 3 and 4 before Requirements 1 and 2, the requirements do not define value before value is needed, which causes a read-before-write error during simulation.

See Also

Requirements Table

Related Examples

- “Use a Requirements Table Block to Create Formal Requirements” on page 2-2
- “Control Requirement Execution by Using Temporal Logic” on page 2-33
- “Detect Data Changes by Using Requirements Table Blocks” on page 2-39
- “Identify Inconsistent and Incomplete Formal Requirement Sets” on page 2-17

Establish Hierarchy in Requirements Table Blocks

In this section...

“Add Child Rows” on page 2-45

“Add Semantic Rows” on page 2-46

When defining requirements and assumptions in a Requirements Table block, you may decide that some requirements or assumptions must relate to one another in a hierarchy. For example, you may have a requirement precondition that you want to check only if another precondition is valid. Alternatively, you may have a requirement where your model must satisfy several preconditions before you check a postcondition. You can establish hierarchies by using child and semantic rows.

For more information on how to define requirements and assumptions, see “Use a Requirements Table Block to Create Formal Requirements” on page 2-2 and “Add Assumptions to Requirements” on page 2-10.

Add Child Rows

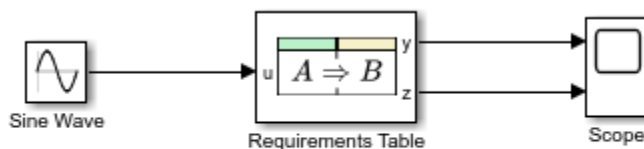
Requirements and assumptions are organized in rows. You can organize the rows into a hierarchy by creating *parent* and *child* rows. Use child rows to determine evaluation hierarchy. For example, if a parent requirement precondition is valid, the block checks the preconditions of the child requirements. Otherwise, the block ignores the child preconditions.

To create child rows, right-click the index of a row and select **Add Child Requirement** or **Add Child Assumption**. If you want to move a row to a higher or lower hierarchy level, click the row index and select **Make Parent** or **Make Child**. You can delete and modify child rows by right-clicking the index of a child row and clicking **Delete Requirement** or **Delete Assumption**.

The index number of a child row is the index of the parent followed by the index of the child. You can have multiple child rows at each hierarchy level and multiple hierarchy levels. Rows in the same hierarchy level that have the same parent, or rows that do not have parent rows, are called *siblings*.

Use Child Rows in an Example Model

This example uses a Requirements Table block that has child requirements. The model tests a signal and outputs two values if the model behavior meets the parent and child requirements.



Copyright 2021 The MathWorks, Inc.

Open the Requirements Table block to see the parent and child requirements. If the input signal satisfies the precondition of the first requirement, the block executes the action and then tests the child requirements. In this example, the child requirements control the output data *z*. The second requirement executes if the input signal does not satisfy the precondition of the first requirement.

Requirements		Assumptions		
Index	Summary	Precondition	Action	
			z	y
1	Requirement 1	$u \geq 0$		1
1.1	child 1	$u > 0.5$	1	
1.2	child 2	$u \geq 0.3 \ \& \ u \leq 0.5$	2	
1.3	child 3	$u < 0.3$	3	
2	Requirement 2	$u < 0$	0	0

Run the simulation and open the Scope block to observe the block output.

Add Semantic Rows

While designing your model, you may need to define rows that have several applicable preconditions. For example, you may have a requirement where only one precondition must be satisfied, or each of the preconditions must be simultaneously satisfied. The Requirements Table block allows you to construct these rows using semantic rows. Semantic rows evaluate two or more preconditions before checking the postconditions or executing the actions of a parent row.

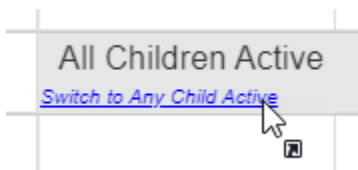
To add a semantic requirement where only one child precondition must be satisfied before the parent requirement executes, right-click the requirement index and click **Add Semantic Requirement > Any Child Active**. You can perform the same task in the **Assumptions Table** by clicking **Add Semantic Assumption > Any Child Active**.

To add a semantic requirement that executes only if each of the child requirement preconditions are satisfied, right-click the requirement index and click **Add Semantic Requirement > All Children Active**. You can perform the same task in the **Assumptions Table** by clicking **Add Semantic Assumption > All Children Active**. For each kind of semantic row, the table dims cells that cannot be filled and indicates the logic used by the children in each parent. This table illustrates what each semantic requirement looks like.

Requirements		Assumptions			
Index	Summary	Precondition	Duration	Postcondition	Action
		1		Any Child Active	
1.1					
1.2					
2		All Children Active			
2.1					
2.2					

You can specify the duration in the parent or in individual children. For more information on the duration, see “Using the Duration Column” on page 2-33. If you specify the duration in a child, the duration applies only to the child. If you specify the duration for the parent, the block checks the duration only if the semantic is satisfied.

If you want to switch the semantic row type, point to the precondition cell of the parent row. The option to switch between **Any Child Active** and **All Children Active** appears at the bottom of the cell.



Create Default Requirements

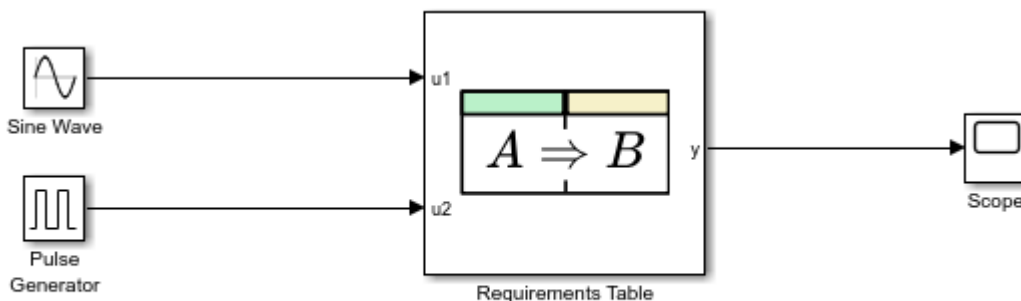
If you want to define behavior that occurs if none of the sibling requirements are met, include a default requirement. Right-click the index of the sibling requirement and click **Add Semantic Requirement > Default**.

You can specify multiple default requirements for each group of siblings. For example, this table has two requirements that each contain two child requirements and a default requirement. The default requirement executes only if the parent precondition is true and if the siblings are false.

Requirements		Assumptions		
Index	Summary	Precondition		Action
1	Requirement 1	$u1 > 1$		
1.1	Child 1		$u2 > 1$	$y = 1$
1.2	Child 2		$u2 < 1$	$y = 2$
1.3	Default Requirement D	Else		$y = 0$
2	Requirement 2	$u1 \leq 1$		
2.1	Child 1		$u2 > 1$	$y = -1$
2.2	Child 2		$u2 < 1$	$y = -2$
2.3	Default Requirement D	Else		$y = 0$

Use Semantic Rows in an Example Model

This example uses a Requirements Table block with semantic requirements. The model tests two signals and adjusts the output if the model behavior meets the semantic requirement preconditions.



Copyright 2021 The MathWorks, Inc.

Open the block. The block contains two semantic requirements and a default requirement:

- If the input $u1$ satisfies the precondition of the first child and the input $u2$ satisfies the precondition of the second child, the block sets the output y to 2.

- Otherwise, if either the first or second preconditions for the children of the second requirement are true, then the block sets y to 1.
- If the semantic requirement preconditions are not satisfied, the block executes the action of the default requirement and sets y to 0.

Requirements		Assumptions		
Index	Summary	Precondition		Action
		u1	u2	y
1	Requirement 1	All Children Active		2
1.1	Child 1	> 0		
1.2	Child 2		<= 0	
2	Requirement 2	Any Child Active		1
2.1	Child 1	> 0	> 0	
2.2	Child 2	<= 0	> 0	
3	Default Requirement	Else		0

Run the model and open the Scope block to observe the block output.

See Also

Requirements Table

Related Examples

- “Control Requirement Execution by Using Temporal Logic” on page 2-33
- “Detect Data Changes by Using Requirements Table Blocks” on page 2-39
- “Add Assumptions to Requirements” on page 2-10

Specify Requirements Table Block Properties

In this section...

“Requirements Table Block Properties” on page 2-50

“Fixed-Point Properties” on page 2-51

“Description and Document Link Properties” on page 2-52

You can specify how a Requirements Table block interfaces with a Simulink model by setting the block properties in the Property Inspector, the Table Properties window, or the Model Explorer.

To use the Property Inspector, click the **Modeling** tab, and in the **Design** section, select **Property Inspector**. Click the Requirements Table block to see the Requirements Table block properties.

To use the Table Properties window, open the block. In the **Simulation** tab, in the **Prepare** section, click **Table Properties**.

To use the Model Explorer, click the **Modeling** tab and in the **Design** section, select **Model Explorer**. In the **Model Hierarchy** pane, expand the model tree view, select the Requirements Table block, and edit the properties in the **Requirements Table** pane.

Requirements Table Block Properties

You can adjust these properties in the **Properties** tab of the Property Inspector or in the **General** tab of the Model Explorer or Table Properties window.

Update method

Specifies the method used to activate the Requirements Table block.

Update Method	Description
Inherited (default)	Input from the Simulink model activates the Requirements Table block. If you define input data, the Requirements Table block samples at the rate of the fastest input data. If you do not define input data, the Requirements Table block samples at the rate defined by the execution behavior of the parent subsystem.
Discrete	Sample the Requirements Table block by using the rate you specify in the Sample time property of the Requirements Table block. The Requirements Table block generates an implicit event at regular time intervals that correspond to the specified rate. Note that other blocks in the model can have different sample times.


Support variable-size arrays

Specifies that the Requirements Table block supports output and local data that varies in dimension during simulation. See “Variable size” on page 2-55.

Saturate on integer overflow

Specifies that integer overflows saturate in the generated code. See “Saturation and Wrapping” (Fixed-Point Designer).

Enable outputs in preconditions

Allow data with **Scope** set to Output to be used in preconditions. If you disable this property, the block highlights the cell and displays an alert icon  in preconditions that use output data, unless the data is an input argument of `getPrevious`. However, if `getPrevious` attempts to return the value of the data at a time step when it was not defined, `getPrevious` returns undefined behavior.

When you use output data in preconditions, you may encounter read-before-write issues in your requirements during simulation. To resolve these issues, specify your requirements in order. For more information, see “Identify Inconsistent and Incomplete Formal Requirement Sets” on page 2-17 and “Leverage Evaluation Order of Formal Requirements” on page 2-43.

Fixed-Point Properties

You can adjust these properties in the **Properties** tab of the Property Inspector or by opening the **Fixed-point properties** tab in the Model Explorer or the Table Properties window.

Requirements Table `fimath`

Specifies the `fimath` properties for the Requirements Table block. The `fimath` defined in the **Requirements Table `fimath`** text box behaves as a global `fimath` for the contents of the Requirements Table block. The block associates the `fimath` properties in the **Requirements Table `fimath`** text box with the fixed-point and integer input signals to the Requirements Table block that you choose to treat as `fi` objects. When you construct `fi` objects in the Requirements Table block, note that:

- If no `fimath` is associated with a `fi` object when it is constructed, then the `fi` constructor uses the default `fimath` settings regardless of the properties in the **Requirements Table `fimath`** text box. However, if you perform additional operations on the `fi` object after it is constructed, the object adopts the properties in the **Requirements Table `fimath`** text box.
- If you specify a `fimath` in the `fi` constructor, then that `fimath` is obeyed when quantizing the value in the `fi` constructor. `fimath` settings not specified in the `fi` constructor use the specified properties in the **Requirements Table `fimath`** text box.

You can select one of these options:

Setting	Description
Same as MATLAB	The block uses the same <code>fimath</code> properties as the current default <code>fimath</code> . The text box is dimmed and displays the current global <code>fimath</code> in read-only form.
Specify Other	You can specify your own <code>fimath</code> object in the text box. You can do so by either: <ul style="list-style-type: none"> • Constructing the <code>fimath</code> object inside the text box. • Constructing the <code>fimath</code> object in the MATLAB or model workspace and then entering its variable name in the text box. If you use this option and plan to share your model with others, define the variable in the model workspace. For more information on <code>fimath</code> objects, see “<code>fimath</code> Object Construction” (Fixed-Point Designer).

Description and Document Link Properties

You can set description and document link properties for the Requirements Table block in the **Info** tab of the Property Inspector, or in the **Documentation** tab of the Model Explorer or Table Properties window.

Description

Specifies the description of the Requirements Table block. You can enter a brief description and comments.

See Also

Requirements Table

Related Examples

- “Use a Requirements Table Block to Create Formal Requirements” on page 2-2
- “Define Data in Requirements Table Blocks” on page 2-53
- “Set Data Types in Requirements Table Blocks” on page 2-59

Define Data in Requirements Table Blocks

In this section...

“Create and Delete Data” on page 2-53

“Set General Data Properties” on page 2-53

“Set Limit Range Properties” on page 2-57


“Set Logging Properties” on page 2-57



“Set Description Properties” on page 2-58




Requirements Table blocks manage simulation information by using data that represents block inputs, outputs, parameters, local data, and constants. You can define or delete data in the Requirements Table block by using the **Symbols** pane or the Model Explorer. Then you can set the properties of the data in the Property Inspector or Model Explorer.

Create and Delete Data

You can define or delete data by using the **Symbols** pane or the Model Explorer.

To use the **Symbols** pane, open the block by double-clicking it. In the **Modeling** tab, in the **Design** section, click **Symbols Pane**. In the **Symbols** pane, click the Create Data button . Delete data by right-clicking the data and clicking **Delete**.

To use the Model Explorer. In the **Modeling** tab, in the **Design** section, click **Model Explorer**. In the **Model Hierarchy** pane, expand the model tree view and select the **Requirements Table** block. Add data by clicking **Add > Data** or the Add Data button . Delete data by selecting the data name and clicking **Edit > Delete** or the Delete button .

You must define and use the data in the block in either table of the **Requirements** or **Assumptions** tabs. If you define data without also entering it in either table, the Unused symbol icon  appears next to the data in the **Symbols** pane. Enter the data in either table to remove the icon. If you enter data in either table without defining it, the Undefined Symbol icon  appears next to data in the **Symbols** pane. To define the data that you have already entered in either table, click the Resolve Undefined Symbols button .

Set General Data Properties

You can modify the properties of data by using the **Symbols** pane and the Property Inspector or by using the Model Explorer.

To edit the properties by using the **Symbols** pane, open the properties in the Property Inspector by using one of these approaches:

- Right-click the data name and click **Inspect**.
- Click the data name. In the **Modeling** tab, in the **Design Data** section, click **Property Inspector**.






To edit the properties by using Model Explorer, click the data, then edit the properties in the **General** tab.

Name

Specifies the name of the data. Use the same naming conventions used in MATLAB. You can also modify this property in the **Symbols** pane directly.

Scope

Specifies where the data resides in memory relative to the block. This property determines the range of functionality of the data. You can also modify this property in the **Symbols** pane in the **Type** column. You can set **Scope** to one of these values:

Scope	Description
Local	The data is defined in the current block only. You must define local data at each time step. The block clears local data from memory at the end of each time step. The Symbols pane indicates data have Scope set to Local with the Local Data icon  .
Constant	The data is a read-only constant value that is visible to the block at each time step. The Symbols pane indicates data have Scope set to Constant with the Constant Data icon  .
Parameter	The data resides in a variable of the same name in the MATLAB workspace, model workspace, or in the workspace of a masked subsystem that contains this block. Parameter data is visible to the block at each time step. If a variable of the same name exists in more than one of the workspaces visible to the block, the block uses the variable closest to the block in the workspace hierarchy. For more information, see "Model Workspaces" (Simulink). The Symbols pane indicates data have Scope set to Parameter with the Parameter Data icon  .
Input	The data is an input signal to the Requirements Table block. The Symbols pane indicates data have Scope set to Input with the Input Data icon  .
Output	The data is an output signal of the Requirements Table block. You must define output data at each time step. The Symbols pane indicates data have Scope set to Output with the Output Data icon  .

The **Precondition**, **Postcondition**, and **Action** columns place some restrictions on the **Scope** the data can use.

- Entries in the **Precondition** column must use at least one input data.
- Entries in the **Postcondition** column must use at least one input data where the **Treat as design model output for analysis** property is enabled.
- Actions cannot reassign values of input data.

Port

Specifies the index of the port associated with the data. You can set this property in the **Symbols** pane in the **Port** column. This property applies only to data with the **Scope** property set to Input or Output.

Data must resolve to signal object

Specifies that the data explicitly inherits properties from Simulink.Signal objects of the same name in the MATLAB base workspace or the Simulink model workspace. When enabled, the data can inherit these properties:

- Size
- Complexity
- Type
- Unit
- Minimum value
- Maximum value
- Initial value
- Storage class
- Sampling mode

This property applies only to data with the **Scope** property set to Output or Local. This property appears only if you set the configuration parameter **Signal resolution** to a value other than None. For more information, see “Symbol Resolution” (Simulink).

Tunable

Specifies whether the parameter used as the source of the data is tunable. For more information, see “Tunable Parameters” (Simulink). This property applies only to data with the **Scope** property set to Parameter. Clear this option if the parameter must be a constant expression.

Size

Specifies the size of the data. This property can be a scalar value or a MATLAB vector of values. **Size** defaults to -1 , which means that the size is inherited. For more information, see “Specify Size of Requirements Table Block Data” on page 2-67.

Variable size

Specifies whether the size of the data is variable. This property only applies to data with the **Scope** property set to Output or Local. Input data inherit size variability from their corresponding signals.

This property appears only if the Requirements Table block property **Support variable-size arrays** is enabled. See “Support variable-size arrays” on page 2-50.

Complexity

Specifies real or complex data. Set **Complexity** to one of these values:


Complexity	Description
Inherited	The data inherits complexity based on the Scope property. Input and output data inherit complexity from the Simulink signals connected to them. Local and parameter data inherits complexity from the parameter to which the data is bound.
Off	The data is a real number.
On	The data is a complex number.

This property does not apply to data with the **Scope** property set to Constant.

Type

Specifies the data type for the data. You can specify the data type by:

- Selecting a built-in type from the **Type** drop-down list.
- Entering an expression in the **Type** field that evaluates to a data type.
- In the Model Explorer, using the Data Type Assistant to specify the **Mode** property, then specifying the data type based on that mode.

Note To display the Data Type Assistant, click the Show data type assistant button .

For more information, see “Set Data Types in Requirements Table Blocks” on page 2-59.

Lock data type against Fixed-Point tools

Prevents replacement of the current fixed-point type with an autoscaled type chosen by the **Fixed-Point Tool**. See “Autoscaling Data Objects Using the Fixed-Point Tool” (Fixed-Point Designer).

Treat as design model output for analysis

Specifies data that the Requirements Table block identifies as output signals from the attached model. If you use postconditions, you must enable this property on at least one data in each postcondition. If you use only preconditions and actions, do not enable this property. This property applies only when the **Scope** property is Input. Input data with this property enabled are called design model outputs.

Unit (e.g. m, m/s², N*m)

Specifies the physical units for the data. By default, this property inherits the unit from the Simulink signal on the corresponding input or output port. This property applies only to data with the **Scope** property set to Input or Output.

Constant value

Specifies the value of the data. Modify this property in the **Symbols** pane directly in the **Value** column. When you leave the **Constant value** field blank, the data resolves to a default value of 0.

Set Limit Range Properties

You can set the range of acceptable values for data in the **Limit range** section of the Property Inspector or Model Explorer. The Requirements Table block uses this range to validate the values used by the data as they enter, are used by, or leave the block. In some instances, you can also define the limit range by using the table in the **Assumptions** tab. See “Add Assumptions to Requirements” on page 2-10. You can enter an expression or parameter that evaluates to a numeric scalar. Specify the range by using two properties. These properties apply only to data with the **Scope** property set to Input, Output, or Local.

Minimum

Specifies the smallest value allowed for the data during simulation. The default value is `-inf`.

Maximum

Specifies the largest value allowed for the data during simulation. The default value is `inf`.

Set Logging Properties

You can set logging properties in the **Properties** tab of the Property Inspector or the **Logging** tab of the Model Explorer. For more information on signal logging, see “Save Signal Data Using Signal Logging” (Simulink). These properties can only be assigned to data with the **Scope** property set to Output or Local.

Log signal data

Whether to enable signal logging. Signal logging saves the values of the data to the MATLAB workspace during simulation.

Logging Name

Signal name used to log the data.

- To use the name of the data, select `Use signal name`.
- To specify a different name, select `Custom` and enter the custom logging name.

Limit data points to last

Whether to limit the number of data points to log. For example, if you set the maximum number of data points to 5000, the block logs only the last 5000 data points generated by the simulation.

Decimation

Whether to limit the amount of logged data by skipping samples using the specified decimation interval. For example, if you set a decimation interval of 2, the block logs every other sample.

Set Description Properties

After clicking the data you want to modify in the Model Explorer, you can set the following properties in the **Description** tab. You can also set the following properties with the Property Inspector by clicking the **Properties** tab.

Description

Specifies the description of the data.

See Also

Requirements Table

Related Examples

- “Specify Requirements Table Block Properties” on page 2-50
- “Set Data Types in Requirements Table Blocks” on page 2-59

Set Data Types in Requirements Table Blocks

In this section...
“Specify Data Types” on page 2-59
“Inheriting Data Types” on page 2-60
“Specify Built-In Data Types” on page 2-60
“Fixed-Point Designer Data Properties” on page 2-61
“Enumerated Data Types” on page 2-65
“Bus Objects” on page 2-65
“Expression Data Types” on page 2-65

When you create data in a Requirements Table block, you can use the **Type** property to set the data type. Data can inherit their data types, or be set to built-in, fixed-point, or enumerated data types. Data can also be nonvirtual buses. By default, Requirements Table block data inherit their data type.

For more information about creating data, see “Define Data in Requirements Table Blocks” on page 2-53.

Specify Data Types

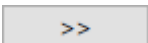
You can specify the data types by using the **Symbols** pane and Property Inspector, or the Model Explorer.

To specify the data type using the **Symbols** pane and Property Inspector:

- 1 Open the Requirements Table block.
- 2 Open the **Symbols** pane. In the **Modeling** tab, in the **Design Data** section, click **Symbols Pane**.
- 3 Right-click the data you want to modify and click **Inspect** to open the data properties in the Property Inspector.
- 4 In the **Properties** tab, select the data type in the **Type** property.

To specify the data type using the Model Explorer:

- 1 Open the Model Explorer. In the **Modeling** tab, in the **Design Data** section, click **Model Explorer**.
- 2 In the **Model Hierarchy** pane, expand the model tree view and select the Requirements Table block.
- 3 Click the data you want to modify.
- 4 Select the data type in the **Type** property.

In the Model Explorer, you can also filter the data type options. In the **General** tab, click the Show data type assistant button  to display the Data Type Assistant. Then choose an option from the **Mode** drop-down menu. The available data types depend on the mode you select:

Mode	What to Specify
Inherit (default)	The data type is inherited based on the Scope property: <ul style="list-style-type: none"> • If Scope is Input, the data type is inherited from the input signal on the designated port. • If Scope is Output, the data type is inherited from the output signal on the designated port. • If Scope is set to the other options, the data type is inherited from the associated parameter, which can be defined in the Simulink masked subsystem or the MATLAB workspace.
Built in	Select a supported built-in data type.
Fixed point	Specify the fixed-point data properties.
Enumerated	Enter the name of a <code>Simulink.IntEnumType</code> object that you define in the base workspace.
Bus Object	Enter the name of a <code>Simulink.Bus</code> object to define the properties of a MATLAB structure. You must define the bus object in the base workspace. Note You can click the Edit button to create or modify <code>Simulink.Bus</code> objects using the Simulink Type Editor.
Expression	Enter an expression that evaluates to a data type.

Inheriting Data Types

Requirements Table block data can inherit their data types, including fixed-point types, from their connected signals. To set data to inherit its data type:

- 1 Select the data in the Model Explorer. Alternatively, select the data in the **Symbols** pane and open the Property Inspector.
- 2 In the Model Explorer or Property Inspector, set **Type** to **Inherit: Same as Simulink**.

Data with the **Scope** property set to **Local**, **Parameter**, **Input**, or **Output** can also inherit complexity from the information sent to it. To inherit complexity, set **Complexity** to **Inherited**.

After you build the model, the **CompiledType** column of the Model Explorer shows the actual data type inherited from Simulink. If the expected type matches the inferred type, the Requirements Table block inherits the data type.

Specify Built-In Data Types

In the Model Explorer, when you expand the Data Type Assistant and set **Mode** to **Built in**, you can set **Type** to these built-in data types. The built-in data types are:

Data Type	Description
double	64-bit double-precision floating point
single	32-bit single-precision floating point

Data Type	Description
half	A half-precision data type occupies 16 bits of memory, but its floating-point representation enables it to handle wider dynamic ranges than integer or fixed-point data types of the same size. See “The Half-Precision Data Type in Simulink” (Fixed-Point Designer).
int64	64-bit signed integer
int32	32-bit signed integer
int16	16-bit signed integer
int8	8-bit signed integer
uint64	64-bit unsigned integer
uint32	32-bit unsigned integer
uint16	16-bit unsigned integer
uint8	8-bit unsigned integer
boolean	Boolean
string	String scalar

Fixed-Point Designer Data Properties

To represent data as fixed-point numbers in Requirements Table blocks, you must install Fixed-Point Designer™.

You can set the following fixed-point properties:

Signedness

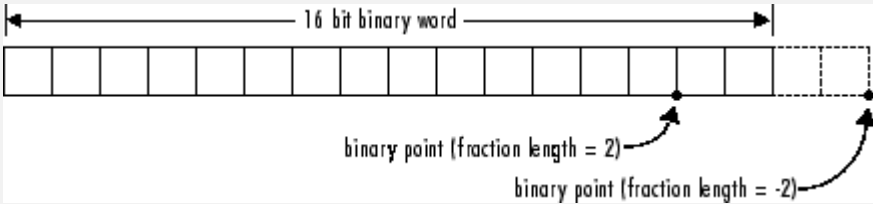
Select whether you want the fixed-point data to be **Signed** or **Unsigned**. Signed data can represent positive and negative quantities. Unsigned data represents positive values only. The default is **Signed**.

Word length

Specify the size, in bits, of the word that will hold the quantized integer. Large word sizes represent large quantities with greater precision than small word sizes. Word length can be any integer between 0 and 128 bits. The default is 16.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization issues. You can select these scaling modes:

Scaling Mode	Description
Binary point (default)	<p>The Data Type Assistant displays the Fraction Length parameter, which specifies the binary point location.</p> <p>Binary points can be positive or negative integers. A positive integer moves the binary point left of the rightmost bit by that amount. For example, an entry of 2 sets the binary point in front of the second bit from the right. A negative integer moves the binary point further right of the rightmost bit by that amount, as in this example:</p>  <p>The diagram shows a horizontal row of 16 boxes representing bits. Above the boxes, a double-headed arrow spans the entire row and is labeled "16 bit binary word". Below the boxes, two arrows point to specific bit positions. The first arrow points to the second bit from the right and is labeled "binary point (fraction length = 2)". The second arrow points to the second bit from the left and is labeled "binary point (fraction length = -2)".</p> <p>The default is 0.</p>
Slope and bias	<p>The Data Type Assistant displays the Slope and Bias parameters:</p> <ul style="list-style-type: none"> • Slope can be any positive real number. The default is 1.0. • Bias can be any real number. The default value is 0.0. <p>You can enter slope and bias as expressions that contain parameters defined in the MATLAB workspace.</p>

Note Use binary-point scaling whenever possible to simplify the implementation of fixed-point numbers in generated code. Operations with fixed-point numbers that use binary-point scaling are performed with simple bit shifts and eliminate the expensive code implementations required for separate slope and bias values.

Data type override

Specify whether the data type override setting is **Inherit** or **Off**. See “Fixed-Point Instrumentation and Data Type Override” (Fixed-Point Designer).

Calculate Best-Precision Scaling

Have Simulink automatically calculate best-precision values for both **Binary point** and **Slope and bias** scaling, based on the **Minimum** and **Maximum** properties you specify.

To automatically calculate best precision scaling values:

- 1 Specify the **Minimum** or **Maximum** properties.
- 2 Click **Calculate Best-Precision Scaling**.

Simulink calculates the scaling values, then displays them in either the **Fraction length**, or the **Slope** and **Bias** fields.

Note The **Minimum** and **Maximum** properties do not apply to data with the **Scope** property set to Constant or Parameter. The software cannot calculate best-precision scaling for these kinds of data.

Fixed-point details

Displays information about the fixed-point data that is defined in the Data Type Assistant:

- Minimum and Maximum show the same values that you specify in the **Minimum** and **Maximum** properties.
- Representable minimum, Representable maximum, and Precision show the minimum value, maximum value, and precision that the fixed-point data can represent.

Data data

General | Logging | Description

Name:

Scope:

Data must resolve to signal object

Size: Variable size

Complexity:

Type:

Data Type Assistant

Mode: Signedness: Word length:

Scaling: Fraction length:

Data type override:

Fixed-point details

Representable maximum:	32767
Maximum:	10000
Minimum:	-20
Representable minimum:	-32768

Precision:

Lock data type against Fixed-Point tools

Limit range

Minimum: Maximum:

If the value of a field cannot be determined without first compiling the model, the **Fixed-point details** subpane shows the value as Unknown. The values displayed by the **Fixed-point details** subpane do not automatically update if you change the values that define the fixed-point data. To update the values shown in the **Fixed-point details** subpane, click **Refresh Details**.

Clicking **Refresh Details** does not modify the data. It changes only the display. To apply the displayed values, click **Apply** or **OK**.

The **Fixed-point details** subpane indicates issues resulting from the fixed-point data specification. For example, this subpane shows two issues.

The screenshot shows the 'Data data' configuration window with the following settings:

- General** tab selected.
- Name: data
- Scope: Local
- Data must resolve to signal object
- Size: [empty] Variable size
- Complexity: Off
- Type: fixdt(1,16,0)
- Data Type Assistant**
 - Mode: Fixed point
 - Signedness: Signed
 - Word length: 16
 - Scaling: Binary point
 - Fraction length: 0
 - Data type override: Inherit
 - Calculate Best-Precision Scaling
- Fixed-point details**

Representable maximum:	32767	
Maximum:	50000	Outside representable range by 17233 (17233 x precision)
Minimum:	MySymbol	Cannot evaluate
Representable minimum:	-32768	

Precision: 1
- Lock data type against Fixed-Point tools
- Limit range**
 - Minimum: MySymbol
 - Maximum: 50000

Buttons: Revert, Help, Apply, Refresh Details

The row labeled **Maximum** indicates that the value specified by the **Maximum** property is not representable by the fixed-point data. To fix the issue, make one of these modifications so the fixed-point data can represent the maximum value:

- Decrease the value in the **Maximum** property.
- Increase **Word length**.
- Decrease **Fraction length**.

The row labeled Minimum shows the message Cannot evaluate because evaluating the expression MySymbol, specified by the **Minimum** property, does not return a numeric value. When an expression does not evaluate, the **Fixed-point details** subpane shows the unevaluated expression (truncating to 10 characters) in place of the unavailable value. To fix this issue, define MySymbol in the base workspace to provide a numeric value.

If you click **Refresh Details**, the issue indicators and descriptions are removed and the value of MySymbol appears in place of the unevaluated text.

Enumerated Data Types

In the Model Explorer or the Property Inspector, you can specify enumerated data explicitly or make the data inherit it. To explicitly set data to an enumerated type, set **Type** to Enum: <class name> and replace <class name> with the name of an enumerated data type that you define in a MATLAB file on the MATLAB path. To inherit the enumerated type from a connected Simulink signal, set **Type** to Inherit: Same as Simulink. You can only inherit enumerated type data when the **Scope** property is Input. For more information, see “Use Enumerated Data in Simulink Models” (Simulink).

Bus Objects

In the Model Explorer or Property Inspector, when you set **Type** to Bus: <object name>, you can set the data type to a bus. Replace <object name> with the name of the Simulink.Bus. Requirements Table blocks support only nonvirtual buses. See “Composite Interface Guidelines” (Simulink). For Requirements Table block bus inputs, incoming virtual bus signals are converted to nonvirtual buses.

You can connect bus inputs and outputs from Requirements Table blocks to other bus signals, including:

- Blocks that output bus signals, such as Bus Creator blocks.
- Blocks that accept bus signals as an input, such as Bus Selector and Gain blocks.
- S-Function blocks.
- Other Requirements Table blocks.

Expression Data Types

You can specify the types of Requirements Table block data as expressions by using the Model Explorer or the Property Inspector.

To use the Model Explorer, set the **Mode** property to Expression. In the **Type** property, replace <data type expression> with an expression that evaluates to a data type.

To use the Property Inspector, double-click the **Type** property, clear the contents, and enter an expression.

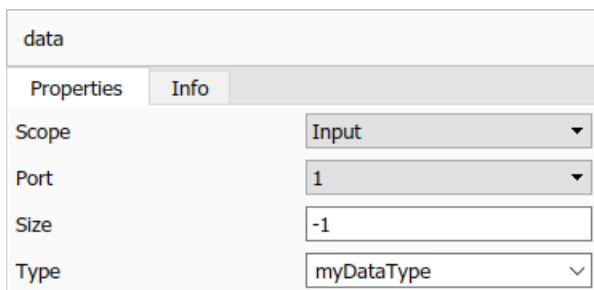
You can use the following expressions:

- Alias type from the MATLAB workspace, as described in `Simulink.AliasType`.
- `fixdt` function to create a `Simulink.NumericType` object describing a fixed-point or floating-point data type.
- `type` operator, to base the type on previously defined data.

For example, suppose you want to designate the variable `myDataType` as an alias for a `single` data type to use as an expression in the **Type** property of an input data. Create an instance of the `Simulink.AliasType` class and set its `BaseType` property by entering these commands:

```
myDataType = Simulink.AliasType;  
myDataType.BaseType = "single";
```

In the Property Inspector, enter the data type alias name, `myDataType`, as the value in the **Type** property.



data	
Properties	Info
Scope	Input
Port	1
Size	-1
Type	myDataType

Note Requirements Table blocks do not support code generation if one of the data uses an alias type and is variable size. This limitation does not apply to block input, output, or local data. For more information on variable-size data, see “Variable size” on page 2-55.

See Also

Requirements Table

Related Examples

- “Use a Requirements Table Block to Create Formal Requirements” on page 2-2
- “Define Data in Requirements Table Blocks” on page 2-53

Specify Size of Requirements Table Block Data

You can specify the size of the data in Requirements Table blocks with the **Symbols** pane and Property Inspector, or with the Model Explorer. Requirements Table blocks can use scalars, vectors, or matrices. For more information about creating, deleting, and setting properties for data, see “Define Data in Requirements Table Blocks” on page 2-53 and “Set Data Types in Requirements Table Blocks” on page 2-59. You can set data to inherit the size or manually specify the size.

Inherit Size from Specified Source

If you want data to inherit its size from a port, constant value, specified value, or a parameter, set the **Size** property to -1. The **Scope** property determines the source of where the data inherits size.

Scope	Description
Input	Inherits size from the Simulink signal connected to the associated input port.
Output	Inherits size from the Simulink signal connected to the associated output port.
Constant	Inherits size from the assigned value shown in the Symbols pane.
Local	Inherits size from the assigned value.
Parameter	Inherits size from the associated Simulink or MATLAB parameter.

Customize Data Sizes

You can manually set the size of data to a scalar, vector, or matrix. To specify the size as a scalar, set **Size** to 1 or clear the property. To specify the size as a vector or a matrix, enter a row vector with positive integers in [row column] format. For example, to define a column vector of size 6, set the **Size** property to [6 1]. To define a row vector of size 5, set the **Size** property to [1 5]. To define a matrix of data size 3-by-3, set the **Size** property to [3 3].

You can also set the **Size** property with an expression. The expressions can include:

- A numeric constant
- Arithmetic operators, restricted to +, -, *, and /
- Names of data with the **Scope** property set to **Parameter**
- Calls to the MATLAB functions `min`, `max`, and `size`

These expressions must output a positive integer or two positive integers in [row column] format. Otherwise, the specified size produces an error at model compilation. For example, consider a Requirements Table block with data `k`, `x`, and `y` that have the **Scope** property set to **Parameter**. Here, `k` is a positive integer and `x` and `y` are matrices. If you create new data for the block, you can define the **Size** property with any of the following expressions:

```
k+1
size(x)
min(size(y))
```

Simulation and Size Matching

After you build the model, the **CompiledSize** column in the Model Explorer displays the actual size used in the simulation. If the value of the **Size** property is not -1 and does not match the actual size, a mismatch error occurs during model compilation.

See Also

Requirements Table

Related Examples

- “Define Data in Requirements Table Blocks” on page 2-53
- “Set Data Types in Requirements Table Blocks” on page 2-59

What Is a Specification Model?

When you systematically verify a model against requirements, you generate test cases for each requirement. These tests validate the model, which you can use to generate production code and build confidence that your model satisfies requirements. To create tests that satisfy your requirements, you can construct a *specification model*. A specification model is an executable entity that you can use to perform requirements-based testing by using Simulink Design Verifier and Requirements Toolbox.

If you have a set of requirements written in natural language text, you can express them as formal requirements by using a Requirements Table block. After defining the requirements in one or more blocks, the blocks and the signals become the specification model. Unlike the model that you want to test, known as the *design model*, the specification model only specifies what to do, not how to do it.

You can use a specification model to:

- Validate the set of requirements in a systematic and quantitative manner.
- Automate requirements-based testing.
- Identify issues with your design model and requirements.

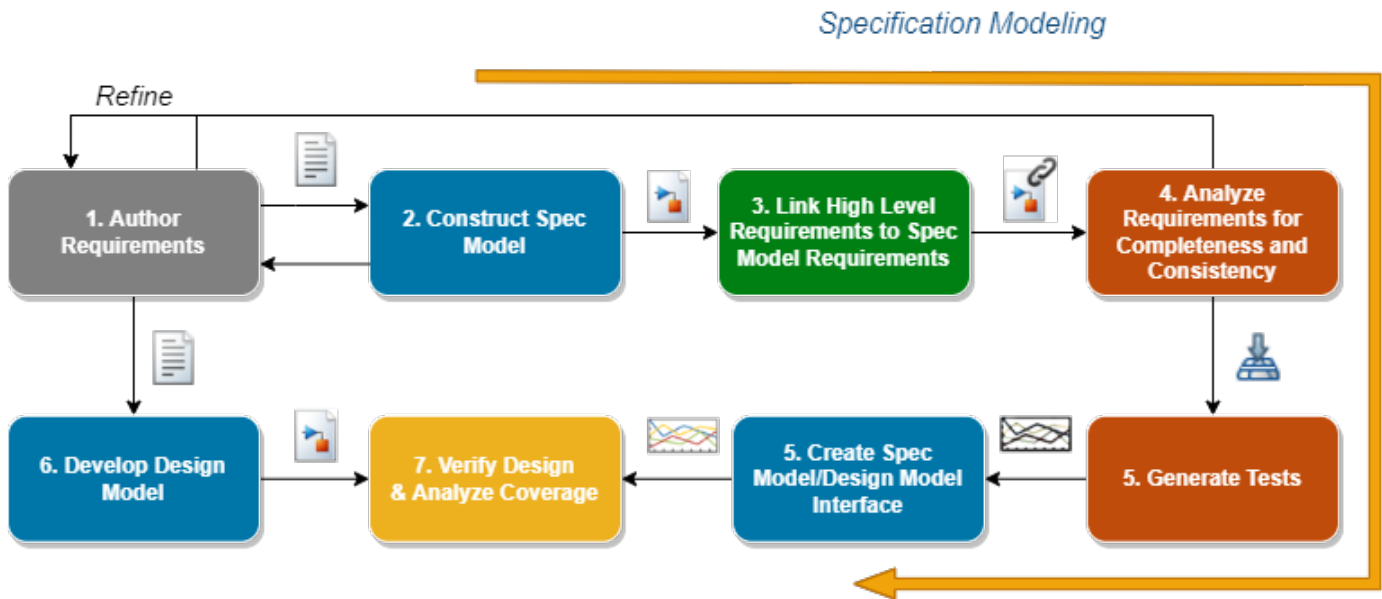
Use Specification Models in Requirements-Based Testing

To create and deploy a specification model, follow these steps:

- 1 Author the requirements — Write your requirements in natural language text that describes the behavior of the system under design. Author them directly in the **Requirements Editor** or import them. For more information on importing requirements, see “Import Requirements from Third-Party Applications” on page 1-8.
- 2 Construct the specification model — Design the specification model as an formal representation of the requirements by using at least one Requirements Table block.
- 3 Link the requirements — Each requirement that you create in the Requirements Table block creates an equivalent requirement in the **Requirements Editor**. See “Configure Properties of Formal Requirements” on page 2-27. Link the high-level requirements to the formal requirements from the specification model.
- 4 Analyze the formal requirements for completeness and consistency — Identifying incomplete and inconsistent requirement sets can be difficult to do manually. The Requirements Table block allows you to automatically analyze the requirements for these issues. See “Identify Inconsistent and Incomplete Formal Requirement Sets” on page 2-17.
- 5 Generate tests for the specification model — Generate at least one test per requirement that demonstrates its conformance to that requirement. For more information on generating tests, see “Generate Test Cases for a Subsystem” (Simulink Design Verifier). Simulink Design Verifier automatically creates test objectives from the requirements defined in Requirements Table blocks.
- 6 Interface the specification model with the design model — The specification and design models often do not use identical input and output signals. Convert the test cases that you generate in step 5 by developing an interface between both models.
- 7 Develop the design model — Develop the design model by using the requirements. Link the requirements to the design model.
- 8 Verify the design and analyze the coverage — Run the tests generated in step 5 on the design model and verify whether the results agree with the specification model and requirements.

Generate a coverage report to identify the missing coverage and refine the requirements, if required.

This flow chart illustrates this process.



Construct a Specification Model

Consider the autopilot controller model described in “Use Specification Models for Requirements-Based Testing” on page 2-91. In this example, you develop requirements that contain logical and temporal conditions that define outputs.

Identify the Specification Model Interface

List the input and output signals for the specification model that are related to the requirements that you want to test. Ignore the signals that the requirements do not specify and that do not affect the tested outputs. In this example, the requirements specify five inputs and two outputs. The specification model input signals are:

- 1 Autopilot Engage Switch — A switch that enables or disables the autopilot controller
- 2 Heading Engage Switch — A switch that specifies the mode of the autopilot controller when the autopilot switch is engaged
- 3 Roll Reference Target Turn Knob — A knob that feeds the desired roll angle value to the autopilot controller
- 4 Heading Reference Turn Knob — A knob that gives the set-point value for heading mode
- 5 Aircraft Roll Angle — The current roll angle of the aircraft

The output signals are:

- 1 Aileron Command — The output to the aileron actuator
- 2 Roll Reference Command — The output on the display window that indicates the set-point value for the aileron actuator

Identify Preconditions, Postconditions, and Actions for Each Requirement

For the requirements that you want to verify, transform the textual requirements into logical expressions that can be represented as preconditions, postconditions, and actions. You define formal requirements as a combination of Preconditions, Postconditions, and Actions:

- **Precondition** — A condition that must be true for a specified duration before evaluating the rest of the requirement
- **Postcondition** — A condition that must be true if the associated precondition is true for the specified duration
- **Action** — A behavior that must be performed if the associated precondition is true for the specified duration

You may find that some requirements can use a postcondition or an action interchangeably, or both postconditions and actions. Specify which you want to use based on the configuration of your design model.

For example consider this high level requirement that specifies the modes of the autopilot controller:

The autopilot controller mode is determined by the following:

- The autopilot controller is OFF when the autopilot engage switch is not engaged.
- The autopilot controller is ROLL_HOLD_MODE when the autopilot engage switch is engaged and the heading engage switch is not engaged.
- The autopilot controller is HDG_HOLD_MODE when the autopilot engage switch and the heading engage switch are both engaged.

You can write these requirements as these logical expressions:

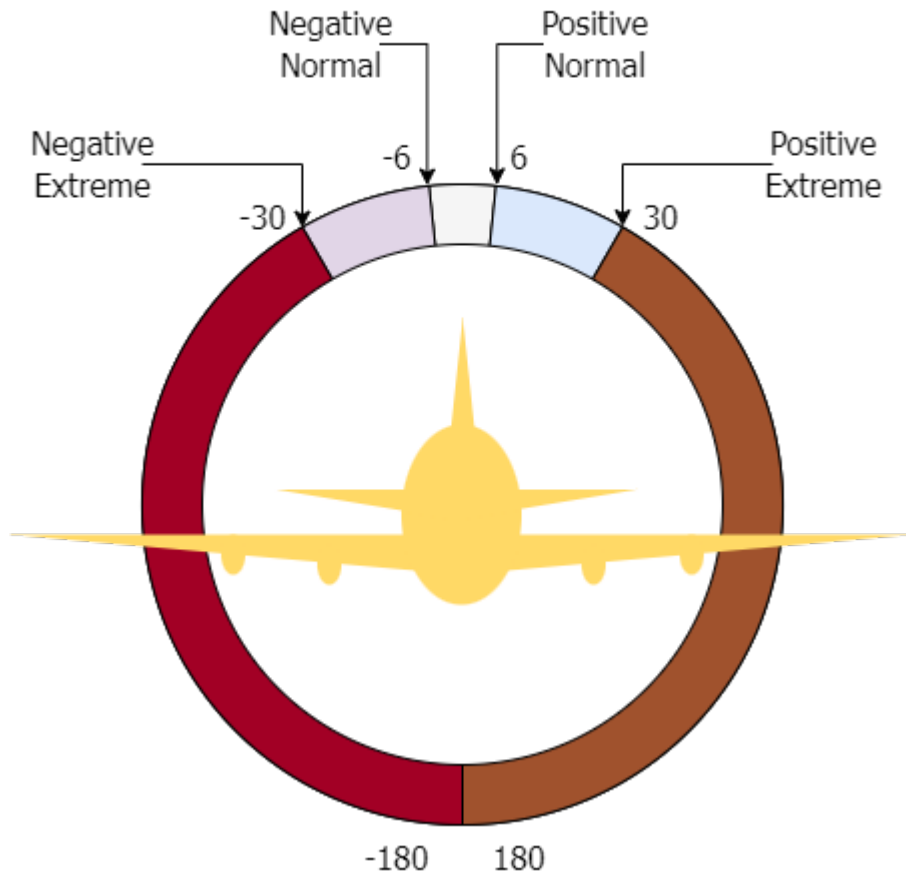
Requirement	Precondition	Action
1	AP_Engage_Switch == false	Mode = Off
2	AP_Engage_Switch == true && HDG_Engage_Switch == false	Mode = ROLL_HOLD_MODE
3	AP_Eng_Switch == true && HDG_Engage_Switch == true	Mode = HDG_Hold_Mode

Repeat this process for the remaining requirements.

Identify Design Values Representations in Requirements

Your requirements may specify ranges of values that your design model must satisfy, or you may want to parameterize the values that you evaluate in each requirement. These values cannot always be described easily with literal values. You can use the Requirements Table block to represent values in the expressions as constant or parameter data. See “Define Data in Requirements Table Blocks” on page 2-53. You can change data throughout simulation. In addition to assigning numerical values to data, the block supports other data types, such as strings, enumerations, or ranges. Use the representation of values that fits your needs.

In the autopilot controller model, the requirements specify threshold values for the aircraft roll angle. This graphic illustrates the numerical and verbal equivalents of the thresholds.



Create the Requirements Table Blocks

After identifying the signal representations, values, and the expressions that you want to use in the formal requirements, write the logical expression of the precondition, postconditions, and actions in the **Precondition Postcondition**, and **Action** columns for each requirement respectively. If your requirements have children or dependencies, you can include those relationships in the block. See “Establish Hierarchy in Requirements Table Blocks” on page 2-45.

Each requirement that you create in the Requirements Table block creates an equivalent requirement in the **Requirements Editor**. Update additional textual properties of the requirements, such as the description, in the editor. See “Configure Properties of Formal Requirements” on page 2-27.

In the autopilot controller model, the specification model includes two Requirements Table blocks. `AP_Mode_Determination` defines the formal requirements for the autopilot controller mode.

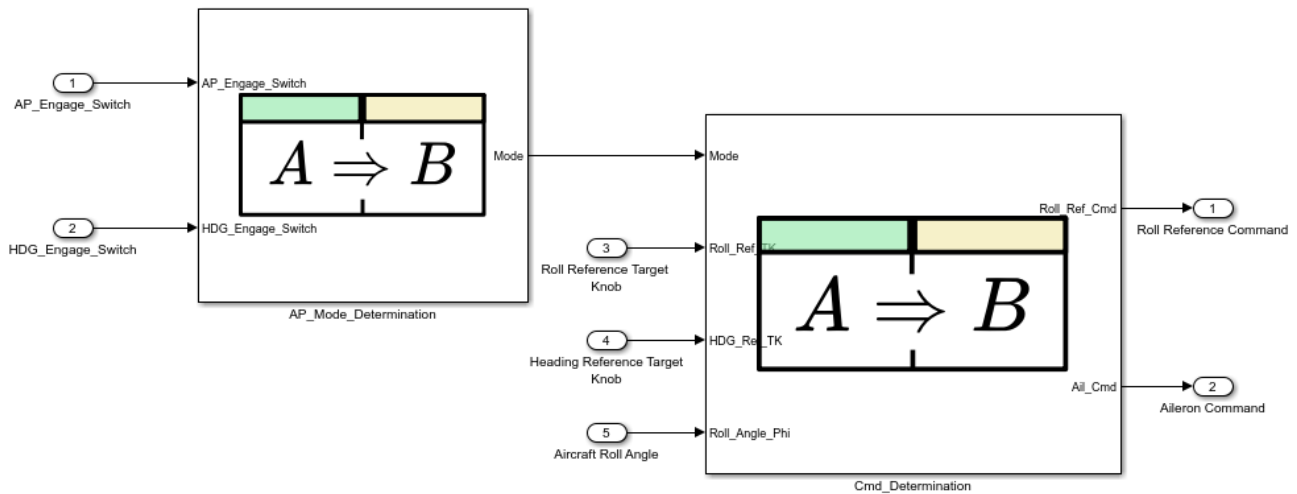
Requirements		Assumptions		
Index	Summary	Precondition		Action
		AP_Engage_Switch	HDG_Engage_Switch	Mode
1	AP_Engage_Switch and HDG_Engage_Switch both engaged	true	true	HDG_HOLD_MODE
2	AP_Engage_Switch is engaged and HDG_Engage_Switch is not engaged	true	false	ROLL_HOLD_MODE
3	AP_Engage_Switch is not engaged	false		OFF

The other Requirements Table block, `Cmd_Determination`, describes the desired output of the aileron command and the roll reference command.

Requirements		Assumptions				
Index	Summary	Precondition			Action	
		Mode	Roll_Ref_TK	prev(Roll_Angle_Phi)	Roll_Ref_Cmd	All_Cmd
1	Autopilot mode is OFF	OFF			0	Zero
2	ROLL_HOLD_MODE becomes active mode	hasChangedTo(X,ROLL_HOLD_MODE)				All
2.1	Roll_Ref_TK between [-30, -3] or [+3, +30] degrees		[TK_neg_extreme, TK_neg_norm] [TK_pos_norm, TK_pos_extreme]			Roll_Ref_TK
2.2	Roll_Ref_TK greater than -3 and less than +3:		(TK_neg_norm, TK_pos_norm)			
2.2.1	Roll_Angle_Phi greater than neg_norm and less than pos_norm			[phi_neg_norm, phi_pos_norm]	0	
2.2.2	Roll_Angle_Phi greater than +30			> phi_pos_extreme	TK_pos_extreme	
2.2.3	Roll_Angle_Phi less than -30			< phi_neg_extreme	TK_neg_extreme	
2.2.4	Otherwise, Roll_Ref_Cmd default setting	Else			Roll_Angle_Phi	
3	HDG_HOLD_MODE becomes active mode	hasChangedTo(X,HDG_HOLD_MODE)				HDG_Ref_TK
4	Otherwise, Roll_Ref_Cmd shall hold the previous value of Roll_Ref_Cmd	Else			prev(Roll_Ref_Cmd)	All

Final Specification Model

After connecting the Requirements Table blocks to the inputs, outputs, and each other, the final specification model is:



Prepare the Specification Model for Test Generation

Simulink Design Verifier automatically creates test objectives from the requirements defined in Requirements Table blocks. If you need to constrain the values of the test objectives, you can specify them either in the signal source, or by including them in the **Assumptions** table of the block. See “Add Assumptions to Requirements” on page 2-10. To prepare the specification model for test generation, set the model coverage objectives. In the **Design Verifier** tab, in the **Prepare** section, click **Test Generation Settings**. In the Configuration Parameters window, expand the **Design Verifier** list and click **Test Generation**. Set **Model coverage objectives** to the option that best captures the desired coverage.

Iterate Through the Steps

As you develop the specification model and test your design model, you typically need to update the requirements, specification model, and design model. This process is iterative. Continue iterating until you reach the desired test outcomes, such as desired model outputs and test coverage.

See Also

Requirements Table

Related Examples

- “Use a Requirements Table Block to Create Formal Requirements” on page 2-2
- “Use Specification Models for Requirements-Based Testing” on page 2-91
- “Export Tests from Models That Contain Requirements Table Blocks with Simulink Design Verifier” on page 2-75

Export Tests from Models That Contain Requirements Table Blocks with Simulink Design Verifier

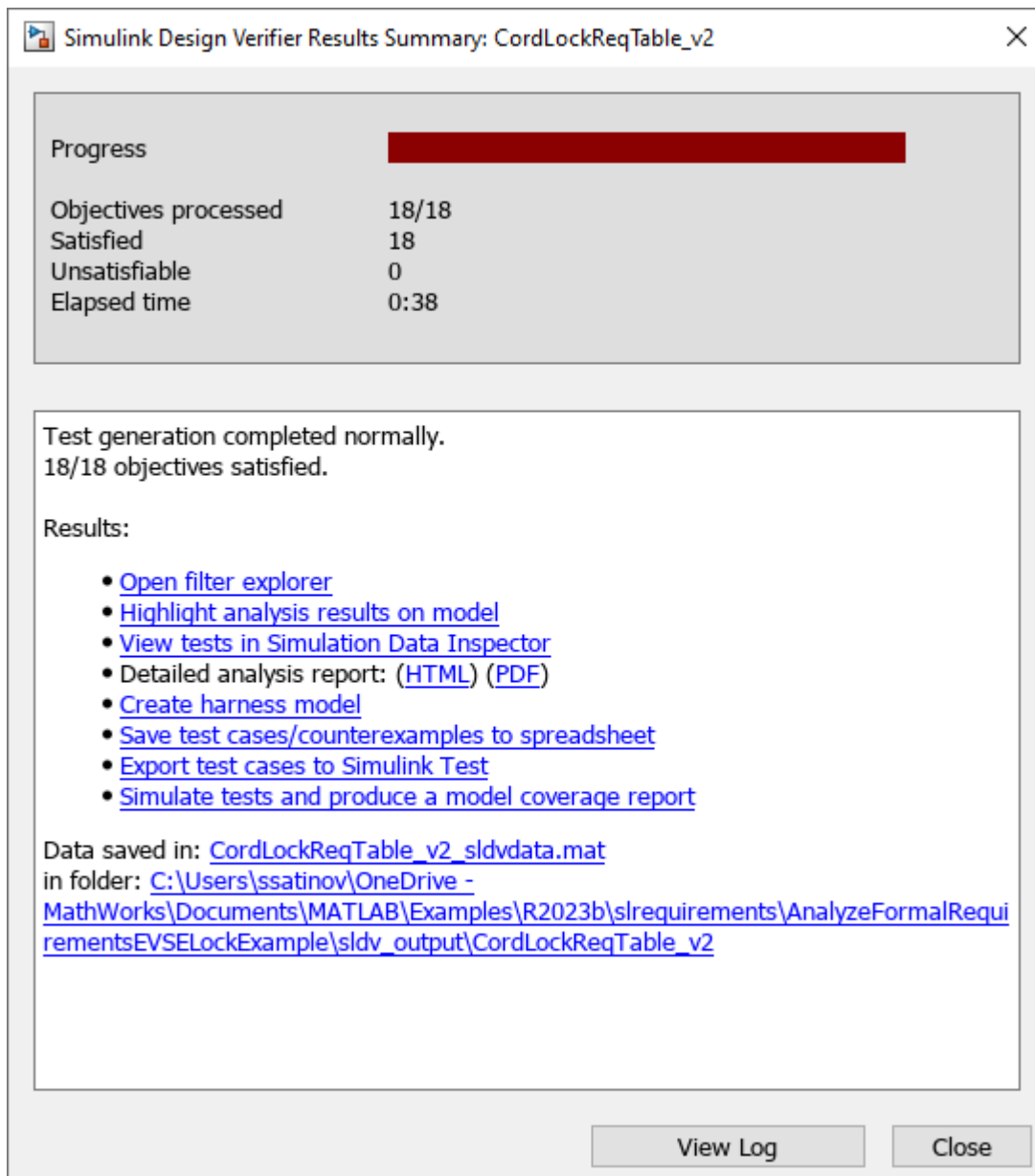
If you create models that contain Requirements Table blocks and you generate tests with Simulink Design Verifier, you can export the tests to the **Test Manager**. When configuring the tests, you can specify the test harness that you want to use. After running the tests, you can determine if the tests fail or pass, and inspect the results further.

Construct the Model and Generate Tests

Simulink Design Verifier creates test objectives from the requirements defined in Requirements Table blocks. To generate tests, construct a model, called the specification model, with Requirements Table blocks and Simulink blocks that do not define test objectives. See “What Is a Specification Model?” on page 2-69. After you create the requirements, confirm that the requirement set in each Requirements Table block is complete and consistent by analyzing them. See “Identify Inconsistent and Incomplete Formal Requirement Sets” on page 2-17. If you do not create complete and consistent requirements, Simulink Design Verifier may not be able to create tests that satisfy the test objectives.

After constructing the requirements, generate tests in the specification model.

- 1 In the **Apps** tab, click **Design Verifier**.
- 2 In the **Mode** section, set the mode to **Test Generation**.
- 3 In the **Prepare** section, click **Test Generation Settings**. The Requirements Table block supports test generation with decision, condition, MCDC, enhanced MCDC, and relational boundary coverage objectives. Specify the objectives in the **Model coverage objectives** parameter. For more information on these options, See “Model Coverage Objectives for Test Generation” (Simulink Design Verifier). Click **OK**.
- 4 In the **Analyze** section, click **Generate Tests**. Simulink Design Verifier indicates how many objectives from the requirements are satisfied.



- 5 If at least one of the objectives is not satisfied, update your requirements. If you did not analyze the requirements before, analyze them now.

Export the Tests to the Test Manager

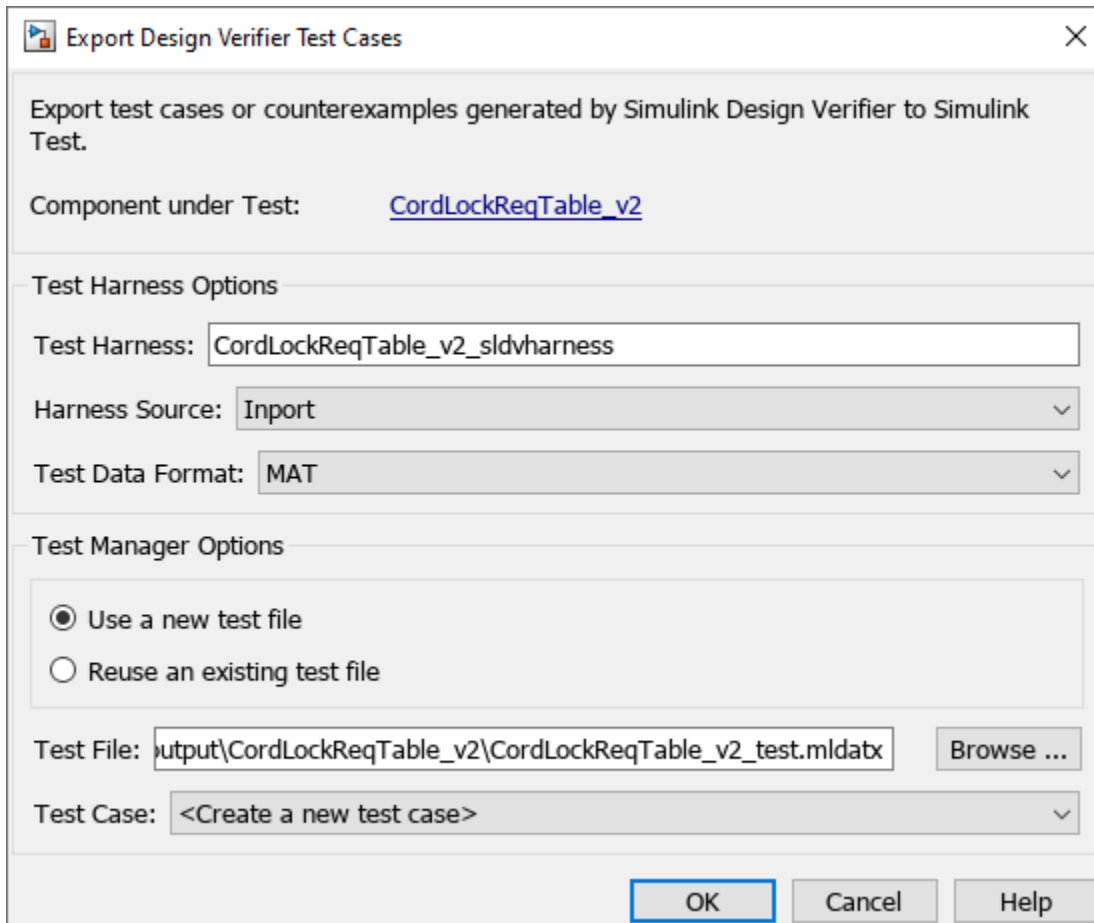
If you have Simulink Test, you can export the tests to the **Test Manager**. In the Simulink Design Verifier Results Summary window, click **Export test cases to Simulink Test**.

Test generation completed normally.
18/18 objectives satisfied.

Results:

- [Open filter explorer](#)
- [Highlight analysis results on model](#)
- [View tests in Simulation Data Inspector](#)
- Detailed analysis report: ([HTML](#)) ([PDF](#))
- [Create harness model](#)
- [Save test cases/counterexamples to spreadsheet](#)
- [Export test cases to Simulink Test](#)
- [Simulate test and produce a model coverage report](#)

The Export Design Verifier Test Cases window displays the properties that you can adjust before exporting.



For more information on the properties you can select, see “Export Test Cases to Simulink Test” (Simulink Design Verifier).

Run the Tests

After exporting the tests, Simulink Test registers the requirements in the Requirements Table blocks to the test cases they generate. To view these assignments, open the **Test Manager**. Then select the test case in the **Test Browser** pane. In the **Test Case** pane, expand the **Iterations** section.

▼ ITERATIONS* ?

▼ TABLE ITERATIONS* ?

<input checked="" type="checkbox"/> NAME	DESCRIPTION	REQUIREMENTS	EXTERNAL INPUT
<input checked="" type="checkbox"/> Test Case:1	None	Requirement 1: Lock when evse c...	Test Case:1
<input checked="" type="checkbox"/> Test Case:2	None	Requirement 6: Unlock during em...	Test Case:2
<input checked="" type="checkbox"/> Test Case:3	None	Requirement 2: Unlock during nor...	Test Case:3
<input checked="" type="checkbox"/> Test Case:4	None	Requirement 7: Unlock SessionSto...	Test Case:4
<input checked="" type="checkbox"/> Test Case:5	None	Requirement 9: Unlock when com...	Test Case:5
<input checked="" type="checkbox"/> Test Case:6	None	Requirement 3: Unlock during em...	Test Case:6

Auto Generate + Add Delete ▼

If you have a manually created test harness that you want to run the tests on, in the **Test Browser** pane, select the test case and expand **System Under Test**. In the **Model** field, specify the model, and clear the model from the **Harness** field.

Inspect Test Failures

If one of your tests fail, you may need investigate causes of the failure. If you use verification blocks in your harness to verify the outputs of your design and specification model, or if you capture design model outputs in requirement postconditions, you can use property proving on the test harness and run **Model Slicer** to identify the conditions that cause an assertion failure.

- 1 Open the test harness model.
- 2 In the **Design Verifier** tab, in the **Mode** section, select **Property Proving**.
- 3 In the **Prepare** section, click **Property Proving Settings**. If your specification model uses at least one postcondition, the Requirements Table block highlights the postconditions green if the associated proof objectives are satisfied, red if they are not, and orange for other conditions.

Requirements		Assumptions	
Index	Summary	Precondition	Postcondition
			deploy
1	The thrust reverser system shall not deploy if the average airspeed is greater than 150 knots.	$\text{mean}(\text{airspeed}) > 150$	false
2	The thrust reverser system shall not deploy if any two WOW sensors are false.	$\text{sum}(\text{wow}) \geq 3$	false
3	The thrust reverser system shall not deploy if the two thrust sensors are greater than 0.	$\text{sum}(\text{throttle}) \geq 3$	false
4	The thrust reverser system shall not deploy if either wheelspeed sensor is less than 10 knots.	$\text{wheelspeed}(1) < 10 \ \&\& \ \text{wheelspeed}(2) < 10$	false

If you use verification blocks, Simulink Design Verifier highlights the blocks that assert a failure in red.

- 4 If you use verification blocks, select the highlighted verification block. In the Results window, click **debug**. The model displays the values that correspond to each signal that caused the failure. These values only display outside of the Requirements Table block.

For more information, see “Debug Property Proving Violations by Using Model Slicer” (Simulink Design Verifier) and “Prove Properties with Requirements Table Blocks” on page 2-102.

See Also

Requirements Table

Related Examples

- “Use a Requirements Table Block to Create Formal Requirements” on page 2-2
- “What Is Property Proving?” (Simulink Design Verifier)
- “What Is a Specification Model?” on page 2-69
- “Use Specification Models for Requirements-Based Testing” on page 2-91

Model Dice Game Rules Using a Requirements Table Block

This example shows how to model some of the rules associated with the craps dice game by using a Requirements Table block. Typically, craps involves a player, an audience, and casino staff. In the game, the player rolls two six-sided dice. The player and audience members bet based on the sum of the dice, and the staff enforce the rules and manage the bets.

On the first role, the player must follow these rules:

- If the player rolls and the sum of the dice is 7 or 11, the game starts over. This is called rolling a natural.
- If the player rolls and the sum of the dice is 2, 3, or 12, the game starts over. This is called rolling craps.
- Otherwise, the sum is assigned to a value known as the point, and the game continues.

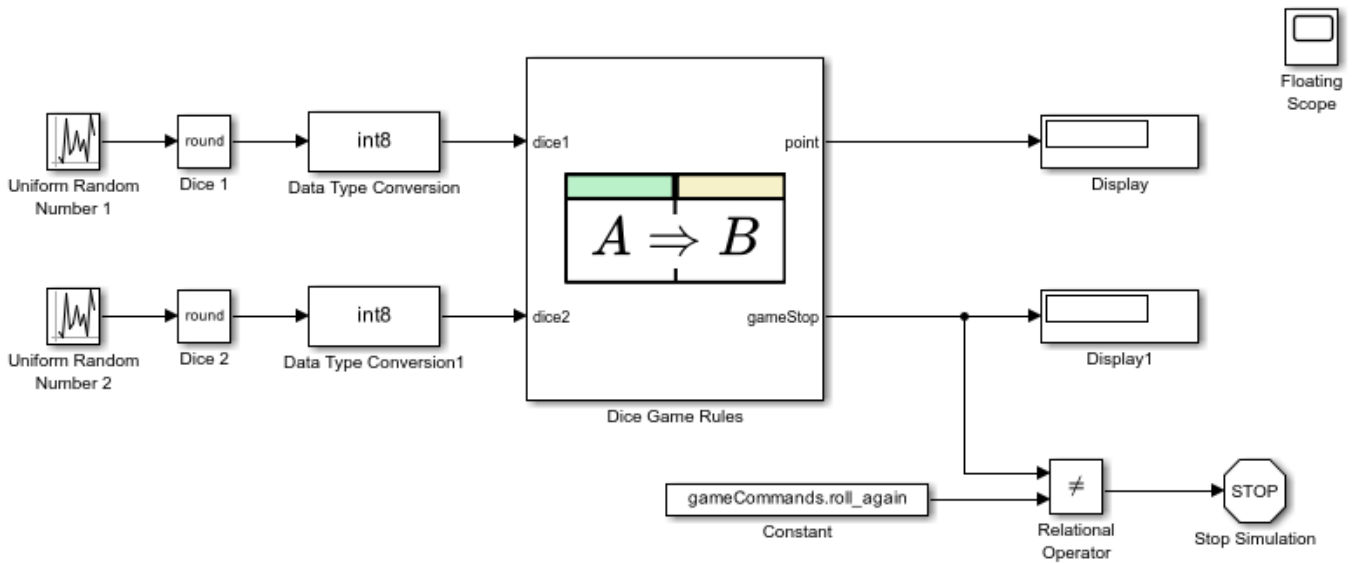
If the player establishes the point, they continue to roll the dice. On the next rolls, they must follow these rules:

- If the player rolls and the sum of the dice is 7, the game starts over. This is called rolling seven-out.
- If the player rolls and the sum of the dice is the value of the point, the game starts over. This is called rolling the point.
- Otherwise, the player rolls the dice again.

In this example, the model determines the outcomes of the rolls and stops the simulation when the game must start over. The Requirements Table block represents these rules and their outcomes as requirements.

View the Model

Open the ReqTableDiceGame model. The model generates two random integers with values that range from 1 to 6 to represent the value of the dice. Based on the roll, the simulation continues or stops.



Copyright 2022, The MathWorks, Inc.

Open the Requirements Table block, **Dice Game Rules**. The requirements model the dice game rules by using a combination of temporal operators and requirement hierarchies. See “Control Requirement Execution by Using Temporal Logic” on page 2-33 and “Establish Hierarchy in Requirements Table Blocks” on page 2-45.

Requirements		Assumptions			
Index	Summary	Precondition		Action	
			dice1 + dice2	point	gameStop
1	Conditions for the first roll	isStartup			
1.1	If the player rolls a 7 or a 11 on the first roll		7,11	0	natural
1.2	If the player rolls a 2, 3, or 12 on the first roll		2,3,12	0	craps
1.3	Otherwise, establish the point value	Else		dice1 + dice2	roll_again
2	Following rolls	~isStartup			
2.1	If the player rolls a 7 before they roll the point		7	prev(point)	seven_out
2.2	If the player rolls the point		prev(point)	prev(point)	rolled_the_point
2.3	Otherwise, roll again	Else		prev(point)	roll_again

The block also includes two assumptions that constrain the possible value of each die. See “Add Assumptions to Requirements” on page 2-10.

Requirements		Assumptions
Index	Summary	Postcondition
1	Dice 1 limitations	$1 \leq \text{dice1} \leq 6$
2	Dice 2 limitations	$1 \leq \text{dice2} \leq 6$

The requirements are complete and consistent. See “Identify Inconsistent and Incomplete Formal Requirement Sets” on page 2-17.

Exit the block and run the simulation. You can view the outcomes of the game in the Floating Scope block. Run the simulation multiple times to generate different game results. The model runs for a maximum 10 rolls. Extend the simulation time to allow for more total rolls.

See Also

Requirements Table

Related Examples

- “Debug Requirements Table Blocks” on page 2-22
- “Use a Requirements Table Block to Create Formal Requirements” on page 2-2

Analyze Formal Requirements of an Electric Vehicle Charger Locking Mechanism

This example uses a Requirements Table block to define a set of formal requirements used by a model of an electric vehicle charger locking system. After creating formal requirements for the locking system, you analyze the requirements for issues that can prevent the requirements from being complete and consistent. Once you identify the issues, you adjust the requirements.

When charging an electric vehicle, the charging port of the vehicle can be exposed to high voltages, which can be dangerous to consumers without proper safeguards. Electric chargers and vehicles deploy locking mechanisms to prevent customers from being exposed to these voltages. To ensure the locking mechanism activates and deactivates without creating hazards, the charging station provides signals to the vehicle before, during, and after charging. The vehicle uses those signals to determine when the charging cord can be unlocked and is safe to handle.

View the Formal Requirements

Open the model, `CordLockReqTable_v1`.

```
open_system("CordLockReqTable_v1");
```

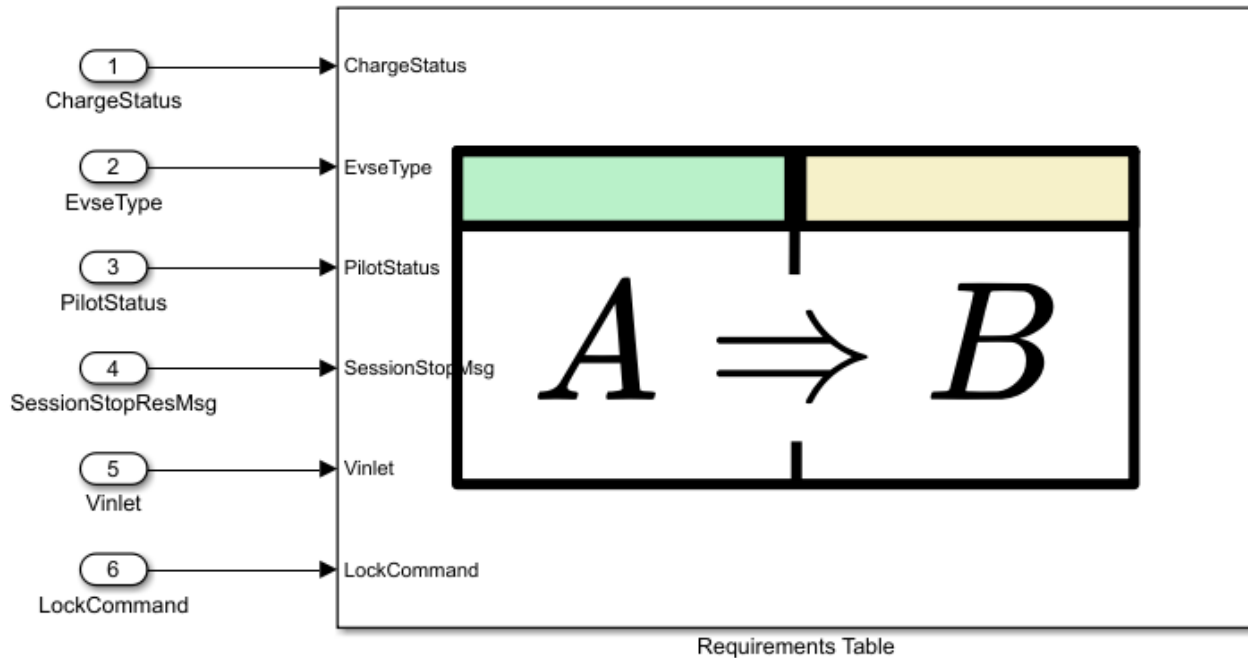
This model adapts the signals used by the charging station from the standard SAE J1772 [1]. To capture the signals, the Requirements Table block uses six input ports that are associated with input data of the same name. See “Define Data in Requirements Table Blocks” on page 2-53.

The input data are:

- **ChargeStatus**: The charging status of the vehicle. The status indicates that the vehicle is not charging (`NotCharging`), charging (`ChargeStart`), shutting down (`NrmlShutDown`), or performing an emergency shut down (`EmrgShutDown`).
- **EvseType**: If the vehicle is compatible with the charging station or not. The vehicle can be `Compatible`, `Incompatible`, or `undecided (NotDecided)`.
- **PilotStatus**: The pilot status of the charging station. The pilot status indicates if the charging station is on standby (`A`), if the charging station detects the vehicle (`B`), if the charging station detects the vehicle and that the vehicle is ready to charge (`C`), if the charging station detects the vehicle and that the vehicle is ready for charging in a ventilated area (`D`), or if there is an error (`E_F`). States `B`, `C`, and `D` each have two substates. Substate 1 indicates that the charging station is not ready to charge, and substate 2 indicates that the charging station is ready to charge.
- **SessionStopResMsg**: Whether the termination of the charging session has been acknowledged by the charging station (`Received`) or not (`NotReceived`).
- **VinLet**: The voltage of the charging port measured by the vehicle.
- **LockCommand**: The status of the lock. The lock can either be `Locked` or `Unlocked`.

The block uses the value of the input data to set the **Type** property of the data to an enumerated value specified in a MATLAB® program file. For example, the `ChrgStat.m` file contains the available statuses for the `ChargeStatus` data.

Charging Cord Locking Mechanism Requirements, Version 1



Copyright 2021 The MathWorks, Inc.

Open the block to view the requirements. The **Requirements** tab lists three simple requirements:

- 1 The electric vehicle shall lock the cord in place if the electric vehicle supply equipment is compatible.
- 2 The electric vehicle shall unlock the cord if the acknowledge terminate charging session signal is received and the vehicle measures an input voltage that is less than 60 V during normal shutdown procedures.
- 3 The electric vehicle shall unlock the cord if the pilot status is not ready and the vehicle measures an input voltage that is less than 60 V during an emergency shutdown.

Each requirement specifies the checked conditions the **Precondition** column. If the requirement precondition is satisfied, the block then verifies that the appropriate locking status occurs in the postconditions, specified in the **Postcondition** column. For example, in the first requirement, the specified precondition checks if the vehicle type is compatible. If true, the block checks if the cord is locked, specified in the postcondition.

Requirements		Assumptions	
Index	Summary	Precondition	Postcondition
1	Requirement 1: Lock when evse compatible	(EvseType == Compatible)	LockCommand Locked
2	Requirement 2: Unlock during normal shutdown	(SessionStopMsg == Received) && (ChargeStatus == NrmalShutDown) && (Vinlet < 60)	Unlocked
3	Requirement 3: Unlock during emergency shutdown static pilot	(PilotStatus ~= C2) && (PilotStatus ~= D2) && (ChargeStatus == EmrgShutDown) && (Vinlet < 60)	Unlocked

Analyze the Requirements for Issues

After creating a requirement set in the block, confirm that the requirements uniquely define the data captured, stored, and generated by the block. In this example, you use Simulink® Design Verifier™ to detect issues in your requirement set for you. These issues are adapted from the ISO/IEC/IEEE 29148 standard [2]. In this example, Simulink Design Verifier detects two kinds of issues:

- 1 **Inconsistency Issues:** The block detects a scenario where the data can equal more than one value during simulation.
- 2 **Incompleteness Issues:** The block detects a scenario where the data is not defined during simulation.

To analyze the table, in the **Table** tab, in the **Analyze** section, click **Analyze Table**.

Inconsistency Issues

Inconsistency 1: 'LockCommand' is inconsistent at time 0 in requirements 1 and 3 for the following inputs:

Time	0
Step	1
ChargeStatus	ChrgStat.EmrgShutDown
EvseType	EvseStat.Compatible
PilotStatus	PilotStat.D1
SessionStopMsg	MsgStat.NotReceived
Vinlet	0

Incompleteness Issues

Incompleteness 1: 'LockCommand' is not specified at time 0.2 for the following inputs:

Time	0	0.2
Step	1	2
ChargeStatus	ChrgStat.ChargeStart	ChrgStat.NotCharging
EvseType	EvseStat.Compatible	EvseStat.NotDecided
PilotStatus	PilotStat.E_F	PilotStat.A
SessionStopMsg	MsgStat.NotReceived	MsgStat.NotReceived
Vinlet	87.6303	0

Update the Requirements

To resolve the issues, you need to update the requirements. For example, to alleviate the inconsistency issue, update the precondition in the first requirement so that the charge status must be charging and the charger must be compatible with the vehicle.

Requirements		Assumptions	
Index	Summary	Precondition	Postcondition
1	Requirement 1: Lock when evse compatible	(EvseType == Compatible) && (ChargeStatus == ChargeStart)	Locked

After updating the precondition, run the analysis again to confirm that you have fixed the inconsistency issue.

This update does not resolve the incompleteness issues. To do that, the requirements must specify values for the data defined in the block throughout the simulation. Iterate through updating and analyzing the requirements to resolve these remaining issues.

Open the model `CordLockReqTable_v2` and open the Requirements Table block to see an example of a complete requirement set.

```
open_system("CordLockReqTable_v2");
```

The table includes additional input data, `ChargePlug`, that specifies if the cord is plugged into the vehicle (`Plugged`) or not (`NotPlugged`). The requirement set also includes five additional requirements. To improve readability, the requirements have been rewritten to use additional column headers.

Requirements		Assumptions						
Index	Summary	Precondition						Postcondition
		EvseType	ChargeStatus	SessionStopMsg	PilotStatus	ChargePlug	Vinlet	LockCommand
1	Requirement 1: Lock when evse compatible	Compatible	ChargeStart					Locked
2	Requirement 2: Unlock during normal shutdown		NrmlShutDown	Received			< 60	Unlocked
3	Requirement 3: Unlock during emergency shutdown static pilot		EmrgShutDown		(X == C2) && (X == D2)		< 60	Unlocked
4	Requirement 4: Lock for unsafe voltage					Plugged	>= 60	Locked
5	Requirement 5: Unlock when unplugged					NotPlugged		Unlocked
6	Requirement 6: Unlock during emergency shutdown oscillating pilot		EmrgShutDown		(X == C2) (X == D2)		< 60	Unlocked
7	Requirement 7: Unlock SessionStop not recieved		NrmlShutDown	NotReceived			< 60	Unlocked
8	Requirement 8: Unlock when not charging		NotCharging				< 60	Unlocked
9	Requirement 9: Unlock when compatibility not decided	NotDecided	ChargeStart				< 60	Unlocked

When you design requirements, you must consider the physical limitations of your system. For example, if you develop a requirement that specifies a diameter, you must ensure the diameter cannot be negative. To account for the physical limitations of the system, specify assumptions in the **Assumptions** tab. See “Add Assumptions to Requirements” on page 2-10. In this example, the Requirements Table block includes three assumptions.

- 1 If the pilot status is on standby (A), then the car is not charging.
- 2 If the vehicle is not compatible with the charger, then the vehicle is not charging.
- 3 If the charger is not plugged in, then the vehicle is not charging.

To view the assumptions, click the **Assumptions** tab.

Requirements		Assumptions	
Index	Summary	Precondition	Postcondition
1	Assumption 1	PilotStatus == A	ChargeStatus == NotCharging
2	Assumption 2	EvseType == Incompatible	ChargeStatus == NotCharging
3	Assumption 3	ChargePlug == NotPlugged	ChargeStatus == NotCharging

If you run the analysis on this requirement set, Simulink Design Verifier does not detect issues.

References

- [1] Hybrid - EV Committee, "SAE Electric Vehicle and Plug in Hybrid Electric Vehicle Conductive Charge Coupler" (SAE International), accessed December 21, 2021, https://doi.org/10.4271/J1772_201710.
- [2] "29148-2018 - ISO/IEC/IEEE International Standard - Systems and Software Engineering -- Life Cycle Processes -- Requirements Engineering" (IEEE), accessed December 21, 2021, <https://doi.org/10.1109/IEEESTD.2018.8559686>.

See Also

Requirements Table

Related Examples

- "Use a Requirements Table Block to Create Formal Requirements" on page 2-2
- "Add Assumptions to Requirements" on page 2-10
- "Identify Inconsistent and Incomplete Formal Requirement Sets" on page 2-17
- "Define Data in Requirements Table Blocks" on page 2-53

Define Formal Requirements with a Duration

This example shows you how to use the Requirements Table block to create formal requirements with a duration. When modeling behaviors that cause brief noise during normal operation, engineers frequently define requirements that must be true for a specified duration. For example, in electrical circuits, closing a mechanical switch may cause the voltage to oscillate or spike before stabilizing, which is referred to as *bounce*. For sensitive control systems, bounce can cause measurements that lead to faulty behavior. When you mitigate bounce, which is known as *debouncing*, you specify controller behavior that executes only when the measurement is a certain value for a specified period of time.

This example shows how to model a requirement with a duration when the Requirement Table block has one or multiple input data. To define a requirement with a duration, use the **Duration** column, child requirements, semantic requirements, and default requirements. See “Using the Duration Column” on page 2-33 and “Establish Hierarchy in Requirements Table Blocks” on page 2-45.

Define a Requirement with a Duration for a Single Input

If you want to create a requirement with a duration that uses a single input, specify the requirement along with a default requirement. Open the model `reqTableDurationModel1`.

```
open_system("reqTableDurationModel1");
```

This model specifies a requirement for a sensor that detects a fault in a circuit. The model has one input data, `current`. The Requirements Table block uses the value of `current` to confirm the anticipated output of the sensor, `overCurrent`. The table defines the following requirement:

- If the measured current is greater than 10 for more than 0.5 seconds, then the sensor shall detect a fault in the circuit.
- Otherwise, the sensor shall not detect a fault in the circuit.

Requirements		Assumptions		
Index	Summary	Precondition	Duration	Postcondition overCurrent
1	Current Sensor			
1.1	Overcurrent Requirement	<code>current > 10</code>	0.5	true
1.2	D	Else		false

Define a Requirement with a Duration for Multiple Inputs

If you have a requirement that relies on multiple inputs that each have a duration, and each of the inputs affects the anticipated output of your model, use a semantic requirement. You can use each child in the semantic requirement to represent the requirements for each input. Open the model `reqTableDurationModel2`.

```
open_system("reqTableDurationModel2");
```

In this example, the Requirements Table block specifies a requirement for overspeed detection in a jet engine. A jet engine experiences overspeed when it spins beyond its physical limits. Prolonged

overspeed can lead to catastrophic failure of the engine. To address this, jet engine control systems use multiple sensors and overspeed detection mechanisms. If at least one of the sensors detects overspeed, the engine control system shuts down the jet engine by cutting off fuel flow to the combustor. However, aircraft operators and designers also do not want the control system to shut down the engine due to a sensor, physical component, or software error caused by bounce. To reduce the chances of unintended engine shutdown, engineers often specify a short duration that each sensor must include before determining if overspeed occurs.

The table uses, `engineSpeed1`, `engineSpeed2`, and `engineSpeed3` input data to define the requirement for the overspeed detection.

- If at least one of the sensors measures the engine speed to be greater than 1000 for more than 0.5 seconds, then the engine control system shall detect engine overspeed.
- Otherwise, the engine control system shall not detect engine overspeed.

Requirements		Assumptions		
Index	Summary	Precondition	Duration	Postcondition overspeedDetected
1	Engine Speed Sensors			
1.1		Any Child Active		true
1.1.1	Sensor 1	<code>engineSpeed1 > 1000</code>	0.5	
1.1.2	Sensor 2	<code>engineSpeed2 > 1000</code>	0.5	
1.1.3	Sensor 3	<code>engineSpeed3 > 1000</code>	0.5	
1.2		Else		false

See Also

Requirements Table

Related Examples

- “Use a Requirements Table Block to Create Formal Requirements” on page 2-2
- “Establish Hierarchy in Requirements Table Blocks” on page 2-45
- “Control Requirement Execution by Using Temporal Logic” on page 2-33
- “Define Data in Requirements Table Blocks” on page 2-53

Use Specification Models for Requirements-Based Testing

This example shows how to use a specification model to model and test formal requirements on a model of an aircraft autopilot controller. The specification model uses two Requirements Table blocks to model the required inputs and outputs of the aircraft autopilot controller model. You generate tests from the specification model, and then run those tests on the aircraft autopilot controller model. The model that you test is the *design model*.

For more information on how to define and configure Requirements Table blocks, see “Use a Requirements Table Block to Create Formal Requirements” on page 2-2 and “Configure Properties of Formal Requirements” on page 2-27.

View the High-Level Requirements

Open the requirements set, AP_Controller_Reqs, in the **Requirements Editor**.

```
slreq.open("AP_Controller_Reqs");
```

The high-level requirements specify the outputs of the model and the autopilot controller mode. Each requirement description uses high-level language that you can use to explicitly define the logic needed in the formal requirements.

The screenshot shows the Requirements Editor interface. On the left, a table lists requirements for the 'AP_Controller_Reqs' set:

Index	ID	Summary
1	1	High Level: Autopilot Controlle...
2	2	High Level: Off Reference
3	3	High Level: Roll Hold Reference
4	4	High Level: Heading hold mode...
5	5	High Level: Default Behavior

The right pane shows the details for 'Requirement: 1':

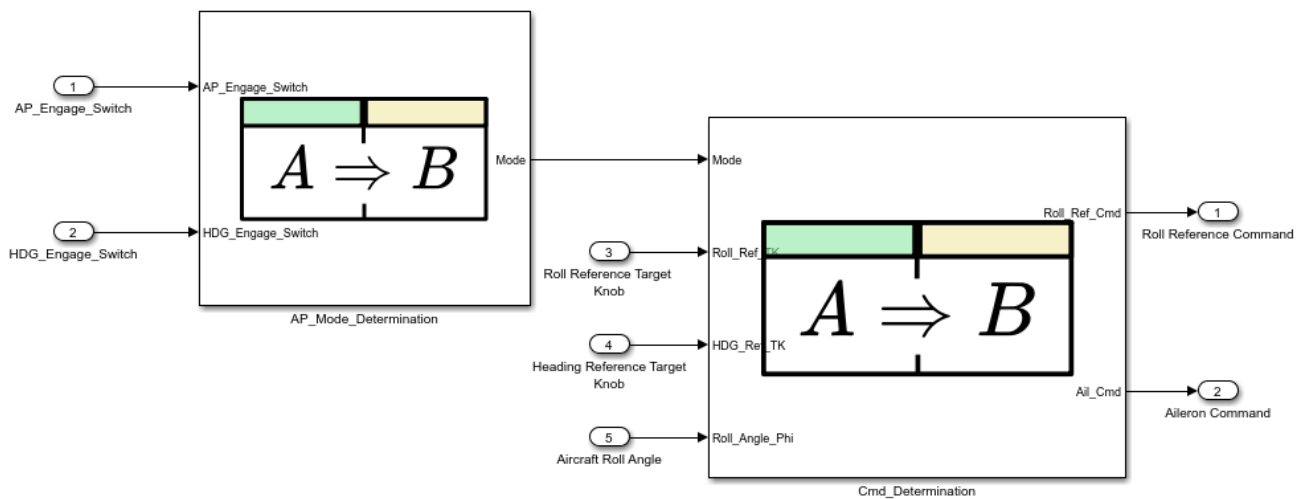
- Properties:** Type: Functional, Index: 1, Custom ID: 1, Summary: High Level: Autopilot Controller Modes
- Description:** The autopilot controller has the following high level system modes:
 - OFF: The autopilot controller is off.
 - ROLL_HOLD_MODE: The autopilot controller is in roll hold mode.
 - HDG_HOLD_MODE: The autopilot controller is in heading hold mode.
- Rationale:** The autopilot controller mode is determined by the following:
 - The autopilot controller is OFF when the autopilot engage switch is not engaged.
 - The autopilot controller is ROLL_HOLD_MODE when the autopilot engage switch is engaged and the heading engage switch is not engaged.
 - The autopilot controller is HDG_HOLD_MODE when the autopilot engage switch and the heading engage switch are both engaged.
- Keywords:** (empty field)
- Revision information:** (empty field)
- Links:** (empty field)
- Comments:** (empty field)

View the First Iteration of the Specification Model

Open the specification model, `spec_model_partial`.

```
spec_model = "spec_model_partial";
open_system(spec_model);
```

The model contains two Requirements Table blocks that define the formal requirements that translate the high-level requirements into testable logical expressions. The block `AP_Mode_Determination` specifies the formal requirements for the autopilot controller mode, and the block `Cmd_Determination` specifies the outputs of the controller.



To view the formal requirements, inspect each Requirements Table block.

Requirements Table Block for Controller Mode

Open `AP_Mode_Determination`. The block specifies the formal requirements for the autopilot controller mode. To determine the output data `Mode`, `AP_Mode_Determination` specifies three requirements by using two input data:

- `AP_Engage_Switch` — The autopilot engage switch
- `HDG_Engage_Switch` — The heading engage switch

Each requirement uses a combination of the inputs to specify a unique output value for `Mode`.

Requirements		Assumptions		
Index	Summary	Precondition		Action
		AP_Engage_Switch	HDG_Engage_Switch	Mode
1	AP_Engage_Switch and HDG_Engage_Switch both engaged	true	true	HDG_HOLD_MODE
2	AP_Engage_Switch is engaged and HDG_Engage_Switch is not engaged	true	false	ROLL_HOLD_MODE
3	AP_Engage_Switch is not engaged	false		OFF

Requirements Table Block for Controller Commands

Open `Cmd_Determination`. `Cmd_Determination` specifies the requirements for the aileron command and roll reference command. `Cmd_Determination` uses four input data:

- `Mode` — The `AP_Mode_Determination` output, `Mode`
- `Roll_Ref_TK` — The setting of the roll reference target knob
- `Roll_Angle_Phi` — The actual aircraft roll angle
- `HDG_Ref_TK` — The setting of the heading reference target knob

The block uses these input data to determine the controller output data:

- `Roll_Ref_Cmd` — Roll reference command
- `Ail_Cmd` — Aileron command

Requirements		Assumptions					
Index	Summary	Precondition		Action			
		Mode	Roll_Ref_TK	prev(Roll_Angle_Phi)	Roll_Ref_Cmd	Ail_Cmd	
1	Autopilot mode is OFF	OFF			0	Zero	
2	ROLL_HOLD_MODE becomes active mode	<code>hasChangedTo(X,ROLL_HOLD_MODE)</code>				All	
2.1	Roll_Ref_TK between [-30, -3] or [+3, +30] degrees		<code>[TK_neg_extreme, TK_neg_norm] [TK_pos_norm, TK_pos_extreme]</code>			Roll_Ref_TK	
2.2	Roll_Ref_TK greater than -3 and less than +3		<code>(TK_neg_norm, TK_pos_norm)</code>				
2.2.1	Roll_Angle_Phi greater than <code>neg_norm</code> and less than <code>pos_norm</code>			<code>[phi_neg_norm, phi_pos_norm]</code>	0		
2.2.2	Roll_Angle_Phi greater than +30			<code>> phi_pos_extreme</code>		TK_pos_extreme	
2.2.3	Roll_Angle_Phi less than -30			<code>< phi_neg_extreme</code>		TK_neg_extreme	
3	HDG_HOLD_MODE becomes active mode	<code>hasChangedTo(X,HDG_HOLD_MODE)</code>				HDG_Ref_TK	All
4	Otherwise, Roll_Ref_Cmd shall hold the previous value of Roll_Ref_Cmd	Else			<code>prev(Roll_Ref_Cmd)</code>	All	

In this example, the expressions use constant data to define the ranges of values for `Roll_Ref_TK` and `Roll_Angle_Phi`. You can also parameterize the values or use literal values. See “Define Data in Requirements Table Blocks” on page 2-53. To view these values, open the **Symbols** pane. In the **Modeling** tab, in the **Design Data** section, click **Symbols Pane**.

In addition to requirements, `Cmd_Determination` also defines the assumptions for the design. See “Add Assumptions to Requirements” on page 2-10. In this example, the assumptions constrain the values for the roll angle and the roll reference target knob based on their physical limitations. The roll angle cannot exceed 180 or fall below -180 degrees, and the roll reference target knob cannot exceed 30 or fall below -30. In the table, click the **Assumptions** tab.

Requirements		Assumptions
Index	Summary	Postcondition
1	Roll angle geometric limitations	Roll_Angle_Phi >= -180 && Roll_Angle_Phi <= 180
2	Reference knob minimum and maximum settings	Roll_Ref_TK <= 30 && Roll_Ref_TK >= -30

You can also specify data range limitations in the **Minimum** and **Maximum** properties of the data or explicitly specify the range from the signal with blocks.

Generate Tests

Simulink® Design Verifier™ automatically creates test objectives from the requirements defined in Requirements Table blocks. To generate tests, use the Configuration Parameter window or specify the tests programmatically. See “Model Coverage Objectives for Test Generation” (Simulink Design Verifier). Select different coverage objectives to determine if you want to minimize the number of tests generated, or if you want to improve test granularity and traceability.

In this example, generate tests with decision coverage and save the output to a MAT-file.

```
opts = sldvoptions;
opts.Mode = "TestGeneration";
opts.ModelCoverageObjectives = "Decision";
[~,files] = sldvrun(spec_model,opts,true);
```

Simulink Design Verifier generates the test objectives and the tests from the requirements, however the requirements satisfy only seven of the test objectives.

The screenshot shows a window titled "Simulink Design Verifier Results Summary: spec_model_partial". It contains a progress bar and a table of statistics. Below the table, there is a text summary and a list of action items.

Progress	
Objectives processed	24/24
Satisfied	7
Unsatisfiable	0
Elapsed time	0:26

Test generation completed normally.
 7/24 objectives satisfied.
 17/24 objectives undecided due to runtime error

Results:

- [Open filter viewer](#)
- [Highlight analysis results on model](#)
- [View tests in Simulation Data Inspector](#)
- Detailed analysis report: ([HTML](#)) ([PDF](#))
- [Create harness model](#)
- [Save test cases/counterexamples to spreadsheet](#)
- [Export test cases to Simulink Test](#)
- [Simulate tests and produce a model coverage report](#)

To satisfy the test objectives, you must revise the specification model. In general, avoid generating tests from a specification model without confirming that the formal requirements are complete, consistent, and correspond to the high-level requirements. Otherwise, the tests that you generate are less likely to satisfy the test objectives.

```
clear("files")
```

Investigate and Update the Specification Model

Investigate the specification model and update the formal requirements. In this example, the requirement set in `Cmd_Determination` is missing the formal requirement that corresponds to the third bullet of requirement 3.

Requirement: 3

▼ Properties

Type:

Index: 3

Custom ID:

Summary:

Description Rationale

Times New Roman 11 **B** *I* U [List Icons]

When roll hold mode becomes the active mode of the autopilot controller, the roll reference command shall be set to the cockpit turn knob command, up to a 30 degree limit, if the turn knob is commanding 3 degrees or more in either direction.

If the turn knob is commanding between -3 and 3 degrees in either direction:

- The roll reference command shall be set to zero if the actual roll angle is less than 6 degrees, in either direction, just before the roll hold mode is received.
- The roll reference command shall be set to 30 degrees in the same direction as the actual roll angle if the actual roll angle is greater than 30 degrees just before the roll hold mode is received.
- The roll reference command shall be set to the actual roll angle when the actual roll angle equals other angles.

Open `Cmd_Determination` in the model `spec_model_final` to view the updated requirement set. The additional requirement has the index 2.2.4.

```
spec_model = "spec_model_final";
load_system(spec_model);
open_system(spec_model + "/Cmd_Determination");
```

2.2.3	Roll_Angle_Phi less than -30			< phi_neg_extreme	TK_neg_extreme
2.2.4	Otherwise, Roll_Ref_Cmd default setting	Else			Roll_Angle_Phi
3	HDG_HOLD_MODE becomes active mode	hasChangedTo(X,HDG_HOLD_MODE)			HDG_Ref_TK

Finding issues in your requirement set can be challenging to do manually. You can use Simulink Design Verifier to analyze the requirement set and identify inconsistencies and incompletenesses. For more information, see “Analyze the Block” on page 2-17.

Link High-Level and Formal Requirements

Loading the specification model loads the formal requirements in the **Requirements Editor**. Closing the specification model also closes the associated requirement set. When developing your formal requirements, link formal requirements to the corresponding high-level requirement to track the

requirements in the specification model. In this example, linking the requirements does not affect test generation or test results.

To link the first formal requirement to the corresponding high-level requirement:

- 1 In `spec_model_final`, expand the requirement set named `Table1`.
- 2 Right-click the formal requirement with the **Index** of 1 and select **Select for Linking with Requirement**.
- 3 Expand the `AP_Controller_Reqs` requirement set.
- 4 Right-click the requirement with an **ID** of 1 and click **Create a link from "1: Autopilot mode is OFF" to "1: High Level: Autopilot Con..."**.

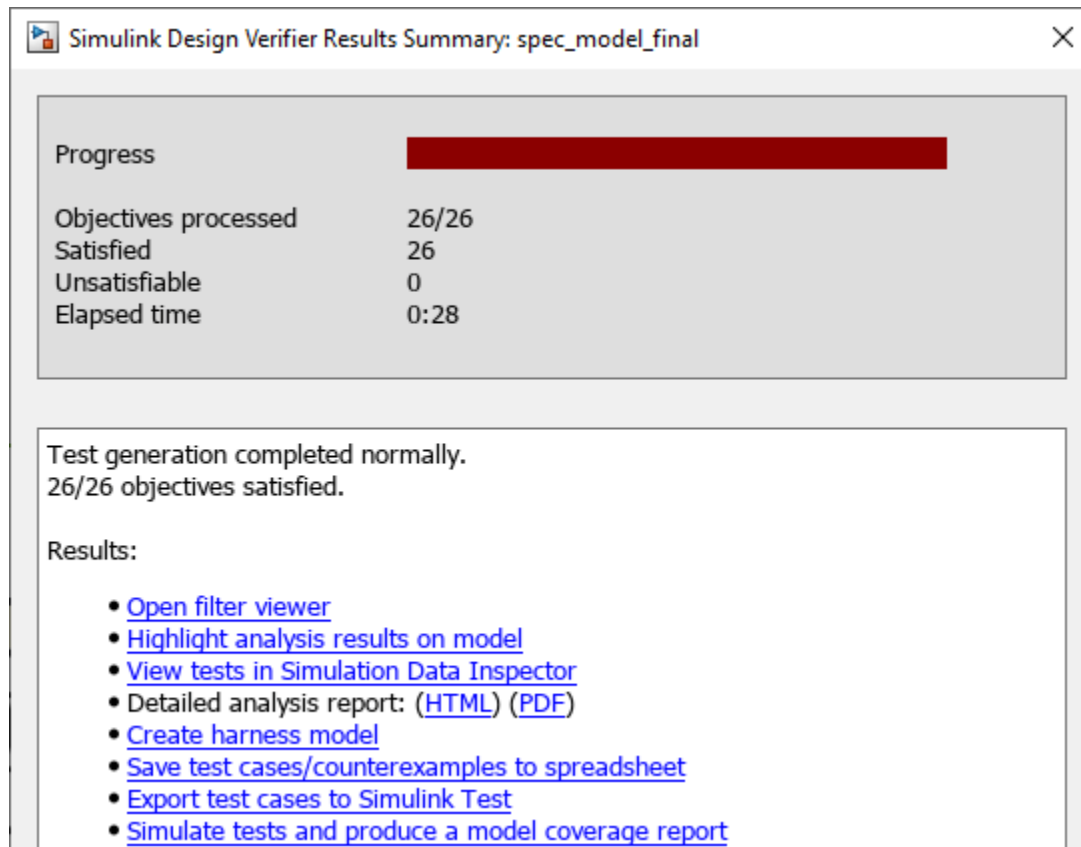
The link type defaults to `Related` to. For more information on link types, see "Link Types" on page 3-34.

Generate Tests from the Updated Model

Generate the tests from the updated specification model by using the options defined previously.

```
opts = sldvoptions;
opts.Mode = "TestGeneration";
opts.ModelCoverageObjectives = "Decision";
[~, files] = sldvrun(spec_model, opts, true);
```

In this version of the specification model, the test objectives are satisfied.



Run the Tests on the Design Model

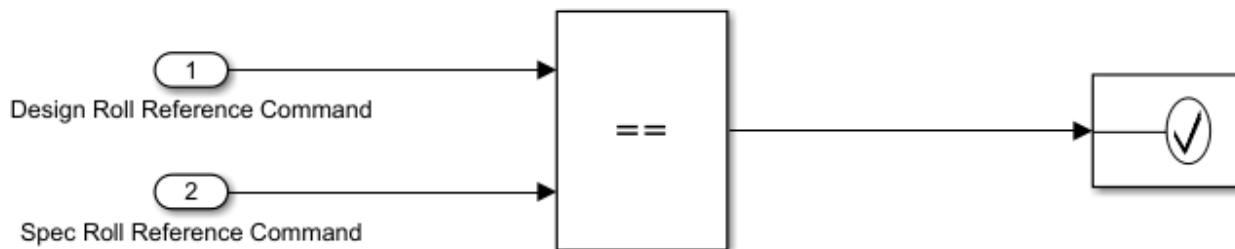
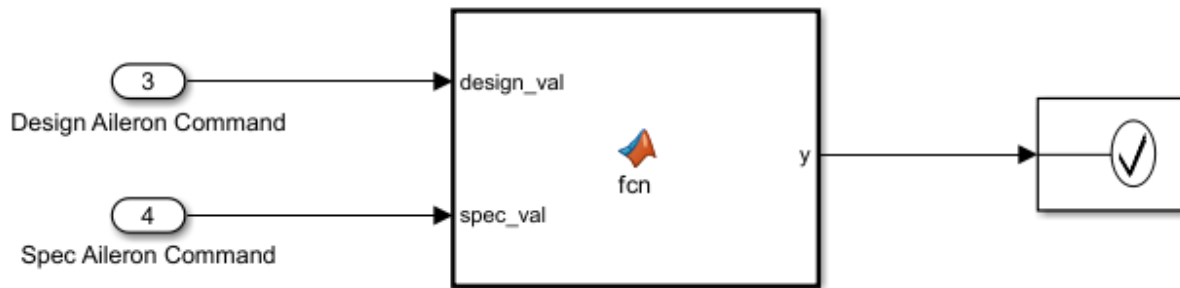
After you create tests that satisfy the test objectives, you can run the tests on the design model. In this example, the design model is the model for the aircraft autopilot controller, `sldvexRollApController`.

Before you run tests on the design model, you must interface the specification model with the design model. Typically, the specification model does not produce or use the same signals as the design model. These differences can be simple or abstract. For example, the design model might use different input and output signal types than the specification model, or you may want to compare a scalar output from the design model against a range in the specification model. As a result, you need to construct an interface between the design model and the specification model.

Interface the Design Model with the Specification Model

In this example, the specification model `spec_model_final` and design model `sldvexRollApController` inputs can interface directly, but one of the outputs is different. `spec_model_final` represents the aileron command as a range of values, but the aileron command value produced by `sldvexRollApController` is a scalar double. The interface uses a MATLAB Function block to compare the aileron command values. The interface then verifies both outputs with Assertion blocks. Open the model, `spec_model_test_interface`, to view the interface.

```
test_interface = "spec_model_test_interface";  
open_system(test_interface);
```



The MATLAB Function block compares the two signals by using this code:

```
function y = fcn(design_val, spec_val)
switch spec_val
case Ail_Cmd.All
    y = true;
case Ail_Cmd.Zero
    y = (design_val == 0);
otherwise
    y = false;
end
```

Run the Updated Tests on the Design Model

To test and verify the design model, create a harness model that contains the:

- 1 Specification model
- 2 Design model

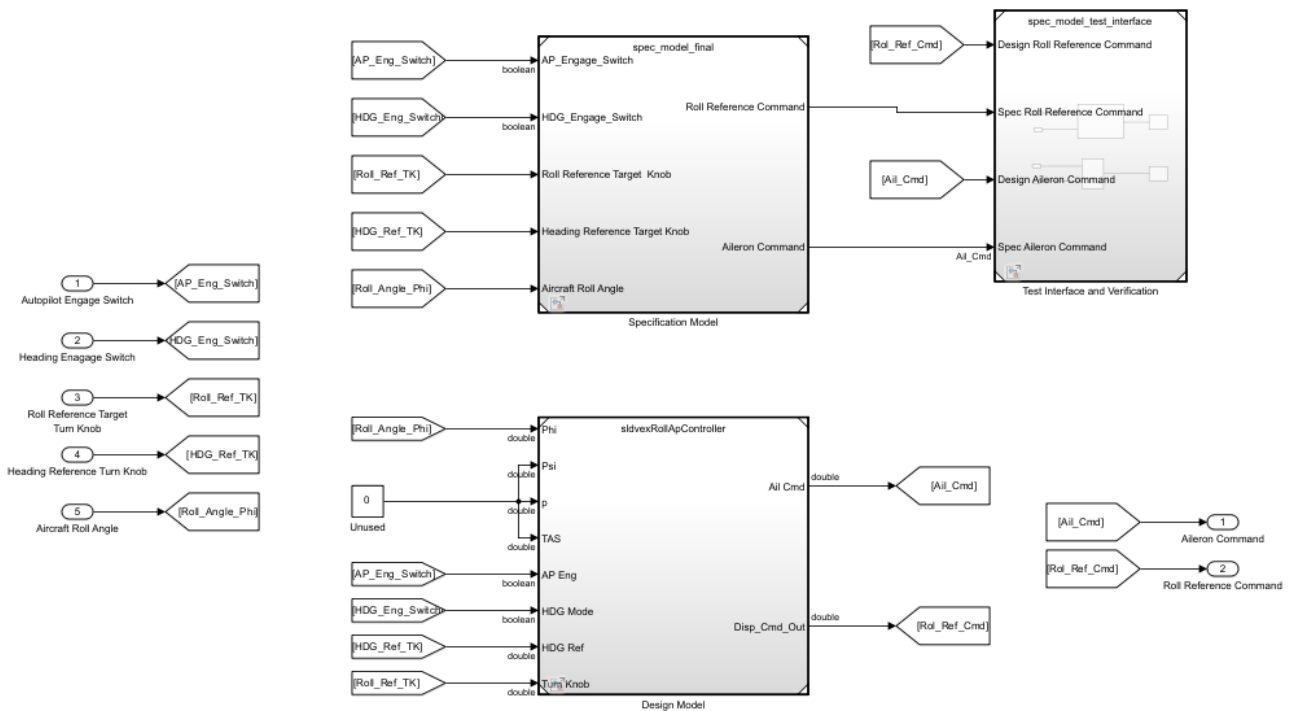
3 Test interface and verification model

In the harness model, attach the models together. Then run the tests on the design model and verify the outputs correspond to the requirements in the harness model.

To view the harness model, open the model, `sldvexDesignHarnessFinal`.

```
harness_model = "sldvexDesignHarnessFinal";
open_system(harness_model);
```

Like with the interface model, not all design model inputs may directly correspond to specification model inputs. In this example, the harness model prepares the design model for testing with the five inputs specified by the specification model.









Run the updated tests on the design model from within the harness model. Use the `sldvrntest` (Simulink Design Verifier) function to run the tests and save the results. If you have Simulink Coverage™, you can view the results of the tests from the output of `sldvrntest` in a coverage report. View the coverage report by using the `cvhtml` (Simulink Coverage) function.

```
cvopts = sldvrntestopts;
cvopts.coverageEnabled = true;
[finalData, finalCov] = sldvrntest(...
    harness_model, files.DataFile, cvopts);
cvhtml("finalCov", finalCov);
```

The coverage report shows that full coverage is achieved on the design model, `sldvexRollApController`.

Summary

Model Hierarchy/Complexity Test 1

		Decision	Execution
1. sldevexRollApController	8	100% 	100% 
2. ... Roll Reference	5	100% 	100% 
3. Latch Phi	1	100% 	100% 

```
bdclose("all");
slreq.clear;
```

See Also

Requirements Table

Related Examples

- “What Is a Specification Model?” on page 2-69
- “Add Assumptions to Requirements” on page 2-10
- “Export Tests from Models That Contain Requirements Table Blocks with Simulink Design Verifier” on page 2-75

Prove Properties with Requirements Table Blocks

This example shows how to use a Requirements Table block and Simulink® Design Verifier™ to prove the properties of an engine thrust reverser system.

When you use a Requirements Table block and Simulink Design Verifier for property proving:

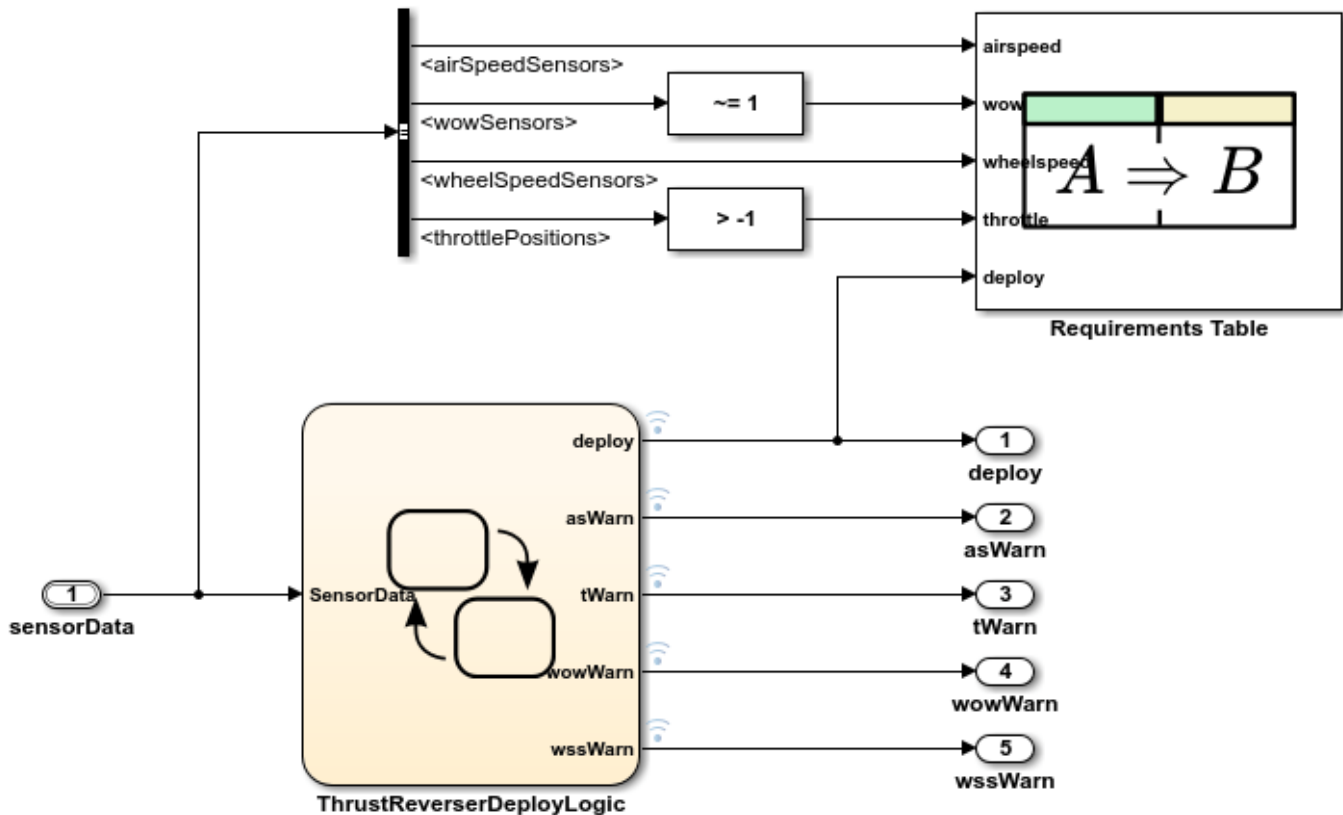
- 1** Each requirement in the block defines a formal requirement that you can use to test properties of a model, subsystem, or block. In this example, the Requirements Table block represents the requirements as preconditions and postconditions.
- 2** Each requirement in the block produces a corresponding requirement in the Requirements Editor. See “Configure Properties of Formal Requirements” on page 2-27.
- 3** Simulink Design Verifier produces proof objectives for the requirements in the requirement set. The postconditions define the logical conditions that you would normally define in Proof Objective (Simulink Design Verifier) blocks. The block evaluates the proof objective associated with a postcondition only if the precondition is true.

For more information, see “Use a Requirements Table Block to Create Formal Requirements” on page 2-2 and “What Is Property Proving?” (Simulink Design Verifier).

View the Requirements in the Requirements Table Block

Open the example model, `property_proving_reqtable`. In this example, you test the properties of the engine thrust reverser system, which is modeled by the chart, `ThrustReverserDeployLogic`. The Requirements Table block uses the chart input signals and deploy output signal to observe the chart behavior. The block defines data in expressions to assess the inputs and outputs of the chart. See “Define Data in Requirements Table Blocks” on page 2-53.

Prove Properties with a Requirements Table Block



Copyright 2022 The MathWorks, Inc.

To view the verification logic associated with each requirement, open the Requirements Table block. The requirements correspond to the requirements in the requirement set used in the example "Validate Requirements by Analyzing Model Properties" (Simulink Design Verifier). The block proves these properties:

- 1 The thrust reverser shall not deploy if the airspeed is greater than 150 knots.
- 2 The thrust reverser shall not deploy if the aircraft is in the air, as indicated by the value of the weight on wheels sensors. If the aircraft is in the air, the signal value for each of two weight on wheels (WOW) sensors is false.
- 3 The thrust reverser shall not deploy if the value of either thrust sensor is positive.
- 4 The thrust reverser shall not deploy if the rotational speed of the landing gear wheels is less than a threshold value.

Each requirement defines a property. If the preconditions are valid, the postcondition must also be satisfied to prove the property.

Requirements		Assumptions	
Index	Summary	Precondition	Postcondition
1	Thrust reverser shall not deploy if airspeed is greater than 150.	mean(airspeed) > 150	false
2	Thrust reverser shall not deploy if the aircraft is in the air, as indicated by the value of the weight on wheels sensors. If the aircraft is in the air, the signal value for each of two weight on wheels (WOW) sensors is false.	sum(wow) >= 3	false
3	Thrust reverser shall not deploy if the value of either throttle position is positive.	sum(throttle) >= 3	false
4	Thrust reverser shall not deploy if landing gear wheels rotational speed is less than 10.	wheelspeed(1) < 10 && wheelspeed(2) < 10	false

Prove Properties

To prove the properties, in the **Design Verifier** tab, click **Prove Properties**. In this example, the properties of the chart are proven. The Requirements Table block highlights the postconditions associated with the proven proof objectives in green.

Postcondition
deploy
false
false
false
false

If the requirement proof objective is falsified, the block highlights the requirement in red. Otherwise, if Simulink Design Verifier is unable to prove or disprove the proof objective, the block highlights the requirement in yellow. You can investigate this behavior by replacing the chart in this example with

the first iteration of the chart used in the “Validate Requirements by Analyzing Model Properties” (Simulink Design Verifier) example.

See Also

Requirements Table

Related Examples

- “Prove Properties in a Model” (Simulink Design Verifier)
- “Use a Requirements Table Block to Create Formal Requirements” on page 2-2
- “Export Tests from Models That Contain Requirements Table Blocks with Simulink Design Verifier” on page 2-75

Requirements Traceability and Consistency

- “Link Blocks and Requirements” on page 3-2
- “Track Requirement Links with a Traceability Matrix” on page 3-5
- “Visualize Links with Traceability Diagrams” on page 3-17
- “Assess Allocation and Impact” on page 3-26
- “Create and Store Links” on page 3-31
- “Load and Resolve Links” on page 3-39
- “Define Custom Requirement and Link Types by Using sl_customization Files” on page 3-42
- “Add Custom Attributes to Links” on page 3-44
- “Requirements Consistency Checks” on page 3-47
- “Manage Navigation Backlinks in External Requirements Documents” on page 3-51
- “Requirements Traceability for Code Generated from MATLAB Code” on page 3-53
- “Link from Sequence Diagrams” on page 3-58
- “Use Command-Line API to Update or Repair Requirements Links” on page 3-61
- “Manage Custom Attributes for Links by Using the Requirements Toolbox API” on page 3-73
- “Make Requirements Fully Traceable with a Traceability Matrix” on page 3-78
- “Modeling System Architecture of Small UAV” on page 3-91
- “Model-Based Systems Engineering for Space-Based Applications” on page 3-97

Link Blocks and Requirements

You can track requirements implementation by linking requirements to model elements that implement the requirements. Linking also enables change notification, so that you can review and act on changes to requirements or models.

In this tutorial, link requirements to a model by using the model requirements perspective. Visual elements highlight links between requirements and blocks.

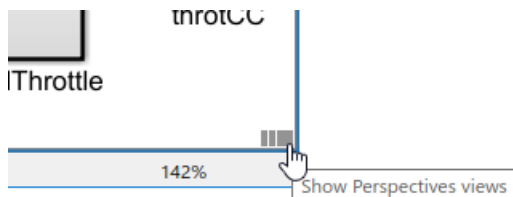
- 1 Open the CruiseRequirementsExample project. At the MATLAB command prompt, enter:

```
slreqCCProjectStart
```

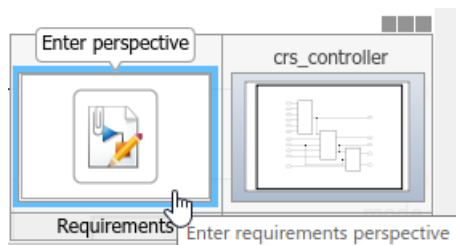
- 2 Open the `crs_controller` model from the `models` folder. At the MATLAB command prompt, enter:

```
open_system("models/crs_controller")
```

- 3 In the model canvas, click the perspectives control in the lower-right corner.



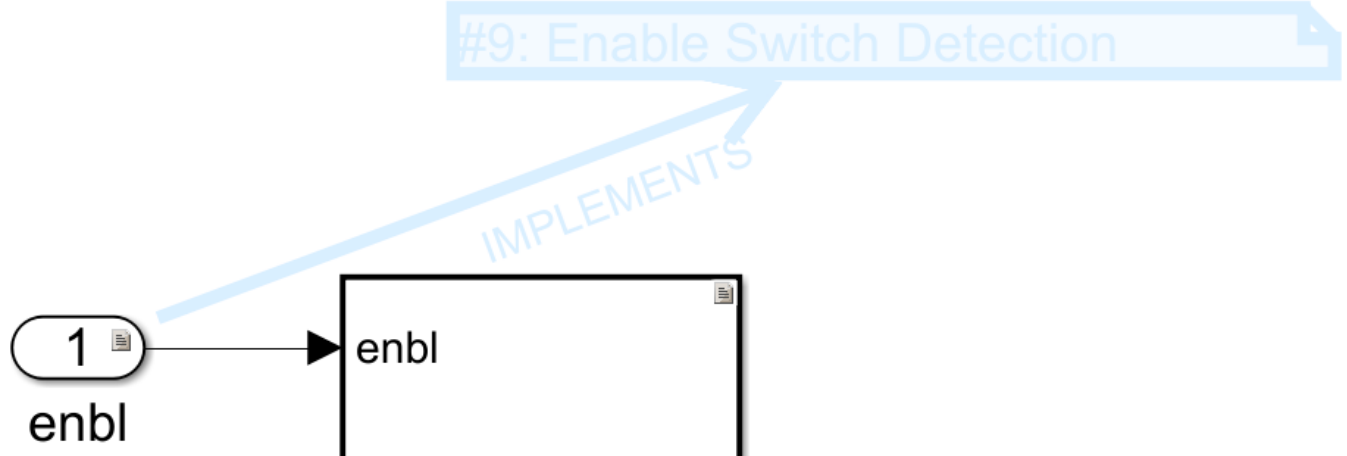
- 4 Open the requirements perspective by clicking the **Requirements** icon.



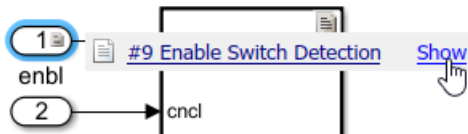
The Requirements Browser appears at the bottom of the model canvas. When you select a requirement, the Property Inspector displays the requirement's properties.

- 5 Link a requirement to a model element:

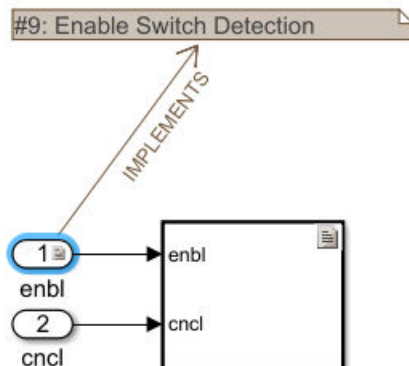
- 1 In the Requirements Browser, search for `Enable Switch Detection`.
- 2 Link to the `enb1` Inport block by clicking and dragging the requirement to the block. An annotation template appears.
- 3 Place the requirement annotation by clicking on the canvas. Create a link without an annotation by clicking outside the canvas.



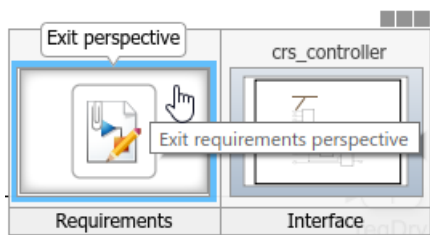
- 6 The block displays a link badge. To display information about the requirement, click the badge and select **Show**.



Clicking **Show** displays the requirement ID, requirement summary, and link type. For information on link types, see “Create and Store Links” on page 3-31.



- To see the requirement description, double-click the annotation.
 - To edit the requirement, right-click the annotation and select **Select in Requirements Browser**. Edit the requirement properties in the Property Inspector.
- 7 Exit the requirements perspective. Click the perspectives control and click the requirements icon.



Work with Simulink Annotations

Convert Simulink Annotations to Requirements

You can convert the annotations in your Simulink models to requirements by using the context menu in the Requirements Perspective View and by using the API. See `slreq.convertAnnotation` for more information on converting annotations to requirements by using the API.

To convert annotations to requirements by using the context menu in the Requirements Perspective View:

- 1 Open the Simulink model and enter the Requirements Perspective View.
- 2 Select a requirement set from the Requirements Browser. This is the destination requirement set for the new requirement.
- 3 Right click the annotation you want to convert to a requirement and click **Convert to Requirement**.
- 4 The annotation is converted to a requirement and is linked to the system or subsystem at which the annotation was present.

Link Requirements to Simulink Annotations

Use the Requirements Perspective View to link requirements to text and area annotations on the Simulink Editor. To create a link, select a requirement and drag it onto the annotation. If you link requirements to an area annotation, a badge appears on the annotation to show that the link was created. You see badges only in the Requirements Perspective View. To see more information about the requirement, click the badge and select **Show**.

See Also

More About

- “Create and Store Links” on page 3-31
- “Requirements Traceability for MATLAB Code” on page 10-2

Track Requirement Links with a Traceability Matrix

Traceability matrices allow you to easily view requirements and their links to other items. Traceability matrices show links between requirements, model or test entities, data dictionaries, and code, and allow you to navigate to link sources or destinations. For example, you can:

- View links between items.
- Create and delete links.
- Inspect and navigate link sources and destinations.
- Focus the display on a hierarchy of a specific artifact or item.
- Apply artifact-specific filters to rows, columns, and cells.
- View and highlight unlinked items.
- View and highlight items with associated change issues and clear the change issues.
- Perform batch operations when you select multiple cells.

Generate a Traceability Matrix

You can create a traceability matrix with two or more artifacts. You can use:

- Requirements Toolbox requirement sets
- Simulink models
- System Composer models
- Simulink Test test files
- Simulink data dictionaries
- MATLAB M-files

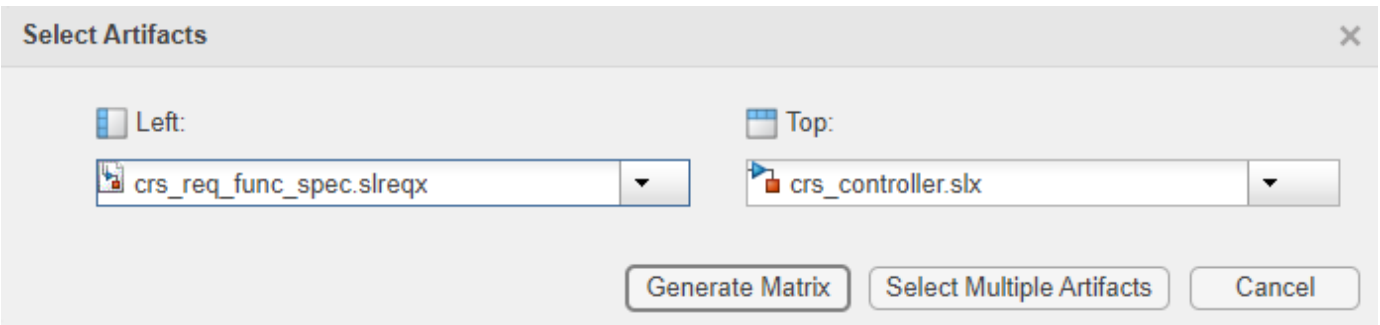
To open the Traceability Matrix window, use one of these approaches:

- In the **Requirements Editor**, click **Traceability Matrix**.
- In a Simulink model, in the **Requirements** tab, select **Share > Open Requirements Traceability Matrix**.
- At the MATLAB command line, enter:

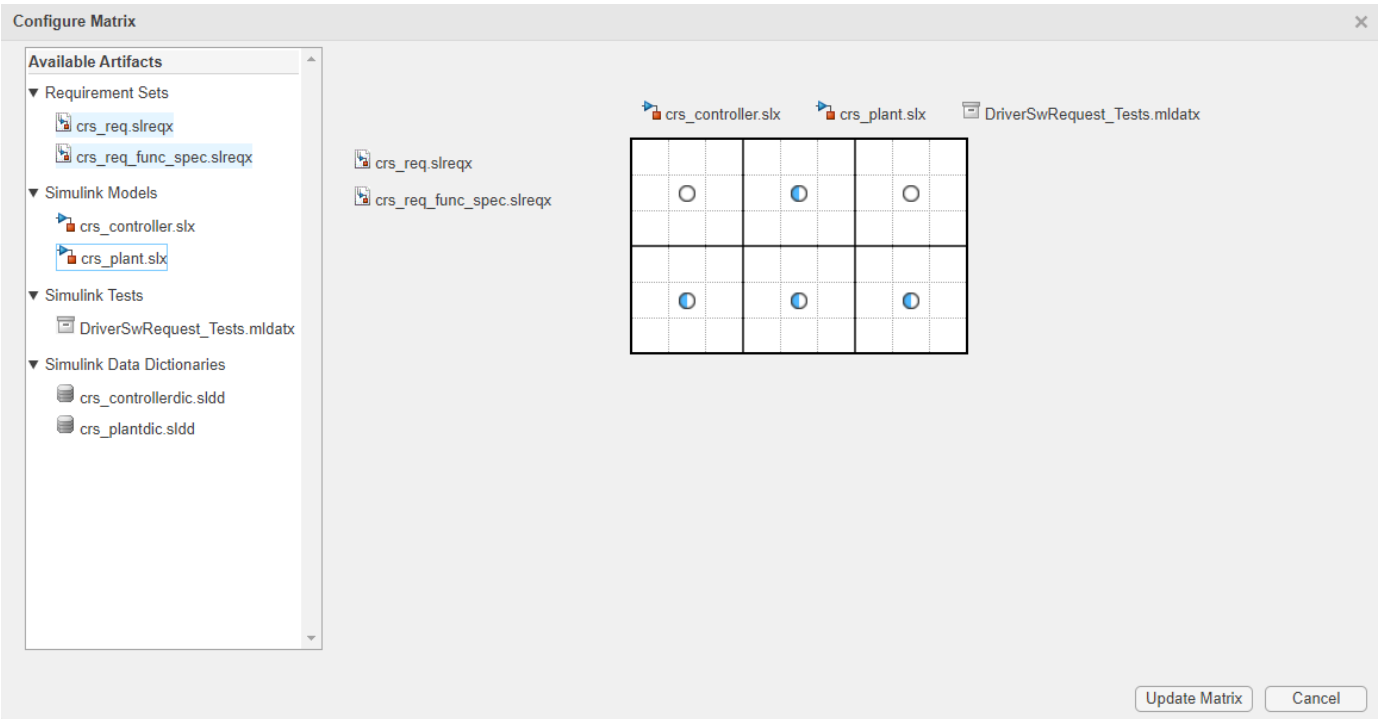
```
slreq.generateTraceabilityMatrix
```

To create a traceability matrix:

- 1** In the Traceability Matrix window, click **Add**.
- 2** Generate a matrix with either two artifacts or multiple artifacts.
 - To create a matrix with only two artifacts, select the **Left** and **Top** artifacts from the Select Artifacts dialog.



- To create a matrix with multiple artifacts, click **Select Multiple Artifacts**. In the Configure Matrix dialog, add artifacts from the **Available Artifacts** pane to the left or top artifact list by clicking and dragging, or by right-clicking the artifact and selecting **Add to the left** or **Add to the top**. Remove an artifact from a list by pointing to the artifact and clicking the remove icon (✖), or by right-clicking the artifact and selecting **Remove Artifacts**.



- Click **Generate Matrix**. You can reconfigure the artifacts in the matrix by clicking **Configure Matrix**, reconfiguring the artifacts, and clicking **Update Matrix**.

The artifacts in this image are a requirement set and a Simulink model. The requirements are listed on the left and the blocks of the Simulink model are listed on the top.

If you make changes to your artifacts, click **Update** to refresh your traceability matrix.

Note Unresolved links are not displayed in the traceability matrix.


When you create a traceability matrix with multiple artifacts, a solid blue line indicates the division between artifacts.

The screenshot displays the Traceability Matrix application. The top navigation bar includes 'HOME' and a help icon. Below it is a toolbar with various actions: 'Add', 'Configure Matrix', 'Highlight Missing Links', 'Create Link', 'Remove Links', 'Clear Change Issue', 'Update', 'Scope', 'Expand All', 'Collapse All', and 'Export'. The main interface is divided into a 'Filter Panel' on the left and a central workspace. The workspace title is 'Requirement Set vs Simulink Model'. It shows a list of artifacts at the top: 'crs_plant, crs_controller, DriverSwRequest Tests' and 'crs_req, crs_req_func_spec'. The central workspace contains a grid with columns for system components and rows for requirements. The components are: crs_plant, crs_controller, CruiseControlMode, DriverSwRequest, TargetSpeedThrottle, enbl, throtDnr, vehSp, cncl, set, and resume. The requirements are: crs_req, References to crs_req.docx, Overview, System overview, Functional Requirements, Interface specification, crs_req_func_spec, Driver Switch Request Handling, Cruise Control Mode, Calculate Target Speed and Th, System Interface, and Justifications. A blue vertical line is positioned between 'crs_controller' and 'CruiseControlMode', and a blue horizontal line is positioned between 'Interface specification' and 'crs_req_func_spec'. Blue circles are present in the grid cells at the intersections of these lines and other components/requirements.

Configure a Matrix with Multiple Artifacts

When you create or update a matrix with multiple artifacts, you can use the Configure Matrix dialog to arrange the artifacts by clicking and dragging to move an artifact from one list to another or reorder a list by dragging an artifact within a list.


You can add, remove, or arrange multiple artifacts at a time when you hold **Ctrl** and select multiple artifacts.

When you select an artifact in the **Available Artifacts** pane, any artifacts that contain links between the selected artifact are highlighted. When you add artifacts to the matrix configuration, the expand icon () in the matrix preview indicates that artifacts have links between them.

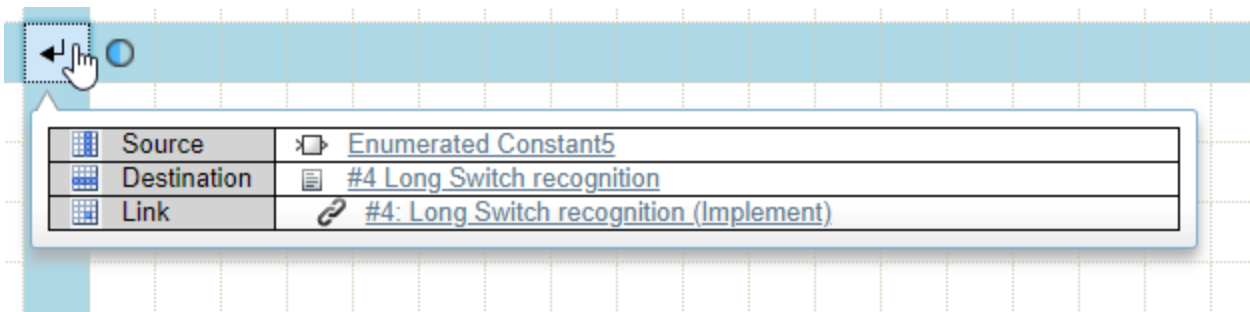
In order to be able to add an artifact to the traceability matrix, the artifact must either:

- Be loaded in your MATLAB workspace or Simulink
- Contain links to a loaded artifact
- Be associated with a loaded link set



Modify the Traceability Matrix View

The traceability matrix is a grid where the rows correspond to items from the left artifact and the columns correspond to items from the top artifact. The arrow icon () in a cell indicates that there is a link between the item in that row and column. The arrow icon points from the source item to the destination item.

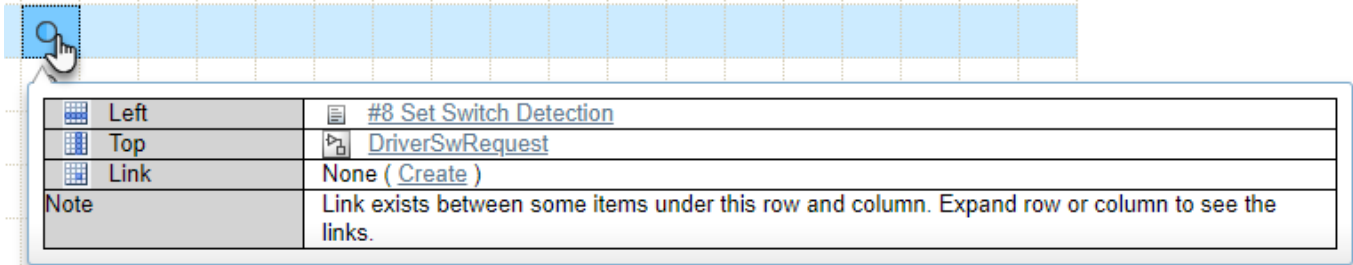
When you click an arrow icon, you see information about the link.



Expand and Collapse Links

Initially, some rows and columns in your matrix may be collapsed. The expand icon () appears when a link is obscured because one or both of the hierarchies in the row or column containing the linked items are collapsed. To expand the hierarchies, double-click the expand icon ().

When you click the expand icon, you see the left and top items that correspond to that cell.



When you click on the items in the information box, the item opens in the associated application for that artifact type. For example, if you click on a requirement, the **Requirements Editor** window opens and displays the specified requirement.

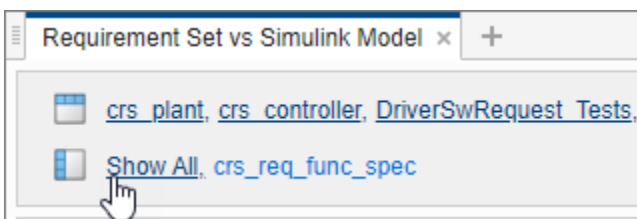
Focus the Display

You can focus the display on the hierarchy of a specific item in your traceability matrix. Select the artifact or item whose hierarchy you want to display. Click **Scope** or right-click the item and click **Focus the display**.



Your traceability matrix only shows the selected part of the hierarchy. To show the entire hierarchy of the artifact, right-click the artifact again and click **Display Entire Hierarchy**.

For matrices with multiple artifacts, you can also focus the display on one of the artifacts by clicking the artifact from the list at the top of the matrix. To remove the focus from just one artifact, click **Show All** in the artifact list at the top of the matrix.



To expand the hierarchy of an artifact, right-click on the artifact whose hierarchy you want to expand and click **Expand All**. To collapse the hierarchy of an artifact, right click on the artifact whose hierarchy you want to collapse and click **Collapse All**.

Apply Filters



You can apply filters from the **Filter Panel** to the top artifact, the left artifact, or the cells. Click the filter to apply it, and click it again to remove it.

Each artifact has type-specific filters. When you create a traceability matrix with multiple artifact types, the pane lists filters by artifact types and uses icons to indicate the type. The **Missing Links** filter and all filters under **Cell** always appear.



Filter Panel


Top


▼ **Link**

Missing Expected Links  
Missing Links

▼ **Type**

Models  
Leaf Block
Stateflow Object
Subsystem

Simulink Data Dictionary 
double
uint8

Simulink Tests 
Test Case
Test File
Test Suite

Left

▼ **Type**

Container
Functional
Justification

▼ **Link**

Missing Links

▼ **Change Tracking**

With Change Issues

Cell

▼ **Type**

Implements
Related to
Verifies

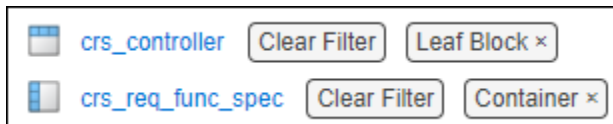
▼ **Change Tracking**

With Change Issues

If you apply a filter to an artifact, the matrix only shows items with those specific properties. For example, if, under **Top**, you click **Missing Links**, the traceability matrix only shows items from the top artifact that are not linked to other items. However, if a parent item does not have these specific properties but one or more of its children does, then the parent item and links to the parent item appear in the matrix, but are dimmed. For example, if you apply the **Leaf Block** filter to a model, the matrix shows subsystem blocks that contain leaf blocks, but dims subsystem blocks and links to subsystem blocks.

If you apply a filter to the cells, the matrix only shows the links that have those properties. However, no rows or columns are omitted. For example, if, under **Cell**, you click **With Change Issues**, the traceability matrix only shows the links that have change issues, but shows all rows and columns.

When you add a filter to the left or top artifacts of the traceability matrix, the filter appears at the top of the matrix next to the artifact name. You can clear the filters by clicking **Clear Filter** or, in the **Filter Panel**, by clicking the filter again.



If one of the artifacts in your traceability matrix is a Simulink model, then you can apply the **Missing Expected Links** filter. This filter displays unlinked Simulink blocks or subsystems that require links to meet HISL 0070 (Simulink).

Highlight Missing Links

To highlight unlinked cells in your traceability matrix, click **Highlight Missing Links**. The unlinked items in your traceability matrix are highlighted in yellow.

The screenshot displays a traceability matrix with a grid of cells. The top row contains system components, and the left column contains requirement items. A dashed blue box highlights the top-left corner of the grid. The requirement item '#2 Switch precedence' is highlighted in yellow. The system components are: crs_controller, CruiseControlMode, disableCaseDetection, IsKeyPositionOn, In1, Constant, Relational Operator, Out1, IsShiftDrive, and In1. The requirement items are: #1 Driver Switch Request Hand, #2 Switch precedence, #3 Avoid repeating command, #4 Long Switch recognition, #7 Cancel Switch Detection, #8 Set Switch Detection, #9 Enable Switch Detection, #10 Resume Switch Detectic, and #11 Increment Switch Detect.

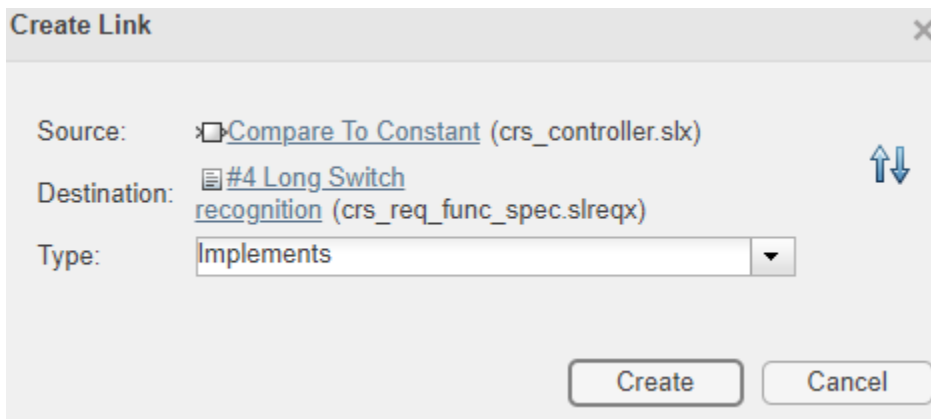
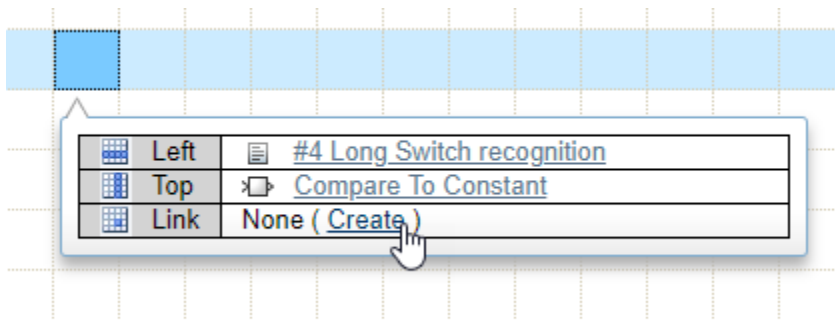
	crs_controller	CruiseControlMode	disableCaseDetection	IsKeyPositionOn	In1	Constant	Relational Operator	Out1	IsShiftDrive	In1
crs_req_func_spec										
#1 Driver Switch Request Hand										
#2 Switch precedence										
#3 Avoid repeating command										
#4 Long Switch recognition										
#7 Cancel Switch Detection										
#8 Set Switch Detection										
#9 Enable Switch Detection										
#10 Resume Switch Detectic										
#11 Increment Switch Detect										


The unlinked items are highlighted even if they are not visible in the current matrix view. View the hierarchy for the entire traceability matrix to see all items with missing links. See “Focus the Display” on page 3-10.

Work with Links in the Traceability Matrix

Add a New Link

Create a link by clicking on a cell, then click **Create Link** or **Create** in the information box to create a link between the item in the row and the item in the column.



The Create Link window populates the link source and destination. You can reverse the link source and destination by clicking the reverse button (). The link is saved in the link set associated with the artifact that the source item belongs to. If there is no link set associated with that artifact, a link set is created with the same name as the artifact.

Note If you create a traceability matrix using the same requirement set for the left and top artifact, you cannot create a link where the source and destination items are the same requirement. You also cannot create a link where the source or destination item is the requirement set.

Remove a Link

Remove a link by clicking on a cell containing a link and clicking **Remove Links** or pressing **Del**. The Remove Links dialog box appears and shows the link artifacts, type, and label. Click **Remove** to remove the link.

View and Clear Change Issues for Links

A link has a change issue if the requirement associated with the link changes. To learn how to enable change tracking and use the **Requirements Editor** to view and clear change issues, see “Track Changes to Requirement Links” on page 5-3.

You can view links with change issues in the traceability matrix by applying the **With Change Issues** filter or by selecting **Highlight Missing Links > Show Changed Links Only**. You can highlight links with change issues by clicking **Highlight Missing Links > Highlight Changed Links**. The row, column, and cell corresponding to the link with a change issue are highlighted in red.

The screenshot shows a software interface for tracking requirement links. The top navigation bar includes a 'HOME' tab and a toolbar with various actions: 'Add', 'Configure Matrix', 'Highlight Missing Links', 'Create Link', 'Remove Links', 'Clear Change Issue', 'Update', 'Scope', 'Expand All', 'Collapse All', and 'Export'. Below the toolbar is a 'Filter Panel' with sections for 'Top' (Type, Link), 'Left' (Type, Link, Change Tracking), and 'Cell' (Source, Destination, Link). The main area displays a traceability matrix with columns for 'crs_controller', 'DriverSwRequest', 'decrement', 'Enumerated Constant', 'Enumerated Constant2', and 'Switch2'. The matrix cells contain requirement names like 'Long Switch recognition' and 'Set Switch Detection'. A tooltip is visible over a cell, showing a table with columns 'Source', 'Destination', and 'Link', and values 'Switch2', 'Set Switch Detection', and '#8: Set Switch Detection (Implement)' respectively.

To clear a change issue for a link, select the cell containing the link and click **Clear Change Issue**.

Perform Batch Operations on Multiple Cells

Create a rectangular cell selection by clicking and dragging, or by pressing **Shift** and clicking the cells. You can press **Ctrl** and click to toggle cells in the selection or to create a selection of individual cells.

You can add or remove links or clear change issues for multiple links at a time when you select multiple cells.

Export the Traceability Matrix

You can export the traceability matrix as an HTML report or as a MATLAB variable that contains the table data.

Generate the HTML report by clicking **Export > Generate HTML Report**. Name and save the report. The report automatically opens.

The HTML report is not interactive. Create the view that you want to export by focusing the display, collapsing or expanding hierarchies, or applying filters and highlighting. The HTML report lists the file path to the artifacts in the matrix, as well as the focused display, applied filters, and highlighting.

Create a MATLAB variable that contains the table data by clicking **Export > Create MATLAB Variable**. The variable `slrtmxData` is created in the base MATLAB workspace. If you have an existing variable `slrtmxData` in your workspace, the variable is overwritten.

The exported MATLAB variable is not interactive, but has the functionality of a MATLAB table. See “Tables”. Create the view that you want to export by focusing the display or applying filters. The MATLAB table includes items in collapsed hierarchies, but does not include highlighting.

Work Programmatically with a Traceability Matrix

In addition to the Traceability Matrix window, you can also create a traceability matrix by using APIs. Use `slreq.getTraceabilityMatrixOptions` to create a structure and set the `leftArtifacts` and `topArtifacts` fields by providing a cell array containing artifact lists. Then use `slreq.generateTraceabilityMatrix` with the structure as an input argument to generate the matrix with the specified artifacts. See “Programmatically Generate a Traceability Matrix”.

See Also

`slreq.generateTraceabilityMatrix` | `slreq.getTraceabilityMatrixOptions`

Related Examples

- “Make Requirements Fully Traceable with a Traceability Matrix” on page 3-78
- “Programmatically Generate a Traceability Matrix”

More About

- “Link Blocks and Requirements” on page 3-2
- “Define Custom Requirement and Link Types by Using `sl_customization` Files” on page 3-42
- “Create and Store Links” on page 3-31
- “Track Changes to Requirement Links” on page 5-3

Visualize Links with Traceability Diagrams

You can visualize the traceability structure of requirements and other Model-Based Design items by using traceability diagrams. A traceability diagram graphically displays the links between an originating Model-Based Design item, such as a requirement, and the items linked to it, such as other requirements or Simulink blocks. For more information, see [Linkable Items](#) on page 3-32.

In the diagram, items are nodes and links are edges. The item that you generate the diagram from is the starting node. You can also generate an artifact-level diagram, where the artifacts, such as requirement sets or Simulink models, are nodes, and the link sets are edges.

A traceability diagram displays all items linked to the starting node, including all upstream nodes and downstream nodes. If an upstream node has further upstream links, the diagram also displays those linked items. Similarly, the diagram displays further downstream linked items for downstream nodes.

Whether nodes are upstream or downstream is determined by the impact direction, which describes how changes propagate between nodes. An upstream node impacts the starting node. A downstream node is impacted by the starting node. The impact direction is determined by the link type on page 3-34. For more information, see [“Impact Direction”](#) on page 3-20.

You can use a traceability diagram to assess requirements allocation and change propagation between linked Model-Based Design items. For more information, see [“Assess Allocation and Impact”](#) on page 3-26.

Generate Traceability Diagrams

You can create a traceability diagram from these objects:

- `slreq.Requirement`, `slreq.Reference`, or `slreq.Justification`
- `slreq.ReqSet`
- `slreq.Link`
- `slreq.LinkSet`

If you create a traceability diagram from a link, then the link source item is the starting node. Similarly, if you create a traceability diagram from a link set, then the link set artifact is the starting node.

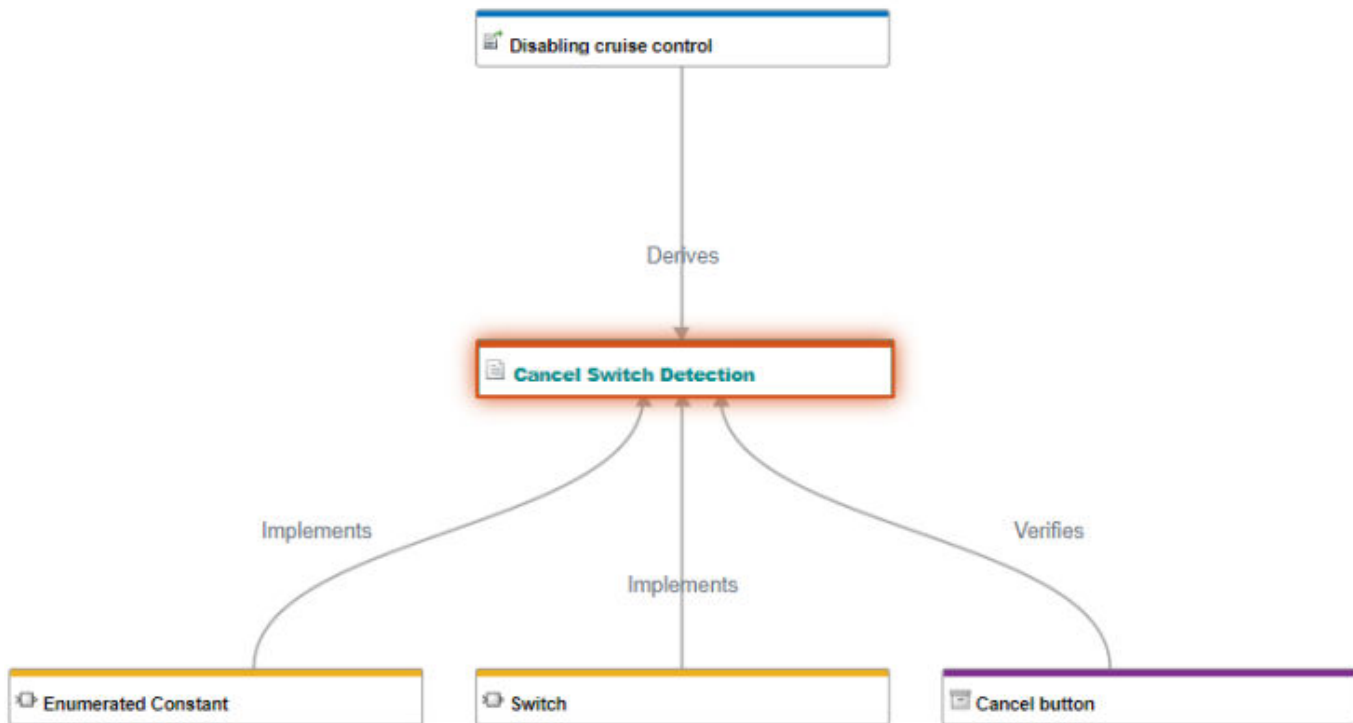
To create a traceability diagram:

- In the **Requirements Editor**, select the item and click **Traceability Diagram**.
- In the **Requirements Editor**, right-click the item and select **View Traceability Diagram**.
- At the MATLAB command line, use `slreq.generateTraceabilityDiagram`.

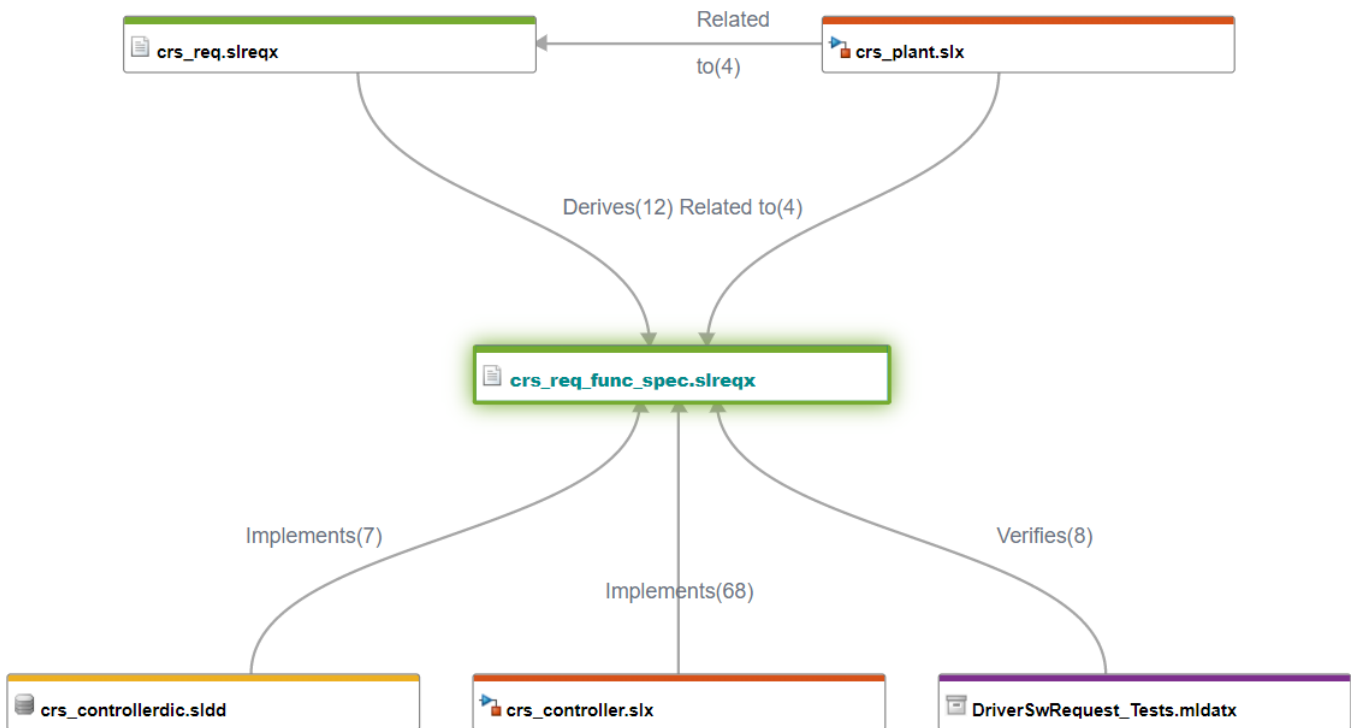
You can create a new diagram from a node in an existing diagram by right-clicking the node and selecting **View Traceability Diagram**.

Types of Traceability Diagrams

When you create a traceability diagram from a requirement, referenced requirement, justification, or link, the diagram is an item-level diagram. The nodes represent Model-Based Design items, such as requirements and Simulink blocks. The edges represent links between those items.



When you create a traceability diagram from a requirement set or link set, the diagram is an artifact-level diagram. The nodes represent Model-Based Design artifacts like requirement sets, Simulink models, and Simulink Test files. The edges represent links between items within the artifacts, such as links between requirements, Simulink blocks, and Simulink test cases.



Elements of a Diagram

Diagrams consist of nodes and edges.

Nodes represent Model-Based Design items or artifacts. The starting node of the diagram has blue text and a surrounding glow.

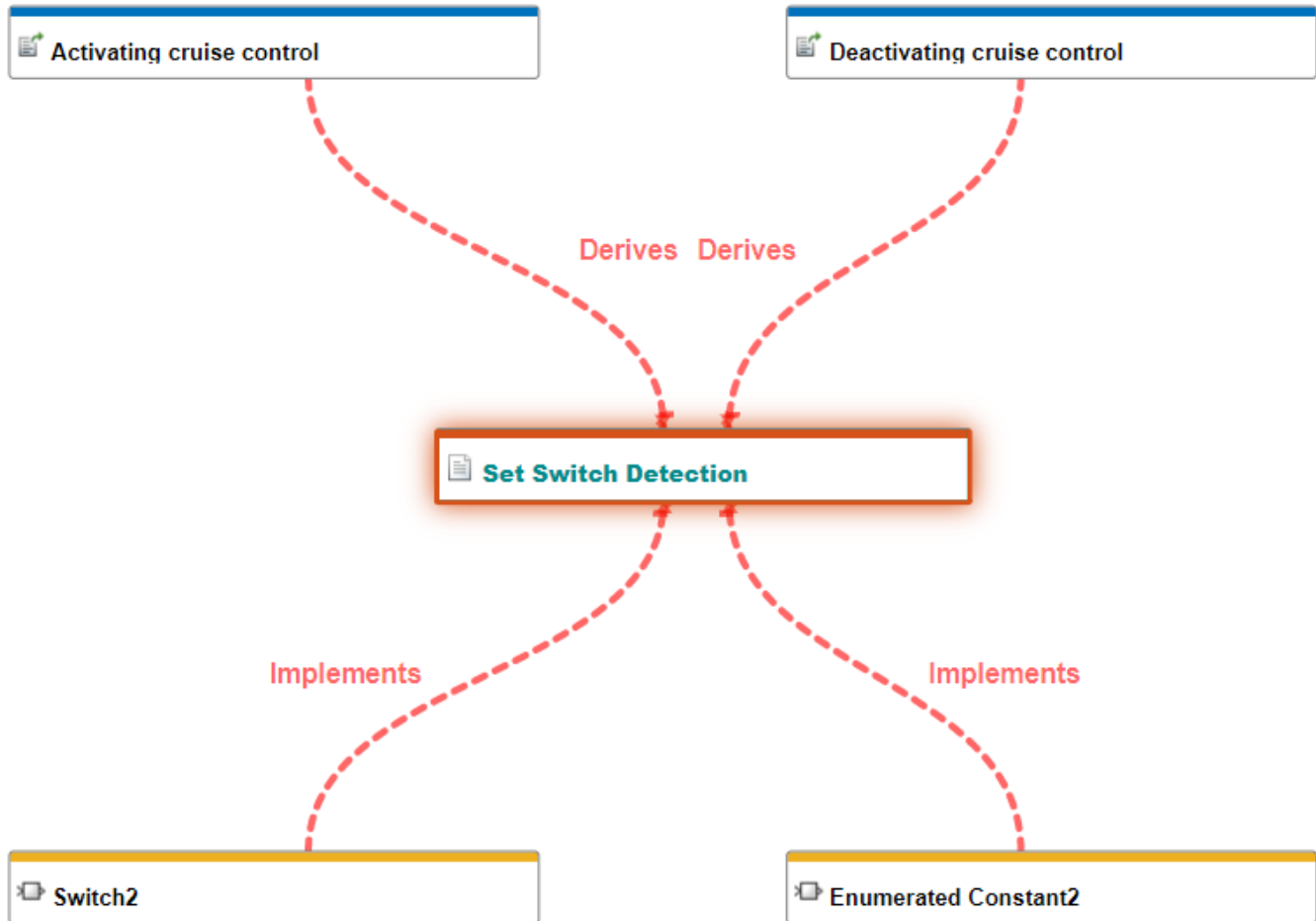


The node border color indicates the artifact that the node belongs to, or the artifact domain, such as Requirements Toolbox files, or Simulink models and libraries. The **Legend** pane displays the artifacts, the artifact colors, and the domain that each artifact belongs to.

For item-level diagrams, the warning icon (⚠) indicates an unavailable item. If the item is unavailable because it is not loaded, double-click the node. If the item is unavailable because the specified ID does not exist, then you must resolve the link. For more information, see “Unresolved Links” on page 3-40.

Edges are arrows represent links between Model-Based Design items or items within artifacts. The label of the edge is the link type on page 3-34 for item-level diagrams, and also includes the number of links of each type for artifact-level diagrams.

If a link has a change issue, the corresponding edge is a dashed red line. For more information about change issues, see “Track Changes to Requirement Links” on page 5-3.



The edge arrow points in the link direction, from the source node to the destination node. The link direction is not necessarily the same as the impact direction. For more information, see “Impact Direction” on page 3-20 and “Link Types” on page 3-34.

Impact Direction

The link type relationship between the starting node and a link determines the impact direction of that edge. For more information, see “Link Types” on page 3-34. Upstream nodes impact the starting node, while downstream nodes are impacted by the starting node.

This table summarizes the relationship between the link type and the impact direction.

Link Type	Upstream	Relationship	Downstream	Impact direction
Relate	Source	Related to	Destination	Same as link direction
Implement	Destination	Implemented by	Source	Opposite of link direction

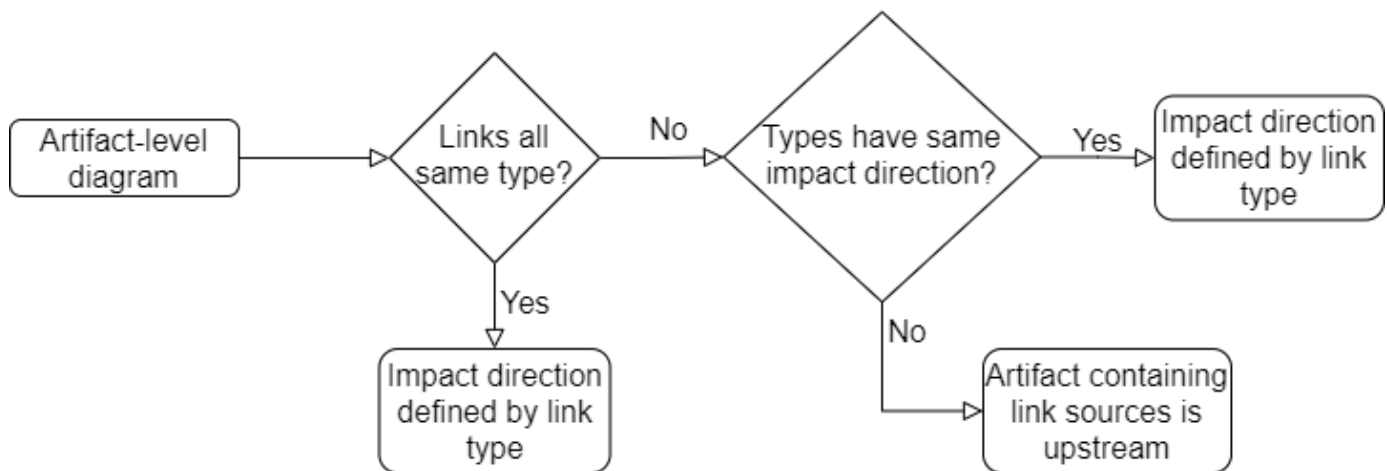
Link Type	Upstream	Relationship	Downstream	Impact direction
Verify	Destination	Verified by	Source	Opposite of link direction
Derive	Source	Derives	Destination	Same as link direction
Refine	Destination	Refined by	Source	Opposite of link direction
Confirm	Source	Confirmed by	Destination	Same as link direction

For example, if a link with the **Implement** type connects two nodes, then the destination is upstream and the source is downstream. The impact direction is opposite of the link direction because the impact is from the destination to the source.

You can use the impact direction to assess how changes propagate upstream and downstream. You can also use impact direction to assess requirements allocation. For more information, see “Assess Allocation and Impact” on page 3-26.

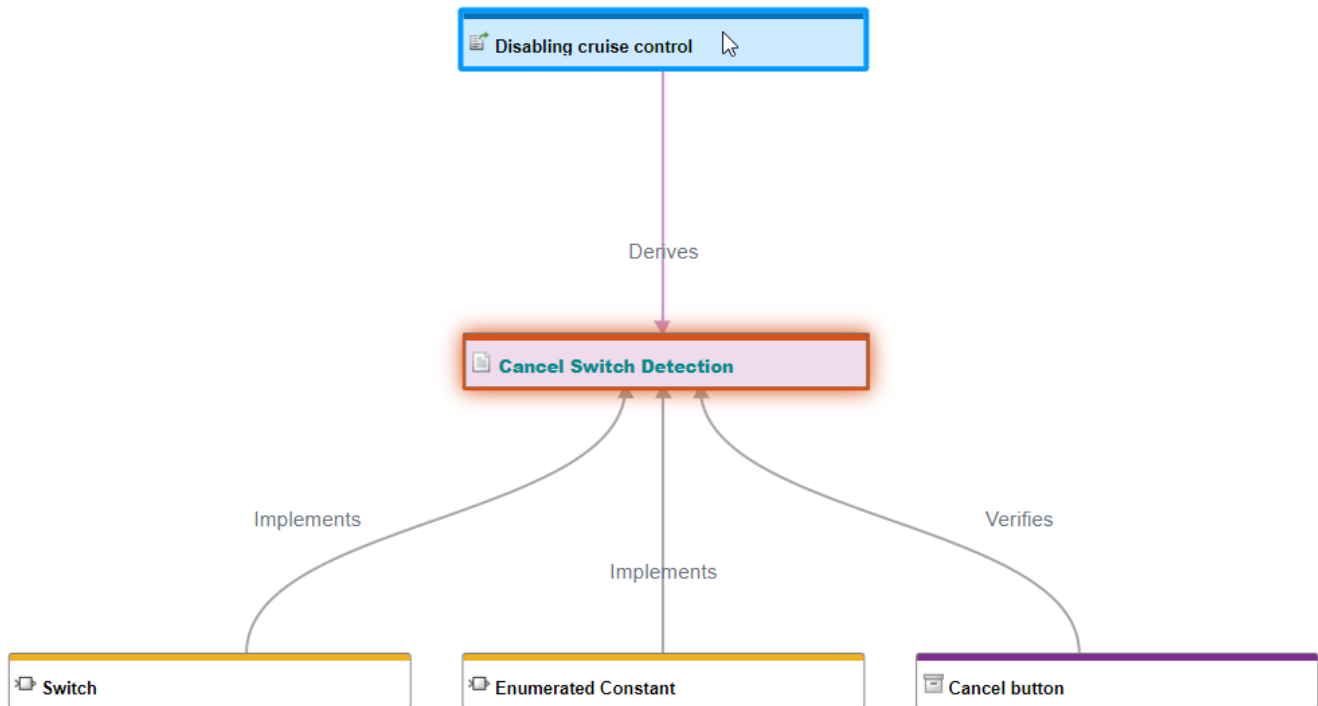
In an artifact-level diagram, if all links between items in two artifacts have the same type, then the link type defines the impact direction. If the links between items in two artifacts have different types, but all types define the impact direction the same way, they use the impact direction defined in the table. For example, both **Derive** and **Relate** link types define the source as upstream.

If the links between items in two artifacts have different types and the types define different impact directions, then the artifact containing the link source is defined as upstream, and the artifact containing the link destination is defined as downstream.



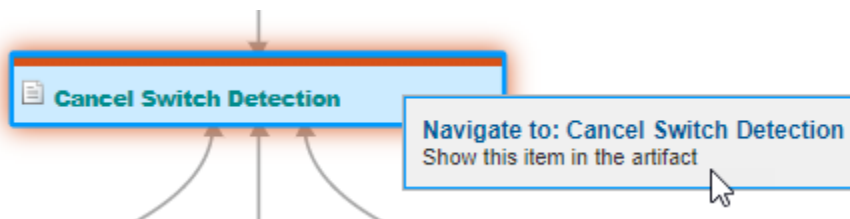
Use the Traceability Diagram

When you select a node, the diagram highlights the edges and nodes connected to the selected node.



Navigate from Node or Edge to Artifact

You can navigate from a node or edge to the corresponding item, artifact, link, or link set when you double-click the node or edge. You can also right-click the node or edge and select **Navigate to**. The node or edge opens in its respective artifact or domain.



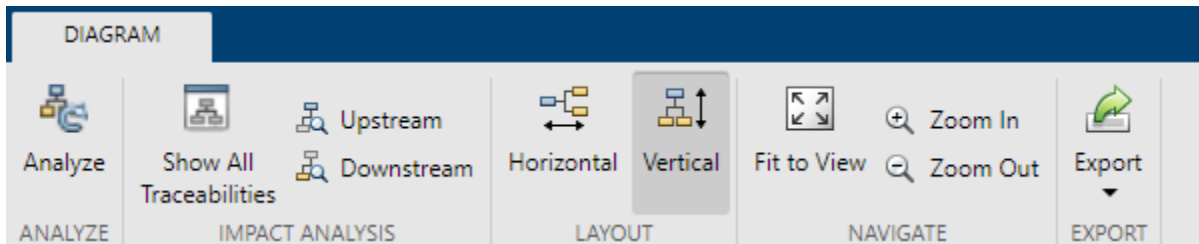
Refresh the Diagram

If you create a traceability diagram and then make a change in the background to any of the items, artifacts, or links, you must refresh the diagram to apply the changes to the diagram. If you load an unloaded item or resolve a link, you must refresh the diagram to remove the warning icon (⚠). For more information, see “Elements of a Diagram” on page 3-19.

Click **Analyze** to refresh the diagram.

Modify the Traceability Diagram View

You can modify the view in the Traceability Diagram window by using the toolstrip or the **Legend** and **Overview** panes.

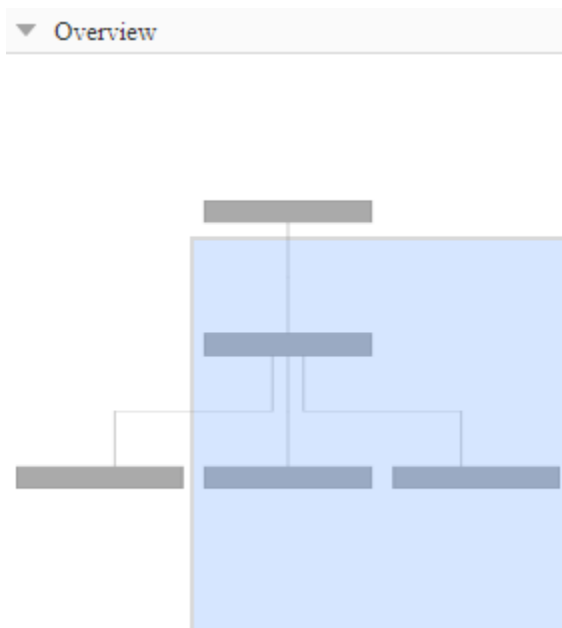


Layout and Navigation

By default, the diagram is laid out vertically. You can change the layout by selecting **Horizontal**.

When you create a diagram, it fits to the work area. You can zoom in by clicking **Zoom In** or by pressing **Ctrl+=** or zoom out by clicking **Zoom Out** or pressing **Ctrl+-**. You can also use the scroll wheel to zoom. You can fit the diagram to the work area again by clicking **Fit to View** or pressing **Space**.

You can also navigate by using the **Overview** pane, which displays a map of the diagram. The map shows which area of the diagram you are currently viewing.



You can navigate to another area by clicking or dragging in the map. You can also navigate to another area of the diagram by pressing **Ctrl** and using a scroll wheel, or clicking and dragging with a scroll wheel.

Filter Nodes by Impact Direction

You can filter nodes from the diagram by impact direction. To only view nodes that are upstream from the starting node, click **Upstream**. To only view nodes that are downstream from the starting node, click **Downstream**.

To clear the upstream or downstream filter, click **Show All Traceabilities**.

Filter Nodes by Artifact or Domain

You can use the **Legend** pane to filter nodes from the diagram by artifact or artifact domain. To filter nodes from a particular artifact, clear the selection for that artifact. To filter nodes from a domain, clear the selection for that domain.



Note For an artifact-level diagram, you can only filter by artifact domain.

Hide Edge Labels

By default, the diagram displays the link type as the label for each edge. You can hide the edge labels by right-clicking an edge or the white space in the diagram and clearing **Always show labels on edges**. After you clear the selection, the edge labels only appear when you select or point to an edge.

Export the Diagram

You can export the traceability diagram to a MATLAB digraph object. To export the diagram, in the Traceability Diagram window, select **Export > Export to MATLAB Variable**. You can use digraph object functions to work with the object and `plot` to visualize it.

You can use this code to create a figure that looks similar to the traceability diagram. In this code, the variable `graph_for_CancelSwitchDetection` is the name of the exported MATLAB digraph object.

```

dg = graph_for_CancelSwitchDetection;

h = plot(dg,NodeLabel=dg.Nodes.Name,EdgeLabel=dg.Edges.Labels, ...
        YData=cell2mat(dg.Nodes.LayerDepths), ...
        XData=cell2mat(dg.Nodes.IndexInCurrentLayer), ...
        interpreter="none",hittest="on",ArrowSize=15,MarkerSize=10);

nodeList = dg.Nodes.Name;

for index = 1:length(nodeList)
    cNode = nodeList(index);
    if (dg.Nodes.isStartingNode{index})
        h.highlight(cNode,NodeColor="b");
    end
end

```

```
    end
end

edgeList = dg.Edges.EndNodes;

for index = 1:length(edgeList)
    if (dg.Edges.HasChanged{index})
        h.highlight(edgeList{index,1},edgeList{index,2},EdgeColor="r");
    end
end
```

See Also

digraph | slreq.generateTraceabilityDiagram

More About

- “Assess Allocation and Impact” on page 3-26
- “Track Requirement Links with a Traceability Matrix” on page 3-5
- “Track Changes to Requirement Links” on page 5-3
- “Perform an Impact Analysis” (Simulink)
- “Trace Artifacts to Units and Components” (Simulink Check)

Assess Allocation and Impact

You can use the Traceability Diagram window to visualize links between Model-Based Design items. The diagram originates from a starting node that corresponds to an item and displays links, also called edges, from the starting node to other nodes that correspond to other linkable items on page 3-32. For more information, see “Visualize Links with Traceability Diagrams” on page 3-17.

Traceability diagrams also allow you to visually inspect the requirements allocation in a requirement set, which is a process of decomposing requirements and linking them to design elements and test for implementation and verification. Requirements allocation allows you to confirm that the design implements and verifies behavior that is required at a high-level, such as requirements that describe end-user needs.

You can also use a traceability diagram to visualize indirect links to assess how a change impacts and propagates between Model-Based Design items, especially when requirements change within a requirements hierarchy that contains multiple levels.

Assess Requirements Allocation

The process of requirements allocation allows you to confirm that functionality required at the high-level is implemented and verified by the design and tests, respectively. To allocate a single high-level requirement, you must:

- 1 Decompose the high-level requirement into one or more low-level requirements that are more detailed in order to allow for final implementation and verification. The low-level requirements should cumulatively capture the functionality required at the high level.
- 2 Create links between the high-level requirement and the decomposed requirements. This creates traceability from the high-level required functionality to low-level implementation and verification.
- 3 Implement and verify the low-level requirements by linking them to design and test items.

After you allocate a requirement, you can use a traceability diagram to create a diagram from that requirement and visually inspect the allocation. You can visualize the links to low-level requirements and to the implementation and verification items, such as Simulink blocks and test cases.

Decomposing Requirements

The first step of requirements allocation is to decompose your high-level requirements. Some requirements are too abstract and must be decomposed into low-level functional requirements that can be implemented and verified.

Decomposing high-level requirements into more detailed low-level requirements allows you to implement the requirements with design components that explicitly carry out the required functionality. Additionally, you can create tests that only require the part of the system that contains that component. This component-level implementation and verification helps requirements to remain implemented and verified when multiple components are integrated into a system.

For example, the files in the CruiseRequirementsExample project describe a cruise control system design. Open the project by entering the following command in the MATLAB command prompt:

```
slreqCCProjectStart
```

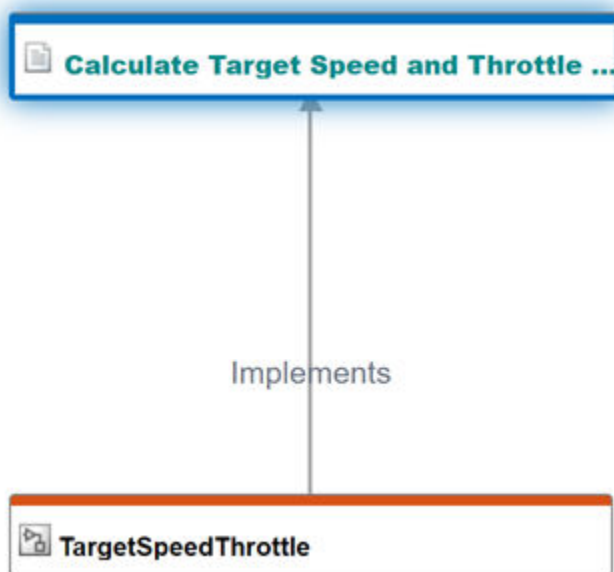
The `crs_req_func_spec` requirement set contains requirements that ensure that the system meets the functional specifications. The Calculate Target Speed and Throttle Value requirement is an example of a decomposed, high-level requirement.

3	#37	Calculate Target Speed and Throttle Value	Functional
3.1	#38	Disabled case	Functional
3.2	#39	Enabled case	Functional
3.3	#40	Activated case	Functional
3.3.1	#41	Throttle Value Computation	Functional
3.3.2	#42	Next Target Speed Computation	Functional
3.4	#43	Resume mode	Functional

Linking, Implementing, and Verifying Requirements

In order for a decomposed high-level requirement to be allocated, it must link to the corresponding low-level requirements. Additionally, each low-level functional requirement must have at least one outgoing link for implementation and one outgoing link for verification. If you intentionally did not implement or verify a low-level functional requirement, you can create a link to a justification and add information about why this requirement is exempt from implementation or verification. For more information, see “Justify Requirements” on page 4-17.

You must fully allocate your high-level requirements to visualize the design items in the Traceability Diagram window. For example, the Calculate Target Speed and Throttle Value is decomposed but it is not linked to its child requirements. The diagram below shows that it only has one link.



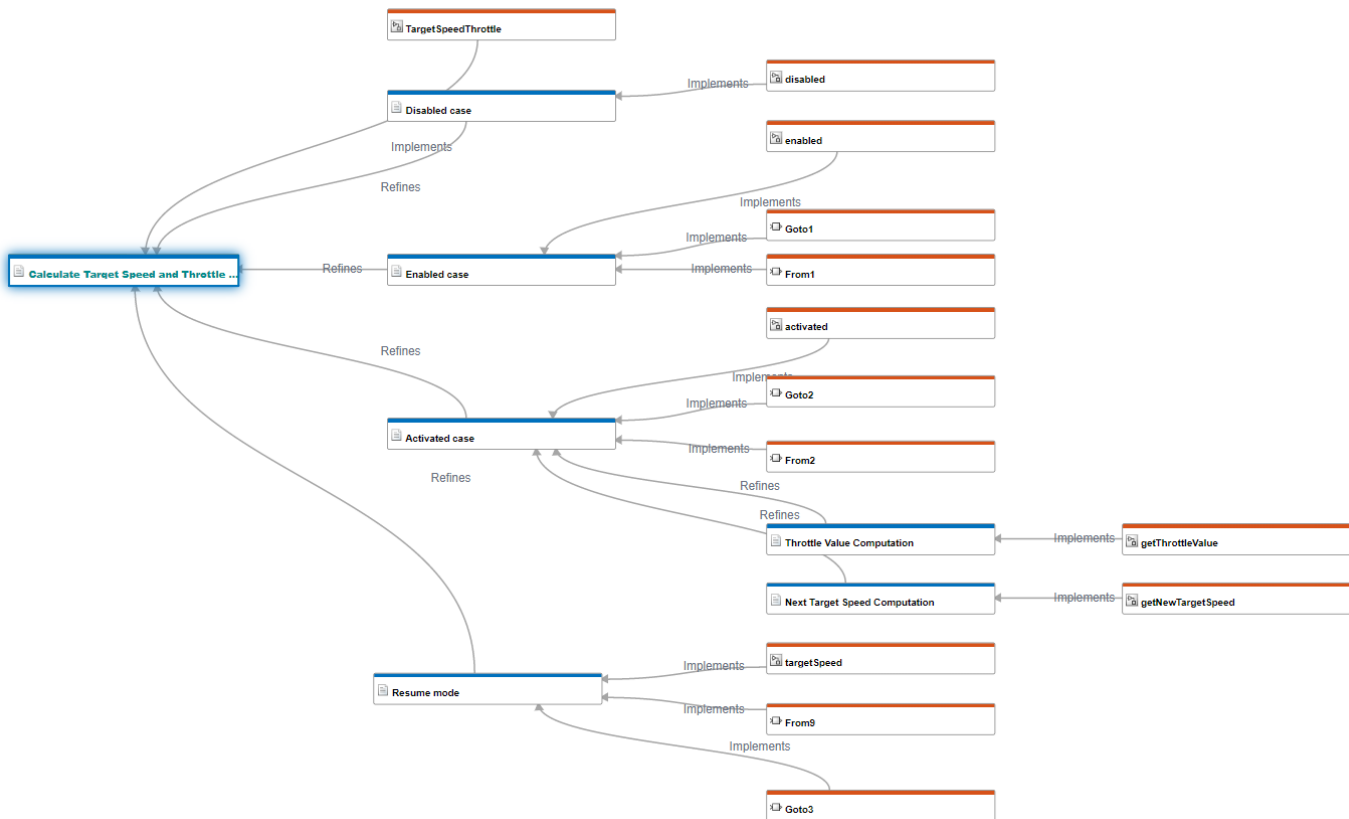
You can use the **Requirements Editor**, Requirements Perspective, or Traceability Matrix to create links. For more information, see:

- “Link Blocks and Requirements” on page 3-2
- “Link Test Cases to Requirements”
- “Track Requirement Links with a Traceability Matrix” on page 3-5

Visualizing Requirements Allocation

After you allocate your requirements, you can visualize the allocation by creating a traceability diagram from the high-level requirement. For more information, see “Generate Traceability Diagrams” on page 3-17.

For example, the diagram below originates from the Calculate Target Speed and Throttle Value requirement and shows added links between Calculate Target Speed and Throttle Value and its child requirements.



Each child requirement has at least one Implement type link to a model element. However, the child requirements do not have Verify type links to test items or to justifications, so the Calculate Target Speed and Throttle Value requirement is not considered fully allocated.

Visualize Change Propagation

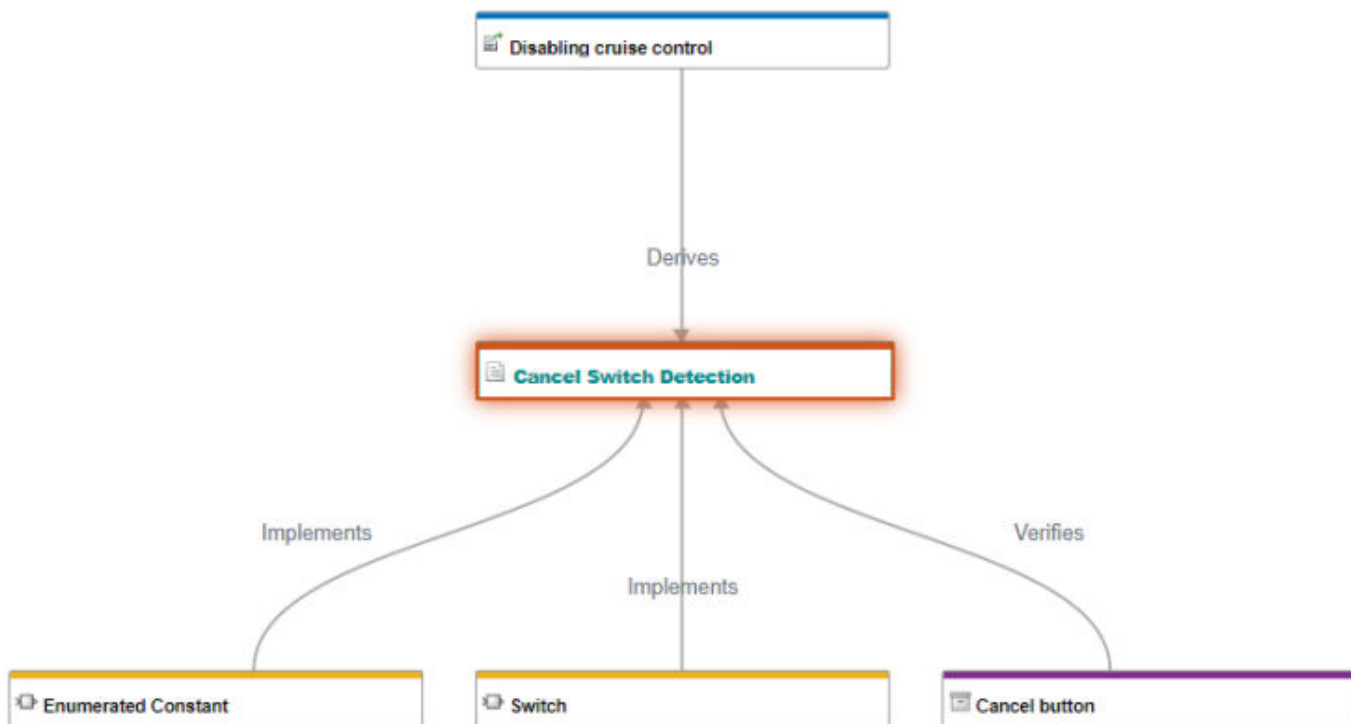
Requirements Toolbox allows you to track changes to requirements. If a linked requirement changes, the associated link has a change issue. For more information, see “Track Changes to Requirement Links” on page 5-3.

In the Traceability Diagram, links with change issues are shown as dashed red lines. Because change issues apply only to the immediate link when you change a linked requirement, you can use the Traceability Diagram to assess the change propagation for links that have change issues and items that indirectly link to a changed requirement.

Visualize Indirect Links

If you make a change to a requirement, a change issue is only applied to that immediate link. However, changes may affect other items that are indirectly linked. Indirect links are links between items that have at least one degree of separation. You can use a traceability diagram to visualize indirect links.

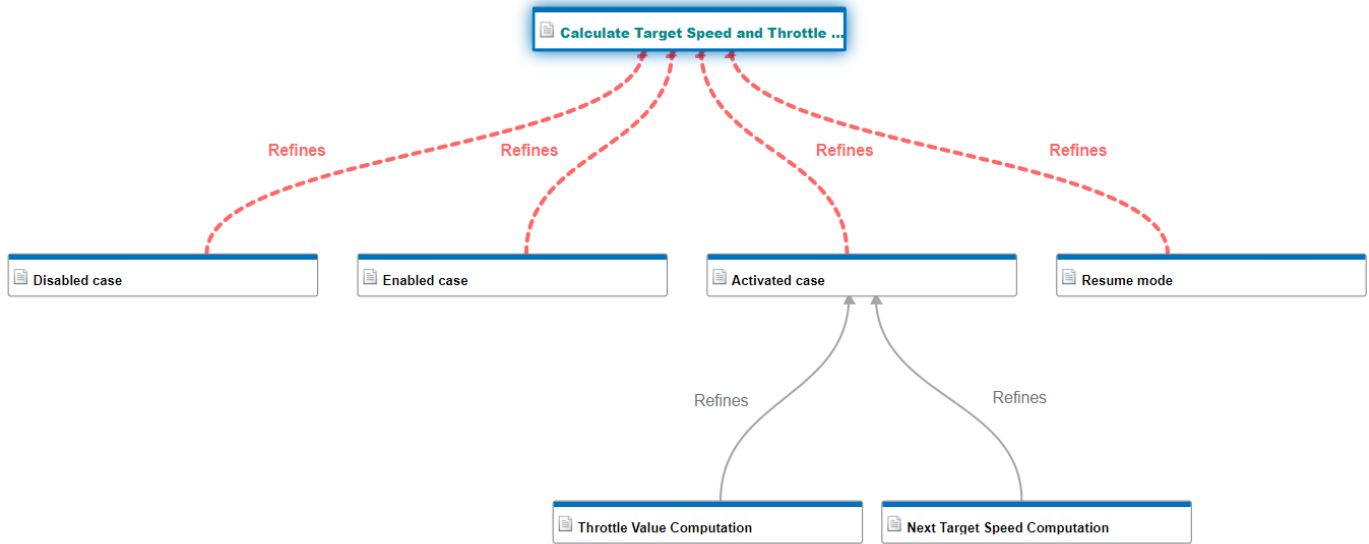
For example, this diagram shows an indirect link between the **Disabling cruise control** requirement and the **Enumerated Constant** Simulink block:



Assess Change Propagation

Changes can flow from the changed node through several layers of upstream or downstream nodes. You can use a traceability diagram to visualize how changes propagate through indirect links, and assess how the changes might affect further upstream or downstream nodes.

For example, the diagram below originates from the **Calculate Target Speed and Throttle Value** requirement and shows links to child requirements. A change has been made to the **Calculate Target Speed and Throttle Value** requirement. The diagram indicates that four outgoing links from the **Calculate Target Speed and Throttle Value** requirement have change issues.



Although the links to the grandchild requirements Throttle Value Computation and Next Target Speed Computation do not have change issues, the diagram shows that they are indirectly linked to the changed requirement - Calculate Target Speed and Throttle Value.

You can assess the impact that the change to the Calculate Target Speed and Throttle Value requirement has on the grandchild requirements by navigating to them in the **Requirements Editor**. For more information, see “Navigate from Node or Edge to Artifact” on page 3-22.

See Also

`slreq.generateTraceabilityDiagram`

More About

- “Visualize Links with Traceability Diagrams” on page 3-17
- “Review Requirements Implementation Status” on page 4-2
- “Review Requirements Verification Status” on page 4-6
- “Track Changes to Requirement Links” on page 5-3

Create and Store Links

You can use links and link sets to trace requirements to the design and test items that implement and verify them. Each link and link set has a corresponding API object. You can use the **Requirements Editor**, Requirements Perspective, a traceability matrix, or the MATLAB command line to create links between requirements, MATLAB code, Simulink blocks, Simulink Test test cases, and other items. You can also view and edit the links.

Link Objects, Sources, and Destinations

Each link has a corresponding `slreq.Link` object. You can provide additional traceability information by using link properties, custom attributes, and stereotypes. For more information, see:

- “Set Link Properties, Custom Attributes, or Stereotype Properties”
- “Add Custom Attributes to Links” on page 3-44
- “Customize Requirements and Links by Using Stereotypes” on page 1-71

Links point from source items to destination items, which are contained in source artifacts and destination artifacts, respectively. For example, consider a link that points from a Simulink block to a requirement. The Simulink model is the source artifact and the block is the source item. The requirement set is the destination artifact and it contains the requirement, which is the destination item.

A linked item has an outgoing link if it is the source of a link. Conversely, an item has an incoming link if it is the destination of a link. For example, if a link points from a Simulink block to a requirement, the Simulink block has an outgoing link and the requirement has an incoming link.

Link Storage

Link sets contain links. Each link set you load has a corresponding `slreq.LinkSet` object. When you create a link, Requirements Toolbox creates a link set and saves it as an SLMX file in the same folder as the source artifact. The link set name is the source artifact base name and the source artifact extension, separated by a tilde. For example, Requirements Toolbox stores outgoing links from a MATLAB function called `myFunction.m` in a link set file called `myFunction~m.slmx`.

Note Link sets that contain links from Simulink models combine the model base name and `~mdl` to prevent link resolution issues if the model file extension changes between `.mdl` and `.slx`. For example, Requirements Toolbox stores outgoing links from `vdp.slx` in a link set file called `vdp~mdl.slmx`.

Save Links

When you create links from artifacts such as Simulink models or Simulink Test files, you can save the changes to the link set by saving the artifact.

To save the changes to a link set when you create or edit links to lines of MATLAB code or plain-text external code, use one of these approaches:

- In the MATLAB Editor, right-click and select **Requirements > Save Links**.

- In the **Requirements Editor**, click **Show Links**. Select the link set, and click **Save**.
- At the MATLAB command line, use `save`.

Linkable Items

You can create links between these requirements items, model entities, test artifacts, and code:

- Requirements Toolbox objects:
 - `slreq.Requirement` objects
 - `slreq.Reference` objects
 - `slreq.Justification` objects
- Simulink entities:
 - Blocks
 - Subsystems
- Simulink data dictionary entries
- Stateflow objects:
 - States
 - Charts and subcharts
 - Transitions
- System Composer architecture entities:
 - Components
 - Ports
 - Views
- System Composer sequence diagram entities:
 - Lifelines
 - Gates
 - Messages
 - Fragments
- Simulink Test objects:
 - Test files
 - Test suites
 - Test cases
 - Iterations
 - Assessments
- Lines of MATLAB code in:
 - MATLAB code files (.m extension). For more information, see “Requirements Traceability for MATLAB Code” on page 10-2.
 - MATLAB Function blocks. For more information, see “Integrate Basic Algorithms Using MATLAB Function Block” (Simulink).

- MATLAB-based Simulink tests. For more information, see “Test Models Using MATLAB-Based Simulink Tests” (Simulink Test).
- Lines of code in plain-text external code files, such as C and H files. You can also link to HTML files, but not XML or JSON.

Note To create links to lines of plain-text external code, you must open the code in the MATLAB Editor. For more information, see “Link Requirements to MATLAB or Plain Text Code”.

Create Links

You can use the **Requirements Editor**, Requirements Perspective, a traceability matrix, or the MATLAB command line to create links.

To create a link from a design or test item to a requirement by using the **Requirements Editor**:

- 1 Select one of these items:
 - Simulink or Stateflow model element
 - System Composer architecture element
 - Simulink Test test case
 - Code range in the MATLAB Editor
 - Simulink data dictionary entry

Tip To link to MATLAB functions and enable change tracking for the entire body of the function, create the link to the line that contains the `function` keyword.

- 2 In the **Requirements Editor**, select the requirement.
- 3 Click **Add Link**, then select the menu option that contains the selection that you want to link to.

To create a requirement-to-requirement link:

- 1 Select the requirement that you want to link as the source item.
- 2 Click **Add Link > Select for Linking with Requirement**.
- 3 Select the requirement that you want to link as the destination item.
- 4 Create the link by selecting **Add Link > Create a link from**.

Tip

- You can create links without leaving the Simulink Editor by using the Requirements Perspective. For more information, see “Link Blocks and Requirements” on page 3-2.
 - You can create links between multiple artifacts in a single window by using a traceability matrix. For more information, see “Track Requirement Links with a Traceability Matrix” on page 3-5.
-

Create Links Programmatically

To create links at the MATLAB command line:

- 1 Get the object or handle for the link source. For example:

```
open_system("vdp")  
src = get_param("vdp/Mu", "Handle");
```

- 2 Get the object or handle for the link destination. For example:

```
dest = slreq.find(Type="Requirement", Summary="My requirement");
```

- 3 Use the `slreq.createLink` function to create the link.

```
newLink = slreq.createLink(src, dest);
```

To create links to MATLAB code or plain-text external code programmatically, use `slreq.TextRange` objects. For more information, see “Requirements Traceability for MATLAB Code” on page 10-2.

Link Types

Each link has a type that describes the relationship between the source and destination items. The `Type` property value of the `slreq.Link` object describes the link type.

Assigned Link Types

Each link type has an intended use case. For example, the **Implement** link type indicates a relationship between a requirement and a design item that implements the requirement. When you create a link between two items, Requirements Toolbox sets the link type and designates the items as the source or destination depending on the type of artifact that they belong to. For example, if you create a link between a requirement and a Simulink model element, Requirements Toolbox assumes that the model element implements the requirement, sets the link type to **Implement**, and designates the model element as the source and the requirement as the destination.

If there is no assumed link type for a link created between two items, then Requirements Toolbox sets the link type to **Relate**. For example, requirement-to-requirement links and links from plain-text external code default to **Relate**.

Edit the Link Type

After you create the link, you can edit the link type in the **Requirements Editor**, the Requirements Perspective, or at the MATLAB command line. In the **Requirements Editor**, click **Show Links**. Select a link and, in the right pane, under **Properties**, select the desired link type from the **Type** list.

Built-in Link Types

Requirements Toolbox provides six built-in link types.

The forward direction indicates how the source relates to the destination. Similarly, the backward direction indicates how the destination relates to the source.

Type	Description	Source-to-Destination Example	Forward Direction	Backward Direction
Relate	<ul style="list-style-type: none"> • General relationship between items for most use cases • Bi-directional link 	Requirement to requirement	The first requirement is related to the second requirement.	The second requirement is related to the first requirement.
Implement	<ul style="list-style-type: none"> • Specifies the source item that implements the requirement • Contributes to the implementation status <p>For more information, see “Review Requirements Implementation Status” on page 4-2.</p>	Simulink model element to requirement	The Simulink model element implements the requirement.	The requirement is implemented by the Simulink model element.
Verify	<ul style="list-style-type: none"> • Specifies which source item verifies the requirement • Contributes to the verification status if the source item is one of the accepted item types <p>For more information, see “Review Requirements Verification Status” on page 4-6.</p>	Simulink test case to requirement	The Simulink test case verifies the requirement.	The requirement is verified by the Simulink test case.

Type	Description	Source-to-Destination Example	Forward Direction	Backward Direction
Derive	Specifies which source item derives the destination item	Imported referenced requirement to requirement	The imported referenced requirement derives the requirement.	The requirement is derived from the imported referenced requirement.
Refine	Specifies which source item adds detail for the functionality specified by the destination item	Low-level requirement to high-level requirement	The low-level requirement refines the high-level requirement.	The high-level requirement is refined by the low-level requirement.
Confirm	<ul style="list-style-type: none"> Specifies a relationship between a requirement and an external test result source Can contribute to the verification status in certain cases <p>For more information, see “Include Results from External Sources in Verification Status” on page 4-28.</p>	Requirement to external test result	The requirement is confirmed by the external test result.	The external test result confirms the requirement.

The **Implement** and **Verify** link types describe requirement-to-model and requirement-to-test relationships. These links affect the implementation status and verification status. For more information, see “Review Requirements Implementation Status” on page 4-2 and “Review Requirements Verification Status” on page 4-6.

The link type also affects the impact direction in the Traceability Diagram window. For more information, see “Visualize Links with Traceability Diagrams” on page 3-17.



Custom Link Types

In addition to the built-in types, you can define custom link types. Custom link types must use one of the built-in types as the base behavior. The custom link type inherits some functionality from the built-in type, including how the link type contributes to the implementation and verification statuses. For more information, see “Choose a Built-in Type as a Base Behavior” on page 1-80.

You can define custom link types by using stereotypes or by using `sl_customization` files. For more information, see “Define Custom Requirement and Link Types and Properties” on page 1-78.

View and Edit Links

To view the loaded link sets and the links they contain, in the **Requirements Editor**, click **View Links**. You can then select a link and view or edit its properties or custom attributes. For more information, see “Set Link Properties, Custom Attributes, or Stereotype Properties”.


You can also view the links for a particular requirement in the **Requirements Editor** when you select a requirement. In the right pane, under **Links**, the outgoing links icon  indicates outgoing links and the incoming links icon  indicates incoming links.

To view links in the Simulink Editor by using the Requirements Perspective:

- 1 Open the Requirements Perspective in a Simulink model by clicking the **Show Perspectives views** icon in the lower-right corner of the model canvas and select **Requirements**.
- 2 In the **Requirements** tab, ensure that **Layout > Requirements Browser** is selected.
- 3 In the **Requirements** pane, in the **View** menu, select **Links**.

To visualize links, create a traceability matrix or traceability diagram. For more information, see “Track Requirement Links with a Traceability Matrix” on page 3-5 and “Visualize Links with Traceability Diagrams” on page 3-17.

Delete Links and Link Sets

To delete a link, in the **Requirements Editor**, click **Show Links**. Select a link and, in the **Links** section, click .

To delete a link set:

- 1 Locate the SLMX file that contains the link set. By default, link set files are in the same folder as the source artifact.
- 2 Close the linked artifacts before deleting a link set, including requirement sets, Simulink Test files, MATLAB code, Simulink data dictionaries, and Simulink, Stateflow or System Composer models.
- 3 At the MATLAB command line, clear the loaded requirements and links by entering:


```
slreq.clear
```
- 4 Delete the SLMX file.

Note If you want to delete a link set file associated with a Simulink model, ensure that the links are stored externally. For more information about how to store links externally, see “Requirements Link Storage” on page 6-4.

See Also

Requirements Editor | `slreq.createLink`

More About

- “Load and Resolve Links” on page 3-39
- “Track Requirement Links with a Traceability Matrix” on page 3-5

- “Add Custom Attributes to Links” on page 3-44

Load and Resolve Links

When you open or load an artifact that has incoming or outgoing links, Requirements Toolbox loads the link sets that contain those links. If the link sets have registered requirement sets, the software also loads those requirement sets. Loading registered requirement sets can further load links if the requirement set has incoming or outgoing links to additional artifacts.

When you load links, the software may identify unresolved links where the source or destination item are not available. You can resolve links by loading the artifact that contains the unloaded item or by repairing the link manually.

Load Artifacts and Associated Link Sets

When you load artifacts such as requirement sets, Simulink models, or MATLAB code that have incoming or outgoing links, Requirements Toolbox loads the link sets that contain those links.

For example:

- 1 At the MATLAB command line, enter this command to open a project:



```
slreqShortestPathProjectStart
```

- 2 Open the `shortest_path_func_reqs` requirement set. At the MATLAB command line, enter:

```
slreq.open("shortest_path_func_reqs");
```

The `shortest_path_func_reqs` requirement set has outgoing links to MATLAB code ranges in `shortest_path.m` and `graph_unit_tests.m`.

- 3 View the loaded link sets. In the **Requirements Editor**, click **Show Links**. The software loaded the link sets associated with `shortest_path.m` and `graph_unit_tests.m`.

	Label	Source	Type	Destination
>	 shortest_path.slmx			
>	 graph_unit_tests.slmx			

Load Registered Requirement Sets

When you create a link to a requirement, the requirement set that contains the requirement becomes registered to the link set. You can view the registered requirement sets for a link set by using the `getRegisteredReqSets` method.

When you load a link set that has registered requirements, Requirements Toolbox also loads those requirement sets. If those requirement sets have links to additional artifacts, the software also loads the link sets that contain those links.

For example:

- 1 Clear the loaded requirement sets and link sets by entering this command at the MATLAB command line:

```
slreq.clear
```

- 2 Open a project by entering this command at the MATLAB command line:

```
slreqShortestPathProjectStart
```

- 3 Open the `graph_unit_tests.m` MATLAB code file. At the MATLAB command line, enter:



```
edit graph_unit_tests.m
```

The `graph_unit_tests.m` file has outgoing links to requirements.

- 4 View the loaded link sets. Open the **Requirements Editor**.



```
slreq.editor
```

In the **Requirements Editor**, click **Show Links**. The software loaded `graph_unit_tests.slmx`, which is the link set that contains the outgoing links from `graph_unit_tests.m`.

Label	Source	Type	Destination
>  shortest_path.slmx			
>  graph_unit_tests.slmx			



The `graph_unit_tests.m` file has outgoing links to the `shortest_path_tests_reqs` and `shortest_path_func_reqs` requirement sets, so they are registered to the `graph_unit_tests` link set.

- 5 View the loaded requirement sets. In the **Requirements Editor**, click **Show Requirements**. Because the `shortest_path_tests_reqs` and `shortest_path_func_reqs` requirement sets are registered to the `graph_unit_tests` link set, they software also loaded them.

Index	ID	Summary
>  shortest_path_func_reqs		
>  shortest_path_tests_reqs		

The `shortest_path_func_reqs` requirement set also has incoming links from `shortest_path.m` stored in `shortest_path.slmx`.

- 6 View the loaded link sets again. In the **Requirements Editor**, click **Show Links**. Because the software loaded the `shortest_path_func_reqs` requirement set, it also loaded the link set that contains its incoming links, `shortest_path.slmx`.

Label	Source	Type	Destination
>  shortest_path.slmx			
>  graph_unit_tests.slmx			

Unregister Requirement Sets

To unregister a requirement set from a link set, use the `updateRegisteredReqSets` method. You can only unregister a requirement set from a link set if the requirement set has no incoming links stored in the link set.

Unresolved Links

An unresolved link has a source item or destination item that is not available. The source or destination items can be unavailable because:

- The artifact that contains the source or destination item is not loaded.


For example, if you load a requirement set that has incoming links from a Simulink model, this also loads the link set that belongs to the model. However, if you do not load the Simulink model, the links are unresolved.

- The artifact is loaded, but the specified ID does not exist. Links with invalid IDs are called broken links.

For example, if you delete a linked requirement, the link becomes unresolved because the stored ID no longer corresponds to a valid item.

To see the unresolved links, in the **Requirements Editor**, click **Show Links**. Unresolved links have the unresolved link icon .

Resolve Links

To resolve a link with an unloaded source or destination, load the artifact that contains the unloaded source or destination. You might be able to load the artifact by selecting the link in the **Requirements Editor** and, in the right pane, under **Properties**, click the source or destination item that has a warning icon .

To resolve a broken link, identify the location of the link source or destination, then use the `setSource` and `setDestination` methods to repair the link. Alternatively, delete the link and re-create it.

Unload Link Information

To unload link sets from memory, close both the source and destination artifacts that contain the linked items. If you close only the source artifact or only the destination artifact, the links remain loaded.

Alternatively, you can unload link sets by using the `slreq.clear` function to clear the loaded requirement sets and link sets and close the **Requirements Editor** and Traceability Matrix windows.

See Also

Requirements Editor | `getRegisteredReqSets` | `updateRegisteredReqSets`

More About

- “Create and Store Links” on page 3-31

Define Custom Requirement and Link Types by Using `sl_customization` Files

All requirement and link objects in Requirements Toolbox have a `Type` property. The `Type` property can be one of the built-in requirement types on page 1-6 or link types on page 3-34 or a custom requirement or link type. Custom requirement and link types must be a subtype of one of the built-in types and inherit functionality from that type.

Alternatively, you can use stereotypes to define custom requirement or link types that also have custom properties. For more information, see “Define Custom Requirement and Link Types and Properties” on page 1-78.

Create and Register Custom Requirement and Link Types

To create a custom requirement or link type:

- 1 Create an `sl_customization.m` file in the current working folder. In MATLAB, in the **Home** tab, click **New Script**. Copy and paste this code and save the file as `sl_customization.m`.

```
function sl_customization(cm)
    cObj = cm.SimulinkRequirementsCustomizer;
end
```

- 2 Add definitions to the customization file to create custom requirement types or custom link types.

Note Custom link types do not inherit the link direction from the built-in link type. When you create subtypes for the `Verify` or `Confirm` built-in types, use the same link direction as the built-in type so that the test item contributes to the verification status. For more information, see “Link Types” on page 3-34.

For example, this code creates a custom requirement type called `Heading` that is a subtype of the built-in requirement type `Container`. It also creates two custom link types called **Satisfy** and **Solve** that are subtypes of the built-in link types `Verify` and `Implement`, respectively. For more information, see “Requirement Types” on page 1-6 and “Link Types” on page 3-34.

```
function sl_customization(cm)
    cObj = cm.SimulinkRequirementsCustomizer;
    cObj.addCustomRequirementType('Heading',slreq.custom.RequirementType.Container,...
    'Headings for functional requirements');
    cObj.addCustomLinkType('Satisfy', slreq.custom.LinkType.Verify,'Satisfies', ...
    'Satisfied by','Links from verification objects to requirements');
    cObj.addCustomLinkType('Solve', slreq.custom.LinkType.Implement,'Solves', ...
    'Solved by','Links from implementation objects to requirements');
end
```

- 3 Register the customization. At the MATLAB command line, enter:

```
slreq.refreshCustomizations
```

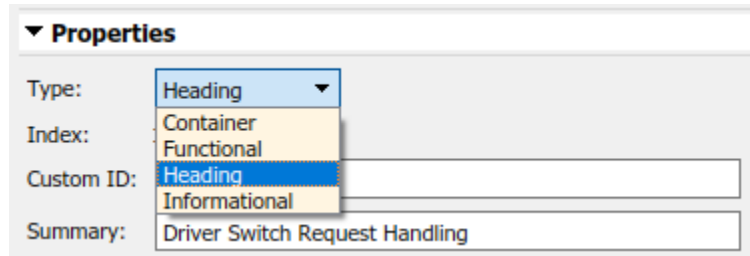
For more information, see `slreq.refreshCustomizations`.

Custom requirement and link types inherit some functionality from the built-in type they are a subtype of, including how they contribute to the implementation and verification status and the

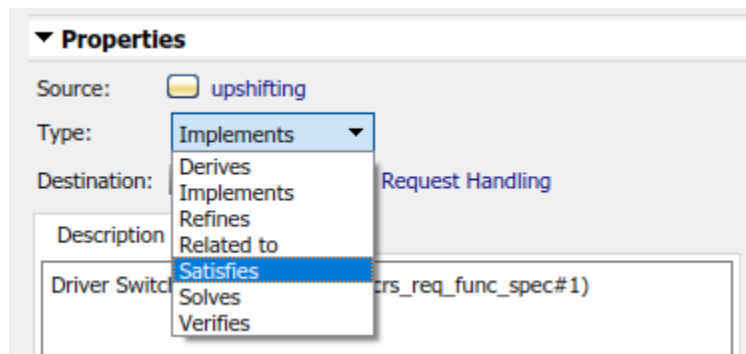
direction and impact of links. For more information, see “Choose a Built-in Type as a Base Behavior” on page 1-80.

Set the Type in the Requirements Editor

You can select the custom requirement or link type from the **Requirements Editor**. To set a requirement to a custom requirement type, click **Show Requirements** and select a requirement. In the right pane, under **Properties**, select the custom requirement type from the **Type** drop-down list.



To set a link to a custom link type, click **Show Links** and select a link. In the right pane, under **Properties**, select the custom link type from the **Type** drop-down list.



See Also

Requirements Editor | `slreq.refreshCustomizations`

More About

- “Requirement Types” on page 1-6
- “Link Types” on page 3-34
- “Customize Requirements and Links by Using Stereotypes” on page 1-71

Add Custom Attributes to Links

In this section...
“Define Custom Attributes for Link Sets” on page 3-44
“Set Custom Attribute Values for Links” on page 3-45
“Edit Custom Attributes” on page 3-46

When you create a link set using Requirements Toolbox, you can create custom attributes that apply to the links contained in the link set. Custom attributes extend the set of properties associated with your links.

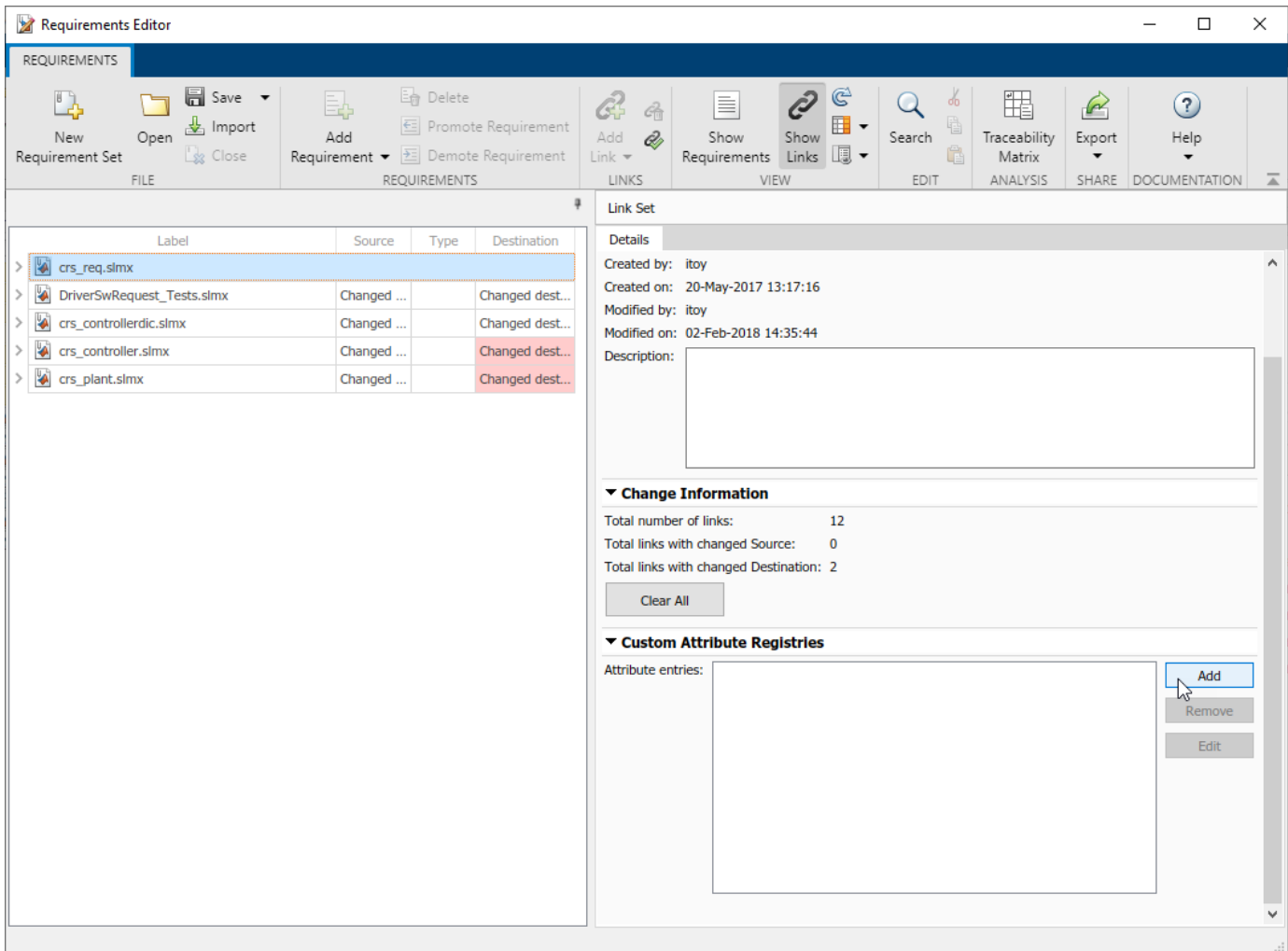
Alternatively, you can customize your links by creating stereotypes that define custom link types and properties. For more information, see “Define Custom Requirement and Link Types and Properties” on page 1-78.

Define Custom Attributes for Link Sets

To define a custom attribute for a link set:

- 1 Open the **Requirements Editor**. At the MATLAB command prompt, enter:

```
slreq.editor
```
- 2 Open a requirement set that contains requirement links, or create a new requirement set and create requirement links. For more information, see “Create and Store Links” on page 3-31.
- 3 Click **Show Links**.
- 4 Select the link set.
- 5 In the right pane, under **Custom Attribute Registries**, click **Add** to add a custom attribute to the link set.
- 6 The **Custom Attribute Registration** dialog box appears. Enter the name of your custom attribute in the **Name** field. Select the type from the **Type** drop-down menu. Enter a description of the custom attribute in the **Description** field.



Alternatively, you can programmatically add a custom attribute to a link set by using `addAttribute`.

Custom Attribute Types

There are four custom attribute types:

- **Edit**: Text box that accepts a character array. There is no default value.
- **Checkbox**: Single check box that can be either checked or unchecked. The default value is unchecked.
- **Combobox**: Drop-down menu with user-defined options. Unset is always the first option in the drop-down menu and the default attribute value.
- **DateTime**: Text box that only accepts a `datetime` array. There is no default value. See `datetime` for more information on `datetime` arrays.

Set Custom Attribute Values for Links

After you define custom attributes for a link set, you can set the custom attribute value for each link. Select the link in the **Requirements Editor**. In the right pane, under **Custom Attributes**, enter the desired value in the field.

Note You can only set the custom attribute value for one link at a time.

You can also set the custom attribute value for a link programmatically by using `setAttribute`.

If you do not define a value for `Checkbox` or `Combobox` type custom attributes for a link, the value will be set to the default. For `Checkbox` custom attributes, the default value is defined for the link set in the right pane, under **Custom Attribute Registries**. For `Combobox` custom attributes, the default value is `Unset`.

Edit Custom Attributes

After you define a custom attribute for a link set, you can make limited changes to the custom attribute. Select the link set in the **Requirements Editor**. In the right pane, under **Custom Attribute Registries**, select the custom attribute you want to edit and click **Edit**.

Alternatively, you can edit custom attributes programmatically by using `updateAttribute`.

For custom attributes of any type, you can edit the name and description. For `Combobox` custom attributes, you can also edit the value of each option in the drop-down menu, or add and remove options. If you remove an option, the custom attribute value for links that previously had that value reset to the default value, `Unset`. Editing the value of multiple options at once resets the custom attribute values of that attribute for all links in the link set. However, you can edit the value of a single option without resetting the custom attribute values. Additionally, if you reorder the existing options without changing the value of any of the options, the links' custom attribute values do not reset.

After you set the custom attribute value for a link, you can change the value at any time by selecting the link in the **Requirements Editor** and setting the updated value in the right pane, under **Custom Attributes**.

See Also

Apps
Requirements Editor

Classes
`slreq.LinkSet` | `slreq.Link`

Related Examples

- “Manage Custom Attributes for Links by Using the Requirements Toolbox API” on page 3-73

More About

- “Add Custom Attributes to Requirements” on page 1-81
- “Define Custom Requirement and Link Types and Properties” on page 1-78
- “Customize Requirements and Links by Using Stereotypes” on page 1-71

Requirements Consistency Checks

Check Requirements Consistency in Model Advisor

- “Identify requirement links with missing documents” on page 3-47
- “Identify requirement links that specify invalid locations within documents” on page 3-47
- “Identify selection-based links having description fields that do not match their requirements document text” on page 3-48
- “Identify requirement links with path type inconsistent with preferences” on page 3-49
- “Identify IBM Rational DOORS objects linked from Simulink that do not link to Simulink” on page 3-50

You can check requirements consistency using the Model Advisor.

Identify requirement links with missing documents

Check ID: mathworks.req.Documents

Verify 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.

Available with Requirements Toolbox.

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 fix the path name of the requirements document or move the document to the specified location.

Capabilities and Limitations

You can exclude blocks and charts from this check.

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.

Identify requirement links that specify invalid locations within documents

Check ID: mathworks.req.Identifiers

Verify 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.

Available with Requirements Toolbox.

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 fix the location reference within the requirements document.

Capabilities and Limitations

You can exclude blocks and charts from this check.

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.

Identify selection-based links having description fields that do not match their requirements document text

Check ID: mathworks.req.Labels

Verify 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.

Available with Requirements Toolbox.

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 Update 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 Edit/Add Links from the Requirements menu, and use the Requirements dialog box that appears to synchronize the text.

Capabilities and Limitations

You can exclude blocks and charts from this check.

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.

Identify requirement links with path type inconsistent with preferences

Check ID: mathworks.req.Paths

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.

Available with Requirements Toolbox.

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 Selection Linking 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"> • Click Fix to change the current path to the valid path. • In the Apps tab, click Requirements Viewer. In the Requirements Viewer tab, click Link Settings. <p>Select the Selection Linking tab, and change the value for the Document file reference option.</p>

Linux Check for Absolute Paths

On Linux systems, this check is named **Identify requirement links with absolute path type**. The check reports warnings for requirements links that use an absolute path.

The recommended action is:

- 1 Right-click the model object and select **Requirements > Edit/Add Links**.
- 2 Modify the path in the Document field to use a path relative to the current working folder or the model location.

Capabilities and Limitations

You can exclude blocks and charts from this check.

Identify IBM Rational DOORS objects linked from Simulink that do not link to Simulink

Identify IBM Rational DOORS objects that are targets of Simulink-to-DOORS requirements traceability links, but that have no corresponding DOORS-to-Simulink requirements traceability links.

Description

You have Simulink-to-DOORS links that do not have a corresponding link from DOORS to Simulink. You must be logged in to the IBM Rational DOORS Client to run this check.

Available with Requirements Toolbox.

Results and Recommended Actions

The Requirements Management Interface (RMI) examines Simulink-to-DOORS links to determine the presence of a corresponding return link. The RMI lists DOORS objects that do not have a return link to a Simulink object. For such objects, create corresponding DOORS-to-Simulink links:

- 1 Click the **Fix All** hyperlink in the RMI report to insert required links into the DOORS client for the list of missing requirements links. You can also create individual links by navigating to each DOORS item and creating a link to the Simulink object.
- 2 Re-run the link check.

Manage Navigation Backlinks in External Requirements Documents

A backlink is a navigation link in an external document that allows you to navigate from a requirement to the linked item in MATLAB or Simulink. A backlink either has a corresponding direct link that points from the item in MATLAB or Simulink to the external requirement, or matches a link that points from an item in MATLAB or Simulink to an imported referenced requirement, which is an `slreq.Reference` object that serves as a proxy object for the external requirement. See “Differences Between Importing and Direct Linking” on page 1-11.

When you create a direct link from an item in MATLAB or Simulink to a requirement in an external document, you can insert a backlink. You cannot insert a backlink when you create a link from an item in MATLAB or Simulink to an imported referenced requirement but you can insert one after you create the link. You can also remove backlinks from an external document if the original link was removed.

You can insert backlinks in:

- Microsoft Word documents
- Microsoft Excel spreadsheets
- IBM Rational DOORS modules
- IBM DOORS Next projects

Insert Backlinks in External Requirements Documents

When you create a link from an item in MATLAB or Simulink to an imported referenced requirement or a requirement in an external document, Requirements Toolbox creates a link as an `slreq.Link` object. In some cases, you can choose to insert a backlink in the external document when you create the direct link.

To update backlinks in the **Requirements Editor**:

- 1 Open the **Requirements Editor**. At the MATLAB command prompt, enter:

```
slreq.editor
```
- 2 In the **Requirements Editor**, click **Show Links** to view the loaded link sets.
- 3 Select the link set that contains the links that you want to use to insert backlinks or to remove stale backlinks. Right-click the link set and select **Update Backlinks**.
- 4 A dialog box displays the number of links checked and the number of backlinks added and removed. Click **OK**.

Alternatively, you can update backlinks programmatically by using `updateBacklinks`.

Note When you select **Update Backlinks** or use `updateBacklinks`, the software does not remove backlinks in IBM DOORS Next. Remove unwanted backlinks in DOORS Next in the DOORS Next interface.

See Also

`updateBacklinks`

More About

- “Navigate to Requirements in Microsoft Office Documents from Simulink” on page 7-11
- “Link to Requirements in Microsoft Word Documents” on page 7-2
- “Link to Requirements in Microsoft Excel” on page 7-7
- “Link and Trace Requirements with IBM DOORS Next” on page 8-4

Requirements Traceability for Code Generated from MATLAB Code

When you generate C/C++ code from MATLAB code that has links to requirements, you can include comments in the generated code that contain information about the requirements and the linked MATLAB code ranges. When you view the generated code from a code generation report, the comments are hyperlinks that you can use to navigate to the requirement and the linked MATLAB code range.

Note To include requirement comments in generated code, you must have MATLAB Coder™ and Embedded Coder®.

Include Requirement Comments in Generated Code

If your MATLAB code contains links to requirements, you can include requirement comments when you generate code by using the **MATLAB Coder** app or the `codegen` function. For more information, see “Generate C Code by Using the MATLAB Coder App” (MATLAB Coder) and “Generate C Code at the Command Line” (MATLAB Coder).

Include Requirement Comments by Using the MATLAB Coder App

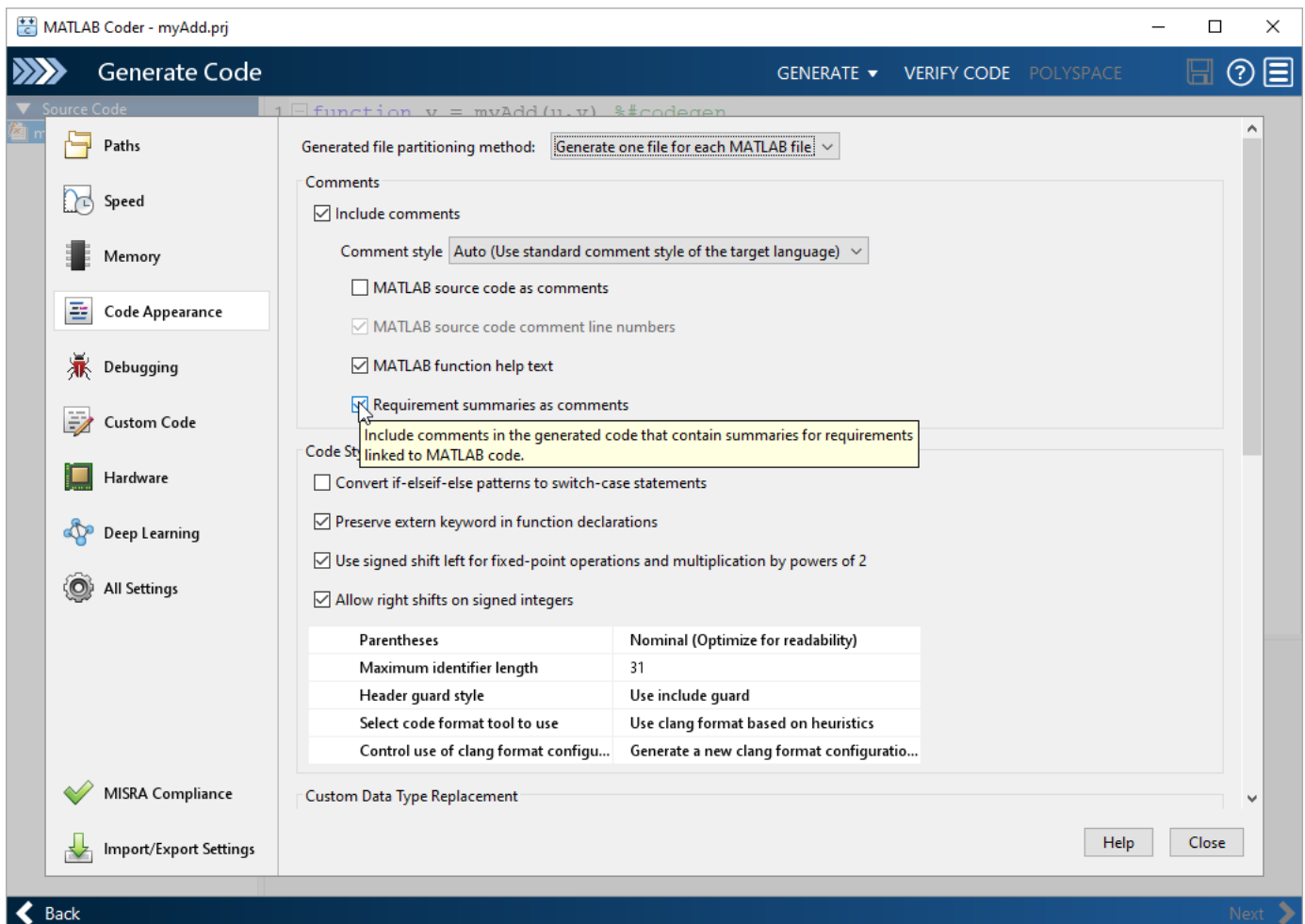
To include requirement comments in generated code by using the **MATLAB Coder** app:

- 1 Open the **MATLAB Coder** app. In the **Apps** tab, under **Code Generation**, click **MATLAB Coder**. Alternatively, at the MATLAB command prompt, enter `coder`.
- 2 In the **Generate code for function** field, enter the name of your MATLAB function, then click **Next**. For more information, see “Open the MATLAB Coder App and Select Source Files” (MATLAB Coder).
- 3 Define the input types for the function, then click **Next**. For more information, see “Define Input Types” (MATLAB Coder).
- 4 Check for run-time issues by clicking **Check for issues**. If no issues are detected, click **Next**. For more information, see “Check for Run-Time Issues” (MATLAB Coder).
- 5 In the **Build type** menu, select one of these options:
 - Source Code
 - Static Library (.lib)
 - Dynamic Library (.dll)
 - Executable (.exe)

For more information, see “Generate C Code” (MATLAB Coder).

Note You cannot include requirement comments in generated MEX functions.

- 6 To include requirement comments in the generated code, click **More Settings**. In the left pane, click **Code Appearance**. Under **Comments**, ensure that **Include comments** is selected, then select **Requirement summaries as comments**.



- 7 To generate a code generation report, in the left pane, click **Debugging**. Under **Code Generation Report**, ensure that **Always create a report** is selected.
- 8 After you configure any additional configuration parameters, click **Close** to close the code configuration parameters menu, then click **Generate** to generate the code. For more information, see “Generate C Code” (MATLAB Coder).

Include Requirement Comments Programmatically

Suppose that you want to generate code and include requirement comments for a MATLAB function called myAdd by using the codegen function.

```
function y = myAdd(u,v) %#codegen
y = u + v;
end
```

Suppose that the function has links to these requirements:

myAddRequirements		
1	#1	Input u
2	#2	Input v
3	#3	Add u and v
4	#4	Output y

To include requirement comments in the generated code:

- 1 Use `coder.config` with the `ecoder` flag set to `true` to create a `coder.EmbeddedCodeConfig` object. You can use LIB, DLL, or EXE build types.

```
cfg = coder.config("lib","ecoder",true);
```

- 2 Set the "ReqsInCode" (MATLAB Coder) property of the `coder.EmbeddedCodeConfig` object to `true`.

```
cfg.ReqsInCode = true;
```

- 3 Set any additional code configuration parameters by modifying the properties of the `coder.EmbeddedCodeConfig` object. For more information, see "Generate C Code at the Command Line" (MATLAB Coder).

- 4 Define function input data types and sizes with `coder.typeof`. For more information, see "Defining Input Types" (MATLAB Coder).

```
utype = coder.typeof(1);
```

```
vtype = coder.typeof(1);
```

- 5 Generate the code by using `codegen`. Use these flags as input arguments:

- `-config` to specify the code configuration object to use during code generation
- `-args` to specify the function input types and sizes
- `-launchreport` to generate and launch the code generation report

```
codegen myAdd -config cfg -args {utype,vtype} -launchreport
```

View Comments in Generated Code

You can view the requirement comments by opening the generated entry-point C file, which has the same name as the MATLAB entry-point function. Each comment corresponds to a requirement link. The comment includes the:

- Full file path to the MATLAB function
- ID that represents the linked code range
- Lines of MATLAB code that are linked to the requirement
- Requirement summary

If multiple requirements are linked to the same code range, the comment lists the requirement summaries as a numbered list under the information about the linked code range.

For example, suppose you include requirement comments when you generate code from the `myAdd` function. The generated `myAdd.c` entry-point file contains comments that correspond to the requirement links:

```
double myAdd(double u, double v)
{
```

```

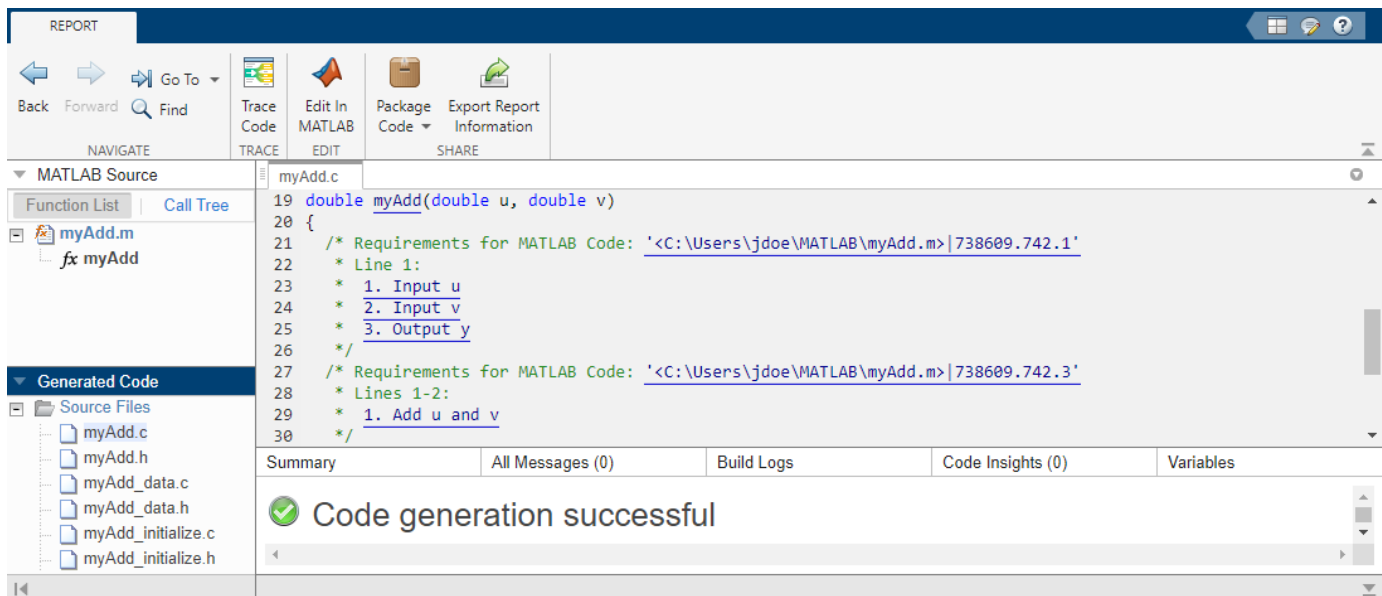
/* Requirements for MATLAB Code: '<C:\Users\jdoe\MATLAB\myAdd.m>|738609.742.1'
 * Line 1:
 * 1. Input u
 * 2. Input v
 * 3. Output y
 */
/* Requirements for MATLAB Code: '<C:\Users\jdoe\MATLAB\myAdd.m>|738609.742.3'
 * Line 2:
 * 1. Add u and v
 */
return u + v;
}

```

Navigate to Requirements from Code Generation Report

MATLAB Coder code generation reports allow you to view the generated C/C++ code and trace the generated code to MATLAB source code. For more information, see “Code Generation Reports” (MATLAB Coder).

When you view the generated entry-point C file in a MATLAB Coder code generation report, the requirement comments are hyperlinks that you can use to navigate to the linked requirements in the **Requirements Editor** and the linked MATLAB code range in the MATLAB Editor.



Alternatively, you can trace between the generated code and the MATLAB source code without leaving the code generation report by using the **Trace Code** button. For more information, see “Interactively Trace Between MATLAB Code and Generated C/C++ Code” (Embedded Coder).

See Also

codegen | coder.EmbeddedCodeConfig

More About

- “Requirements Traceability for MATLAB Code” on page 10-2

- “Generate Code for Models with Requirements Links” on page 12-8

Link from Sequence Diagrams

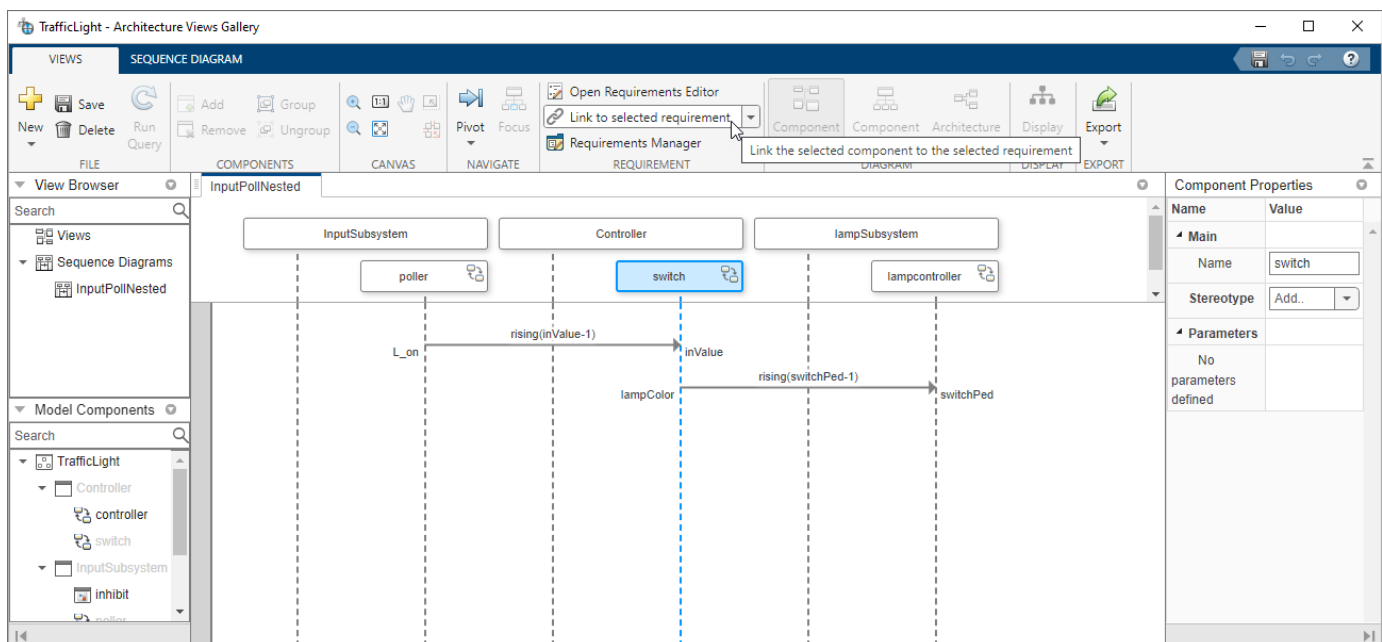
You can associate requirements for system behavior interactions with the model elements that implement the behavior by creating links from sequence diagrams. You can create links from lifelines, gates, messages, fragments, and sequence diagrams themselves. Links from sequence diagrams to requirements contribute to the requirements implementation status in the **Requirements Editor**.

You can use System Composer sequence diagrams to describe expected system behavior as a sequence of interactions between the components of an architecture model. For more information, see “Describe System Behavior Using Sequence Diagrams” (System Composer).

Create Links

To create links:

- 1 Open a sequence diagram in the **Architecture Views Gallery**.
- 2 Open the **Requirements Editor**. In the **Views** tab, in the **Requirement** section, click **Open Requirements Editor**.
- 3 In the **Requirements Editor**, open a requirement set.
- 4 Select a requirement.
- 5 In the **Architecture Views Gallery**, select an element in the sequence diagram, or select the sequence diagram itself by clicking the white space in the diagram.
- 6 Create the link. In the **Views** tab, in the **Requirement** section, click **Link to selected requirement**.



Requirements Toolbox sets the link type depending on the type of model element that you link from.



Model Element	Link Type
<ul style="list-style-type: none"> Lifelines Gates 	Implement
<ul style="list-style-type: none"> Messages Fragments Sequence diagrams 	Verify

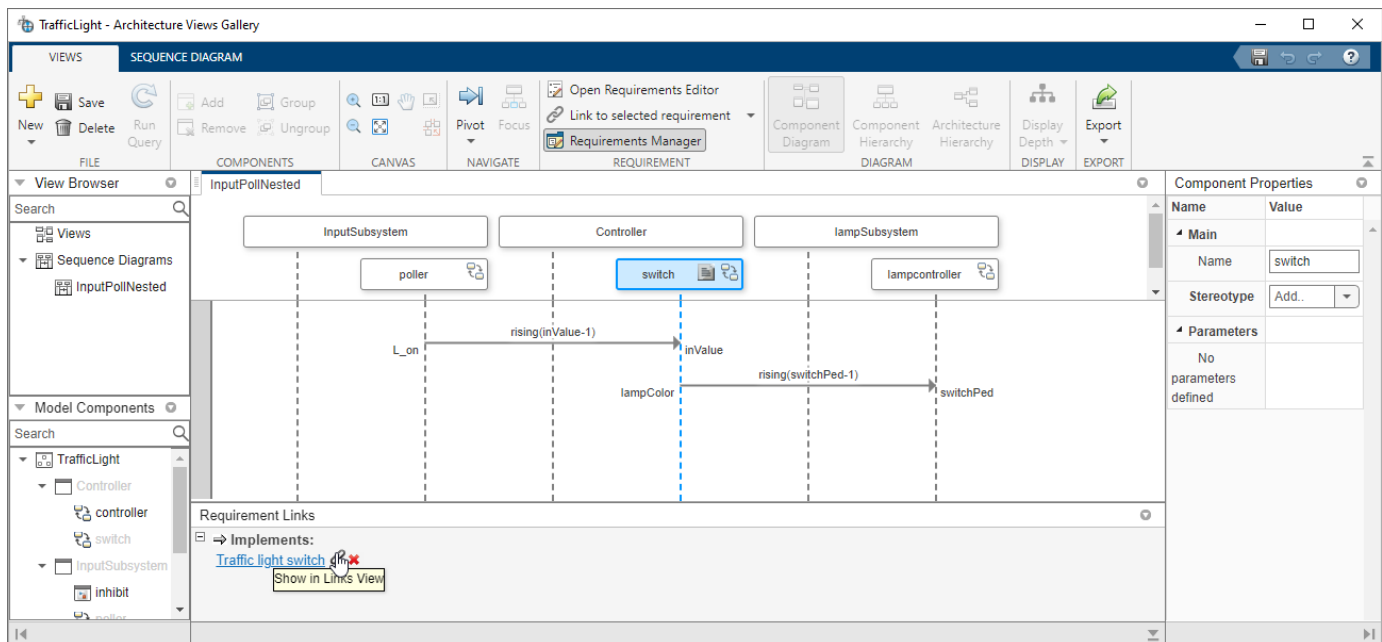
Implementation links contribute to the requirements implementation status. For more information, see “Review Requirements Implementation Status” on page 4-2.

Note Verification links from sequence diagrams do not contribute to requirements verification status. For a list of valid requirement verification items, see “Review Requirements Verification Status” on page 4-6.

View Links

To view the links from the **Architecture Views Gallery**:

- 1 In the **Views** tab, in the **Requirement** section, click **Requirements Manager**. Linked model elements have the Requirements icon .
- 2 Select a model element that has the Requirements icon . The **Requirement Links** pane at the bottom of the screen displays the summary of the linked requirement and the link type.
- 3 Click the hyperlink in the **Requirement Links** pane to view the linked requirement in the **Requirements Editor**.



See Also

Apps

Requirements Editor | Architecture Views Gallery

More About

- “Create and Store Links” on page 3-31
- “Describe System Behavior Using Sequence Diagrams” (System Composer)
- “Use Sequence Diagrams with Architecture Models” (System Composer)
- “Review Requirements Implementation Status” on page 4-2

Use Command-Line API to Update or Repair Requirements Links

This example covers a set of standard situations when links between design artifacts and requirements become stale after one or more artifacts are moved or renamed. Rather than deleting broken links and creating new ones, we want to update existing links so that creation/modification history and other properties (description, keywords, comments,..) are preserved. Use of the following APIs is demonstrated:

- `slreq.find` to get hold of Requirements Toolbox™ entries and links
- `find` to locate the wanted entry in a given ReqSet
- `getLinks` to query all outgoing Links in LinkSet
- `source` brief information about link source
- `destination` brief information about link destination
- `slreq.Link` for "as stored" target info, which is different from "as resolved" `Link.destination()`
- `slreq.LinkSet` to update link destinations when target document moved
- `slreq.ReqSet` to update previously imported set when source document moved
- `updateFromDocument` to update previously imported References from updated document
- `slreq.LinkSet` to convert existing "direct links" to "reference links"
- `slreq.show` used to view either the source or the destination end of a given `slreq.Link`

In a few places we also use the legacy `rmi` APIs that are inherited from Requirements Management Interface (RMI) part of the retired SLVnV Product.

Example Project Files

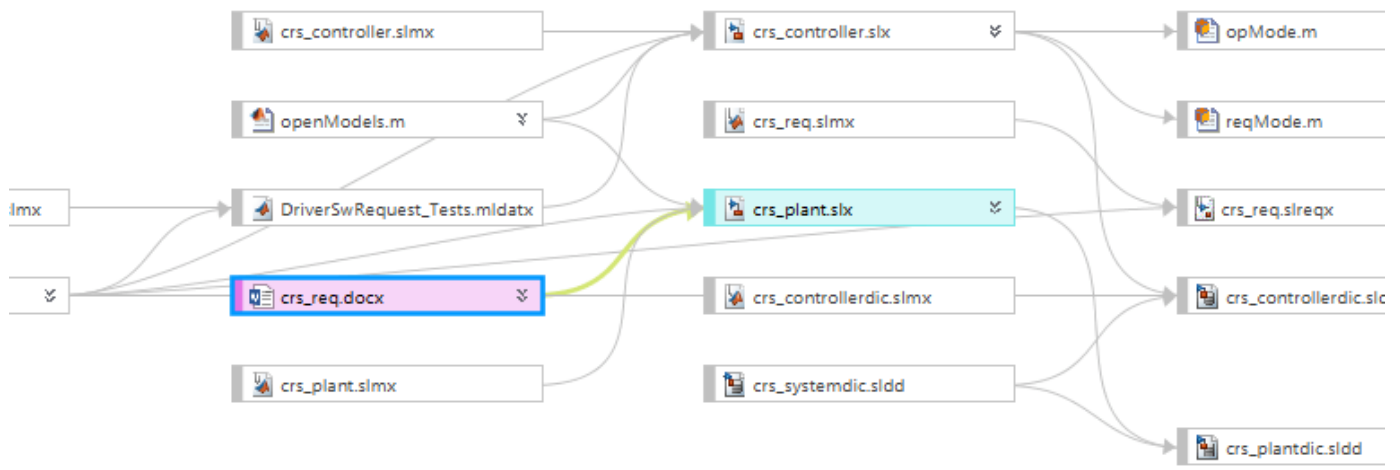
Before you begin, ensure a clean initial state by running `slreq.clear` command. Then type `slreqCCProjectStart` to open the `CruiseRequirementsExample` project. This will unzip a collection of linked artifact files into a new subfolder under your `MATLAB/Projects` folder.

```
slreq.clear();
slreqCCProjectStart();
```

Simulink Model Linked to Requirements

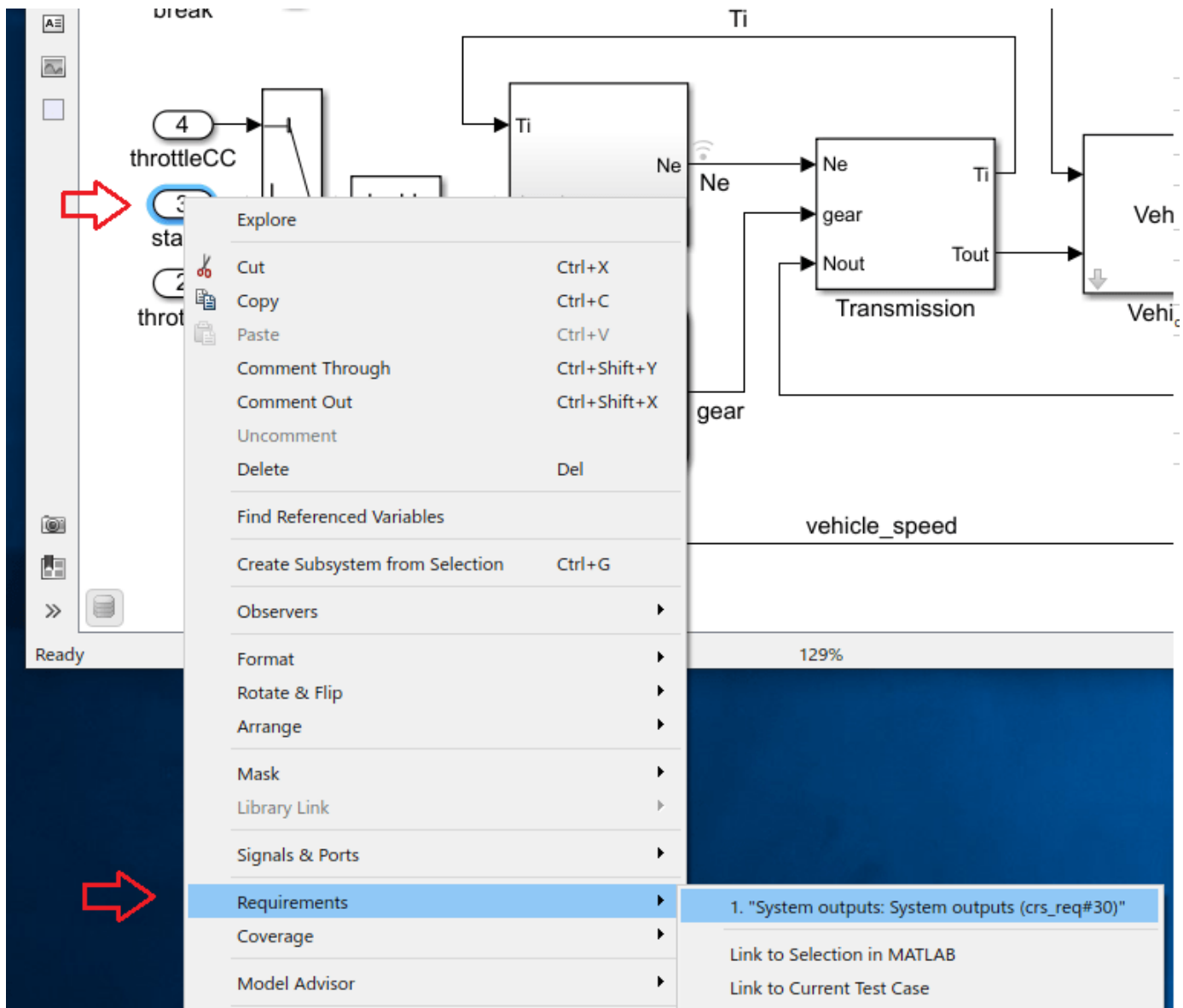
We will focus on a small part of this Project's Dependency Graph: open the `crs_plant.slx` Simulink® model, that has several links to an external Microsoft® Word document `crs_reqs.docx`.

```
open_system('crs_plant');
```

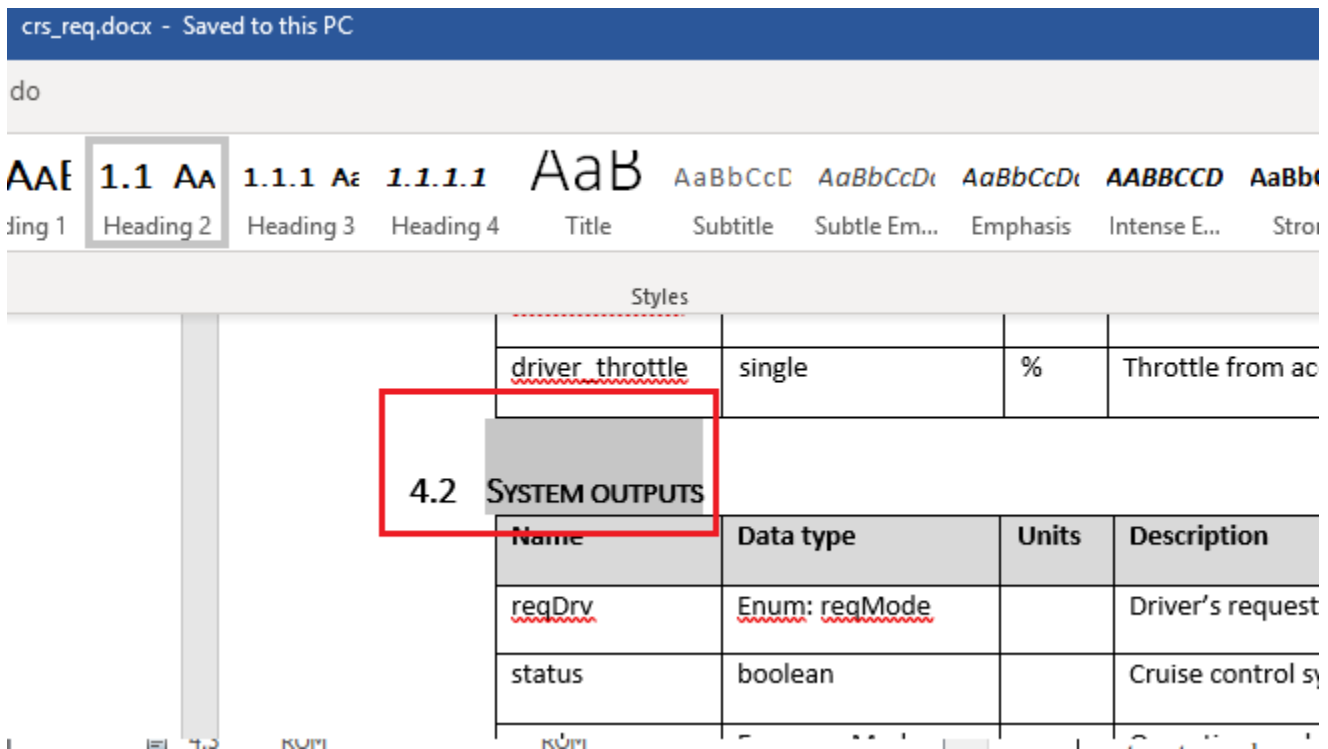


Navigate one of the links to open the linked document.

```
rmi('view', 'crs_plant/status', 1);
```

Word document opens to the corresponding section:



Here is how to use command-line APIs and check for links from `crs_plant.slx` to `crs_reqs.docx`.

```
linkSet = slreq.find('type', 'LinkSet', 'Name', 'crs_plant');
links = linkSet.getLinks();
disp('Original Links to Word document:');

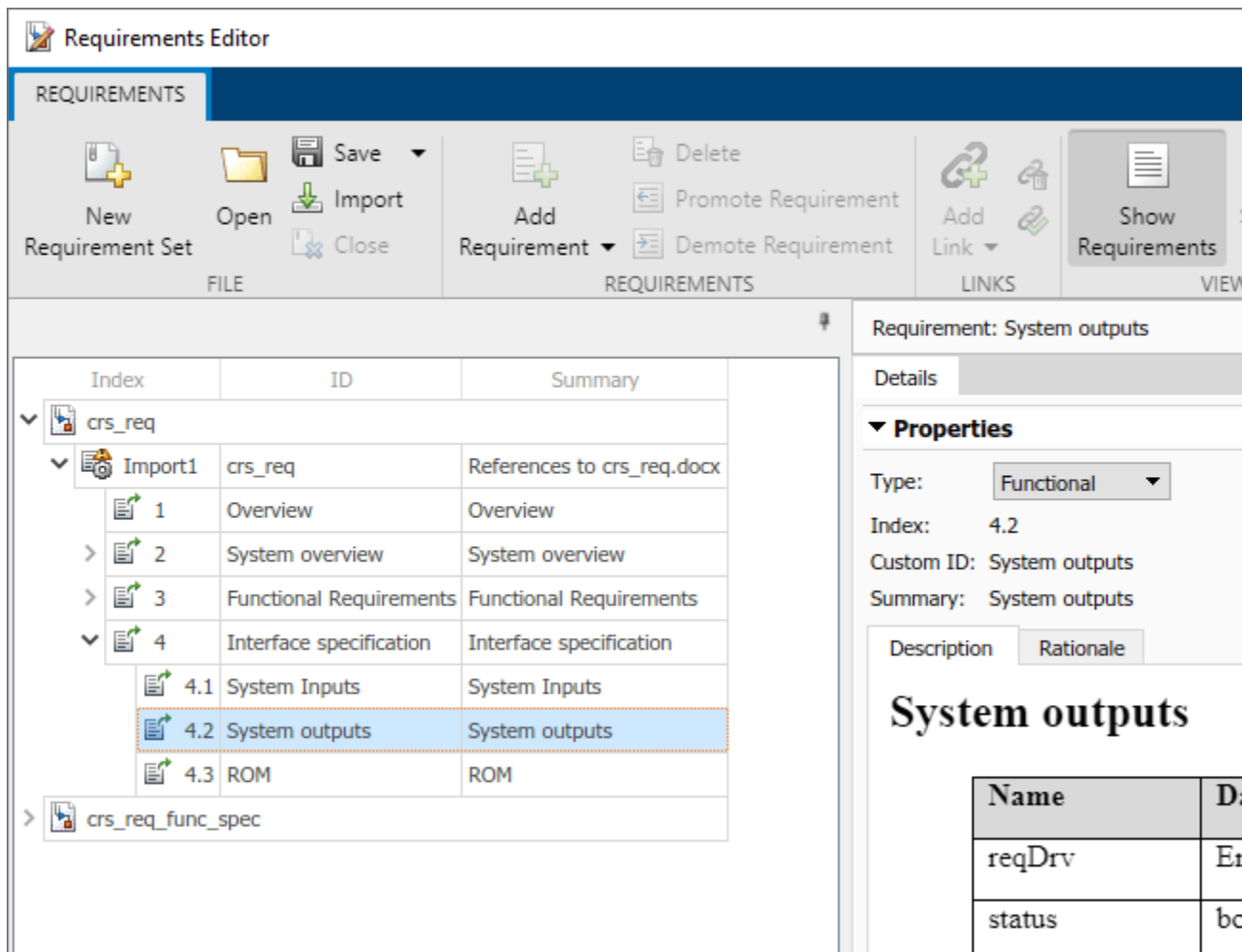
Original Links to Word document:

for i = 1:numel(links)
    linkTarget = links(i).getReferenceInfo();
    if contains(linkTarget.artifact, 'crs_req.docx')
        source = links(i).source;
        disp(['    found link from ' strep(getfullname([broot source.id]),newline,'') ...
            ' to crs_req.docx']);
    end
end

    found link from crs_plant/Vehicle1/vehiclespeed to crs_req.docx
    found link from crs_plant/throttDrv to crs_req.docx
    found link from crs_plant/status to crs_req.docx
    found link from crs_plant/throttleCC to crs_req.docx
```

Navigation of Direct Links in the Presence of Imported References

Open the **Requirements Editor** by entering `slreq.editor` at the MATLAB® command line. You will see two Requirement Sets loaded: `crs_req.slreqx` and `crs_req_func_spec.slreqx`. The first Requirement Set is a collection of references imported from `crs_req.docx`, and the second was manually created in the **Requirements Editor**. If you now close the Word document and navigate the same link from `crs_plant/status` Inport block, the corresponding imported reference is highlighted in the **Requirements Editor**, because navigation action finds the matching reference in a loaded imported Requirement Set.



You can still use the **Show in document** button to see the linked Requirement in the context of original document.

```
slreq.editor();
rmdotnet.MSWord.application('kill');
rmi('view', 'crs_plant/status', 1);
```

Use Case 1: Batch-Update Links After Document Renamed

Suppose that an updated version of the requirements document is received, named `crs_req_v2.docx`. We now want the links in `crs_plant.slx` to target the corresponding sections of the updated document. For the purpose of this example, we will make a copy of the original document in same folder with a modified name. We then use `slreq.LinkSet` API to batch-update all links in a given `LinkSet` to connect with the newer copy of the document:

```
copyfile(fullfile(pwd, 'documents/crs_req.docx'), fullfile(pwd, 'documents/crs_req_v2.docx'));
linkSet.updateDocUri('crs_req.docx', 'crs_req_v2.docx');
```

Verify the Update of Matching Links

Now we can navigate the same link and confirm that the appropriate version of the external document opens. If we iterate all links as before, this confirms that all 4 links updated as intended:

```

rmi('view', 'crs_plant/status', 1); % updated document opens
links = linkSet.getLinks();
disp('Links to Word document after update:');

Links to Word document after update:

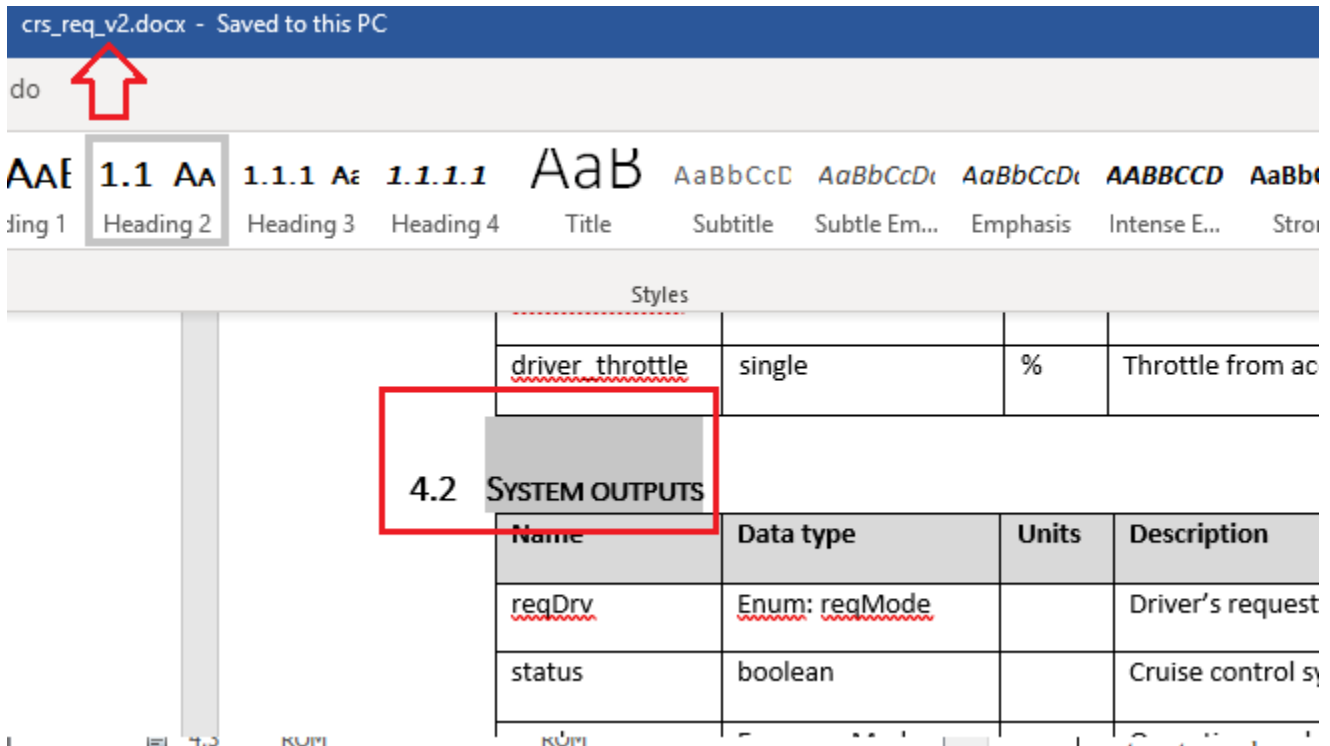
for i = 1:numel(links)
    source = links(i).source;
    linkTarget = links(i).getReferenceInfo();
    if contains(linkTarget.artifact, 'crs_req.docx')
        warning(['link from ' source.id ' still points to crs_req.docx']); % should not happen
    elseif contains(linkTarget.artifact, 'crs_req_v2.docx')
        disp(['    found link from ' strrep(getfullname([bdroot source.id]),newline,' ')...
            ' to crs_req_v2.docx']);
    end
end

found link from crs_plant/Vehicle1/vehicle speed to crs_req_v2.docx
found link from crs_plant/throttDrv to crs_req_v2.docx
found link from crs_plant/status to crs_req_v2.docx
found link from crs_plant/throttleCC to crs_req_v2.docx

```

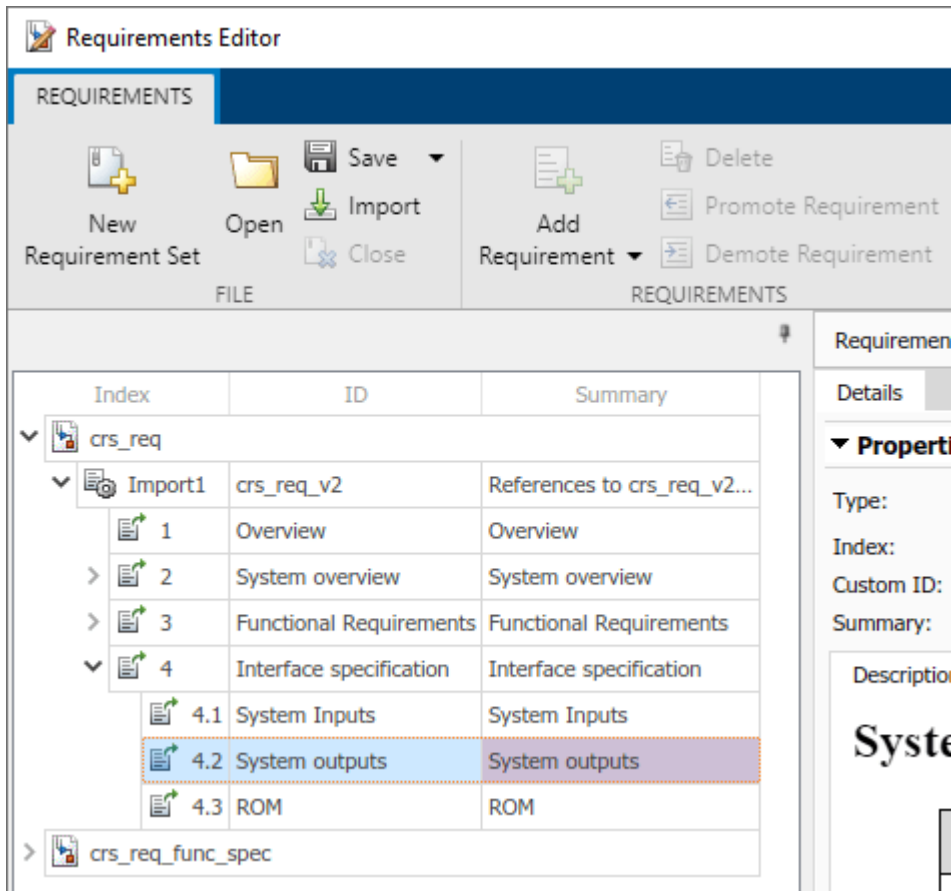
Navigate to Imported References After Updating Links

As demonstrated above, when imported references are available in the **Requirements Editor**, navigating a link will select the matching reference object. However, we have just updated links for a new version of the document `crs_req_v2.docx`, and there are no imported references for this document. Navigation from Simulink blocks in the presence of the **Requirements Editor** brings you directly to the external Word document.



To avoid this inconsistency we need to update the previously imported references for association with the updated document name. We use the `slreq.ReqSet` API to accomplish this task. Additionally,

because the updated document may have modified Requirements, we must use `updateFromDocument` API to pull-in the updates for reference items stored on Requirements Toolbox side. After this is done, navigating from Simulink model will locate the matching imported reference.



Find the Requirement Set with imported references. Update the source file location and find the top-level Import node.

```
reqSet = slreq.find('type', 'ReqSet', 'Name', 'crs_req');
reqSet.updateSrcFileLocation('crs_req.docx', 'crs_req_v2.docx');
importNode = reqSet.find('CustomId', 'crs_req_v2');
```

Update the imported references from the source file. Close Microsoft Word. Then, navigate to the updated reference in the **Requirements Editor**.

```
importNode.updateFromDocument();
rmidotnet.MSWord.application('kill');
rmi('view', 'crs_plant/status', 1);
```

Cleanup After Use Case 1

Discard link data changes to avoid prompts on Project close. Close the project (also cleans-up MATLAB path changes). Close Microsoft Word.

```
slreq.clear();
prj = simulinkproject(); prj.close();
rmdir(net.MSWord.application('kill'));
```

Use Case 2: Batch-Update Links to Fully Rely on Imported References

As demonstrated in Use Case 1 above, additional efforts are required to maintain "direct links" to external documents when documents are moved or renamed. A better workflow is to convert the existing "direct links" into "reference links", which are links that point to the imported References in *.slreqx files and no longer duplicate information about the location or name of the original document. When using this option, the external source document association is stored only in the Requirement Set that hosts the imported References.

To demonstrate this workflow, restart from the same initial point by reopening the CruiseRequirementsExample project in a new subfolder. Copy the Simulink model to your directory and open it.

```
slreqCCProjectStart();
copyfile(fullfile(pwd, 'documents/crs_req.docx'), fullfile(pwd, 'documents/crs_req_v2.docx'));
open_system('crs_plant');
```

Find the crs_plant link set and crs_req requirement set with imported References.

```
linkSet = slreq.find('type', 'LinkSet', 'Name', 'crs_plant');
reqSet = slreq.find('type', 'ReqSet', 'Name', 'crs_req');
```

We then use slreq.LinkSet API to update all the direct links in crs_plant.slmx. Then create an array of all the links in the link set.

```
linkSet.redirectLinksToImportedReqs(reqSet);
links = linkSet.getLinks();
```

After updating the LinkSet in this way, loop over all the links to confirm the absence of "direct" links to crs_req.docx file.

```
disp('Check for links to original external document:');
```

Check for links to original external document:

```
counter = 0;
for i = 1:numel(links)
    linkTarget = links(i).getReferenceInfo();
    if contains(linkTarget.artifact, 'crs_req.docx')
        source = links(i).source;
        warning(['link from ' source.id ' still points to crs_req.docx']);
        counter = counter + 1;
    end
end
disp(['    Total ' num2str(counter) ' links to external document']);
```

```
    Total 0 links to external document
```

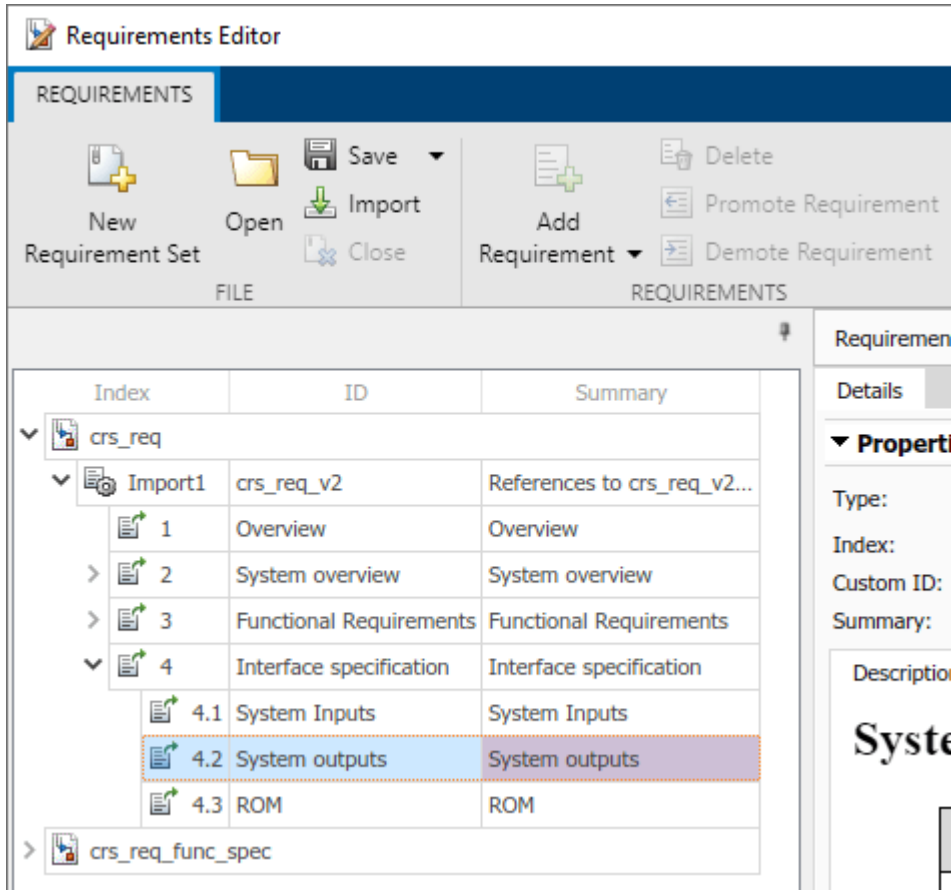
Navigate from the Simulink model to the updated reference.

```
rmi('view', 'crs_plant/status', 1);
```

Links to References and External Document Rename

Now, when all links point to imported References and not to the external document, traceability data remains consistent after document rename, as long as the Import node is updated for the new

external document name. As in the Use Case 1, we will pretend there is an updated version of the external requirements document, by resaving our Word document with a new name. We then perform the required update for the Import node by using the same APIs as before. Now, because the links rely on imported References, and do not store information about imported document, navigation from Simulink model brings us to the updated reference, same as after performing all the steps of Use Case 1.



The Reference is now associated with the updated external document, [Show in document] button opens the updated (renamed) document, and no further adjustment on the LinkSet side is required.

Find the Requirement Set with imported references. Update the source file location and find the top-level Import node.

```
reqSet = slreq.find('type', 'ReqSet', 'Name', 'crs_req');
reqSet.updateSrcFileLocation('crs_req.docx', 'crs_req_v2.docx');
importNode = reqSet.find('CustomId', 'crs_req_v2');
```

Update the imported references from the source file. Then, navigate to the updated reference in the **Requirements Editor**.

```
importNode.updateFromDocument();
rmi('view', 'crs_plant/status', 1);
```

Cleanup After Use Case 2

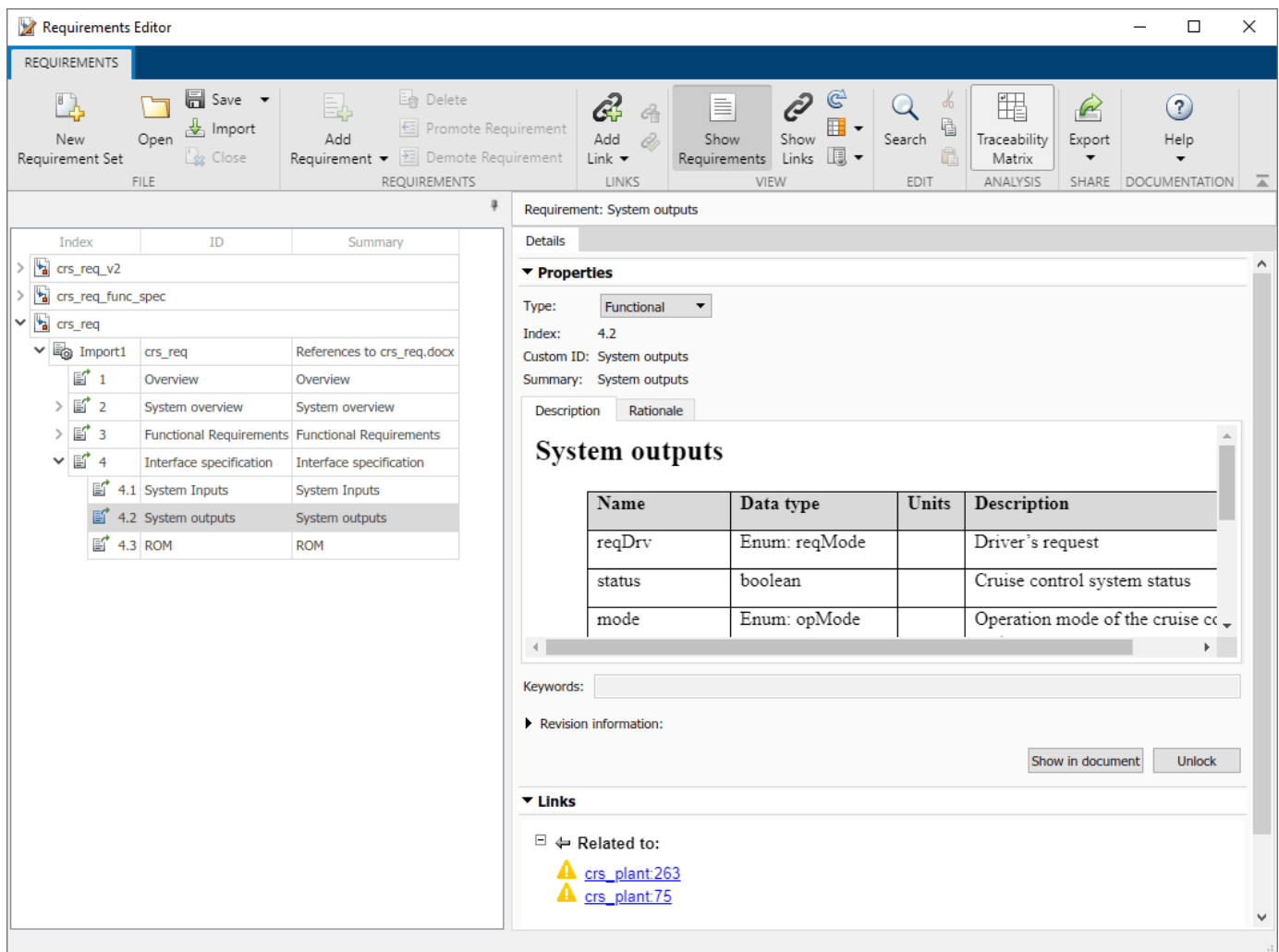
Discard link data changes to avoid prompts on Project close. Close the project (also cleans-up MATLAB path changes). Close Microsoft Word.

```
slreq.clear();
prj = simulinkproject(); prj.close();
rmidotnet.MSWord.application('kill');
```

Use Case 3: Moving Linked Artifacts to a New Project

Now suppose that we are branching an existing project with linked artifacts, and we need to create a new set of renamed artifacts with all the traceability links as in the original Project. As before, we will extract the CruiseRequirementsExample project into a new subfolder, and convert the "direct links" to "reference links", as we have done in Use Case 2 above. We then go ahead and create "new versions" of the linked artifacts by resaving each one with the **_v2.** name.

After creating renamed copies of Simulink model, the imported external document, and the Requirement Set with the imported Requirements, there is one problem: renamed model is linked to the references in the original Requirement set, not in the renamed Requirement set. In the **Details** pane, under **Links**, the links appear unresolved because the original model is not loaded.



Open the `CruiseRequirementsExample` project and open the `crs_plant` model. Find the link set for `crs_plant` and the requirement set `crs_req`.

```
slreqCCProjectStart();
open_system('models/crs_plant.slx');
linkSet = slreq.find('type', 'LinkSet', 'Name', 'crs_plant');
reqSet = slreq.find('type', 'ReqSet', 'Name', 'crs_req');
```

Convert the direct links to reference links. Create renamed copies of the files and save them.

```
linkSet.redirectLinksToImportedReqs(reqSet);
mkdir(fullfile(pwd, 'copied'));
save_system('crs_plant', fullfile(pwd, 'copied/crs_plant_v2.slx'));
reqSet.save(fullfile(pwd, 'copied/crs_req_v2.slreqx'));
copyfile('documents/crs_req.docx', 'copied/crs_req_v2.docx');
```

Associate the renamed requirement set with the renamed document. Find the top level Import node.

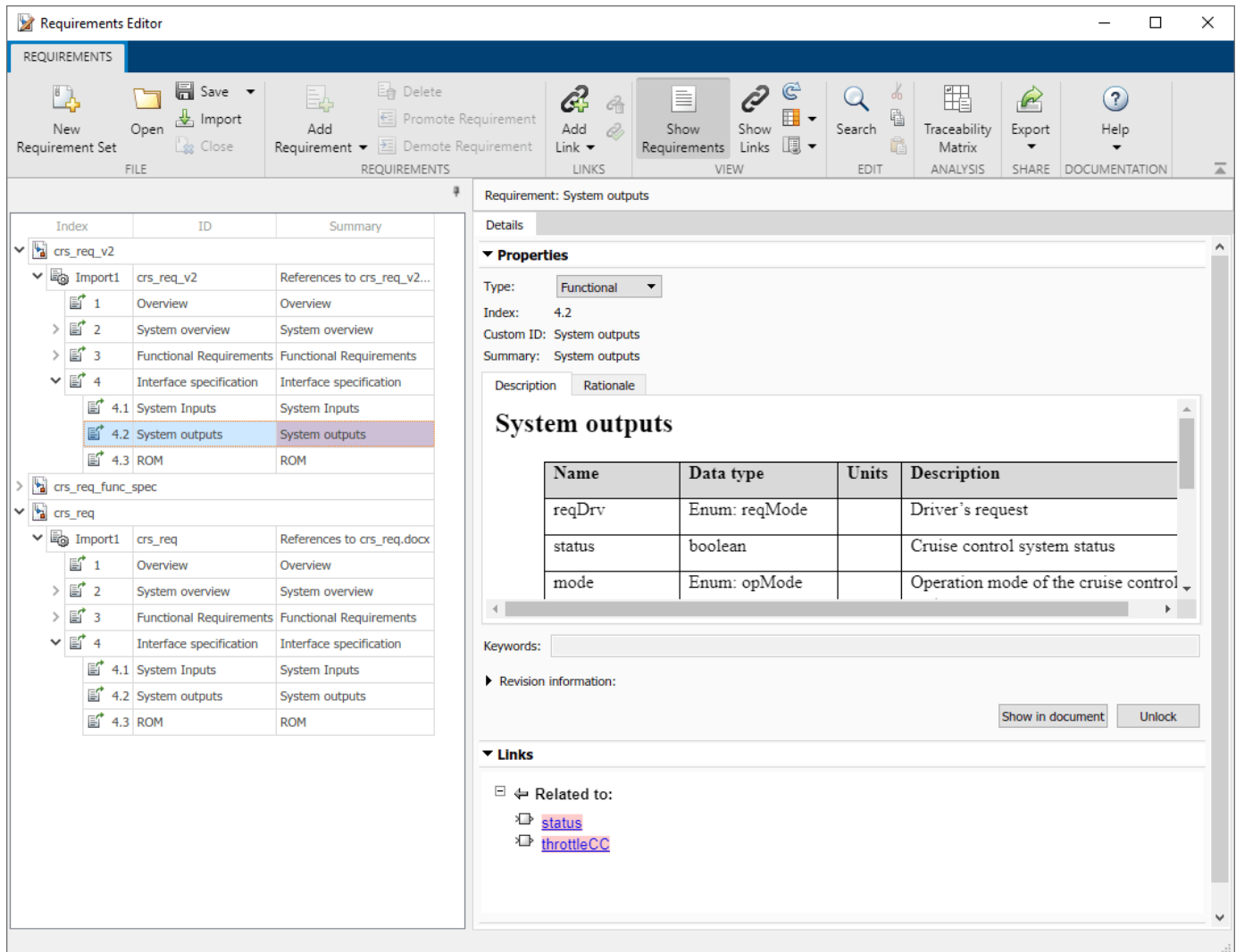
```
reqSet.updateSrcFileLocation('crs_req.docx', fullfile(pwd, 'copied/crs_req_v2.docx'));
importNode = reqSet.find('CustomId', 'crs_req_v2');
```

Ensure the contents in the renamed requirement set match the contents of the renamed document by updating the imported References. Navigate from the renamed Simulink model to the item in the renamed requirement set. The old item in the original requirement set is highlighted, which is incorrect.

```
importNode.updateFromDocument();
rmi('view', 'crs_plant_v2/status', 1);
```

Update Links in Renamed Source to Use the Renamed Destination as the Target

Similarly to Use Case 1, we can use `LinkSet.updateDocUri(OLD, NEW)` API to update links in `crs_plant_v2.slmx` to use the renamed Requirement Set `crs_req_v2.slreqx` as the link target, instead of the original `crs_req.slreqx`. Once this is done, navigate again from the block in the renamed model. The requirement in the renamed Requirement Set is selected, and the links in the **Details** pane under **Links** at bottom-right are resolved.



Find the link set for the new copy of the model, `crs_plant_v2`. Update the name of the requirement set linked with the new copy of the model. Navigate from the renamed Simulink model to the item in the renamed requirement set. This time, it highlights the correct item.

```
linkSet = slreq.find('type', 'LinkSet', 'Name', 'crs_plant_v2');
linkSet.updateDocUri('crs_req.slreqx', 'crs_req_v2.slreqx');

rmi('view', 'crs_plant_v2/status', 1);
```

Manage Custom Attributes for Links by Using the Requirements Toolbox API

This example shows how to use the Requirements Toolbox™ API to create and manage custom attributes for link sets and set custom attribute values for links.

Establish Link Set

Load the `crs_req` requirement file, which describes a cruise control system. Find the link set named `crs_req` and assign it to a variable.

```
slreq.load('crs_req');
ls = slreq.find('Type','LinkSet','Name','crs_req')

ls =
    LinkSet with properties:
        Description: 'crs_req'
        Filename: 'C:\TEMP\Bdoc23a_2213998_3568\ib570499\8\tp24dd4731\slrequirements-ex2
        Artifact: 'C:\TEMP\Bdoc23a_2213998_3568\ib570499\8\tp24dd4731\slrequirements-ex2
        Domain: 'linktype_rmi_slreq'
        Revision: 11
        Dirty: 0
        CustomAttributeNames: {'Target Speed Change'}
```

Delete a Custom Attribute

There is an existing custom attribute in the link set called `Target Speed Change`. Delete the custom attribute and confirm the results by checking the existing custom attribute names for the link set.

```
deleteAttribute(ls,'Target Speed Change','Force',true);
ls.CustomAttributeNames

ans =

    0x0 empty cell array
```

Add a Custom Attribute of Each Type

Add a custom attribute of each type to the link set. Create an `Edit` custom attribute with a description.

```
addAttribute(ls,'MyEditAttribute','Edit','Description',['You can enter text as' ...
    ' the custom attribute value.'])
```

Create a `Checkbox` type attribute and set its `DefaultValue` property to `true`.

```
addAttribute(ls,'MyCheckboxAttribute','Checkbox','DefaultValue',true)
```

Create a `Combobox` custom attribute. Because the first option must be `Unset`, add the options `'Unset'`, `'A'`, `'B'`, and `'C'`.

```
addAttribute(ls,'MyComboboxAttribute','Combobox','List',{'Unset','A','B','C'})
```

Create a `DateTime` custom attribute.

```
addAttribute(ls, 'MyDateTimeAttribute', 'DateTime')
```

Check the custom attributes for the link set. Get information about `MyComboboxAttribute` to see the options you added to the `Combobox` attribute.

```
ls.CustomAttributeNames
```

```
ans = 1x4 cell
      {'MyCheckboxAttribute'}    {'MyComboboxAttribute'}    {'MyDateTimeAttribute'}    {'MyEditAtt
```

```
atrb = inspectAttribute(ls, 'MyComboboxAttribute')
```

```
atrb = struct with fields:
      name: 'MyComboboxAttribute'
      type: Combobox
      description: ''
      list: {'Unset' 'A' 'B' 'C'}
```

Set a Custom Attribute Value for a Link

Find a link in the link set and set the custom attribute value for all four custom attributes that you created.

```
lk = find(ls, 'SID', 3);
setAttribute(lk, 'MyEditAttribute', 'Value for edit attribute. ');
setAttribute(lk, 'MyCheckboxAttribute', false);
setAttribute(lk, 'MyComboboxAttribute', 'B');
```

Set `MyDateTimeAttribute` with the desired locale to ensure that the date and time is set in the correct format on systems in other locales. See “Locale” for more information.

```
localDateTimeStr = datestr(datetime('15-Jul-2018 11:00:00', 'Locale', 'en_US'), 'Local');
setAttribute(lk, 'MyDateTimeAttribute', localDateTimeStr);
```

View the attribute values.

```
getAttribute(lk, 'MyEditAttribute')
```

```
ans =
'Value for edit attribute.'
```

```
getAttribute(lk, 'MyCheckboxAttribute')
```

```
ans = logical
      0
```

```
getAttribute(lk, 'MyComboboxAttribute')
```

```
ans =
'B'
```

```
getAttribute(lk, 'MyDateTimeAttribute')
```

```
ans = datetime
    15-Jul-2018 11:00:00
```

Edit Custom Attributes

After you define a custom attribute for a link set, you can make limited changes to the custom attribute.

Add a description to `MyCheckboxAttribute` and `MyComboboxAttribute`, then change the list of options for `MyComboboxAttribute`. Because you cannot update the default value of `Checkbox` attributes, you can only update the description of `MyCheckboxAttribute`. View the changes.

```
updateAttribute(ls,'MyCheckboxAttribute','Description',['The checkbox value can be ' ...
    ' true or false.']);
updateAttribute(ls,'MyComboboxAttribute','Description',['Choose an option from the ' ...
    'list.'],'List',{'Unset','1','2','3'});
atrb2 = inspectAttribute(ls,'MyCheckboxAttribute')

atrb2 = struct with fields:
    name: 'MyCheckboxAttribute'
    type: Checkbox
    description: 'The checkbox value can be true or false.'
    default: 1

atrb3 = inspectAttribute(ls,'MyComboboxAttribute')

atrb3 = struct with fields:
    name: 'MyComboboxAttribute'
    type: Combobox
    description: 'Choose an option from the list.'
    list: {'Unset' '1' '2' '3'}
```

Find Links That Match Custom Attribute Value

Search the link set for all links where `'MyEditAttribute'` is set to `'Value for edit attribute.'`

```
lk2 = find(ls,'MyEditAttribute','Value for edit attribute.')
```

```
lk2 =
    Link with properties:
        Type: 'Derive'
        Description: '#8: Set Switch Detection'
        Keywords: {}
        Rationale: ''
        CreatedOn: 20-May-2017 13:14:40
        CreatedBy: 'itoy'
        ModifiedOn: 04-Mar-2023 01:15:54
        ModifiedBy: 'batserve'
        Revision: 5
        SID: 3
        Comments: [0x0 struct]
```

Search the link set for all links where `MyCheckboxAttribute` is set to `true`.

```
lkArray = find(ls, 'MyCheckboxAttribute', true)
```

```
lkArray=1x11 object
```

```
1x11 Link array with properties:
```

```
Type  
Description  
Keywords  
Rationale  
CreatedOn  
CreatedBy  
ModifiedOn  
ModifiedBy  
Revision  
SID  
Comments
```

Search the link set for all links where MyComboboxAttribute is set to 'Unset'.

```
lkArray2 = find(ls, 'MyComboboxAttribute', 'Unset')
```

```
lkArray2=1x12 object
```

```
1x12 Link array with properties:
```

```
Type  
Description  
Keywords  
Rationale  
CreatedOn  
CreatedBy  
ModifiedOn  
ModifiedBy  
Revision  
SID  
Comments
```

Delete Custom Attributes

You can use `deleteAttribute` to delete attributes. However, because the custom attributes created in this example are assigned to links, you must set `Force` to `true` to delete the attributes. Delete `MyEditAttribute` and confirm the change.

```
deleteAttribute(ls, 'MyEditAttribute', 'Force', true);  
ls.CustomAttributeNames
```

```
ans = 1x3 cell  
    {'MyCheckboxAttribute'}    {'MyComboboxAttribute'}    {'MyDateTimeAttribute'}
```

Add a new custom attribute, but don't set any custom attribute values for links.

```
addAttribute(ls, 'NewEditAttribute', 'Edit');  
ls.CustomAttributeNames
```

```
ans = 1x4 cell  
    {'MyCheckboxAttribute'}    {'MyComboboxAttribute'}    {'MyDateTimeAttribute'}    {'NewEditAttribute'}
```

Because `NewEditAttribute` is not used by any links, you can delete it with `deleteAttribute` by setting `Force` to `false`. Confirm the change.

```
deleteAttribute(ls, 'NewEditAttribute', 'Force', false);  
ls.CustomAttributeNames
```

```
ans = 1x3 cell  
    {'MyCheckboxAttribute'}    {'MyComboboxAttribute'}    {'MyDateTimeAttribute'}
```

Cleanup

Clear the open requirement sets and link sets, and close the open models without saving changes.

```
slreq.clear;  
bdclose all;
```

See Also

`slreq.LinkSet` | `addAttribute` | `updateAttribute` | `inspectAttribute` | `deleteAttribute` | `getAttribute` | `setAttribute`

Related Examples

- “Manage Custom Attributes for Requirements by Using the Requirements Toolbox API” on page 1-105

More About

- “Add Custom Attributes to Links” on page 3-44

Make Requirements Fully Traceable with a Traceability Matrix

This example shows how to find requirements that are not traceable to Model-Based Design items, and how to trace those requirements by creating links with a traceability matrix.

A traceability matrix displays links between items in Model-Based Design artifacts such as Requirements Toolbox™ objects, Simulink® model elements, Simulink Test™ objects, and MATLAB® code lines. You can apply filters and focus only on the items that you want to see. You can use the matrix to identify unlinked items and implement them in your design.

To read more about how to use the traceability matrix, see “Track Requirement Links with a Traceability Matrix” on page 3-5.

Open the Requirements Definition for a Cruise Control Model project. Load the `crs_req_func_spec` requirement set.

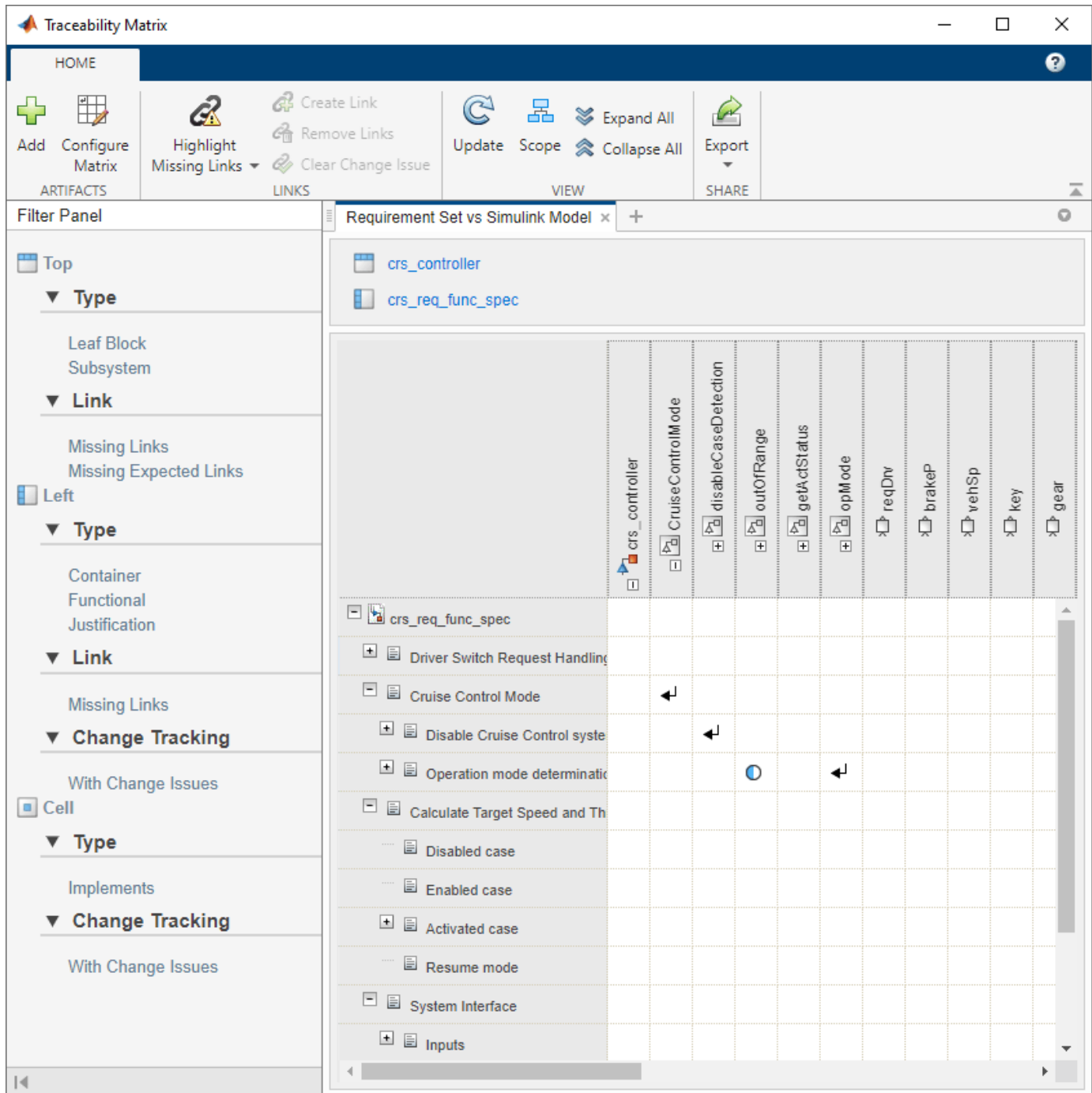
```
slreqCCProjectStart;  
slreq.load('crs_req_func_spec');
```

Generate a Traceability Matrix

Open the Traceability Matrix window.

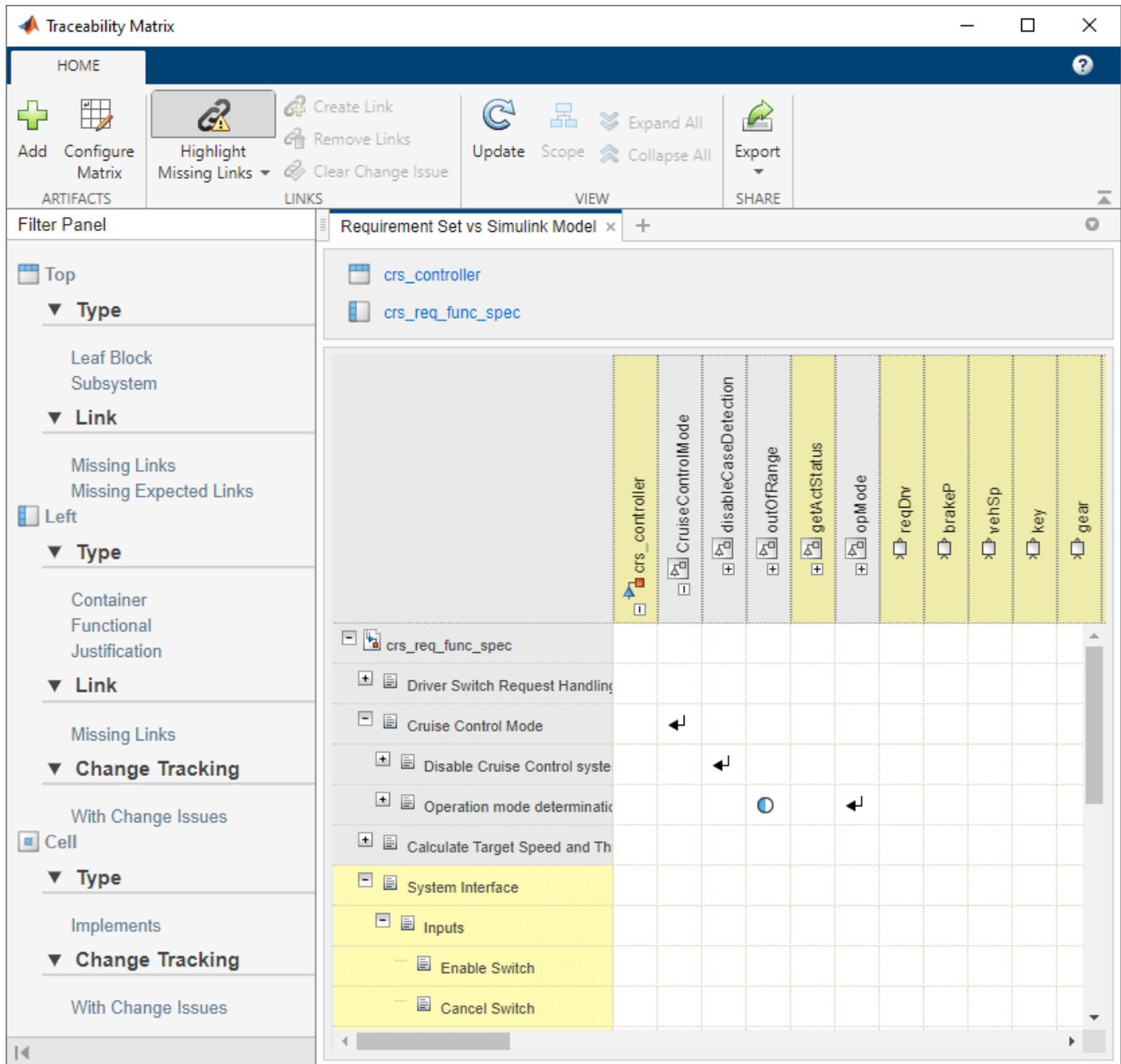
```
slreq.generateTraceabilityMatrix;
```

In the Traceability Matrix window, click **Add**. In the Select Artifacts dialog, set **Left** to `crs_req_func_spec.slreqx` and set **Top** to `crs_controller.slx`. Then click **Generate Matrix**. A traceability matrix is generated with the specified requirement set on the left and the Simulink model on the top.



Identify Unlinked Requirements


To identify unlinked items, click **Highlight Missing Links**. Unlinked requirements are highlighted in yellow in the left column and unlinked model elements are highlighted in the top row.

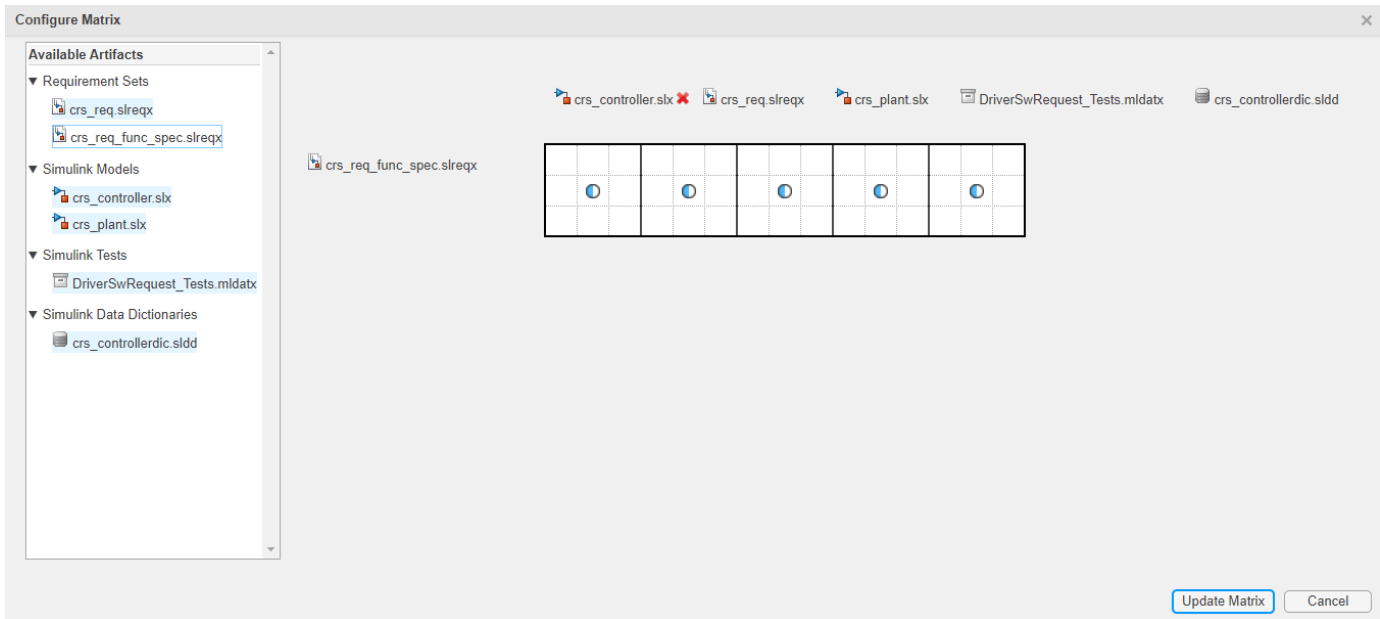


Scroll to the System Interface > Inputs parent requirement. Click **Scope** to focus the matrix view on that hierarchy. The child requirements under Inputs do not have links to the blocks in the Simulink model. However, the traceability matrix that you created only shows links between the crs_req_func_spec requirement set and the crs_controller model. The crs_req_func_spec requirement set may have more links to other artifacts within your project.

Generate a Traceability Matrix with Multiple Artifacts

To view links between multiple artifacts at the same time, you can create a multi-artifact matrix. Click **Configure Matrix** to add more artifacts to your matrix. In the Configure Matrix dialog box, in the

Available Artifacts pane, select `crs_req_func_spec.slreqx`. The artifacts that have links between the selected artifact are highlighted in the **Available Artifacts** pane. In this case, each artifact contains links between the `crs_req_func_spec` requirement set, except for `crs_req_func_spec.slreqx` itself. Drag all of the highlighted artifacts to the top artifact list. The expand icon () in the matrix preview indicates that there are links between items in these artifacts.



Click **Update Matrix** to add the artifacts to your traceability matrix. Starting from the far-left column in the top row, select each artifact and click **Collapse All**. The blue lines in the matrix indicate where one artifact ends and another begins.

The screenshot shows the Traceability Matrix application window. The main area displays a traceability matrix with the following structure:

Requirement	crs_controller	crs_req	crs_plant	DriverSwRequest_Tests	crs_controllerdic.sidd
crs_req_func_spec					
Driver Switch Request Handling					
Switch precedence					
Avoid repeating commands					
Long Switch recognition					
Waiting state for Long Incr					
Waiting state for Long Dec					
Cancel Switch Detection					
Set Switch Detection					
Enable Switch Detection					
Resume Switch Detection					
Increment Switch Detection					
Decrement Switch Detection					
Cruise Control Mode					

The interface includes a Filter Panel on the left with sections for Change Tracking, Link, and Type. The main toolbar contains options like Add, Configure Matrix, Highlight Missing Links, Create Link, Remove Links, Clear Change Issue, Update, Scope, Expand All, Collapse All, and Export.

Select the Inputs parent requirement and click **Scope** to focus on the Inputs child requirements. Click **Highlight Missing Links**. Now you can see that some of the child requirements under Inputs link to items in the crs_plant model.

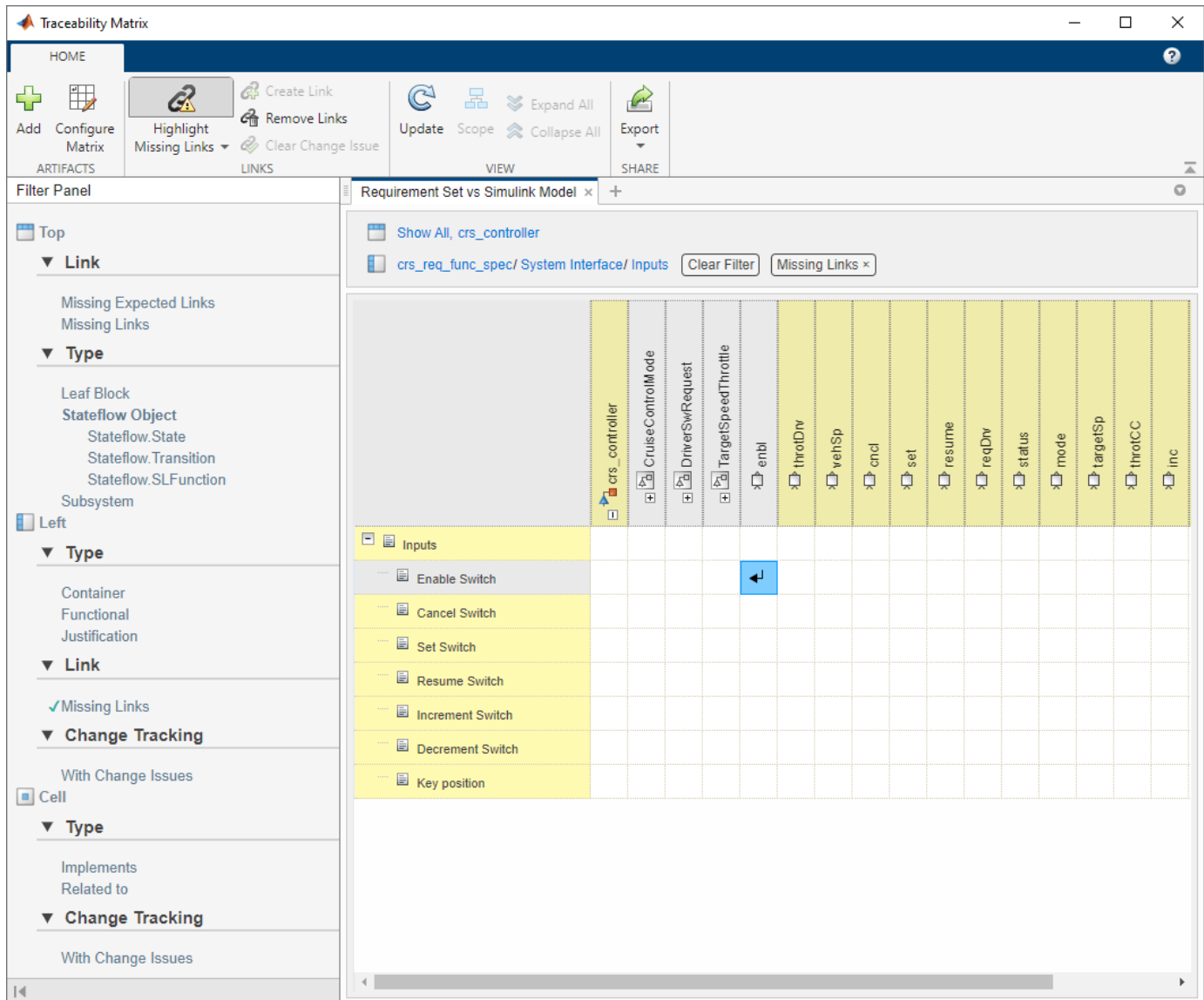
Link Unlinked Inputs to Model Elements

The `crs_controller` and `crs_plant` models contain model elements that are related to the `Inputs` child requirements, however not all of the `Inputs` child requirements are linked. Link all of the `Inputs` child requirements to the model elements for full traceability. First, click **Configure Matrix** and remove all of the artifacts from the Traceability Matrix except for `crs_req_func_spec` on the left, and `crs_controller` and `crs_plant` on the top by right-clicking the artifacts and selecting **Remove Artifacts**. Click **Update Matrix**. In the updated matrix, select the `Inputs` parent requirement and click **Scope** to focus on the `Inputs` child requirements.

Some of the child requirements link to items in `crs_plant`. Link the remaining unlinked `Inputs` child requirements to model elements in `crs_controller`. Select the cell corresponding to `crs_controller` and click **Scope**.

To focus on the unlinked requirements, apply the **Missing Links** filter. In the **Filter Panel**, under **Left**, under **Link**, click **Missing Links**. The filter omits rows with linked items. You can verify this by clicking **Highlight Missing Links**.

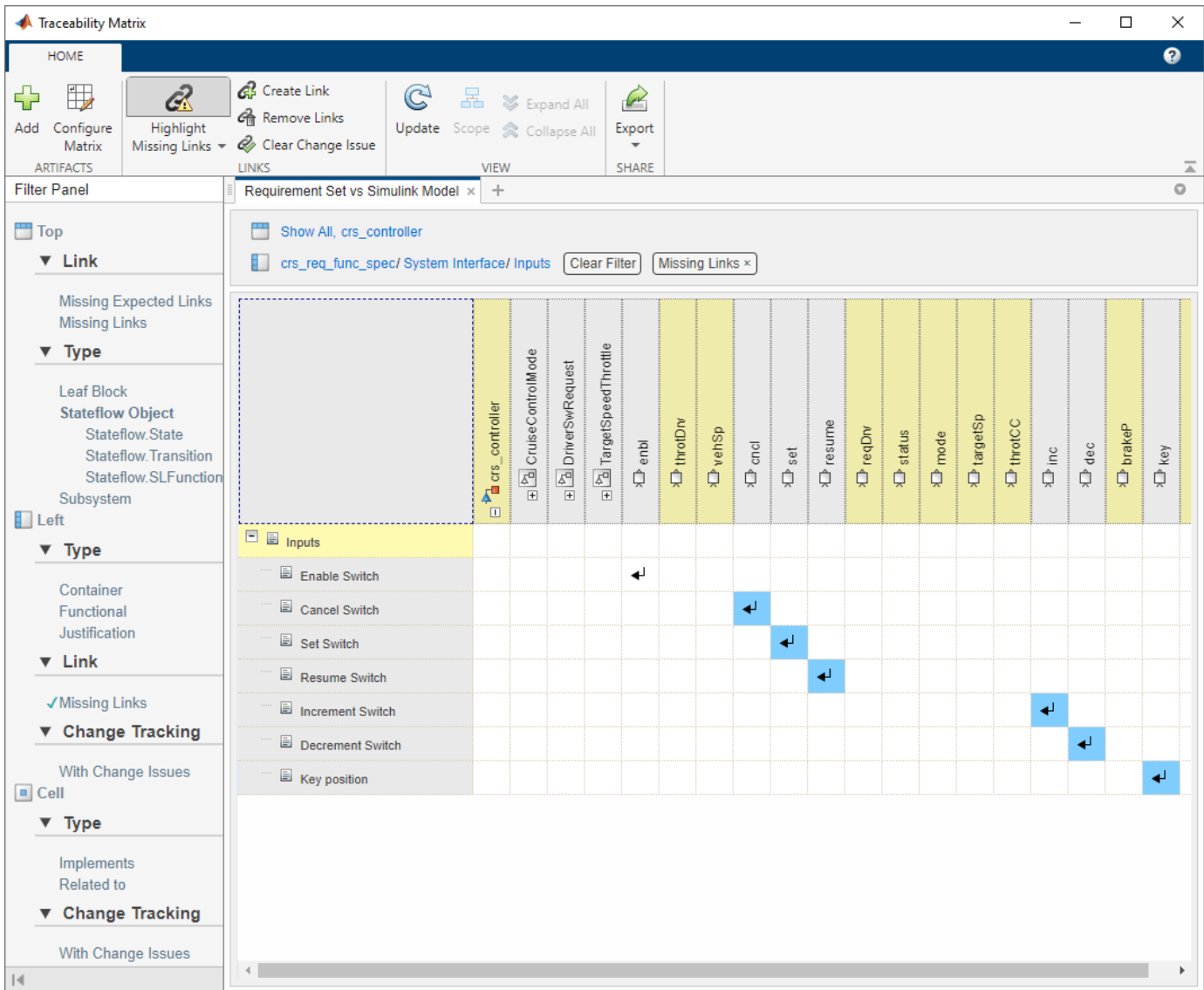
Collapse the `CruiseControlMode`, `DriverSwRequest` and `TargetSpeedThrottle` subsystems by select each subsystem and clicking **Collapse All**. Create a link between the `Enable Switch` requirement and the `enbl` block by selecting the cell corresponding to those two items and clicking **Create**. In the Create Links dialog box, set **Type** to `Implements`, then click **Create** to create a link between the two items.



You can create multiple links at a time when you hold **Ctrl**, select the cells where you want to create links, and click **Create Links**. Create links between the remaining requirements and the corresponding model element:

- The Cancel Switch requirement and the cncl block
- The Set Switch requirement and the set block
- The Resume Switch requirement and the resume block
- The Increment Switch requirement and the inc block
- The Decrement Switch requirement and the dec block
- The Key Position requirement and the key block

In the Create Links dialog, set **Type** to Implements for all of the links.



Clear the **Missing Links** filter by clicking **Clear Filter** in the top artifact list. Click **Show All** to show all of the artifacts. All of the Inputs child requirements link to design items, so they are no longer highlighted. Collapse the hierarchies under `crs_controller` and `crs_plant`. The expand icon (🔍) indicates that all of the Inputs child requirements are linked.

The screenshot displays the Traceability Matrix application window. The interface is divided into several sections:

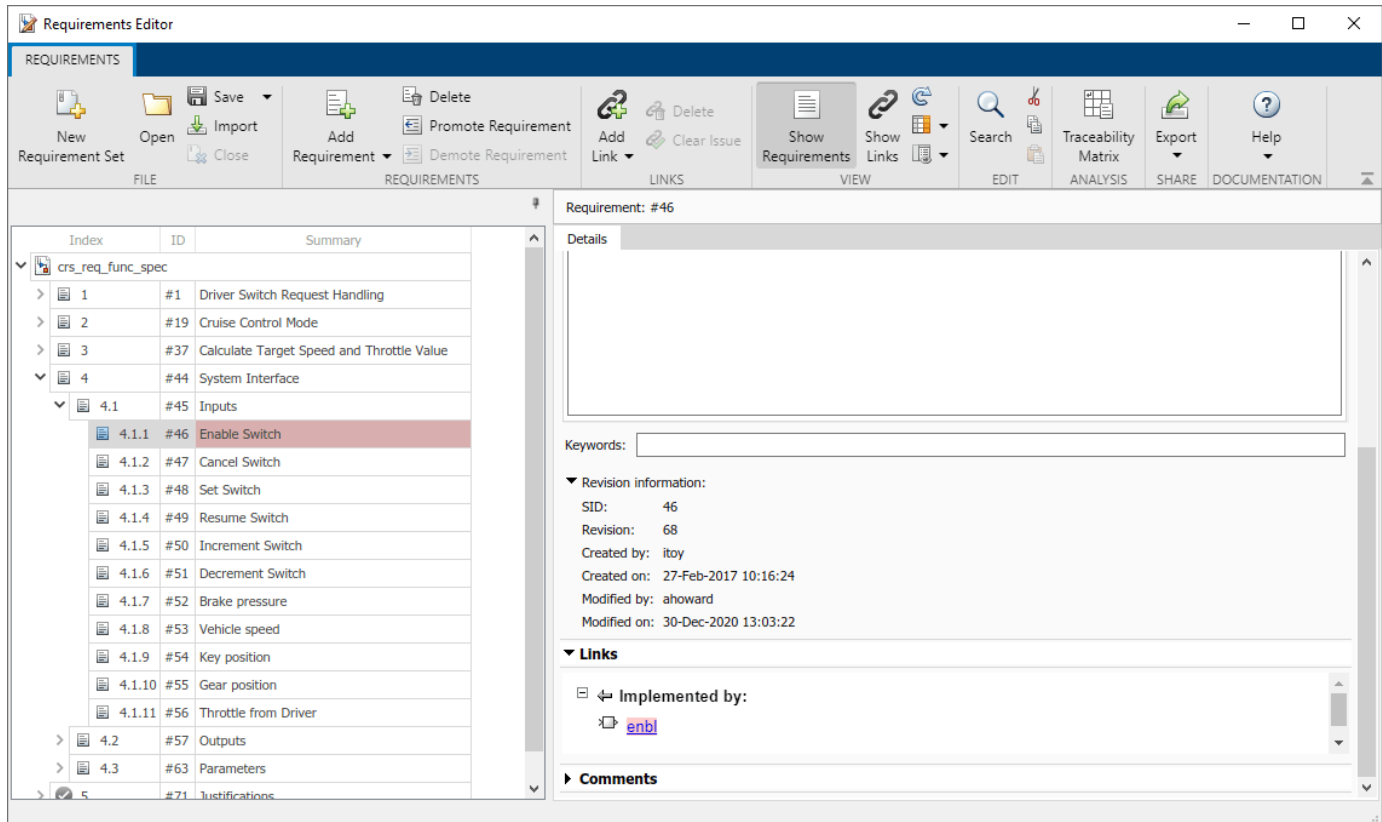
- HOME (Toolbar):** Contains icons for 'Add', 'Configure Matrix', 'Highlight Missing Links', 'Create Link', 'Remove Links', 'Clear Change Issue', 'Update', 'Scope', 'Expand All', 'Collapse All', 'Export', and 'SHARE'.
- Filter Panel (Left):**
 - Top:** 'Link' section with 'Missing Expected Links' and 'Missing Links'.
 - Type:** 'Leaf Block', 'Stateflow Object' (Stateflow.State, Stateflow.Transition, Stateflow.SLFunction), 'Subsystem'.
 - Left:** 'Type' section with 'Container', 'Functional', 'Justification'.
 - Link:** 'Missing Links'.
 - Change Tracking:** 'With Change Issues'.
 - Cell:** 'Type' section with 'Implements', 'Related to'.
 - Change Tracking:** 'With Change Issues'.
- Main View (Requirement Set vs Simulink Model):**
 - Shows a comparison between 'crs_controller, crs_plant' and 'crs_req_func_spec/ System Interface/ Inputs'.
 - A table of inputs is displayed below:

Inputs	crs_controller	crs_plant
Enable Switch	<input type="radio"/>	
Cancel Switch	<input type="radio"/>	
Set Switch	<input type="radio"/>	
Resume Switch	<input type="radio"/>	
Increment Switch	<input type="radio"/>	
Decrement Switch	<input type="radio"/>	
Brake pressure		<input type="radio"/>
Vehicle speed		<input type="radio"/>
Key position	<input type="radio"/>	
Gear position		<input type="radio"/>
Throttle from Driver		<input type="radio"/>

Open Items in Artifact Context

You can open items in rows and columns in their artifact context by double-clicking the cell corresponding to an item. For example, double-clicking a cell corresponding to a Simulink block opens the Simulink model and subsystem that the block is in.

Open the **Enable Switch** requirement in the **Requirements Editor** by double-clicking it. Add additional text to the requirement **Description**: "The **Cruise** button enables the cruise control as long as all other conditions are met." Then click **Save**.



In the **Requirements Editor**, the requirement summary and the associated link (listed in the right pane, under **Links**) are highlighted in red because the link associated with this requirement has a change issue.

View and Clear Change Issues

When you change a requirement that is linked to another item, the requirement is highlighted in red to indicate that there is a change issue associated with the link. The link has a change issue because you changed the description for the **Enable Switch** requirement.

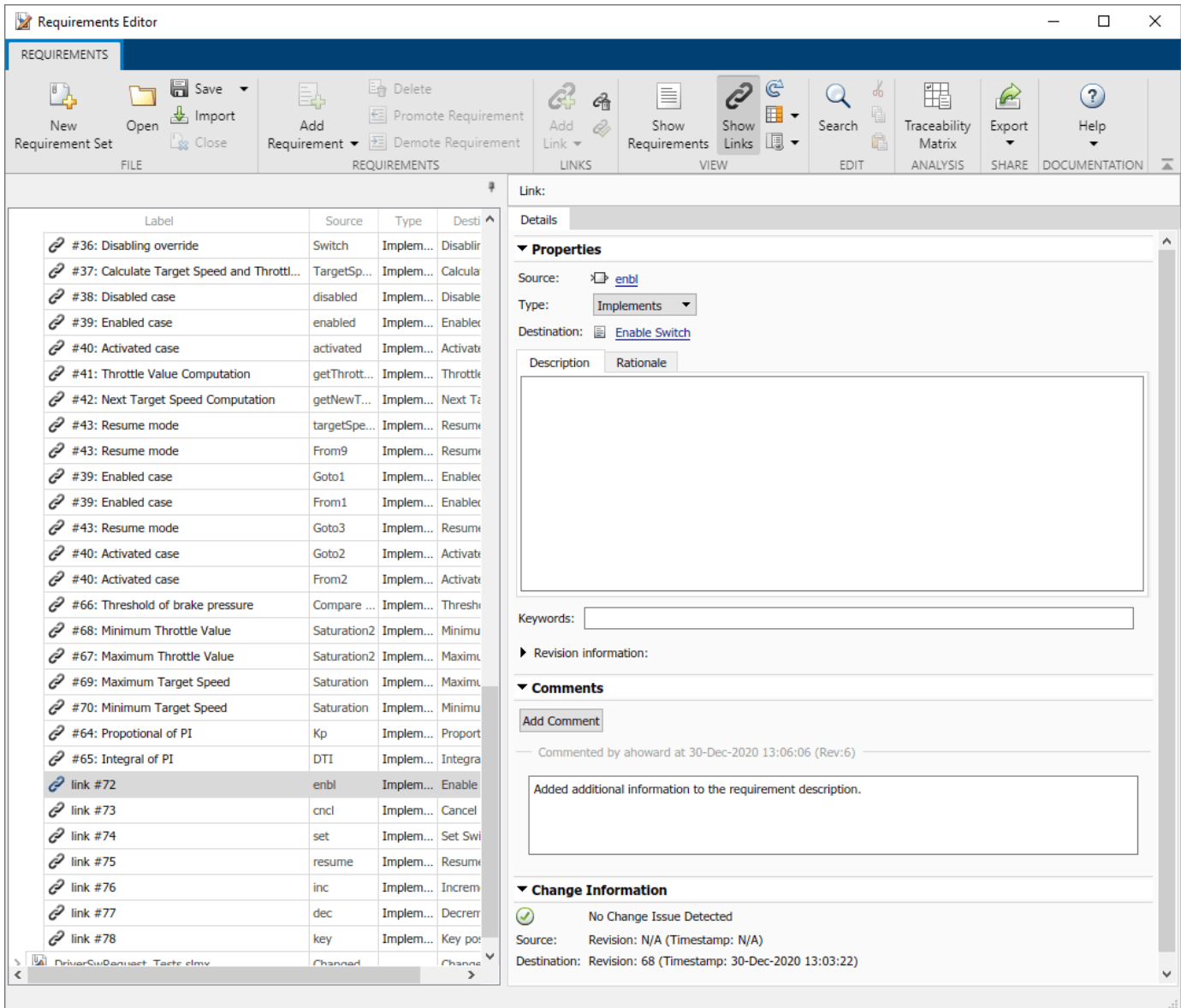
Return to the Traceability Matrix. Click **Update** to refresh the matrix. Select the **Inputs** parent requirement and click **Scope** to focus on the **Inputs** child requirements. Click **Highlight Missing Links > Highlight Changed Links**, then click **Highlight Missing Links > Show Changed Links Only**. The links that have associated change issues are shown, and the requirement, linked item, and link are highlighted in red.

The screenshot displays the Traceability Matrix application window. The title bar reads "Traceability Matrix". The interface is divided into several sections:

- HOME (Toolbar):** Contains icons for "Add", "Configure Matrix", "Highlight Missing Links", "Create Link", "Remove Links", "Clear Change Issue", "Update", "Scope", "Expand All", "Collapse All", and "Export".
- Filter Panel (Left):**
 - Top:** Includes "Link" (Missing Expected Links, Missing Links) and "Type" (Leaf Block, Stateflow Object, Stateflow.State, Stateflow.Transition, Stateflow.SLFunction, Subsystem).
 - Left:** Includes "Type" (Container, Functional, Justification) and "Link" (Missing Links).
 - Change Tracking:** Includes "With Change Issues".
 - Cell:** Includes "Type" (Implements, Related to) and "Change Tracking" (With Change Issues).
- Main View (Requirement Set vs Simulink Model):**
 - Top row: `crs_controller, crs_plant`
 - Second row: `crs_req_func_spec`
 - Diagram area: Shows a grid with a red vertical bar labeled "enbl" and a blue vertical bar labeled "crs_controller". A dashed blue box highlights a portion of the grid.
 - Bottom row: "Enable Switch" with a red background.

Because you changed only the description, the change did not affect the requirement implementation or verification. Clear the change issue by selecting the cell containing the link, then click **Clear Change Issue**. Under **Comment**, enter "Added additional information to the requirement description." Then click **Clear All**.

You can view the comment when you select the link in the **Requirements Editor**, in the right pane, under **Comments**.



Generate a Report from the Traceability Matrix

Update the matrix to reflect the cleared change issues by clicking **Update**. Select Inputs parent requirement and click **Scope**. Expand all links by selecting the cell containing the expand icon (⊕) and clicking **Expand All**. Collapse any hierarchies that don't contain links by clicking **Collapse All**. This view shows the links to the Inputs child requirements. Generate an HTML report that contains

a static snapshot of the current view of the traceability matrix by clicking **Export > Generate HTML Report**. Select a location to save the file and click **Save**.

See Also

`slreq.generateTraceabilityMatrix` | `slreq.getTraceabilityMatrixOptions`

More About

- “Track Requirement Links with a Traceability Matrix” on page 3-5

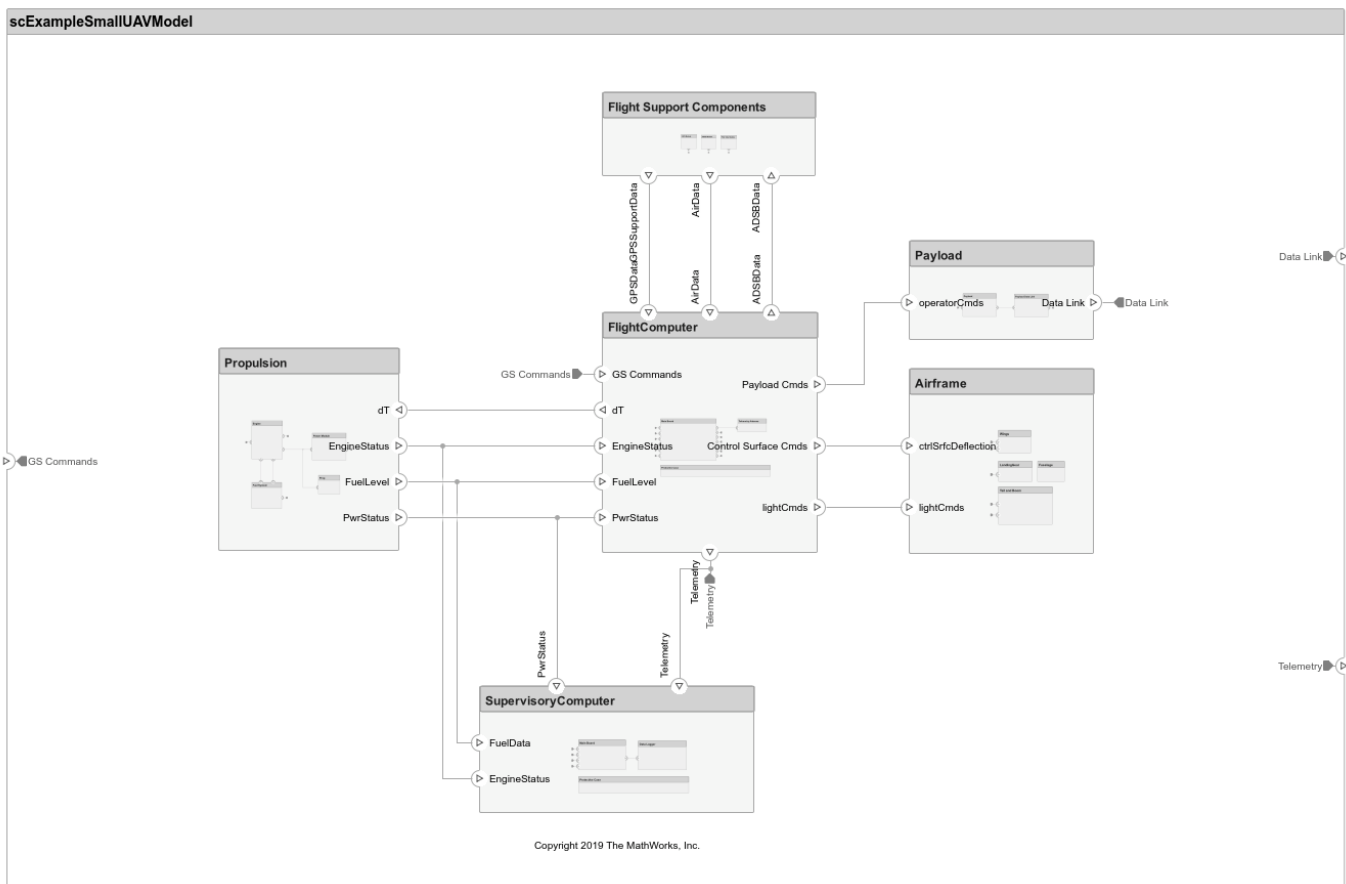
Modeling System Architecture of Small UAV

Overview

This example shows how to use System Composer™ to set up the architecture for a small unmanned aerial vehicle, composed of six top-level components. Learn how to refine your architecture design by authoring interfaces, inspect linked textual requirements, define profiles and stereotypes, and run a static analysis on such an architecture model.

Open the project. `scExampleSmallUAV`

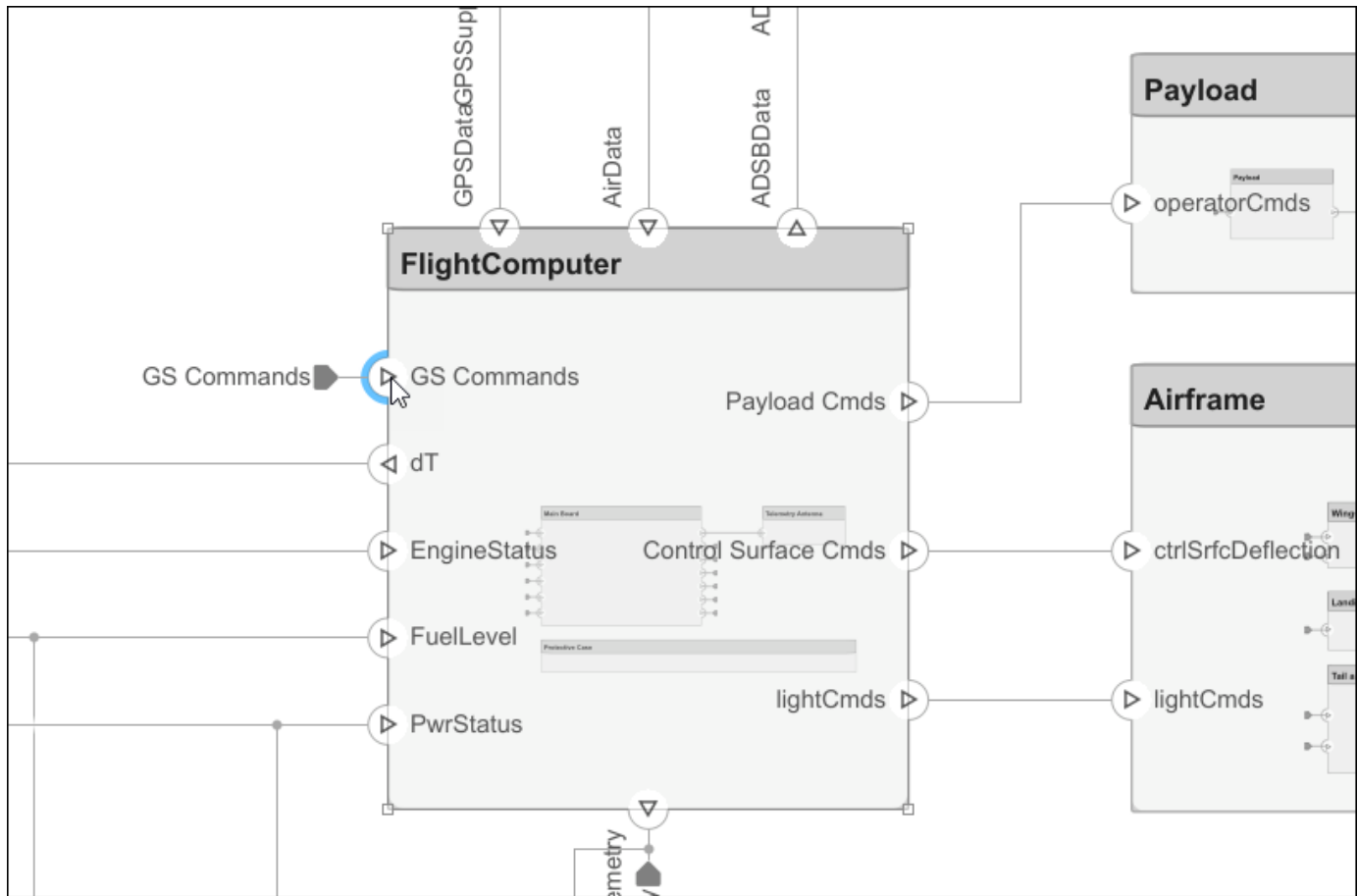
Starting: Simulink



Author Interfaces

Define interfaces for domain-specific data between connections. The information shared between two ports defined by interface element property values further enhances the specification. To open the Interface Editor (System Composer), in the **Modeling** tab in the toolstrip, click **Interface Editor**.

Click the **GS Commands** port on the architecture model to highlight the **architecture_gsCommands** interface and indicate the assignment of the interface.



Interfaces

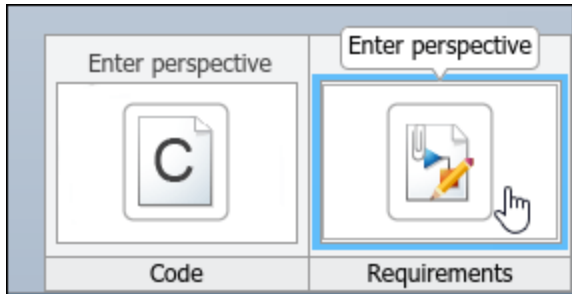
Dictionary View

	Type	Dimensions	Units	Complexity	Minimum	Maximum	Description
scExampleSmallUAVModel.slx							
architecture_gsCommands							
apConfigParams (param_value_bus)							
param_count	uint16	1		real	0	0	Total number of onboard parameters
param_id	int8	16		real	0	0	Onboard parameter id, terminated by NULL if the length is less than 1
param_index	uint16	1		real	0	0	Index of this onboard parameter
param_type	uint8	1		real	0	0	Onboard parameter type: see the MAV_PARAM_TYPE enum for sup
param_value	single	1		real	0	0	Onboard parameter value
gsCommands (gs_commands_bus)							
RTB	uint8	1		real	0	0	Return to Base Command
U_c	single	1		real	0	0	Airspeed Commanded by the GS
guidanceMode	uint8	1		real	0	0	
h_c_midLevel	single	1		real	0	0	Commanded altitude when in Mid Level Commands Mode. Also used
isManualModeOn	uint8	1		real	0	0	
psiDot_c_mdLevel	single	1		real	0	0	Turnrate command when in mid Level Commands guidance mode
viewPointIdx	uint8	1		real	0	0	
uavMission (mission_item_bus)							

Inspect Requirements

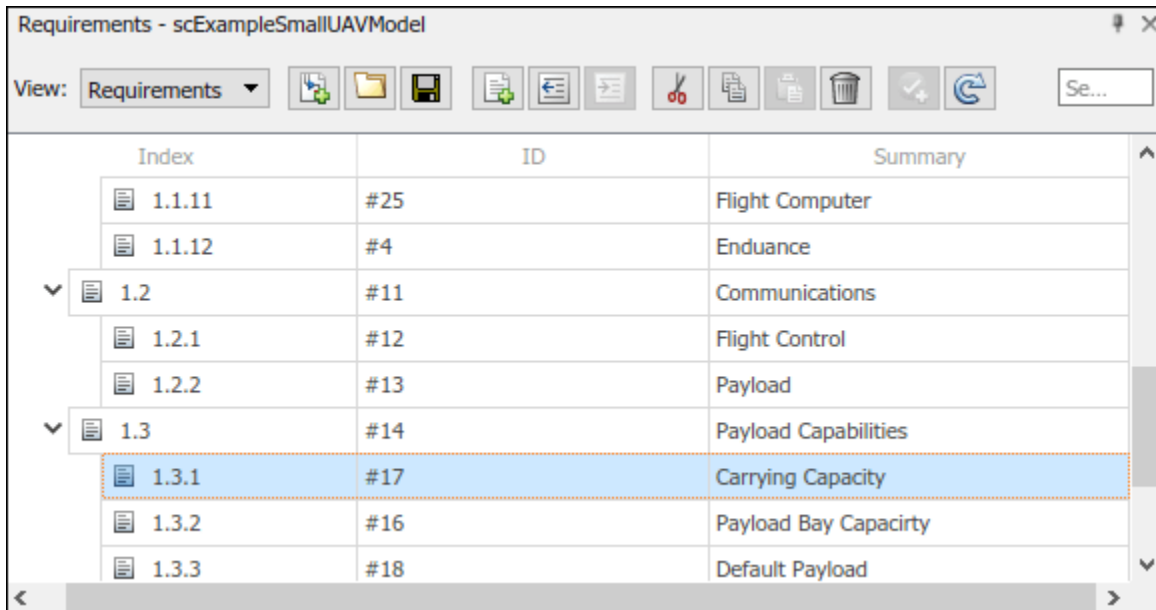
A Requirements Toolbox™ license is required to inspect requirements in a System Composer architecture model.

Components in the architecture model link to system requirements defined in `scExampleSmallUAVModel.s1reqx`. Open the **Requirements Manager**. In the bottom right corner of the model pane, click **Show Perspectives views**. Then, click **Requirements**.



Select components on the model to see the requirement they link to, or, conversely, select items in the **Requirements** view to see which components implement them. Requirements can also be linked to connectors or ports to allow traceability throughout your design artifacts. To edit the requirements in `smallUAVReqs.s1reqx`, select the Requirements Editor from the menu.

The Carrying Capacity requirement highlights the total mass able to be carried by the aircraft. This requirement, along with the weight of the aircraft, is part of the mass rollup analysis performed for early verification and validation.



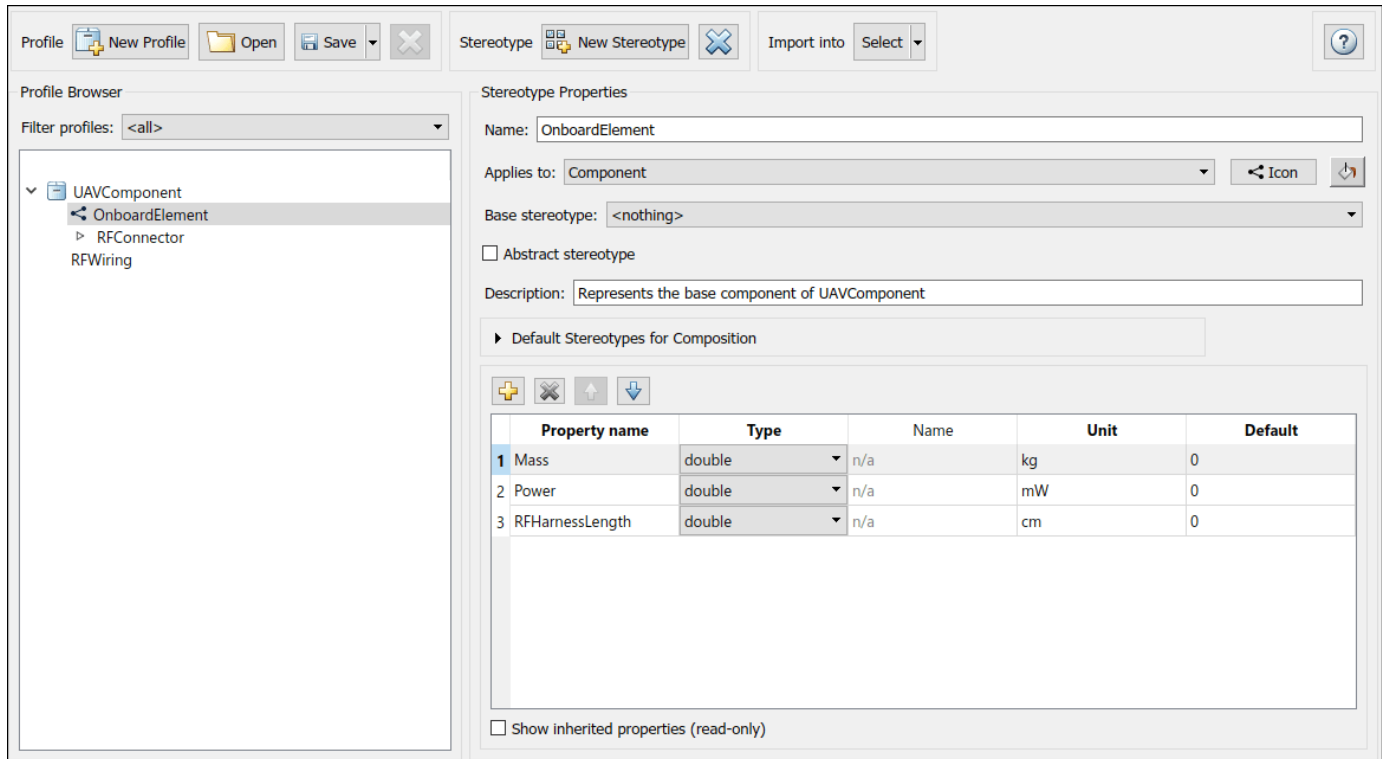
Define Profiles and Stereotypes

To complete specifications and enable analysis later in the design process, stereotypes add custom metadata to architecture model elements. This model has stereotypes for these elements:

- On-board element, applicable to components
- RF connector, applicable to ports
- RF wiring, applicable to connectors

Stereotypes are defined in XML files by using profiles. The profile `UAVComponent.xml` is attached to this model. Edit a profile by using the Profile Editor (System Composer). On the **Modeling** tab, click **Profile Editor**.

The display appears below.

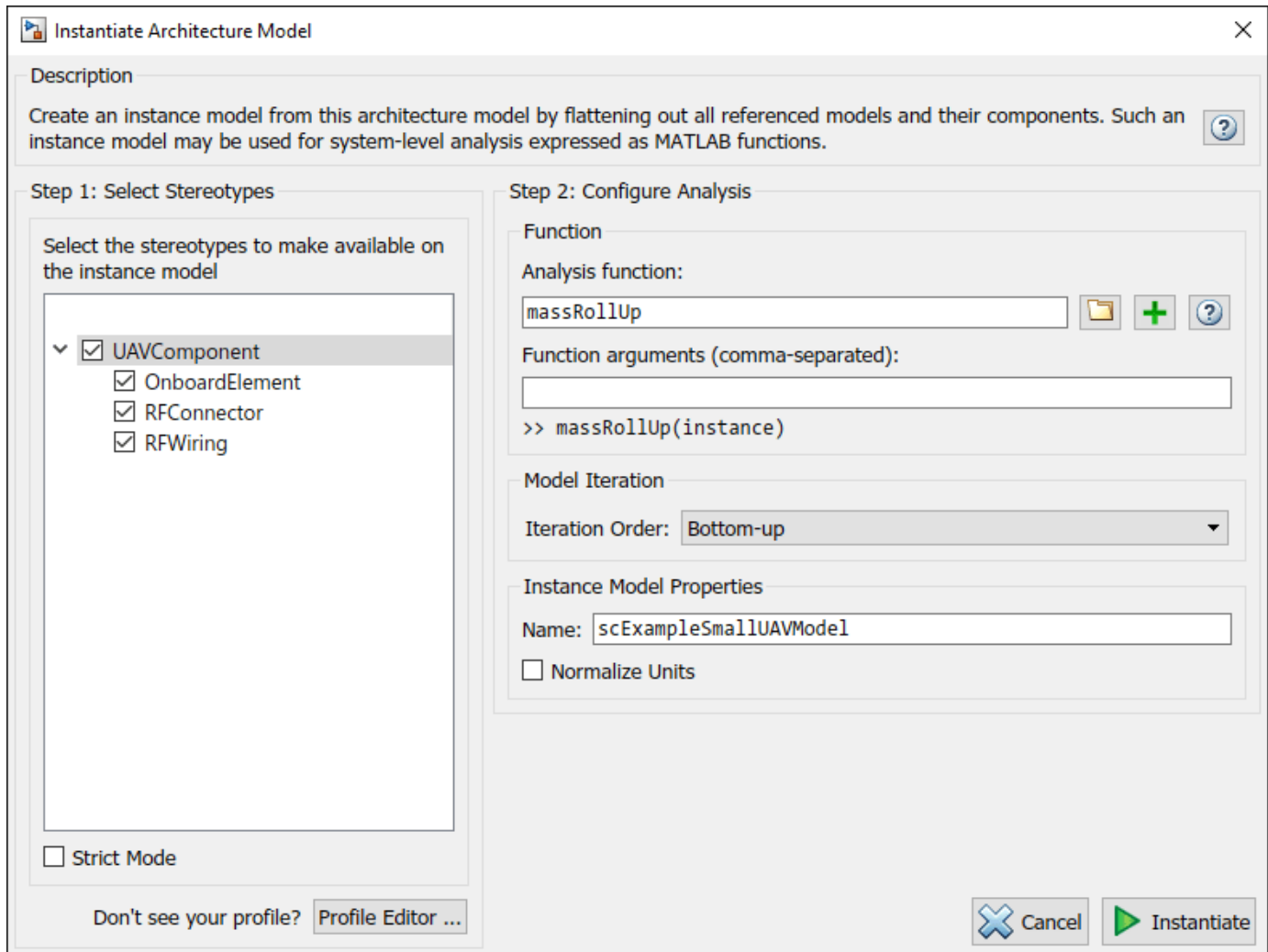


Analyze the Model

To run static analyses on your system, create an analysis model from your architecture model. An analysis model is a tree of instances generated from the elements of the architecture model in which all referenced models are flattened out, and all variants are resolved.

To open the Instantiate Architecture Model (System Composer) tool, click **Analysis Model** on the **Views** menu.

Run a mass rollup on this model. In the dialog, select the stereotypes that you want to include in your analysis. Select the analysis function by browsing to `utilities/massRollUp.m`. Set the model iteration mode to **Bottom-up**.



Uncheck **Strict Mode** so that all components can have a **Mass** property instantiated to facilitate calculation of total mass. Click **Instantiate** to generate an analysis.

Instances	Mass	Power	RFHarnessLength	Length
scExampleSmallUAVModel	15.462	0	0	
Airframe	9.25	0	0	
Fuselage	1.7	0	0	
LandingGear	1.65	0	0	
Tail and Boom	2.7	0	0	
Wings	3.2	0	0	
Airframe:ctrlSrfcDeflection->LandingGear:Brake				0
Airframe:ctrlSrfcDeflection->Tail and Boom:dR_dE				0
Airframe:ctrlSrfcDeflection->Wings:dA_dF				0
Airframe:lightCmds->Tail and Boom:Landing Strobe				0
Airframe:lightCmds->Wings:Navigation Lights				0
Flight Support Components	0.629	0	0	
ADSB Module	0.156	0	0	
ABDSB Antenna	0.058	0	0	
ADSB Board	0.098	0	0	
ADSB Board:RFSignal->ABDSB Antenna:RFSignal				75
ADSB Module:ADSBData->ADSB Board:ADSBData				0
GPS Module	0.398	0	0	
GPS Antenna	0.128	0	0	
GPS Board	0.27	0	0	
GPS Board:GPSData->GPS Module:GPSSupportData				0
GPS Board:RFSignal->GPS Antenna:RFSignal				38
Pitot Tube Module	0.075	0	0	
Flight Support Components:ADSBData->ADSB Module:ADSBData				0
GPS Module:GPSSupportData->Flight Support Components:GPSSupportData				0
Pitot Tube Module:AirData->Flight Support Components:AirData				0
FlightComputer	0.388	0	0	
Main Board	0.145	0	0	
Protective Case	0.195	0	0	
Telemetry Antenna	0.048	0	0	
FlightComputer:AirData->Main Board:AirData				0

Once on the Analysis Viewer (System Composer) screen, click **Analyze**. The analysis function iterates through model elements bottom up, assigning the **Mass** property of each component as a sum of the **Mass** properties of its subcomponents. The overall weight of the system is assigned to the **Mass** property of the top level component, `scExampleSmallUAVModel`.

See Also

More About

- “Create and Store Links” on page 3-31
- “Link Blocks and Requirements” on page 3-2
- “Review Requirements Implementation Status” on page 4-2

Model-Based Systems Engineering for Space-Based Applications

This example provides an overview of the **CubeSat Model-Based System Engineering Project** template, available from the Simulink® start page, under Aerospace Blockset™. It demonstrates how to model a space mission architecture in Simulink with System Composer™ and Aerospace Blockset for a 1U CubeSat in low Earth orbit (LEO). The CubeSat's mission is to image MathWorks Headquarters in Natick, Massachusetts at least once per day. The project references the Aerospace Blockset *CubeSat Simulation Project*, reusing the vehicle dynamics, environment models, data dictionaries, and flight control system models defined in that project.

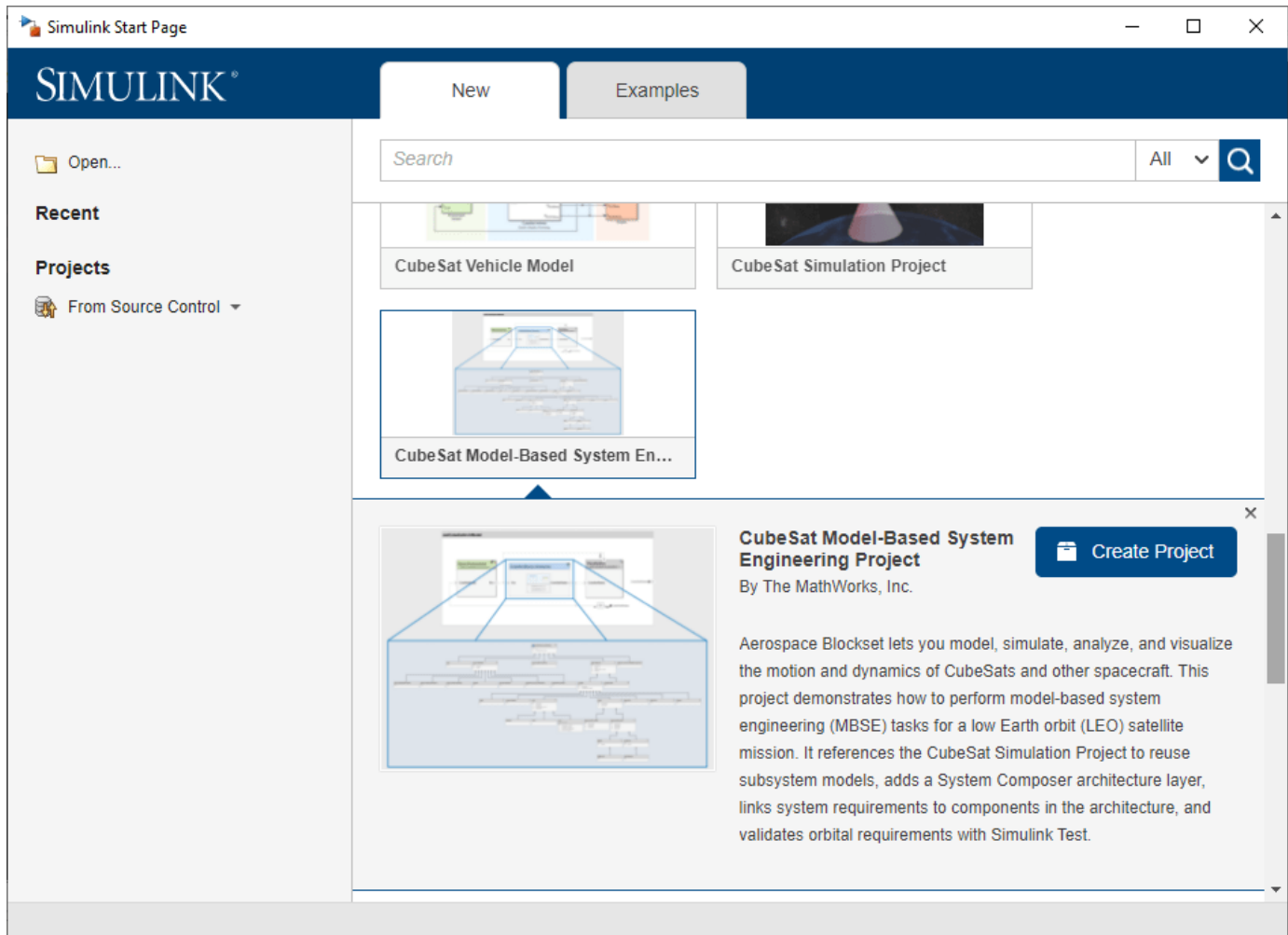
This project demonstrates how to:

- Define system level requirements for a CubeSat mission in Simulink
- Compose a system architecture for the mission in System Composer
- Link system-level requirements to components in the architecture with Requirements Toolbox™
- Model vehicle dynamics and flight control systems with Aerospace Blockset
- Validate orbital requirements using mission analysis tools and Simulink Test™

Open the Project

To create a new instance of the **CubeSat Model-Based System Engineering Project**, select **Create Project** in the Simulink start page. When the project is loaded, an architecture model for the CubeSat opens.

```
open("asbCubeSatMBSEProject.sltx");
```



Define System-level Requirements

Define a set of system-level requirements for the mission. You can import these requirements from third-party requirement management tools such as ReqIF (Requirements Interchange Format) files or author them directly in the Requirements Editor.

This example contains a set of system-level requirements stored in *SystemRequirements.sreqx*. Open this requirement specification file in the **Requirements Editor**. Access the **Requirements Editor** from the **Apps** tab or by double-clicking on *SystemRequirements.sreqx* in the project folder browser.

Our top level requirement for this mission is:

- 1 The system shall provide and store visual imagery of MathWorks headquarters [42.2775 N, 71.2468 W] once daily at 10 meters resolution.

Additional requirements are decomposed from this top-level requirement to create a hierarchy of requirements for the architecture.

The screenshot displays the Requirements Editor application. The main window is titled "Requirements Editor" and features a ribbon-style toolbar with the following sections:

- FILE:** New Requirement Set, Open
- PROFILE:** Profile Editor
- REQUIREMENTS:** Add Requirement
- LINKS:** Add Link
- VIEW:** Show Requirements, Show Links
- EDIT:** Search
- ANALYSIS:** ANALYSIS
- SHARE:** SHARE

The left pane shows a tree view of requirements:

Index	ID	Summary
SystemRequirements		
1	#1	Provide visual imagery
1.1	#2	Visual imagery collection
1.1.1	#10	Orbit Selection
1.1.2	#11	CubeSat
1.1.2.1	#4	Imaging payload performance
1.1.2.2	#5	GNC pointing accuracy
1.1.2.3	#6	GNC slew rate
1.1.2.4	#7	Image downlink
1.1.2.5	#8	On-board image management
1.1.2.6	#23	Power for on-board imaging tasks
1.1.2.6.1	#24	Power System Control
1.1.2.6.2	#25	Power System Plant
1.1.2.6.2.1	#26	Solar Panel
1.1.2.6.2.1.1	#28	Solar Panel Cells
1.1.2.6.2.2	#27	Battery
1.1.2.6.2.2.1	#29	Battery Cell
1.2	#3	Visual imagery ground storage

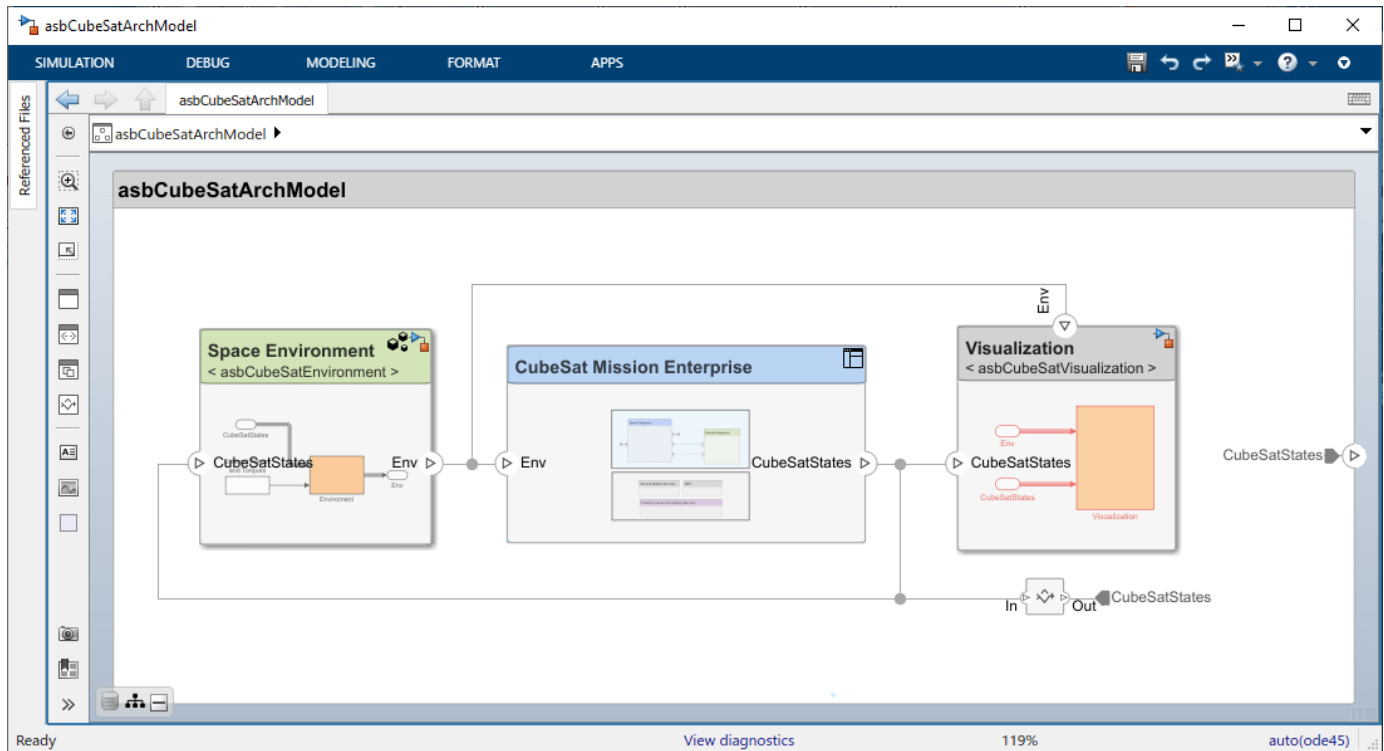
The right pane shows the details for Requirement #1:

- Details:** Requirement: #1
- Properties:**
 - Type: Functional
 - Index: 1
 - Custom ID: #1
 - Summary: Provide visual imagery
- Description / Rationale:**

The system shall provide and store visual imagery of MathWorks headquarters [42.2775 N, 71.2468 W] 1 times daily at 10 meters resolution.
- Keywords:** [Empty field]
- Revision information:** [Collapsible section]
- Links:** [Collapsible section]
- Comments:** [Collapsible section]

Compose a System Architecture

System Composer enables the specification and analysis of architectures for model-based systems engineering. Use the system-level requirements defined above to guide the creation of an architecture model in System Composer. The architecture in this example is based on *CubeSat Reference Model (CRM)* developed by the International Council on Systems Engineering (INCOSE) Space Systems Working Group (SSWG) [1].



The architecture is composed of components, ports, and connectors. A component is a part of a system that fulfills a clear function in the context of the architecture. It defines an architectural element, such as a system, subsystem, hardware, software, or other conceptual entity.

Ports are nodes on a component or architecture that represent a point of interaction with its environment. A port permits the flow of information to and from other components or systems. Connectors are lines that provide connections between ports. Connectors describe how information flows between components in an architecture.

Extend the Architecture with Stereotypes and Interfaces

You can add additional levels of detail to an architecture using stereotypes and interfaces.

Stereotypes

Stereotypes extend the architectural elements by adding domain-specific metadata to each element. Stereotypes are applied to components, connectors, ports, and other architectural elements to provide these elements with a common set of properties such as mass, cost, power, etc.

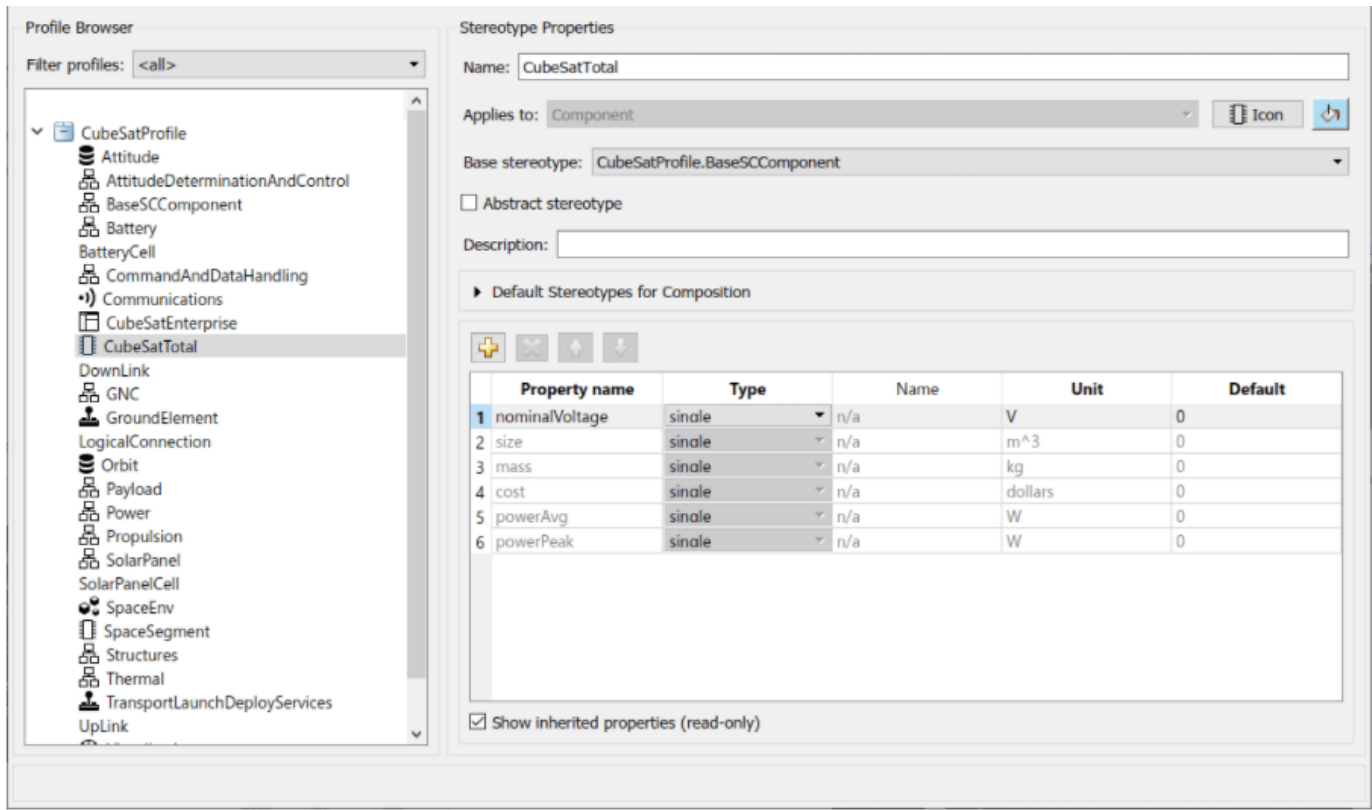
Packages of stereotypes used by one or more architectures are stored in profiles. This example includes a profile of stereotypes called *CubeSatProfile.xml*. To view, edit, or add new stereotypes to the profile, open this profile in the **Profile Editor** from the **Modeling** Tab.

This profile defines a set of stereotypes that are applied to components and connectors in the CubeSat architecture.

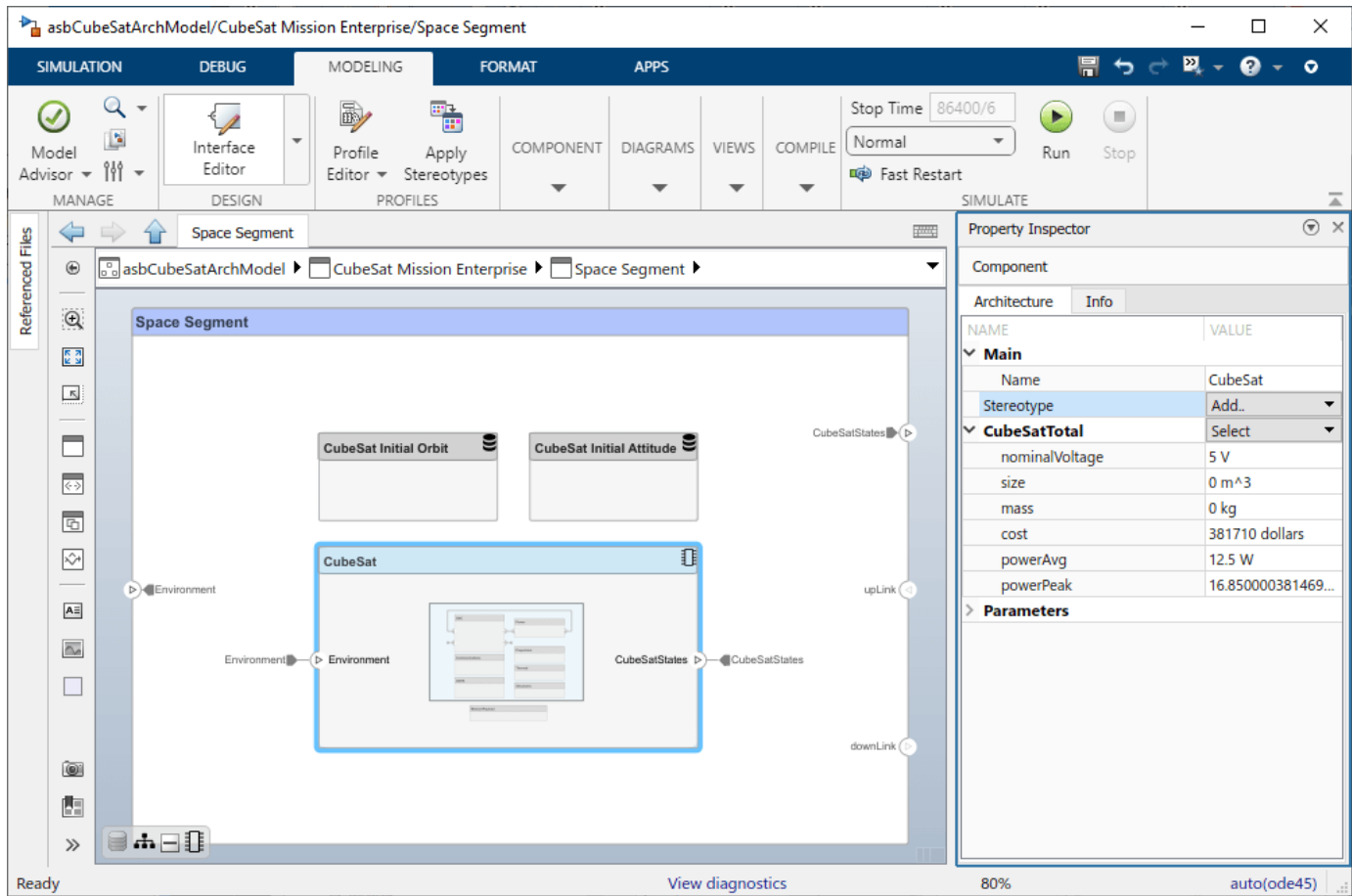
The screenshot displays a software interface with two main panels. On the left is the 'Profile Browser' showing a tree view of components under 'CubeSatProfile'. The 'BaseSCComponent' is selected. On the right is the 'Stereotype Properties' panel for 'BaseSCComponent'. It includes a 'Name' field, an 'Applies to' dropdown set to 'Component', and a 'Base stereotype' dropdown set to '<nothing>'. The 'Abstract stereotype' checkbox is checked. Below is a 'Description' field and a section for 'Default Stereotypes for Composition' containing a table of properties.

	Property name	Type	Name	Unit	Default
1	size	single	n/a	m ³	0
2	mass	single	n/a	kg	0
3	cost	single	n/a	dollars	0
4	powerAvg	single	n/a	W	0
5	powerPeak	single	n/a	W	0

Stereotypes can also inherit properties from abstract base stereotypes. For example, BaseSCComponent in the profile above contains properties for size, mass, cost, and power demand. We can add another stereotype to the profile, CubeSatTotal, and define BaseSCComponent as its base stereotype. CubeSatTotal adds in its own property, nominalVoltage, but also inherits properties from its base stereotype.

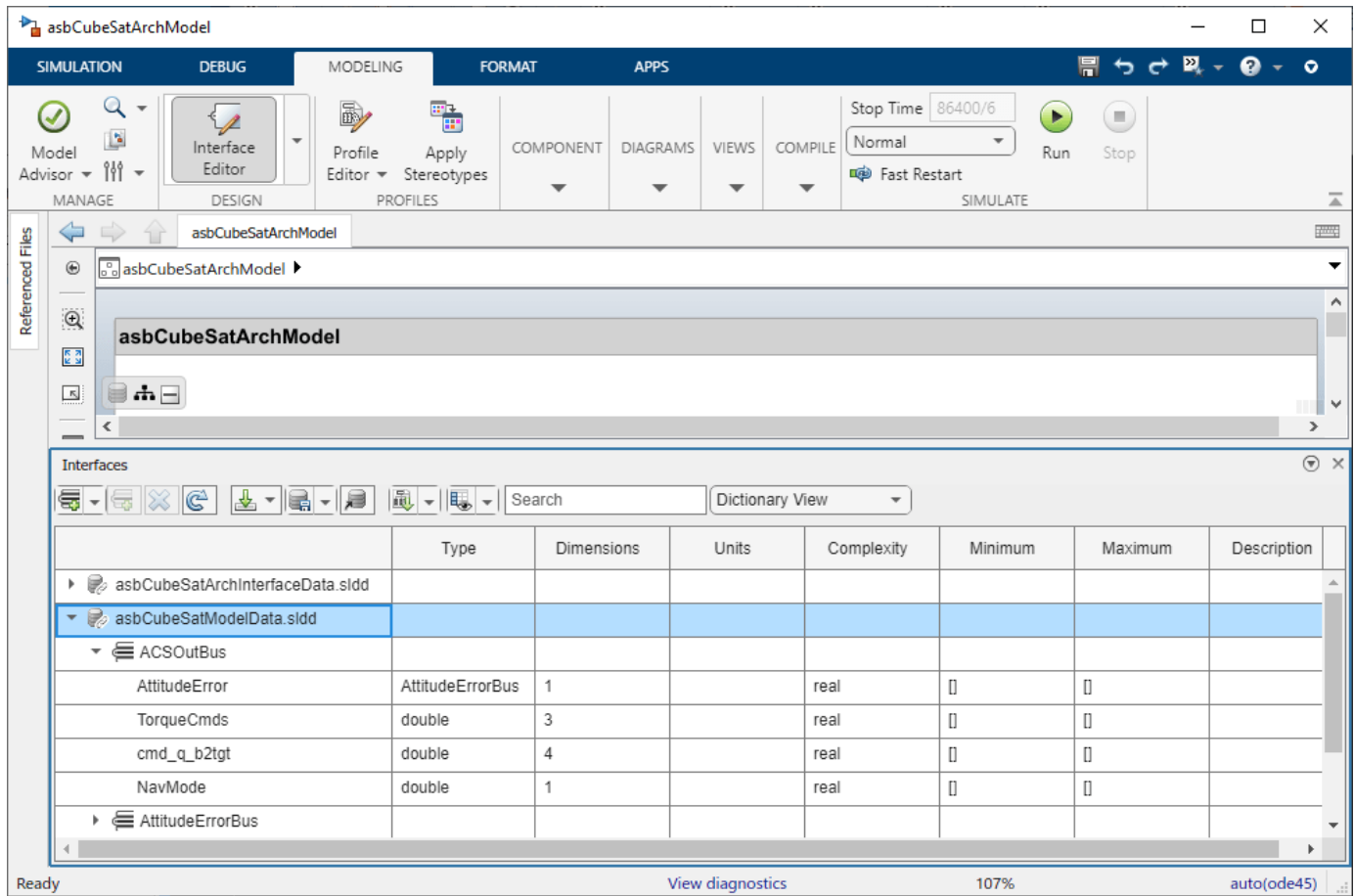


In the architecture model, apply the `CubeSatTotal` stereotype to CubeSat system component (asbCubeSatArchModel/CubeSat Mission Enterprise/Space Segment/CubeSat). Select the component in the model. In the Property Inspector, select the desired stereotype from the drop-down window. Next, set property values for the CubeSat component.



Interfaces

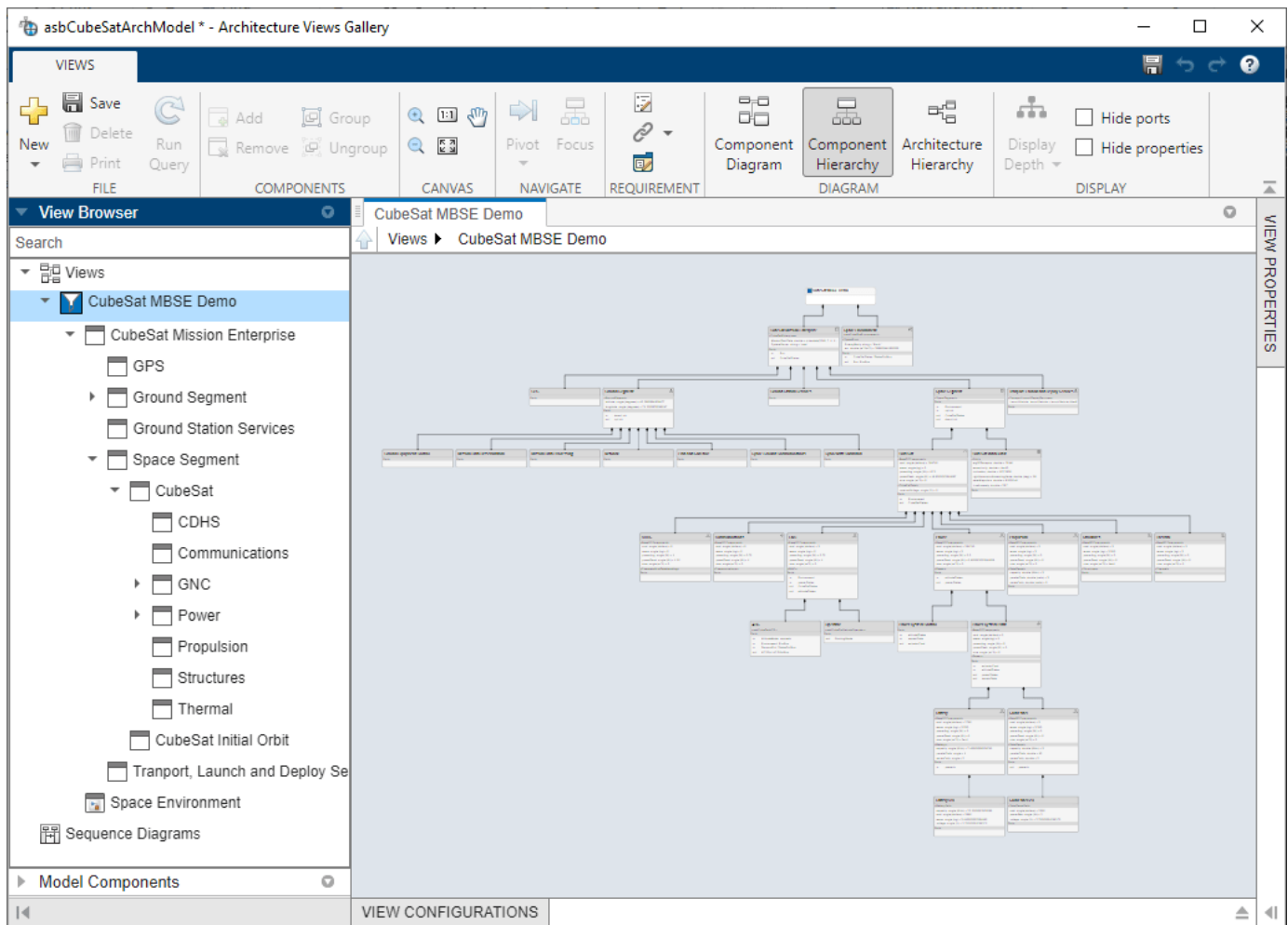
Data interfaces define the kind of information that flows through a port. The same interface can be assigned to multiple ports. A data interface can be composite, meaning that it can include data elements that describe the properties of an interface signal. Create and manage interfaces from the **Interface Editor**. Existing users of Simulink can draw a parallel between interfaces in System Composer and buses in Simulink. In fact, buses can be used to define interfaces (and vice versa). For example, the data dictionary *asbCubeSatModelData.sldd* contains several bus definitions, including **ACSOutBus**, that can be viewed in the **Interface Editor** and applied to architecture ports.



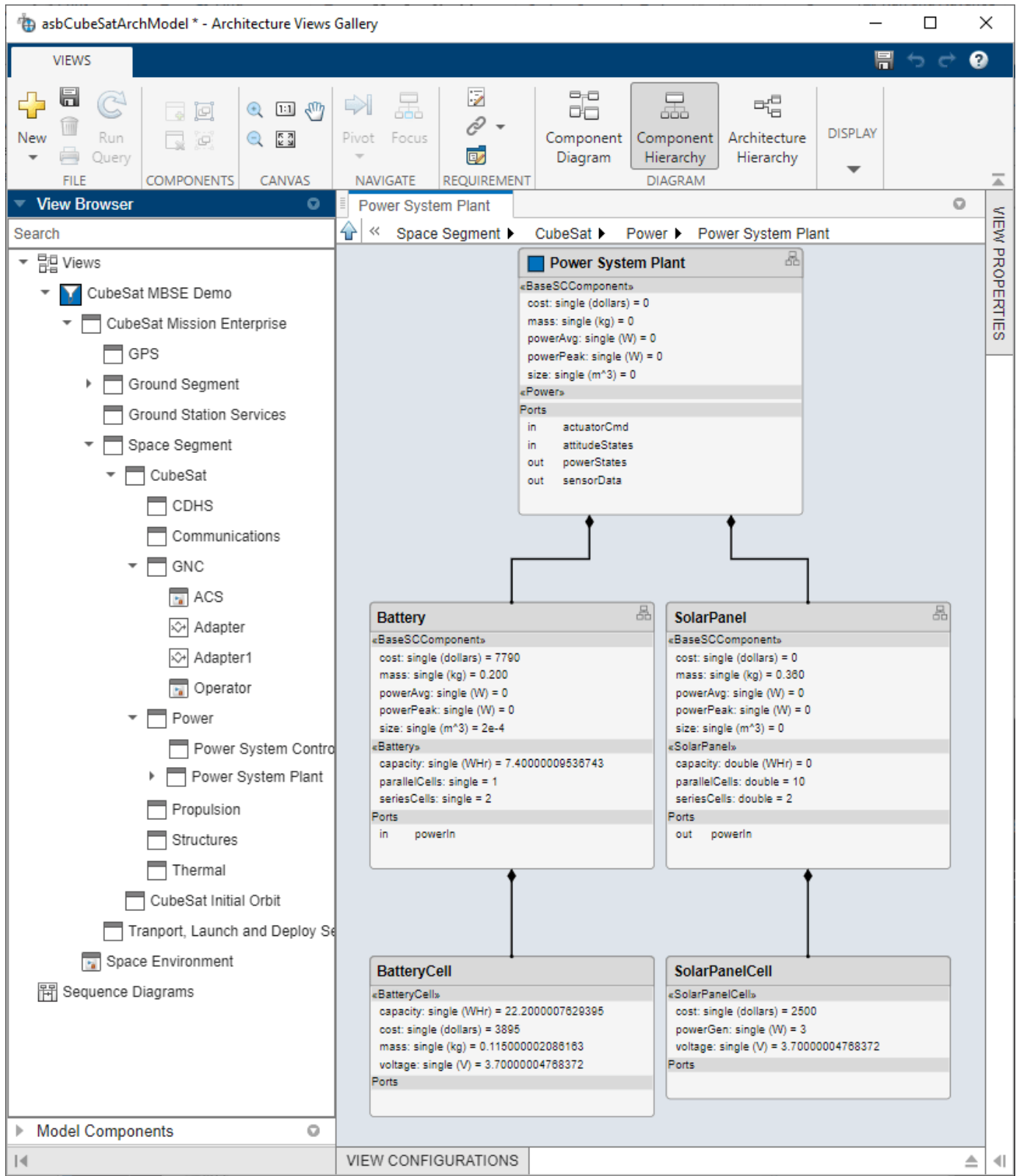
Visualize the System with Architecture Views

Now that we have implemented our architecture using components, stereotypes, ports, and interfaces, we can visualize our system with an architecture view. In the **Modeling** Tab, select **Views**.

Use the **Component Hierarchy** view to show our system component hierarchy. Each component also lists its stereotype property values and ports.



You can also view the hierarchy at different depths of the architecture. For example, navigate to the **Power System Plant** component of the architecture by double-clicking the component in the **View Browser**.



Link Requirements to Architecture Components

To link requirements to the architectural elements that implement them, use the **Requirements Manager**. Drag the requirement onto the corresponding component, port, or interface. Using this linking mechanism, we can identify how requirements are met in the architecture model. The column labeled "Implemented" in the **Requirements Manager** shows whether a textual requirement has been linked to a component in the given model. For example, our top-level requirement "Provide visual imagery" is linked to our top-level component CubeSat Mission Enterprise with decomposed requirements linked to respective decomposed architectural components.

The screenshot displays the Requirements Manager interface within a modeling environment. The main workspace shows an architecture diagram with components like 'environment', 'CubeSat Mission Enterprise', and 'Visualization'. A requirement '#1: Provide visual imagery' is highlighted and linked to the 'CubeSat Mission Enterprise' component. The 'Requirements Manager' pane at the bottom shows a table of requirements, and the 'Property Inspector' pane on the right shows details for requirement #1, including its description and rationale.

Index	ID	Summary	Implemented
SystemRequirements			
1	#1	Provide visual imagery	Implemented
1.1	#2	Visual imagery collection	Implemented
1.2	#3	Visual imagery ground storage	Implemented

Property Inspector - Requirement: #1

Details

Properties

Type: Functional
 Index: 1
 Custom ID: #1
 Summary: Provide visual imagery

Description | **Rationale**

The system shall provide and store visual imagery of MathWorks headquarters [42.2775 N, 71.2468 W] 1 times daily at 10 meters resolution.

Keywords:

Revision information:

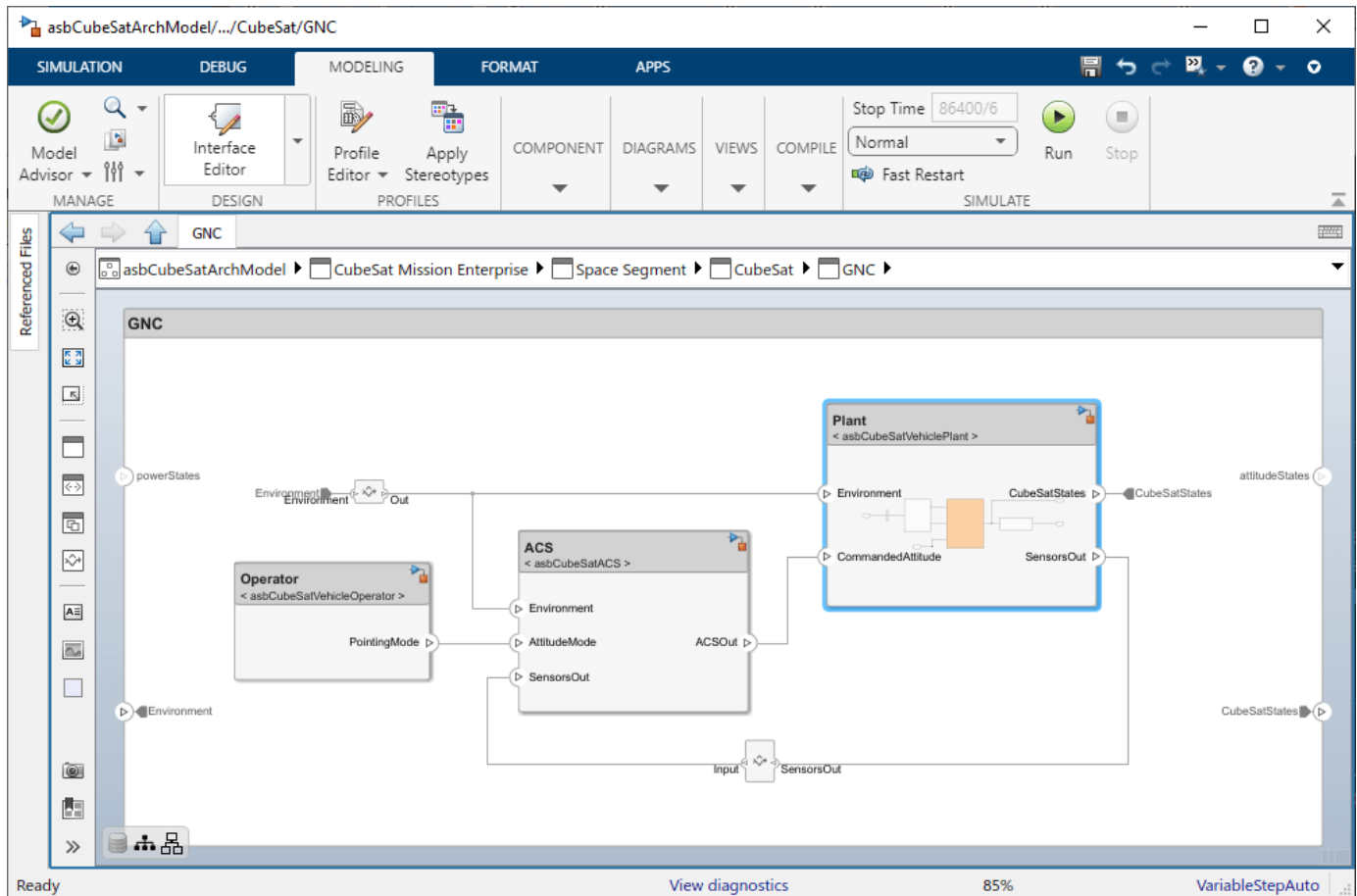
Links

Implemented by:
 CubeSat Mission Enterprise

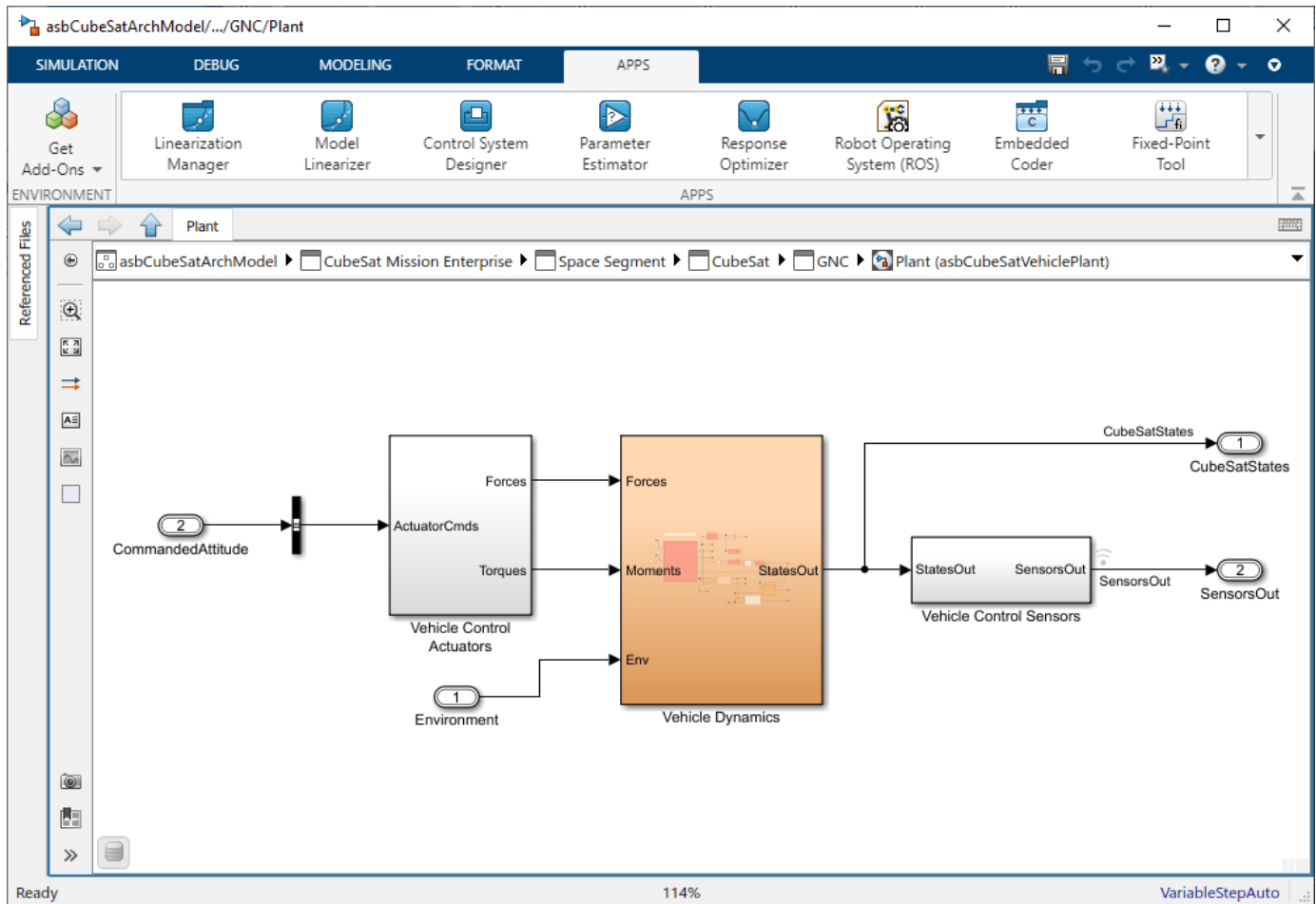
Verified by:
 Verify visual imagery

Connecting the Architecture to Design Models

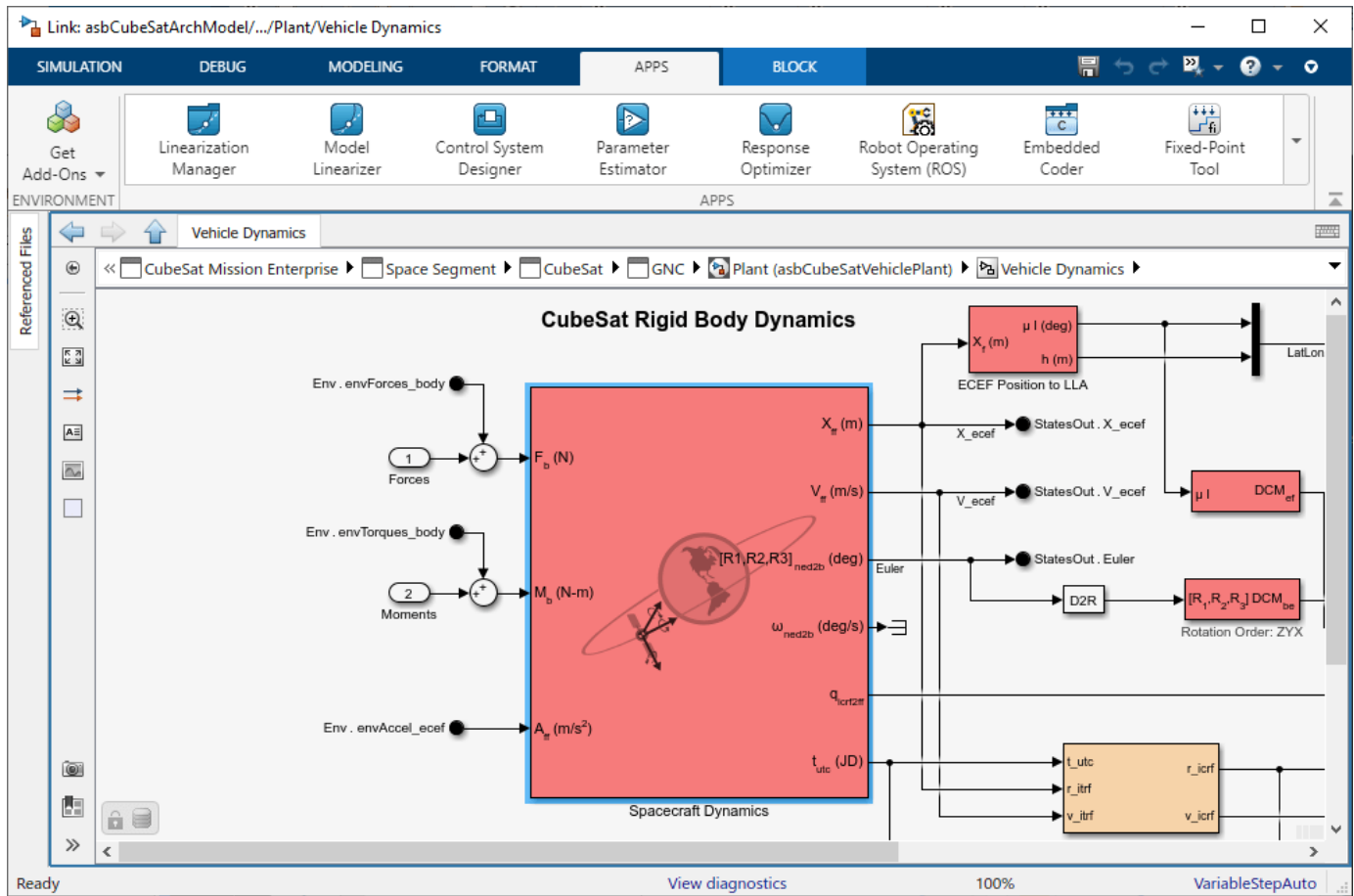
As the design process matures through analysis and other systems engineering processes, we can begin to populate our architecture with dynamics and behavior models. System Composer is built as a layer on top of Simulink, which enables Simulink models to be directly referenced from the components we have created. We can then simulate our architecture model as a Simulink model and generate results for analysis. For example, the GNC subsystem component contains 3 Simulink model references that are part of the *CubeSat Simulation Project*.



Double-click these reference components to open the underlying Simulink models. Notice that the interfaces defined in the architecture map to bus signals in the Simulink model.

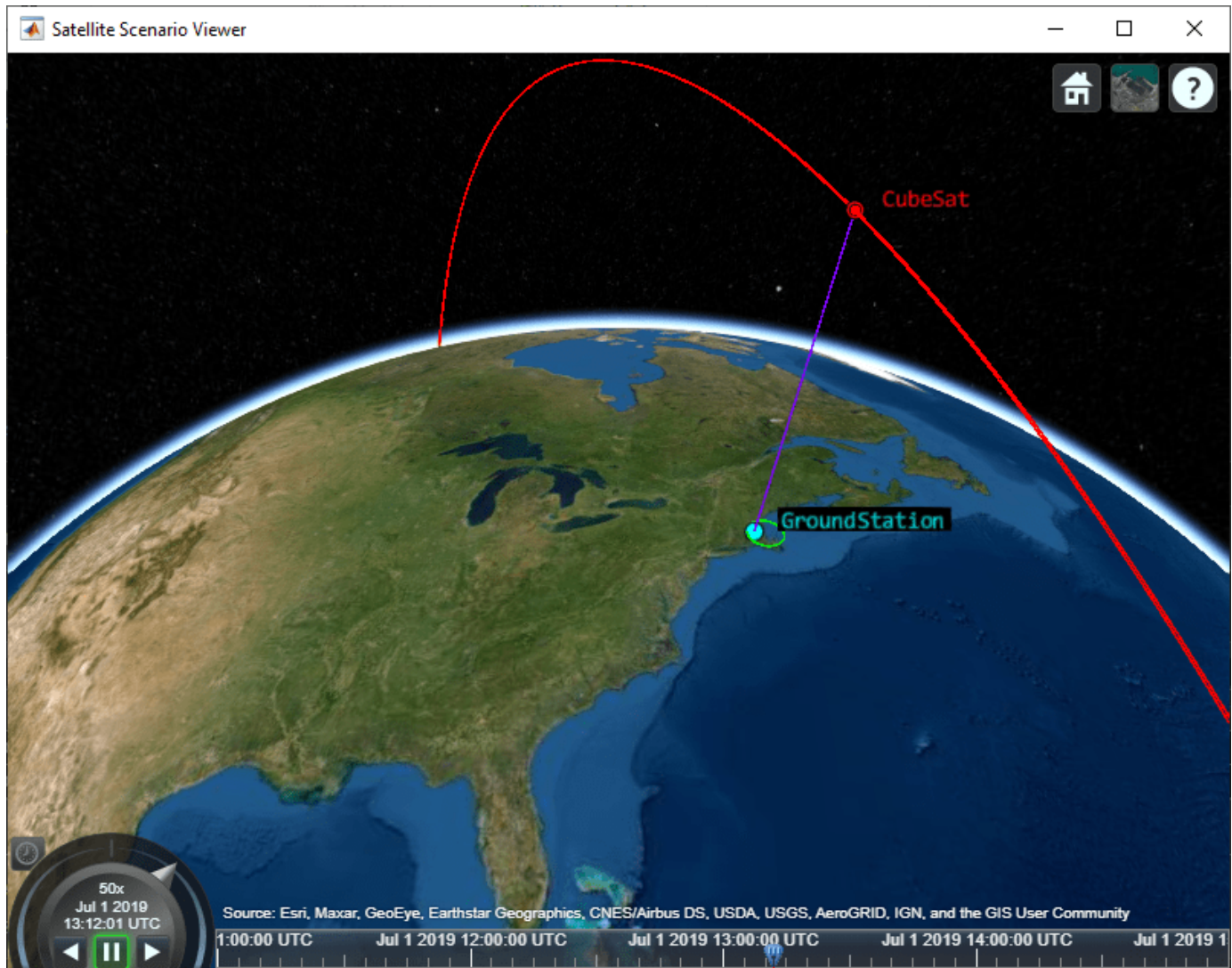


This example uses the **Spacecraft Dynamics** block from Aerospace Blockset to propagate the CubeSat orbit and rotational states.



Simulate System Architecture to Validate Orbital Requirements

We can use simulation to verify our system-level requirements. In this scenario, our top level requirement states that the CubeSat onboard camera captures an image of MathWorks Headquarters at [42.2775 N, 71.2468 W] once daily at 10 meters resolution. We can manually validate this requirement with various mission analysis tools. For examples of these analyses, click on the project shortcuts *Analyze with Mapping Toolbox* and *Analyze with Satellite Scenario*.

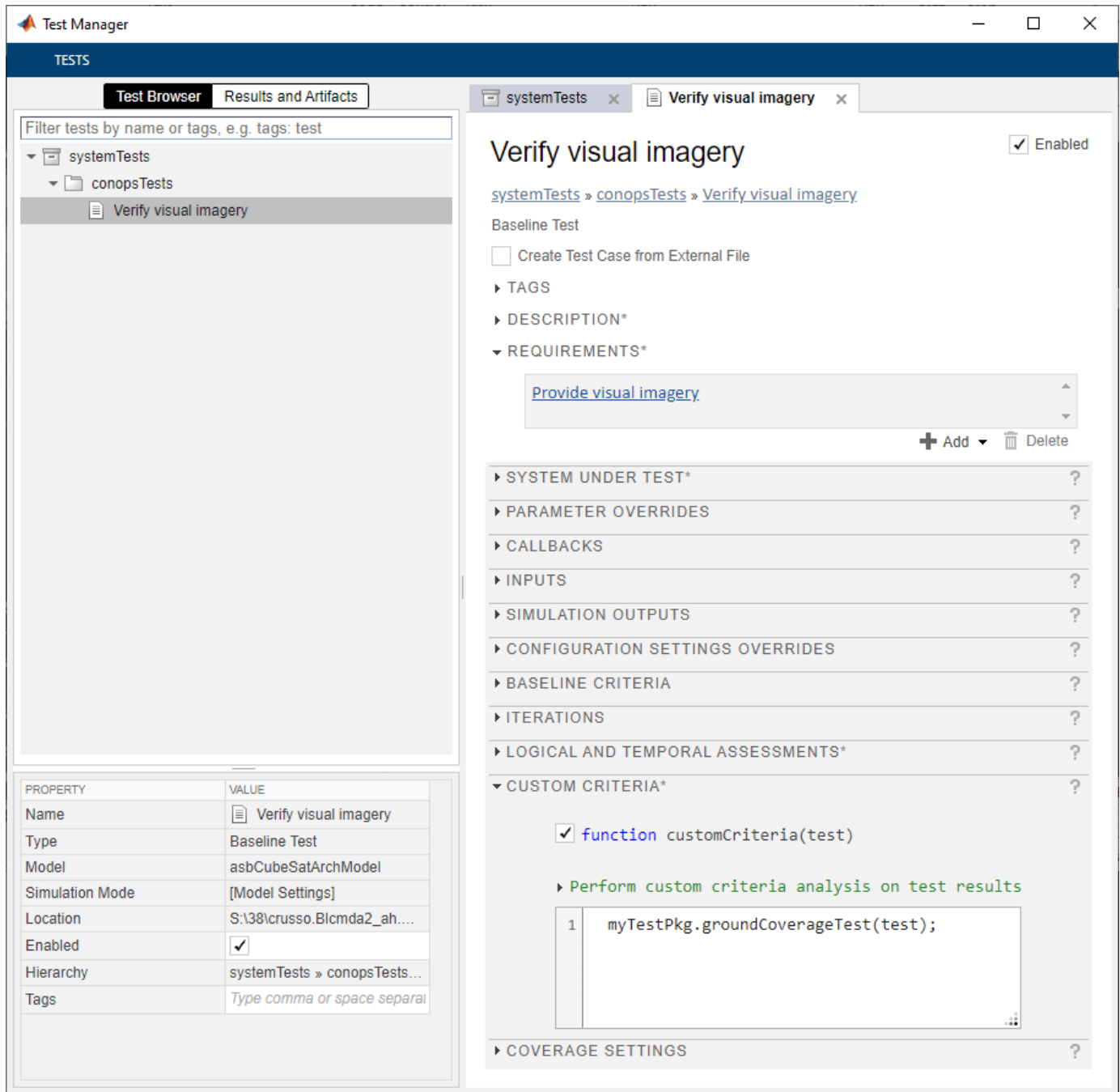


The satellite scenario created in the *Analyze with Satellite Scenario* shortcut example is shown above.

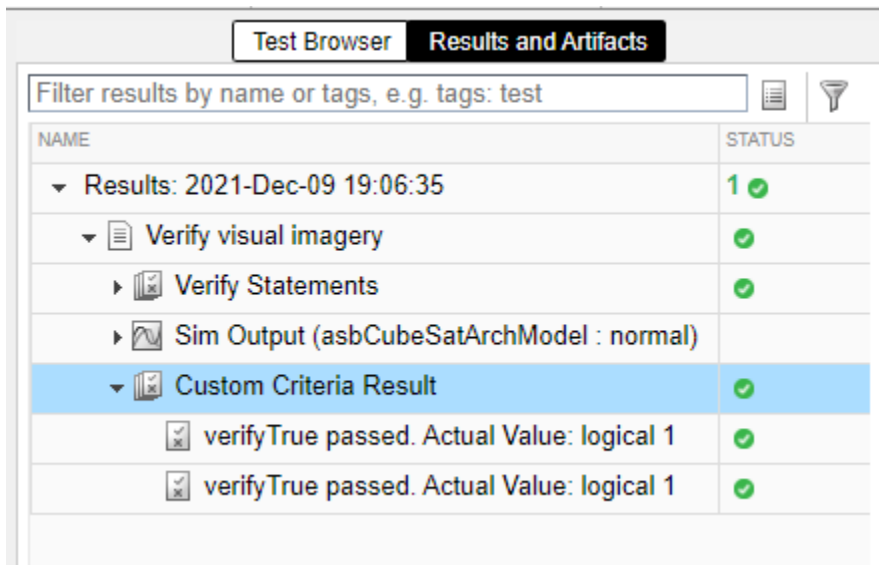
Validate Orbital Requirements using Simulink Test

Although we can use MATLAB to visualize and analyze the CubeSat behavior, we can also use Simulink Test to build test cases. This test case automates the requirements-based testing process by using the testing framework to test whether our CubeSat orbit and attitude meet our high-level requirement. The test case approach enables us to create a scalable, maintainable, and flexible testing infrastructure based on our textual requirements.

This example contains a test file *systemTests.mldatx*. Double-click this file in the project folder browser to view it in the **Test Manager**. Our test file contains a test to verify our top-level requirement. The "Verify visual imagery" testpoint is mapped to the requirement "Provide visual imagery" and defines a MATLAB function to use as custom criteria for the test. While this test case is not a comprehensive validation of our overall mission, it is useful during early development to confirm our initial orbit selection is reasonable, allowing us to continue refining and adding detail to our architecture.



Run the test point in the **Test Manager** and confirm that the test passes. Passing results indicate that the CubeSat onboard camera as visibility to the imaging target during the simulation window.



NAME	STATUS
▼ Results: 2021-Dec-09 19:06:35	1 ✓
▼ Verify visual imagery	✓
▶ Verify Statements	✓
▶ Sim Output (asbCubeSatArchModel : normal)	
▼ Custom Criteria Result	✓
verifyTrue passed. Actual Value: logical 1	✓
verifyTrue passed. Actual Value: logical 1	✓

References

[1] "Space Systems Working Group." INCOSE, 2019, <https://www.incose.org/incose-member-resources/working-groups/Application/space-systems>.

See Also

Orbit Propagator | Spacecraft Dynamics | Attitude Profile

Related Examples

- "Compose Architectures Visually" (System Composer)
- "Define Profiles and Stereotypes" (System Composer)
- "Manage Requirements" (System Composer)

Requirements-Based Verification

- “Review Requirements Implementation Status” on page 4-2
- “Review Requirements Verification Status” on page 4-6
- “Validate Requirements by Analyzing Model Properties” on page 4-10
- “Justify Requirements” on page 4-17
- “Linking to a Test Script” on page 4-20
- “Include Results from External Sources in Verification Status” on page 4-28
- “Linking to a Result File” on page 4-31
- “Integrating Results from a Custom-Authored MATLAB Script as a Test” on page 4-37
- “Integrating Results from an External Result File” on page 4-41
- “Integrating Results from a Custom Authored MUnit Script as a Test” on page 4-45
- “Fix Requirements-Based Testing Issues” on page 4-49

Review Requirements Implementation Status

In this section...
“Implement Functional Requirements by Linking to Model Elements” on page 4-2
“Run Link Implementation Analysis” on page 4-3
“View the Implementation Status” on page 4-4

Requirements Toolbox provides you with implementation status summaries for your requirement sets. You can use these status summaries to identify requirement implementation gaps in your design.

Implement Functional Requirements by Linking to Model Elements

The requirement type specifies the role that a requirement has. Functional requirements are meant to be implemented and contribute to the implementation status, as well as requirements with a custom type that is a subtype of `Functional`. For more information, see “Define Custom Requirement and Link Types by Using `sl_customization` Files” on page 3-42. When you select a requirement in the **Requirements Editor**, the requirement type is displayed in the right pane, under **Properties**. When you add a requirement, it is created with the `Functional` type by default. If a requirement is not meant to be implemented, you can change the requirement type. To read more about requirement types, see “Requirement Types” on page 1-6.

To implement a functional requirement, you can link it with a Simulink, Stateflow, or System Composer model element. Requirements that have an incoming link with the `Implement` type or a custom link type that is defined as a subtype of `Implement` are considered implemented by the implementation status. For more information, see “Link Types” on page 3-34 and “Define Custom Requirement and Link Types by Using `sl_customization` Files” on page 3-42.

The implementation status for a requirement set is cumulatively aggregated over the requirements in the set. Each child requirement must be implemented for the parent requirement to be considered implemented. If you need to manually implement a requirement, you can link it to a justification object for implementation. The implementation status considers this requirement's lack of implementation to be justified. To read more about justifying requirements, see “Justify Requirements” on page 4-17.

Note The implementation status will consider any requirement to be implemented if it has an incoming link of the `Implement` type, regardless of the link source item (unless the link source is a justification, in which case it will be considered justified). To read about how to change an existing link type, see “Link Types” on page 3-34.

When you link a requirement to a Simulink, Stateflow, or System Composer model element, the link is created with the `Implement` type by default. When you select a requirement in the **Requirements Editor**, associated links and the link type are displayed in the right pane, under **Links**.


The screenshot displays the Requirements Editor interface. The main window is titled 'Requirements Editor' and contains a tree view of requirements. The selected requirement is #3, 'Avoid repeating commands'. The right-hand pane shows the details for this requirement, including a 'Links' section with a link to 'doNot Repeat'.

Index	ID	Summary	Implemented	Verified
crs_req_func_spec				
1	#1	Driver Switch Request Handling		
1.1	#2	Switch precedence		
1.2	#3	Avoid repeating commands		
1.3	#4	Long Switch recognition		
1.4	#7	Cancel Switch Detection		
1.5	#8	Set Switch Detection		
1.6	#9	Enable Switch Detection		
1.7	#10	Resume Switch Detection		
1.8	#11	Increment Switch Detection		
1.9	#15	Decrement Switch Detection		
2	#19	Cruise Control Mode		
3	#37	Calculate Target Speed and Thro...		
4	#44	System Interface		
4.1	#45	Inputs		
4.2	#57	Outputs		
4.3	#63	Parameters		
5	#71	Justifications		
5.1	#72	Non-functional requirement		
crs_req				

Tip If a requirement can be implemented by multiple items and you want to get the detailed status of the implementation of each item, you can split a requirement into smaller requirements and implement each requirement separately.

Run Link Implementation Analysis


Requirements Toolbox does not perform link implementation analysis until you run it. You can run the analysis in the **Requirements Editor** or Requirements Perspective.

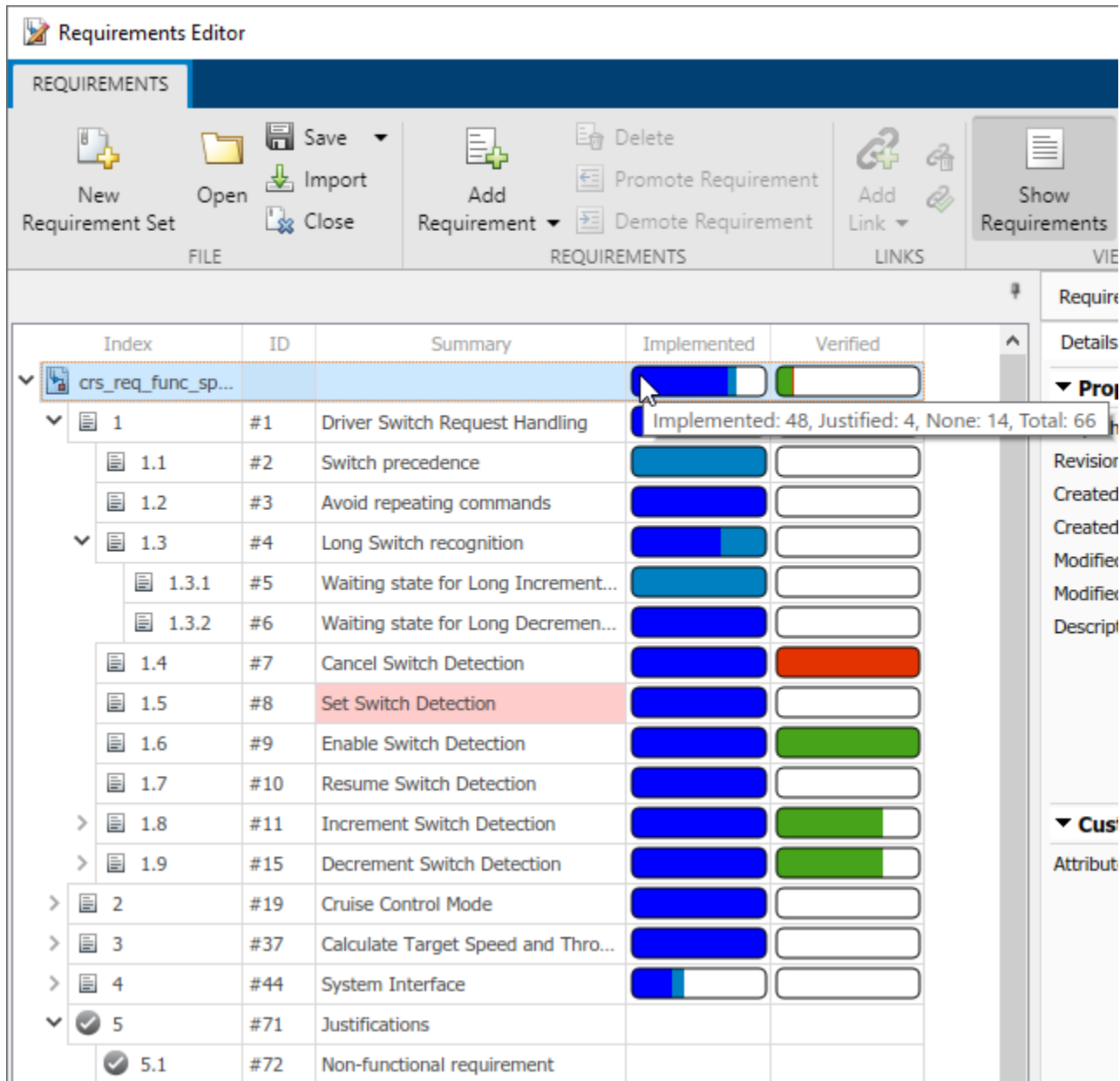
A banner in the **Requirements Editor** or Requirements Perspective indicates when results are pending. To run the analysis, click **Analyze now** in the banner. Alternatively, click **Refresh** in the **Requirements Editor** or the refresh button  in the Requirements Perspective.

Link implementation analysis continuously runs in the background until you use `slreq.clear`.

Alternatively, you can use `updateImplementationStatus` and `getImplementationStatus` to view the implementation status at the MATLAB command line without running the analysis in the **Requirements Editor** or Requirements Perspective.

View the Implementation Status

You can view the implementation status for your requirement sets from both the **Requirements Editor** and the Requirements Browser in the Requirements Perspective View. To toggle the status display in the **Requirements Editor**, select  **Columns > Implementation Status**. In the **Requirements Editor** or the Requirements Browser, point to the **Implemented** column for each requirement or requirement set to view the implementation status associated with it.



The screenshot shows the Requirements Editor interface. The top ribbon includes tabs for REQUIREMENTS, LINKS, and VIEW. The REQUIREMENTS tab is active, showing a ribbon with options like New Requirement Set, Open, Save, Import, Close, Add Requirement, Delete, Promote Requirement, Demote Requirement, Add Link, and Show Requirements. Below the ribbon is a table of requirements with columns for Index, ID, Summary, Implemented, and Verified. The Implemented column contains progress bars indicating the status of each requirement. A tooltip is visible over the 'Implemented' bar for requirement #1, showing 'Implemented: 48, Justified: 4, None: 14, Total: 66'.

Index	ID	Summary	Implemented	Verified
crs_req_func_sp...				
1	#1	Driver Switch Request Handling		
1.1	#2	Switch precedence		
1.2	#3	Avoid repeating commands		
1.3	#4	Long Switch recognition		
1.3.1	#5	Waiting state for Long Increment...		
1.3.2	#6	Waiting state for Long Decremen...		
1.4	#7	Cancel Switch Detection		
1.5	#8	Set Switch Detection		
1.6	#9	Enable Switch Detection		
1.7	#10	Resume Switch Detection		
1.8	#11	Increment Switch Detection		
1.9	#15	Decrement Switch Detection		
2	#19	Cruise Control Mode		
3	#37	Calculate Target Speed and Thro...		
4	#44	System Interface		
5	#71	Justifications		
5.1	#72	Non-functional requirement		

The fullness of the bar indicates how many requirements in a group (including the parent requirement and child requirements) are linked to implementation items. The color indicates the level of implementation:

- **Implemented** (blue): The requirement is linked to an item with an `Implement` type link.
- **Justified** (light blue): The requirement is linked to a justification with an `Implement` type link. For more information, see “Justify Requirements” on page 4-17.
- **None** (colorless): The requirement does not have any `Implement` type links.

See Also

Requirements Editor

More About

- “Review Requirements Verification Status” on page 4-6
- “Justify Requirements” on page 4-17
- “Requirement Types” on page 1-6
- “Link Types” on page 3-34

Review Requirements Verification Status

In this section...

- “Verify Functional Requirements” on page 4-6
- “Run Link Verification Analysis” on page 4-7
- “Display Verification Status” on page 4-7
- “Update Verification Status by Running Tests or Analyses” on page 4-8
- “Include Verification Status in Report” on page 4-9

You can view the verification status of your requirements in the Requirements Browser and **Requirements Editor**. The verification status reflects results from simulation testing using Simulink Test or property proving using Simulink Design Verifier. Use `Verify` type links from requirements to simulation assessments or proof objectives. For more information, see “Link Types” on page 3-34.

Verify Functional Requirements

The requirement type specifies the role of a requirement. Functional requirements as well as requirements with a custom type that is a subtype of the `Functional` type, are meant to be implemented and contribute to the verification status summary. Other requirement types do not contribute to the verification status. For more information, see “Requirement Types” on page 1-6 and “Define Custom Requirement and Link Types by Using `sl_customization` Files” on page 3-42.

You can verify functional requirements by linking them with certain verification items with `Verify` type links.

- **MATLAB testing:** Requirement verification status reflects the results of MATLAB unit tests run in the **Requirements Editor** or tun by using `slreq.ReqSet.runTests`. For more information, see “Verify Requirements with MATLAB Tests” on page 10-4 and “Ways to Write Unit Tests”.

Note Test results from the **Test Browser** and **MATLAB Test Manager** do not affect the verification status in the **Requirements Editor**.

- **Simulation testing:** Requirement verification status reflects the results of the following linkable Simulink Test items after you run these items in the Test Manager:
 - Test files
 - Test suites
 - Test cases
 - Iterations
 - Assessments

To learn how to verify requirements with Simulink Test items, see “Test Model Against Requirements and Report Results” on page 13-2.

Run tests from the Simulink Test Manager, or using `sltest.testmanager.run`. For a brief tutorial on creating and running a test case, follow the first part of “Create and Run a Baseline Test Case” (Simulink Test).

Capture run-time assessments from verify statements or “Model Verification Blocks” (Simulink Test) by monitoring those assessments through test cases in the Test Manager.

If you link a step in Test Sequence block that contains a verify statement to a requirement, then you can visualize the verification status of that corresponding verify statement when:

- **Test Harness:** You simulate the test harness containing the Test Sequence block and open the simulation results in Simulink Data Inspector (SDI)
- **Test Manager:** You run tests in Simulink Test which executes the Test Sequence block with step containing verify statement.

Alternatively, use the **Run Tests** option from the Requirements menu or the `runTests` function to run batch tests for the specified requirements.

If you manually set the link type for a step to be **verified by** when the step does not contain any verify statements, then the verification status will remain **unexecuted**. Assert statements will not be marked as **verified by** in the Requirements Editor. For more information, see Assess Model Simulation Using verify Statements (Simulink Test).

- **Property proving:** Verification status reflects the analysis results of properties modeled using:
 - Simulink Design Verifier Proof Objective blocks
 - Model Verification blocks


Link blocks to requirements, then analyze the properties. For more information, see “Validate Requirements by Analyzing Model Properties” on page 4-10.

You can also verify requirements by linking to external result sources with Confirm type links. For more information, see “Include Results from External Sources in Verification Status” on page 4-28

Run Link Verification Analysis

Requirements Toolbox does not perform link verification analysis until you run it. You can run the analysis in **Requirements Editor** or Requirements Perspective.

A banner in the **Requirements Editor** or Requirements Perspective indicates when results are pending. To run the analysis, click **Analyze now** in the banner. Alternatively, click **Refresh** in the


Requirements Editor or the refresh button  in the Requirements Perspective.

Link verification analysis continuously runs in the background until you use `slreq.clear`.

Alternatively, you can use `updateVerificationStatus` and `getVerificationStatus` to view the verification status at the MATLAB command line without running the analysis in the **Requirements Editor** or Requirements Perspective. You can also use `slreq.ReqSet.runTests` to run tests linked to requirements in the requirement set and get the verification status at the command line.

Display Verification Status

The verification status is summarized in the **Verified** column of the Requirements Browser and **Requirements Editor**. To display the column:

- In the **Requirements Editor**, select  **Columns > Verification Status**
- In the Requirements Browser pane of the model window, right-click a requirement and select **Verification Status**.

For example, the **Verified** column shows partial verification links for this requirement set, with one failed result.

Index	ID	Summary	Implemented	Verified
crs_req_func_spec*			[Full Blue Bar]	[Partial Bar: Green, Red, Yellow]
1	#1	Driver Switch Request Handling	[Full Blue Bar]	[Partial Bar: Green, Red, Yellow]
1.1	#2	Switch precedence	[Full Blue Bar]	[None]
1.2	#3	Avoid repeating commands	[Full Blue Bar]	[Full Green Bar]
1.3	#4	Long Switch recognition	[Full Blue Bar]	[None]
1.4	#7	Cancel Switch Detection	[Full Blue Bar]	[Full Red Bar]
1.5	#8	Set Switch Detection	[Full Blue Bar]	[Full Yellow Bar]
1.6	#9	Enable Switch Detection	[Full Blue Bar]	[Full Green Bar]
1.7	#10	Resume Switch Detection	[Full Blue Bar]	[None]
1.8	#11	Increment Switch Detection	[Full Blue Bar]	[Partial Green Bar]
1.9	#15	Decrement Switch Detection	[Full Blue Bar]	[Partial Green Bar]
2	#19	Cruise Control Mode	[Full Blue Bar]	[None]
2.1	#20	Disable Cruise Control system	[Full Blue Bar]	[None]
2.2	#24	Operation mode determination	[Full Blue Bar]	[None]
3	#37	Calculate Target Speed and Thr...	[Full Blue Bar]	[None]
3.1	#38	Disabled case	[Full Blue Bar]	[None]
3.2	#39	Enabled case	[Full Blue Bar]	[None]

The fullness of the bar indicates how many requirements in a group including the parent requirement and any children requirements are linked to verification items. The color indicates the test or analysis results:

- **Passed** (green): The linked test(s) passed, or the analysis proved the objective(s).
- **Failed** (red): The linked test(s) failed, or the analysis falsified the objective(s).
- **Justified** (light blue): The requirement is excluded from the status with a justification. For more information, see “Justify Requirements” on page 4-17.
- **Unexecuted:** (yellow): The linked test(s) or objective(s):
 - Have not run or executed
 - Have been updated more recently than the most recent result
- **None** (colorless): The requirement does not have `Verify` type links.

Update Verification Status by Running Tests or Analyses

You can update the verification status by running tests or analyses linked to your requirements.

- 1 In the **Requirements Editor**, right-click the requirement and select **Run Tests**.
- 2 In the Run Tests dialog box, select the tests.
- 3 Click **Run Tests**.

You can also update verification status by running tests or analysis outside of the **Requirements Editor**.

- In Simulink Test, run the tests in the Test Manager.
- In Simulink Design Verifier, run property proving analysis.
- In Simulink, run the model that contains the Model Verification blocks.

Note If you have linked requirements to Simulink Design Verifier Proof Objective blocks in multiple models, the **Run Tests** dialog box runs a Simulink Design Verifier analysis when the corresponding models are open.

Include Verification Status in Report

You can include verification status in your requirements report.

- 1 In the **Requirements Editor** menu, select **Report > Generate Report**.
- 2 Select **Verification Status**.
- 3 Click **Generate Report**.

For more information, see “Report Requirements Information” on page 5-12.

See Also

Requirements Editor

Related Examples

- “Validate Requirements by Analyzing Model Properties” on page 4-10

More About

- “Review Requirements Implementation Status” on page 4-2
- “Link Test Cases to Requirements”

Validate Requirements by Analyzing Model Properties

Validate a requirement set by analyzing properties that model individual requirements. Falsified properties indicate design and requirement set incompleteness.

Overview

In this example, you analyze model properties that are based on four requirements of an engine thrust reverser system. Falsified results from the property analysis suggest that the system design requirements are incomplete -- the system allows behavior that violates several of the following requirements:

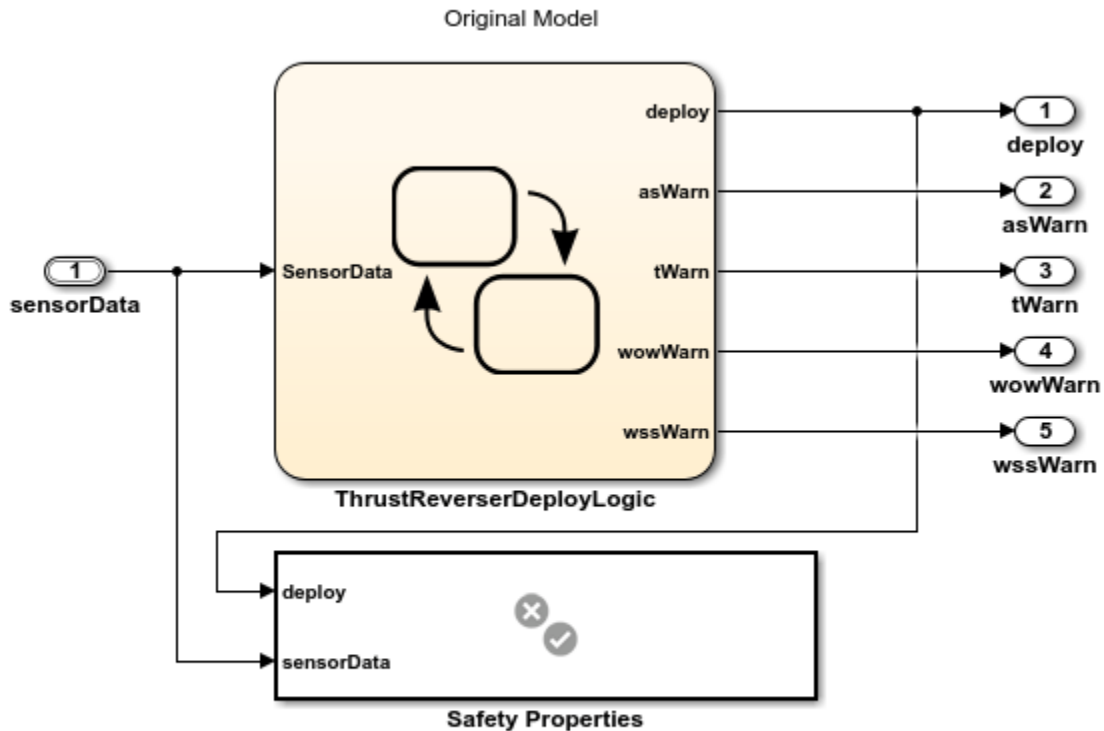
- 1 The thrust reverser shall not deploy if the airspeed is greater than 150 knots.
- 2 The thrust reverser shall not deploy if the aircraft is in the air, as indicated by the value of the weight on wheels sensors. If the aircraft is in the air, the signal value for each of two weight on wheels (WOW) sensors is `false`.
- 3 The thrust reverser shall not deploy if the value of either thrust sensor is positive.
- 4 The thrust reverser shall not deploy if the rotational speed of the landing gear wheels is less than a threshold value.

To better understand the model behavior, you analyze dependencies for a time series input that causes undesirable model behavior because the system lacks required control logic. Then, you analyze a revised control system model which passes the property analysis.

Analyze the Safety Properties

1. Click the **Open Model** button to open the original model and create a working directory of the example files.

Requirements Validation Using Property Analysis Example



Copyright 2019-2020 The MathWorks, Inc.

The Safety Properties subsystem is a Verification Subsystem block from the Simulink® Design Verifier™ library. The verification logic in Safety Properties models the safety requirements. For example, the airspeed requirement is verified by:



If average airspeed > 150 knots, deploy cannot be true.

For more information about Verification Subsystem blocks, see Verification Subsystem (Simulink Design Verifier).

2. View the requirements. In the model, click the **Show Perspectives views** icon at the lower right and select **Requirements**. The **Requirements** pane opens. Expand **thrust_reverser_safety_requirements**.

The safety requirements link to the Assertion blocks in the Safety Properties subsystem. The Assertion blocks are considered proof objectives. The verification status for each requirement reflects the property analysis results of its corresponding Assertion block.

3. Display the verification status for the requirements. Right-click one of the requirements in the browser and select **Verification Status**. The **Verified** column indicates that the requirements are unexecuted.

Index	ID	Summary	Verified
▼ thrust_reverser_safety_requirements			
1	R1.1	Airspeed Condition	
2	R1.2	WOW Condition	
3	R1.3	Throttle Condition	
4	R1.4	Wheelspeed Condition	

4. Analyze the model properties. In the **Apps** tab, click **Design Verifier**. In the **Design Verifier** tab, click **Prove Properties**.

When the property analysis completes, click the **Refresh** button to update the status in the **Verified** column. The results show that three out of four objectives are falsified -- analysis found a signal condition that falsifies the properties, and therefore violates the requirements.

Index	ID	Summary	Verified
▼ thrust_reverser_safety_requirements			
1	R1.1	Airspeed Condition	
2	R1.2	WOW Condition	
3	R1.3	Throttle Condition	
4	R1.4	Wheelspeed Condition	

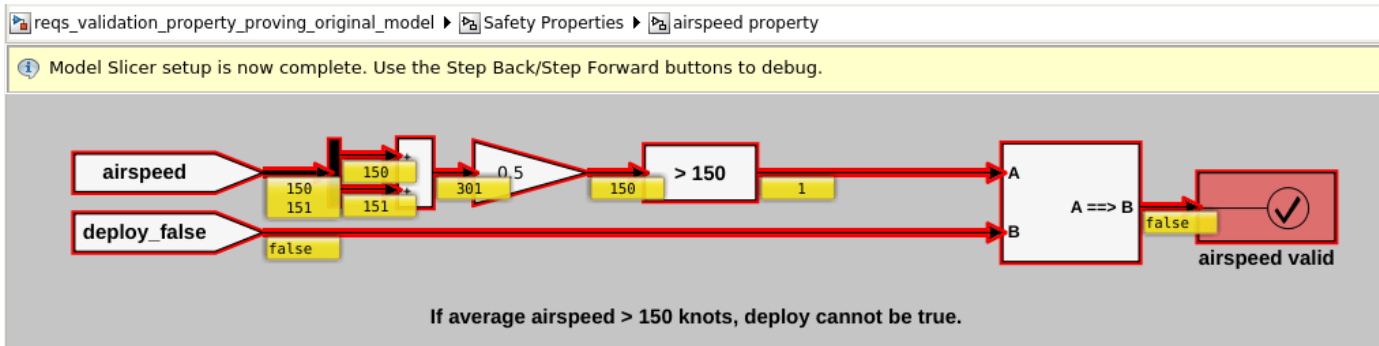
Analyze Model Behavior with Counterexamples

From the Design Verifier Results Summary window, click **HTML** to open the detailed analysis report. In Chapter 4, each falsified property lists a counterexample. For example, in the counterexample that falsifies the airspeed requirement:

- At $t = 0.1$, the thrust reverser is deployed with airspeed below the threshold.
- At $t = 0.2$, the thrust reverser is still deployed with airspeed above the threshold.

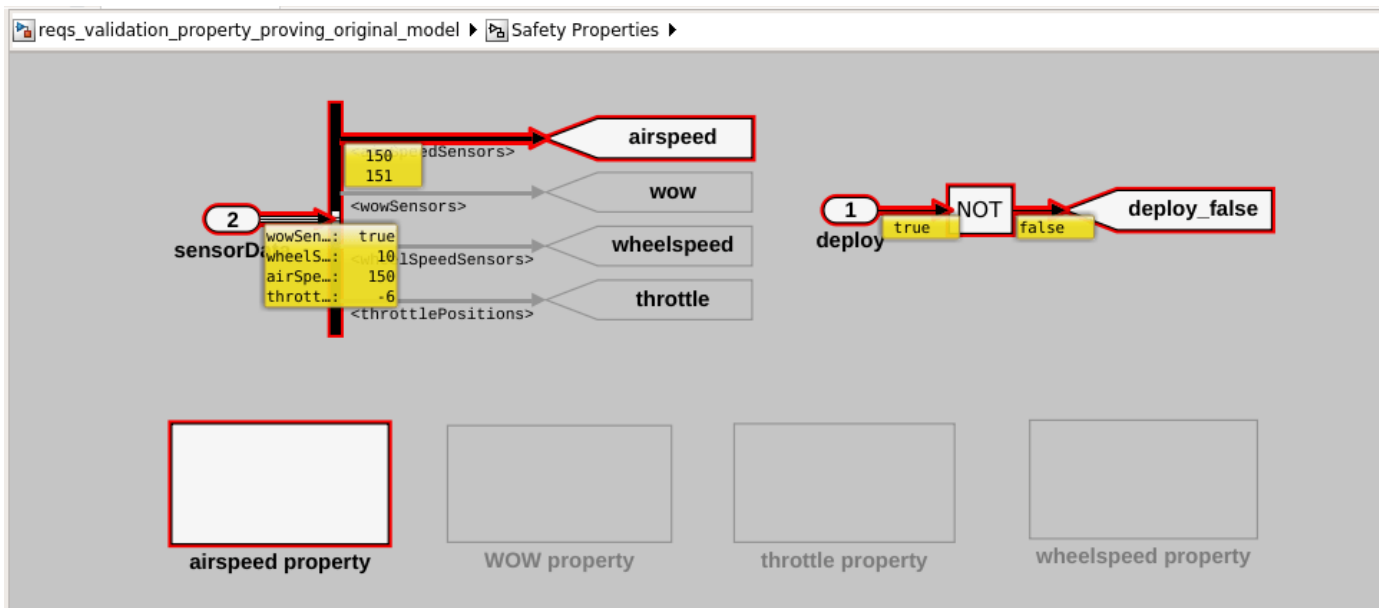
The counterexample time series indicates a condition that might be difficult to encounter in simulation, but nonetheless causes model behavior that violates a requirement. Investigate the behavior by analyzing signal dependencies in the counterexample:

1. In the **Design Verifier** tab, click the **Highlight in Model** button.
2. Select the airspeed valid assertion block in the **Test Unit > Safety Properties > airspeed property** subsystem.
3. In the **Design Verifier** tab, click the **Debug Using Slicer** button. The model highlights dependencies of the airspeed valid assertion, and displays signal values at $T = 0.2$.



4. Move up one level in the model, to the **Safety Properties** subsystem. Step back through the counterexample simulation time. In the **Simulation** tab, click **Step Back**.

5. At T = 0.1, the average airspeed is below the threshold, and the thrust reverser is deployed. Stepping forward, at T = 0.2, the average airspeed is above the threshold, and the thrust reverser is still deployed. This violates a requirement.



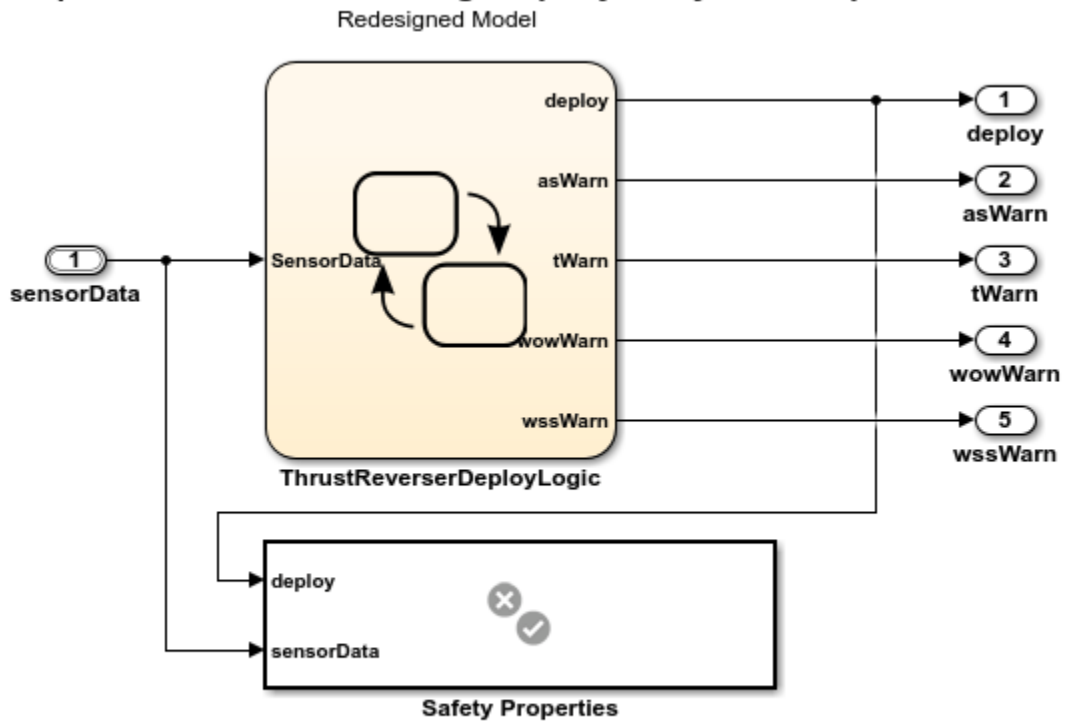
The falsified property and the dependency analysis suggest that the control system algorithm is incompletely designed, and the requirements are incomplete.

Analyze the Redesigned System

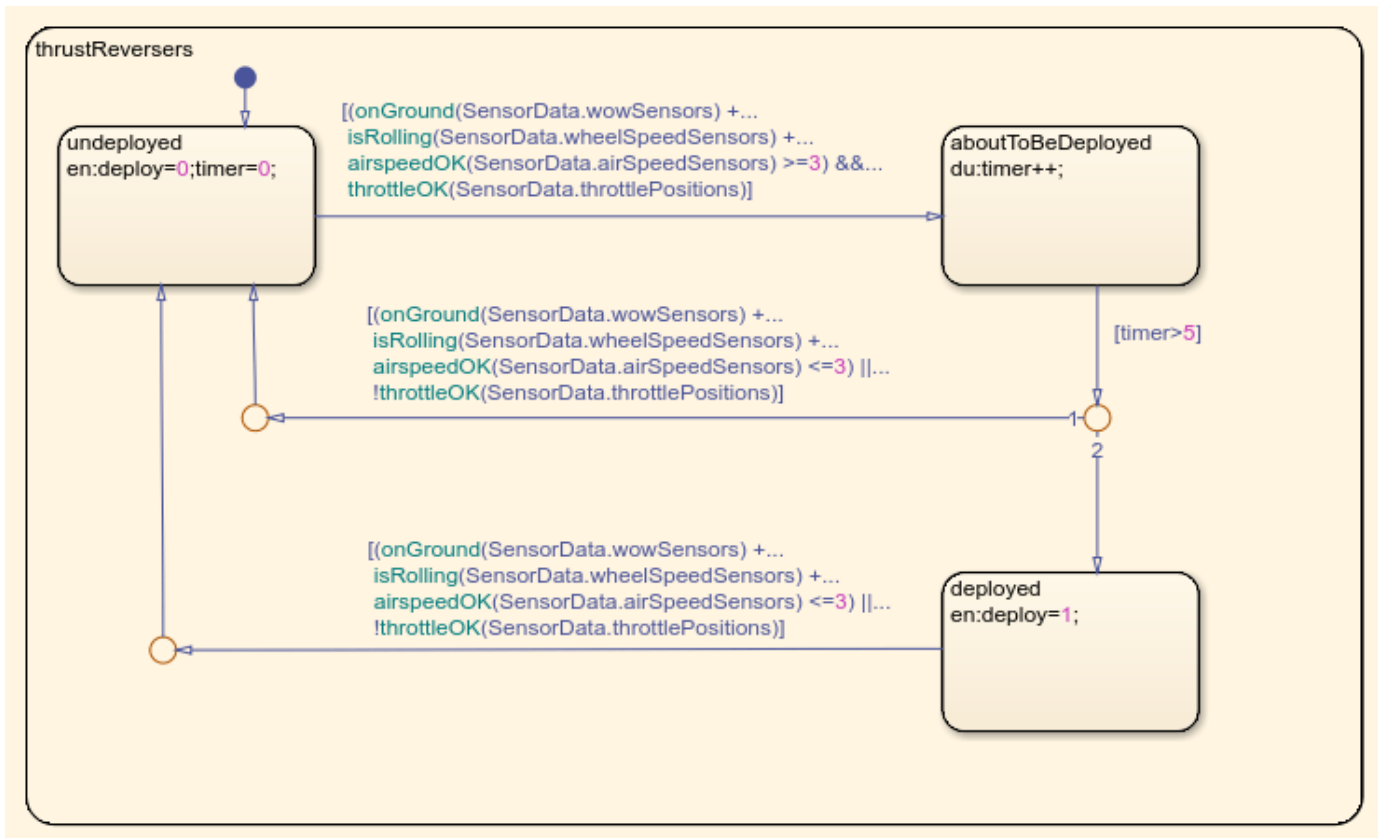
Redesigning a control system requires rethinking requirements. In this case, the lack of an intermediate standby state allows undesirable system behavior when inputs change suddenly. Adding an intermediate deployment mode which delays thrust reverser response resolves the issue.

Open the reqs_validation_property_proving_redesigned_model model. Open the thrustReversers chart.

Requirements Validation Using Property Analysis Example



Copyright 2019-2020 The MathWorks, Inc.



The additional design requirement states that the thrust reverser shall deploy after a pause. The redesigned model includes:

- An additional `aboutToBeDeployed` state.
- Expanded transition conditions that return to `undeployed`.

Create links from the verification blocks in the redesigned model to the requirements:

1. In the model, from the **Apps** tab, click **Requirements Manager**.
2. In the **Requirements** tab, click **Requirements Editor**.
3. Open `thrust_reverser_safety_requirements` in the **Requirements Editor**.
4. For requirement 1.1, Airspeed Condition, link to the airspeed valid block in the Safety Properties > airspeed property subsystem. Drag R1.1 from the requirements browser to the airspeed valid block in the model.
5. The new link appears in the Requirements Editor, in the right pane, under **Links**.
6. Delete the link to the assert block in the original model. If the original model is closed, this link appears unresolved. Next to the link, click the **Delete Link** icon.

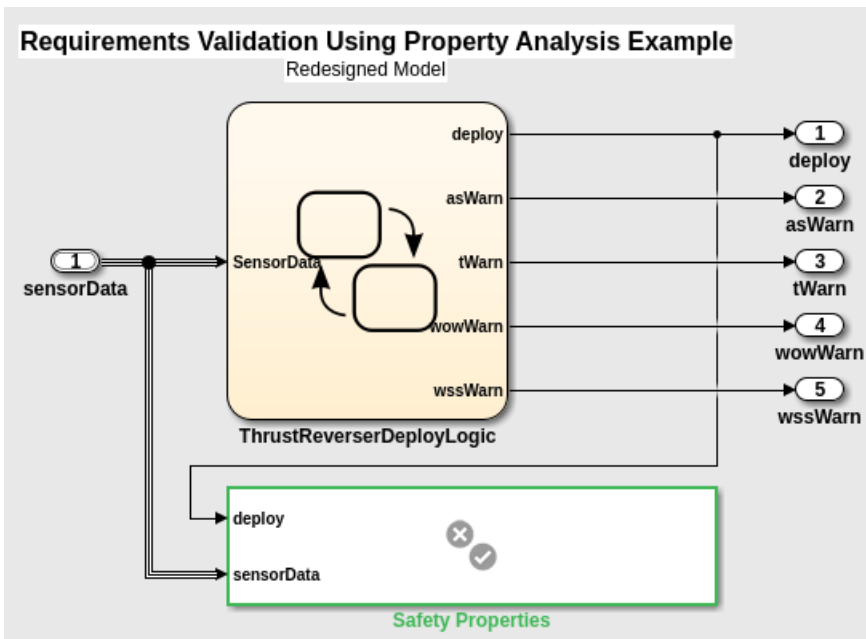
▼ Links

Verified by:

- reqs_validation_property_proving_original_model:5:29
- airspeed valid

Delete Link

7. Repeat for the other three requirements and verification blocks in the redesigned model. Run the property analysis on the revised model. View the results in the Requirements pane.



Index	ID	Summary	Verified
thrust_reverser_safety_requirements			
1	R1.1	Airspeed Condition	Verified
2	R1.2	WOW Condition	Verified
3	R1.3	Throttle Condition	Verified
4	R1.4	Wheelspeed Condition	Verified

The results show that the properties are valid.

See Also

More About

- “What Is Property Proving?” (Simulink Design Verifier)
- “Review Requirements Verification Status” on page 4-6

Justify Requirements

Use requirement justifications to exclude requirements from the implementation and verification status for your requirement sets. Functional requirements contribute to the implementation and verification status for the requirement set. For more information, see “Requirement Types” on page 1-6.


You might have functional requirements in your model design specification that cannot be implemented in your design. You might also have requirements that require manual testing, instead of linking to test cases or verification subsystems. You can justify these requirements to override their implementation and verification statuses and iterate more effectively on your model design.

A justification is an object associated with a requirement. All justification objects in a requirement set are grouped under a single top-level justification object as its children. Any requirement can be justified for implementation, verification, or both. Justified requirements do not contribute to the overall aggregation of implementation and verification status and appear light blue in the **Implemented** and **Verified** columns of the **Requirements Editor**.

The screenshot shows the Requirements Editor interface. The top menu bar includes options like New Requirement Set, Open, Save, Import, Close, Add Requirement, Delete, Promote Requirement, Demote Requirement, Add Link, and Show Requirements. Below the menu is a table of requirements with columns for Index, ID, Summary, Implemented, and Verified. A tooltip is visible over the 'Implemented' progress bar for requirement #1, showing 'Implemented: 48, Justified: 4, None: 14, Total: 66'.

Index	ID	Summary	Implemented	Verified
crs_req_func_sp...				
1	#1	Driver Switch Request Handling	Implemented: 48, Justified: 4, None: 14, Total: 66	
1.1	#2	Switch precedence		
1.2	#3	Avoid repeating commands		
1.3	#4	Long Switch recognition		
1.3.1	#5	Waiting state for Long Increment...		
1.3.2	#6	Waiting state for Long Decremen...		
1.4	#7	Cancel Switch Detection		
1.5	#8	Set Switch Detection		
1.6	#9	Enable Switch Detection		
1.7	#10	Resume Switch Detection		
1.8	#11	Increment Switch Detection		
1.9	#15	Decrement Switch Detection		
2	#19	Cruise Control Mode		
3	#37	Calculate Target Speed and Thro...		
4	#44	System Interface		
5	#71	Justifications		
5.1	#72	Non-functional requirement		

There are two workflows for justifying requirements in Requirements Toolbox. You can create a justification object, create a link to an existing justification, or create a link to a new justification in one step.

- Create a justification object by clicking **Add Requirement > Add Justification** in the **Requirements Editor** or the  icon in the Requirements Browser.
- Link to an existing justification by selecting it in the **Requirements Editor** or Requirements Browser by right-clicking it and selecting **Select for Linking with Requirement**. Then, right-click the requirement and select **Create a Link From....** By default, the link has **Type** set to **Implements**.

- In the **Requirements Editor**, create a link to a new justification by right-clicking the requirement and selecting **Justification > Link with new Justification for implementation** or **Link with new Justification for verification**.

To justify a parent requirement and all its child requirements, select the Hierarchical Justification option in the right pane of the **Requirements Editor**.

Note You cannot link justification objects to objects that are not requirements.

See Also

More About

- “Review Requirements Implementation Status” on page 4-2
- “Review Requirements Verification Status” on page 4-6

Linking to a Test Script

In this workflow, you link a requirement to a MATLAB script using the “Outgoing Links Editor” on page 11-6 and the API. The verification status in the **Requirements Editor** reflects the test results. These illustrations follow the workflow for including external test results in the requirement verification status. For more information, see “Include Results from External Sources in Verification Status” on page 4-28.

Linking to a Test Script Using the Outgoing Links Editor

Create a requirement set called `counter_req.slreqx` in the **Requirements Editor** and save it in a writable location. This requirement set has child requirements that have requirement IDs and descriptions. For more details on how to create requirement sets, see “Work with Requirements in the Requirements Editor”.

Index	ID	Summary
▼ counter_req		
▼ 1	#1	Counter
1.1	#2	Counter default
1.2	#3	Counter increment
1.3	#4	Counter Set

You have a MATLAB script called `runmytests.m` that runs a test for the `Counter` class in `Counter.m`. The test script contains custom methods that write results a TAP format to a file named `results.tap`. Assume that you have run the test and it has produced the `results.tap` file that contains the results of the test. You want to link the results of the test to a requirement in `counter_req.slreqx`. Follow these steps to create and view the verification status with a test case called `counterStartsAtZero` in `runmytests.m` script:

- “Create the Register the Link Type” on page 4-20
- “Create the Link” on page 4-21
- “View the Verification Status” on page 4-22

Create the Register the Link Type

Open the template file at `matlabroot/toolbox/slrequirements/linktype_examples/linktype_TEMPLATE.m`. Follow these steps:

- 1 Create a new MATLAB file.
- 2 Copy the contents of `linktype_TEMPLATE` into the new file. Save the file as `linktype_mymscripttap.m`.
- 3 In `linktype_mymscripttap.m`
 - a Replace the function name `linktype_TEMPLATE` with `linktype_mymscripttap.m`.
 - b Set `linkType.Label` as 'MScript TAP Results'.
 - c Set `linkType.Extensions` as `{'.M'}`.

- d Uncomment the command for `GetResultFcn` in order to use it in `linktype_mymscripttap` and enter:

```
linktype.GetResultFcn = @GetResultFcn;
.....
function result = GetResultFcn(link)
    testID = link.destination.id;
    testFile = link.destination.artifact;
    resultFile = getResultFile(testFile);

    if ~isempty(resultFile) && isfile(resultFile)
        tapService = slreq.verification.services.TAP();
        result = tapService.getResult(testID, resultFile);
    else
        result.status = slreq.verification.Status.Unknown;
    end

end

function resultfile = getResultFile(testFile)
    resultMap = ["runmytests.m", "results.tap";...
                "othertests.m", "results2.tap"];
    resultfile = resultMap(resultMap(:,1) == testFile,2);
end
```

`GetResultFcn` uses the utility `slreq.verification.services.TAP` to interpret the result files for verification. See `slreq.verification.services.TAP` for more details. For more information about `GetResultFcn`, see “Links and Link Types” on page 11-2.

- 4 Save `linktype_mymscripttap.m`.
 5 Register the link type. At the command line, enter:

```
rmi register linktype_mymscripttap
```

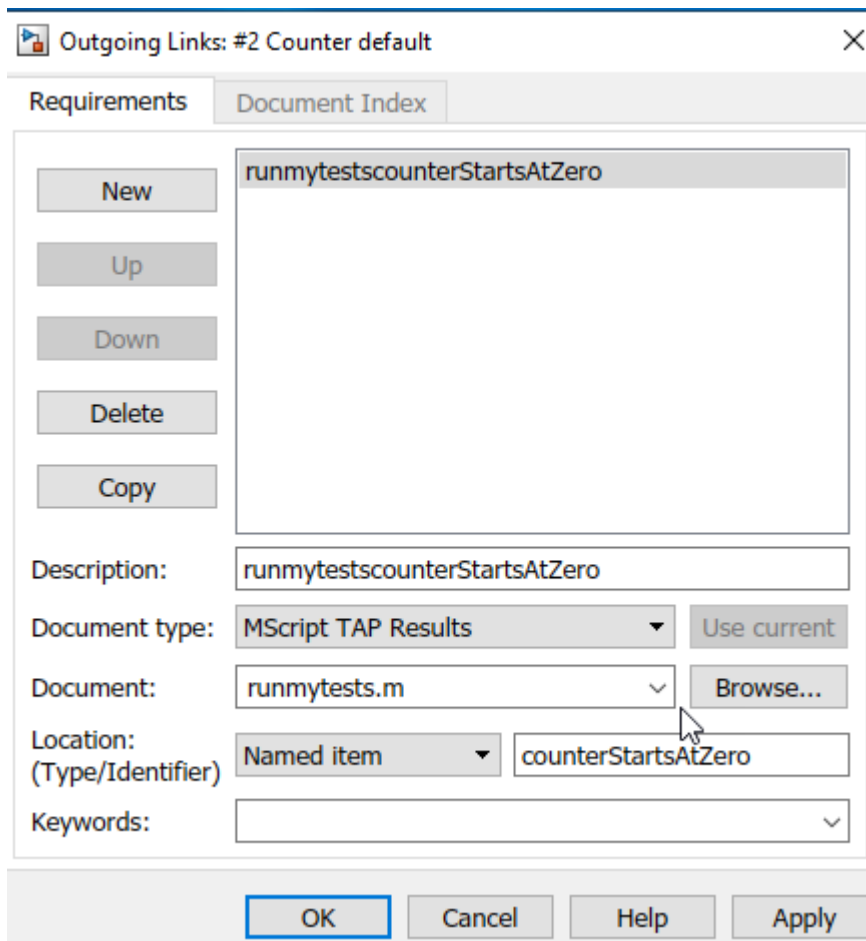
Note If the command returns a warning, then you must unregister the file and follow step 5 again. Unregister the file by entering:

```
rmi unregister linktype_mymscripttap
```

Create the Link

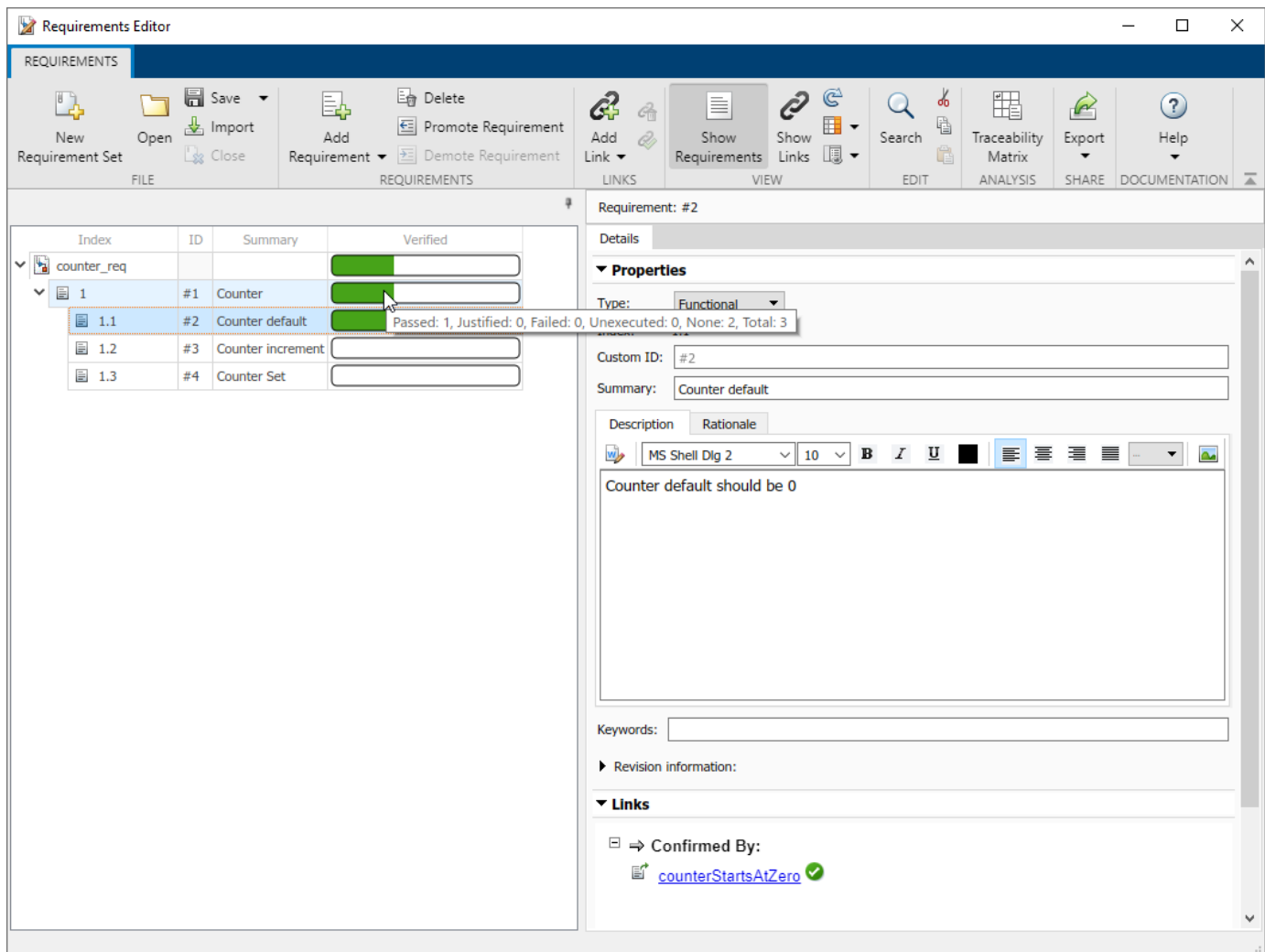
Follow these steps to add the link manually in the Outgoing Links Editor:

- 1 Open the **Requirements Editor** and, in the `counter_req.slreqx` requirement set, right-click the child requirement 1.1 and select **Open Outgoing Links dialog**.
- 2 In the Outgoing Links Editor dialog box, in the **Requirements** tab, click **New**.
- 3 Enter these details to establish the link:
 - Description: `runmytestscounterStartsAtZero`
 - Document Type: MScript TAP Results
 - Document: `runmytests.m`
 - Location: `counterStartsAtZero`
- 4 Click **OK**. The link is highlighted in the **Links** section of the **Requirements Editor**.



View the Verification Status

Update the verification status in the **Requirements Editor**. Click  **Refresh** to see the verification status for the requirements in the **Requirements Editor**. This shows the verification status for entire requirement set that passed or failed.



The requirements for counterStartsAtZero are fully verified. Here, the verification status shows that out of three tests, one test passed.

Linking to a Test Script Using the API

Create a requirement set called counter_req.s1reqx in the **Requirements Editor** and save it in a writable location. This requirement set has child requirements that have requirement IDs and descriptions. For more details on how to create requirement sets, see “Work with Requirements in the Requirements Editor”.

Index	ID	Summary
counter_req		
1	#1	Counter
1.1	#2	Counter default
1.2	#3	Counter increment
1.3	#4	Counter Set

You have a MATLAB script called `runmytests.m` that runs a test for `Counter` class in `Counter.m`. The test script contains custom methods that write results in a TAP format to a file named `results.tap`. Assume that you have run the test and it has produced the `results.tap` file that contains the results of the test. You want to link the results of the test to a requirement in `counter_req.slreqx`. Follow these steps to create and view the verification status with a test case called `counterStartsAtZero` in `runmytests.m` script:

- “Create and Register the Link Type” on page 4-24
- “Create the Link” on page 4-25
- “View the Verification Status” on page 4-25

Create and Register the Link Type

Open the template file at `matlabroot/toolbox/slrequirements/linktype_examples/linktype_TEMPLATE.m`. Follow these steps:

- 1 Create a new MATLAB file.
- 2 Copy the contents of `linktype_TEMPLATE` into the new file. Save the file as `linktype_mymscripttap.m`.
- 3 In `linktype_mymscripttap.m`:
 - a Replace the function name `linktype_TEMPLATE` with `linktype_mymscripttap.m`.
 - b Set `linkType.Label` as 'MScript TAP Results'.
 - c Set `linkType.Extensions` as `{'.M'}`.
 - d Uncomment the command for `GetResultFcn` in order to use it in `linktype_mymscripttap` and enter:

```
linktype.GetResultFcn = @GetResultFcn;
.....
function result = GetResultFcn(link)
    testID = link.destination.id;
    testFile = link.destination.artifact;
    resultFile = getResultFile(testFile);

    if ~isempty(resultFile) && isfile(resultFile)
        tapService = slreq.verification.services.TAP();
        result = tapService.getResult(testID, resultFile);
    else
        result.status = slreq.verification.Status.Unknown;
    end
end
```

```
function resultfile = getResultFile(testFile)
    resultMap = ["runmytests.m", "results.tap";...
                "othertests.m", "results2.tap"];
    resultfile = resultMap(resultMap(:,1) == testFile,2);
end
```

GetResultFcn uses the utility `slreq.verification.services.TAP` to interpret the result files for verification. See `slreq.verification.services.TAP` for more details. For more information about GetResultFcn, see “Links and Link Types” on page 11-2.

- 4 Save `linktype_mymscripttap.m`.
- 5 Register the link type. At the command line, enter:

```
rmi register linktype_mymscripttap
```

Note If the command returns a warning, then you must unregister the file and follow step 5 again. Unregister the file by entering:

```
rmi unregister linktype_mymscripttap
```

Create the Link

Follow these steps to create the link:

- 1 From the MATLAB command prompt, enter:

```
externalSource.id = 'counterStartsAtZero';
externalSource.artifact = 'runmytests.m';
externalSource.domain = 'linktype_mymscripttap';
```

- 2 Find the requirement related to the link by typing:

```
requirement = reqSet.find('Type', 'Requirement', 'SID', 2);
```

- 3 Create the link by entering:

```
link = slreq.createLink(requirement, externalSource);
```

This creates the link as test case `counterStartsAtZero` for the requirement `SID`. In **Requirements Editor**, the link appears in the **Links > Confirmed By** section.



View the Verification Status


Update the verification status. At the MATLAB command prompt, type:

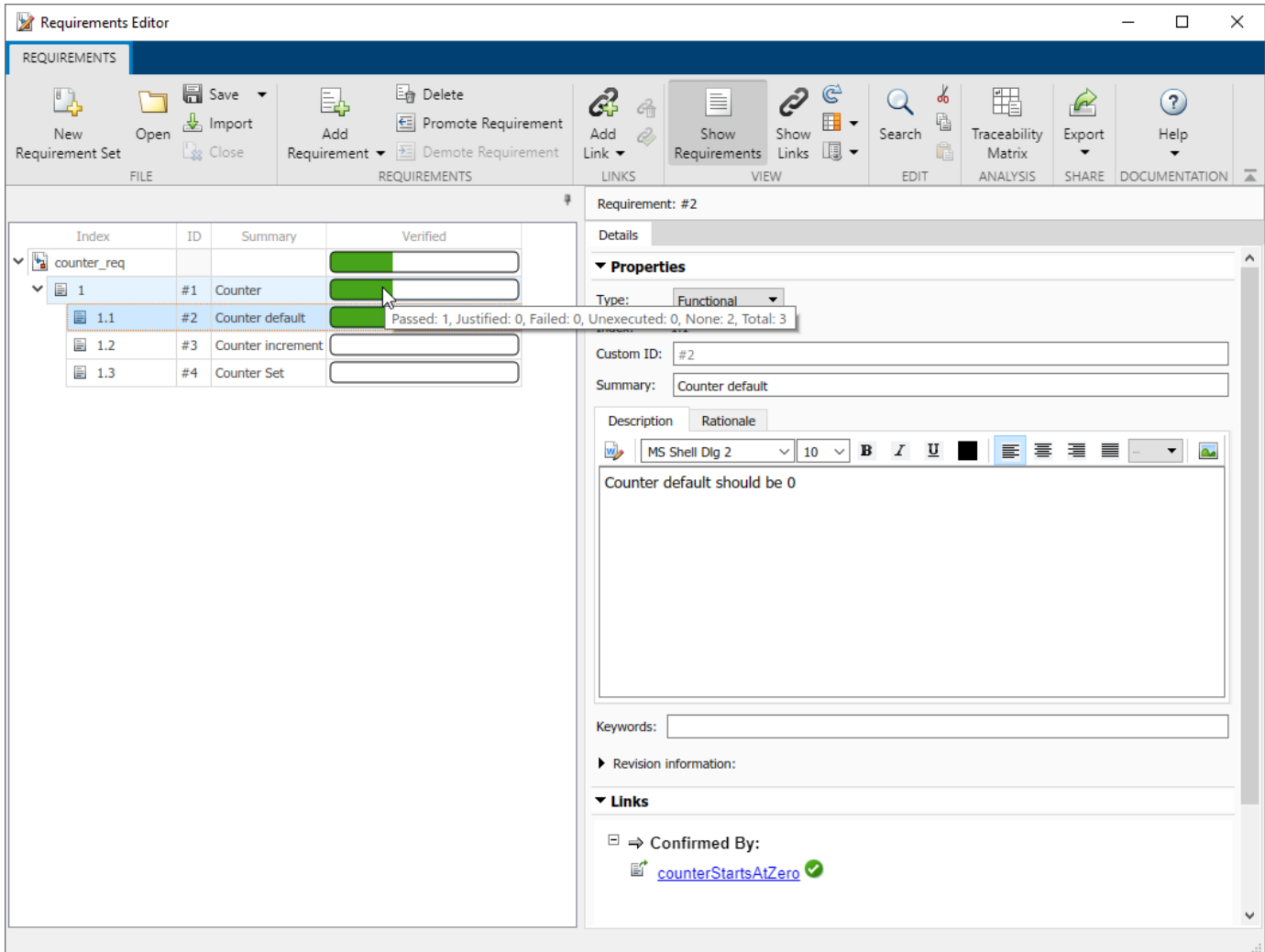
```
reqSet.updateVerificationStatus
```

Fetch the verification status for the requirement by entering :

```
status = reqSet.getVerificationStatus
```

This shows which of the requirements in the requirement set have passed or fail. Click on **Refresh**

 button to see the verification status for the requirements in the **Requirements Editor**.



The screenshot shows the Requirements Editor interface. On the left, a table lists requirements under the 'counter_req' set:

Index	ID	Summary	Verified
1	#1	Counter	
1.1	#2	Counter default	Passed: 1, Justified: 0, Failed: 0, Unexecuted: 0, None: 2, Total: 3
1.2	#3	Counter increment	
1.3	#4	Counter Set	

The right pane shows the details for Requirement #2:

- Details:** Requirement: #2
- Properties:** Type: Functional
- Custom ID:** #2
- Summary:** Counter default
- Description:** Counter default should be 0
- Keywords:**
- Revision information:**
- Links:** Confirmed By: [counterStartsAtZero](#)

The requirements for counterStartsAtZero are fully verified. Here, the verification status shows that out of three tests, one test passed.

Integrating Results from a MATLAB Unit Test Case

You can also integrate the results from a MATLAB Unit Test case by linking to a test script. The test is run with a customized test runner using a XML plugin that produces a JUnit output. The XMLPlugin class creates a plugin that writes test results to an XML file. For more information, see `matlab.unittest.plugins.XMLPlugin.producingJUnitFormat`.

You can register the domain and create the links in the same way as with the test script. The verification status for a set of requirements is shown in the **Requirements Editor**.

The screenshot shows the Requirements Editor interface. On the left, a table lists requirements under the 'counter_req' category:

Index	ID	Summary	Verified
counter_req			
1	#1	Counter	
1.1	#2	Counter default	
1.2	#3	Counter increment	
1.3	#4	Counter Set	

The right pane shows the details for Requirement #2:

- Summary:** Counter default
- Description:** Counter default should be 0
- Keywords:** (empty field)
- Revision information:** (collapsed)
- Links:** Confirmed By: [testCounterStartsAtZero](#) Passed
- Comments:** (collapsed)

See Also

More About

- “Include Results from External Sources in Verification Status” on page 4-28

Include Results from External Sources in Verification Status

Requirements Toolbox allows you to include the verification status of results from external sources in the **Requirements Editor**. You can summarize requirements verification status, author your custom domain registration, and write custom logic to fetch the results. For more information, see “Review Requirements Verification Status” on page 4-6.

You can also include test results from:

- Continuous integration (CI) servers such as Jenkins
- Custom results updated manually or with test scripts

You can create custom link type registrations that interpret test results from the external environment into language specific to your development environment. See, “Custom Link Types” on page 11-8.

You can use built-in verification services to interpret result files for most common cases, such as JUnit and TAP (Test Anything Protocol), to include external test results in the requirements verification status.

When you include the verification status of external test results in your requirements:

- The external results are listed in the **Verified** column of the **Requirements Editor**, along with results from other sources, such as Model Verification blocks and Simulink Test test files.
- Pass/fail indication is reflected in requirement links.
- Result status is automatically aggregated across requirement hierarchies.
- Result status automatically updates as requirements are added or removed.

How to Populate Verification Results from External Sources

Commonly, external test results are run and managed outside of the MATLAB environment. Test results can be the product of:

- Running test scripts or other programs that generate a result file
- Running a MATLAB Unit Test test case with a custom TestRunner object, with or without a CI server

You can create links to the test results by either:


- Linking directly to a result file. The external result artifact is used as the link destination and the requirements are used as the links source. To create custom link type, you must know:

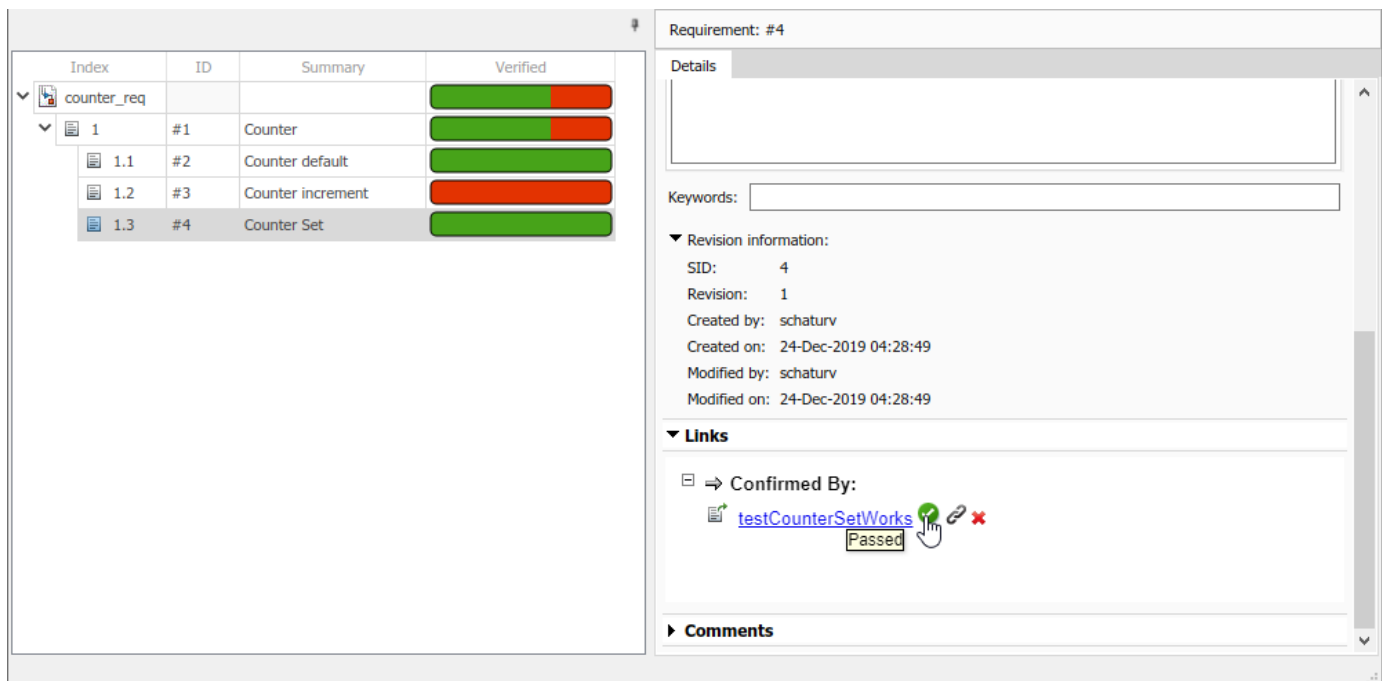
- 1 The file location
- 2 The file format (for example, JUnit or Excel)

For details, see “Linking to a Result File” on page 4-31.

- Linking to a test script and providing code that fetches results based on that test location. The external test artifacts are used as the link destination and the requirements are used as the link source. Your custom logic in the `GetResultFcn` function should locate the result artifact that corresponds to the test artifact and fetch results from that result artifact. See “Linking to a Test Script” on page 4-20.

The following steps are used to create the links from external sources and populate verification statuses from them:

- 1 **Create a custom link type:** In the Requirements Management Interface (RMI), create a custom link type for your test result file:
 - a Write a MATLAB function that implements the custom link type. The `GetResultFcn` is implemented in the custom link type. For more information, see “Links and Link Types” on page 11-2.
 - b Save the function on the MATLAB path.
- For details, see “Custom Link Type Registration” on page 11-15.
- 2 **Register the custom link type:** See “Custom Link Type Registration” on page 11-15. After registration, the link type is available in the Outgoing Links Editor in the **Document type** menu.
 - 3 **Link from the requirement to the test result file or test script:** Use the Outgoing Links Editor or `slreq.createLink` to link from the requirements to the results file.
 - 4 **Display the verification status:** In the Requirements Editor, view the **Verified** column to view the verification status. For details, see “Display Verification Status” on page 4-7.
 - 5 **Refresh the requirements view:** After the tests run, refresh the verification status by clicking the **Refresh**  button.



The screenshot displays the Requirements Management Interface (RMI) with a table of requirements and a detailed view of requirement #4.

Index	ID	Summary	Verified
counter_req			
1	#1	Counter	
1.1	#2	Counter default	
1.2	#3	Counter increment	
1.3	#4	Counter Set	


The detailed view for Requirement: #4 shows the following information:

- Details:** A text area for additional information.
- Keywords:** A text input field.
- Revision information:**
 - SID: 4
 - Revision: 1
 - Created by: schaturv
 - Created on: 24-Dec-2019 04:28:49
 - Modified by: schaturv
 - Modified on: 24-Dec-2019 04:28:49
- Links:**
 - Confirmed By: [testCounterSetWorks](#)
 - Passed
- Comments:** A section for adding comments.

You can include the verification status from external sources in your requirements report by clicking **Report > Generate Report** from the Requirements Editor.

When populating verification results from external sources:

- Test the `GetResultFcn` code before integrating the code with `rmi register`. For more information about `GetResultFcn`, see “Links and Link Types” on page 11-2.

- Confirm the custom link type registration in the **Outgoing Links Editor**.
- Use caching to improve the performance for cases where a single file contains a result for many links.
- Insert break points into the `GetResultFcn` code and use the **Refresh**  button to re-execute it.
- When using Projects, register and unregister the custom link type when using in project startup or shutdown scripts.

See Also

Related Examples

- “Integrating Results from a Custom-Authored MATLAB Script as a Test” on page 4-37
- “Integrating Results from an External Result File” on page 4-41
- “Integrating Results from a Custom Authored MUnit Script as a Test” on page 4-45

More About

- “Linking to a Test Script” on page 4-20
- “Linking to a Result File” on page 4-31

Linking to a Result File

You can link a requirement to a test result file that is in Microsoft Excel format using the “Outgoing Links Editor” on page 11-6 and the API. The verification status in the **Requirements Editor** reflects the test results. These illustrations follow the workflow for including external test results in the requirement verification status. For more information, see “Include Results from External Sources in Verification Status” on page 4-28.

Open Example Files

Open the “Integrating Results from an External Result File” on page 4-41 example.

```
openExample(['\requirements/' ...
  'IntegratingResultsFromAnExternalResultFileExample'])
```

Open the counter_req requirement set in the **Requirements Editor**. This requirement set has child requirements that have requirement IDs and descriptions. For more details on how to create requirement sets, see “Work with Requirements in the Requirements Editor”.

Index	ID	Summary
counter_req		
1	#1	Counter
1.1	#2	Counter default
1.2	#3	Counter increment
1.3	#4	Counter Set

The external test results are contained in an Excel file called `results.xlsx`. The verification status in Requirements Toolbox updates based on the values of the cells in the Excel sheet. A unique ID in the **Test** column identifies each result in the **Status** column. The **Test** and **Status** labels are contained in a header row.

	A	B
	results	
	Test	Status
	Text	Text
1	Test	Status
2	counterStartsAtZero	passed
3	counterIncrements	failed
4	counterSetsValue	passed

Create and Register a Custom Link Type

Before creating the links to the external result file, first create and register a custom link type.

Open the template file at `matlabroot/toolbox/slrequirements/linktype_examples/linktype_TEMPLATE.m`. Follow these steps:

- 1 Create a new MATLAB file.
- 2 Copy the contents of `linktype_TEMPLATE` into the new file. Save the file as `linktype_myexternalresults.m`.
- 3 In `linktype_myexternalresults.m`:
 - a Replace the function name `linktype_TEMPLATE` with `linktype_myexternalresults`.
 - b Set `linkType.Label` as 'Excel Results'.
 - c Set `linkType.Extensions` as `{'.xlsx'}`.
 - d Uncomment the command for `GetResultFcn` in order to use it in `linktype_myexternalresults` and enter:

```
linktype.GetResultFcn = @GetResultFcn;
.....
function result = GetResultFcn(link)
    testID = link.destination.id;
    if testID(1) == '@'
        testID(1) = [];
    end
    resultFile = link.destination.artifact;

    if ~isempty(resultFile) && isfile(resultFile)
        resultTable = readtable(resultFile);
        testRow = strcmp(resultTable.Test, testID);
        status = resultTable.Status(testRow);

        if status{1} == "passed"
            result.status = slreq.verification.Status.Pass;
        elseif status{1} == "failed"
            result.status = slreq.verification.Status.Fail;
        else
            result.status = slreq.verification.Status.Unknown;
        end
    else
        result.status = slreq.verification.Status.Unknown;
    end
end
```

For more information about `GetResultFcn`, see “Links and Link Types” on page 11-2.

- 4 Save `linktype_myexternalresults.m`.
- 5 Register the link type. At the command line, enter:

```
rmi register linktype_myexternalresults
```

Note If the command returns a warning, then you must unregister the link type and register it again. Unregister the link type by entering:

```
rmi unregister linktype_myexternalresults
```

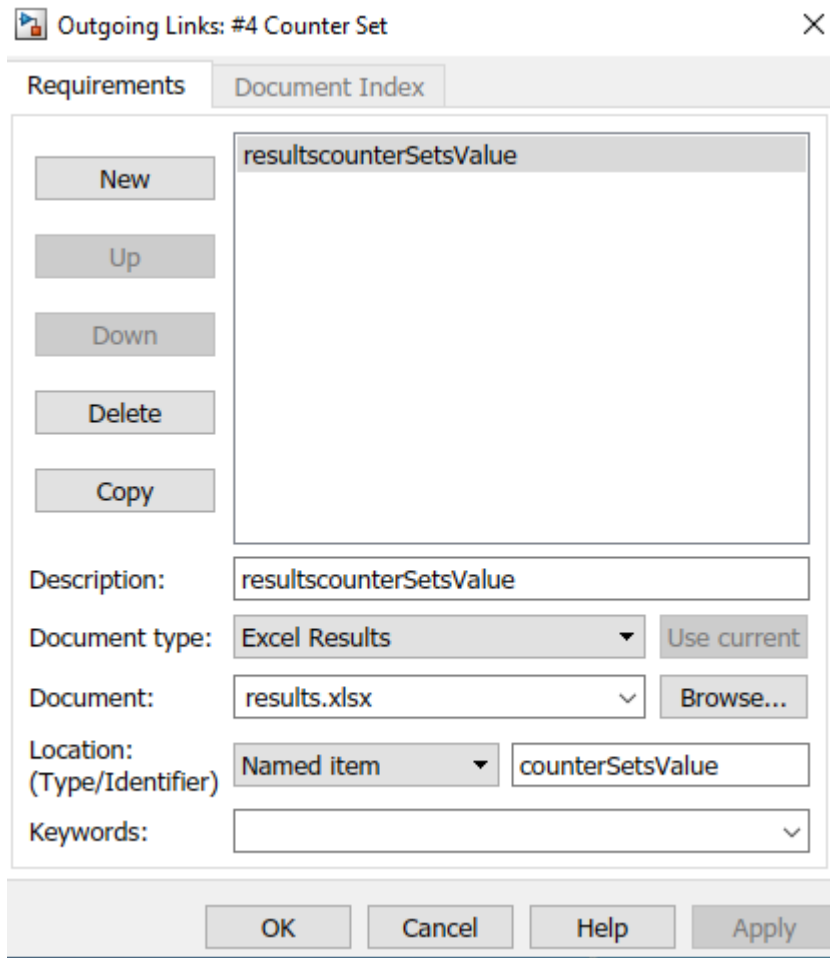
Create a Requirement Link

You can create a link from a requirement to a test result for a test case from the external result file to confirm the requirement. You can create the link by using the Outgoing Links Editor, or by using the Requirements Toolbox API.

Create a Link by Using the Outgoing Links Editor

Create the link from a requirement to the external results file by using the Outgoing Links Editor:

- 1 Open the **Requirements Editor** and, in the `counter_req.slreqx` requirement set, right-click the child requirement 1.3 and select **Open Outgoing Links dialog**.
- 2 In the Outgoing Links Editor dialog box, in the **Requirements** tab, click **New**.
- 3 Enter these details to establish the link:
 - **Description:** resultcounterSetsValue
 - **Document type:** Excel Results
 - **Document:** results.xlsx
 - **Location:** counterSetsValue



- 4 Click **OK**. The link is highlighted in the **Links** section of the **Requirements Editor**.

Create a Link by Using the API

Create the link from a requirement to the external results file by using the API:

- 1 From the MATLAB command prompt, enter:

```
externalSource.id = 'counterSetsValue';
externalSource.artifact = 'results.xlsx';
externalSource.domain = 'linktype_myexternalresults';
```

- 2 Open the requirement set and find the requirement related to the link:

```
reqSet = slreq.open('counter_req.slmx');
requirement = find(reqSet, 'Type', 'Requirement', 'SID', 4);
```

- 3 Create the link by entering:



```
link = slreq.createLink(requirement, externalSource);
```

This creates the link from the requirement with SID 4 to the result for the test case in the external result file called counterSetsValue. In the **Requirements Editor**, the link appears in the **Links > Confirmed By** section.



View the Verification Status

Update the verification information for the counterSetsValue test case based on the Excel status log by updating the verification status for the requirement set.

You can update the verification status in the **Requirements Editor** by clicking **Refresh** . Ensure that **Columns** +  > **Verification Status** is selected to view the verification status for entire requirement set.

Index	ID	Summary	Verified
counter_req			<input type="checkbox"/>
1	#1	Counter	<input type="checkbox"/>
1.1	#2	Counter default	<input type="checkbox"/>
1.2	#3	Counter increment	<input type="checkbox"/>
1.3	#4	Counter Set	<input checked="" type="checkbox"/>

Requirement: #4

Details

Summary: Counter Set

Description Rationale



Arial 10 B I U

Keywords:

Revision information:

Links

Confirmed By:

counterSetsValue  

Passed

Comments

The verification status shows that one of the three requirements is verified.

You can also update the verification status and fetch the current status by entering the following at the MATLAB command prompt:

```
updateVerificationStatus(reqSet)
status = getVerificationStatus(reqSet)
```

See Also

More About

- “Include Results from External Sources in Verification Status” on page 4-28

Integrating Results from a Custom-Authored MATLAB Script as a Test

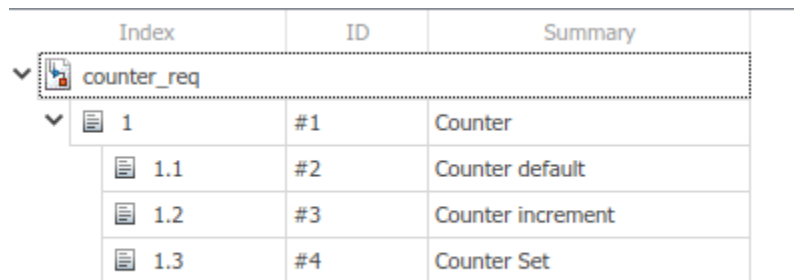
In this example, you link a requirement to a MATLAB® script. The verification status in the Requirements Editor reflects the test results. This example performs the steps described in “Linking to a Test Script” on page 4-20. To run this example, click **Open Example** and run it. This example uses:

- A requirements set file named `counter_req.slreqx`.
- A MATLAB script called `runmytests.m` that runs a test for the Counter class in `Counter.m`. The test script contains custom methods that write results a TAP format to a file named `results.tap`.

Register the Link Type

Before creating the links, you need to register the link type from the requirements set file. Open the requirements file `counter_req.slreqx` in the Requirements Editor:

```
reqSet = slreq.open('counter_req.slreqx');
```



Index	ID	Summary
counter_req		
1	#1	Counter
1.1	#2	Counter default
1.2	#3	Counter increment
1.3	#4	Counter Set

Register the link type that is specific to the external test file. The domain registration needed for this example is `linktype_mymscripttap.m`. To register the custom link type `linktype_mymscripttap.m`, type:

```
rmi register linktype_mymscripttap;
```

The custom logic in the `GetResultFcn` function locates the test file that corresponds to the test case and fetches the results from that test file. For more information about `GetResultFcn`, see “Links and Link Types” on page 11-2.

Note: If the register command returns any warning, then you must unregister the file and run the command again. To unregister the file, enter `rmi unregister linktype_mymscripttap`.

Create the Link

Make the struct containing properties of the external test. To create the link, at the command prompt, enter:

```
externalSource.id = 'counterStartsAtZero';
externalSource.artifact = 'runmytests.m';
externalSource.domain = 'linktype_mymscripttap';
```

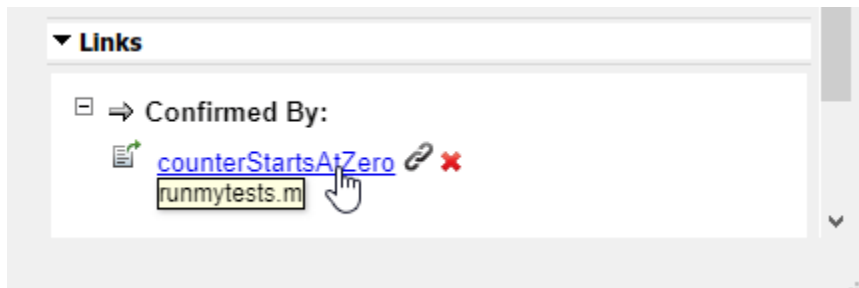
The requirement related to the link has its SID set to 2. To find the requirement related to the link, enter:

```
requirement = reqSet.find('Type', 'Requirement', 'SID', 2);
```

To create the link, enter:

```
link = slreq.createLink(requirement, externalSource);
```

This command creates the link between the test case `counterStartsAtZero` and the requirement with the SID of 2. In the Requirements Editor, the link appears in the **Details** pane, under **Links**.



View the Verification Status

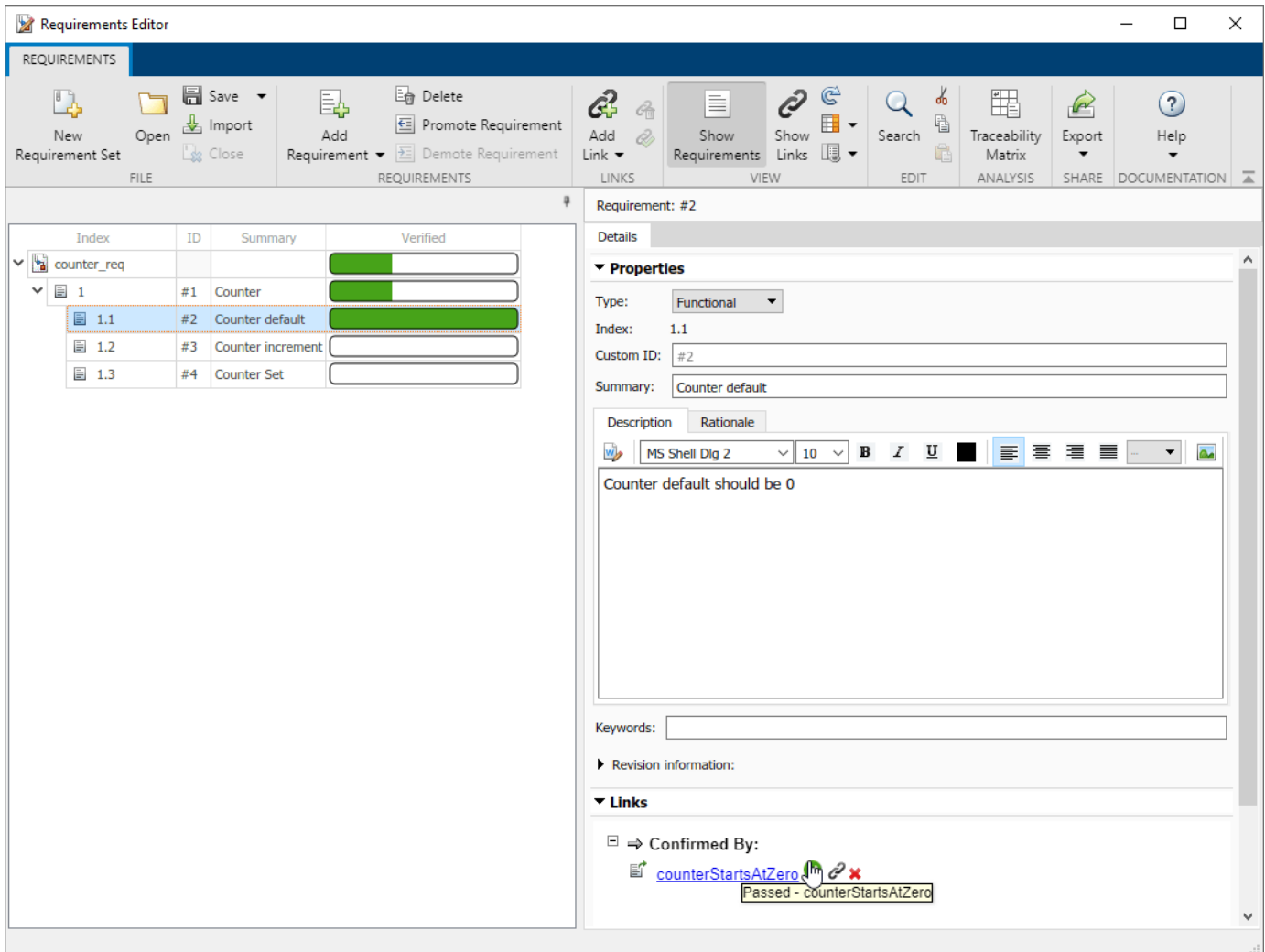
To view the verification status, you need to first update the verification status for the requirement set. At the MATLAB command prompt, type:

```
reqSet.updateVerificationStatus;
```

To see the verification status column in the Requirements Editor, ensure that **Columns > Verification Status** is selected. After the update, fetch the verification status for the requirement:

```
status = reqSet.getVerificationStatus;
```

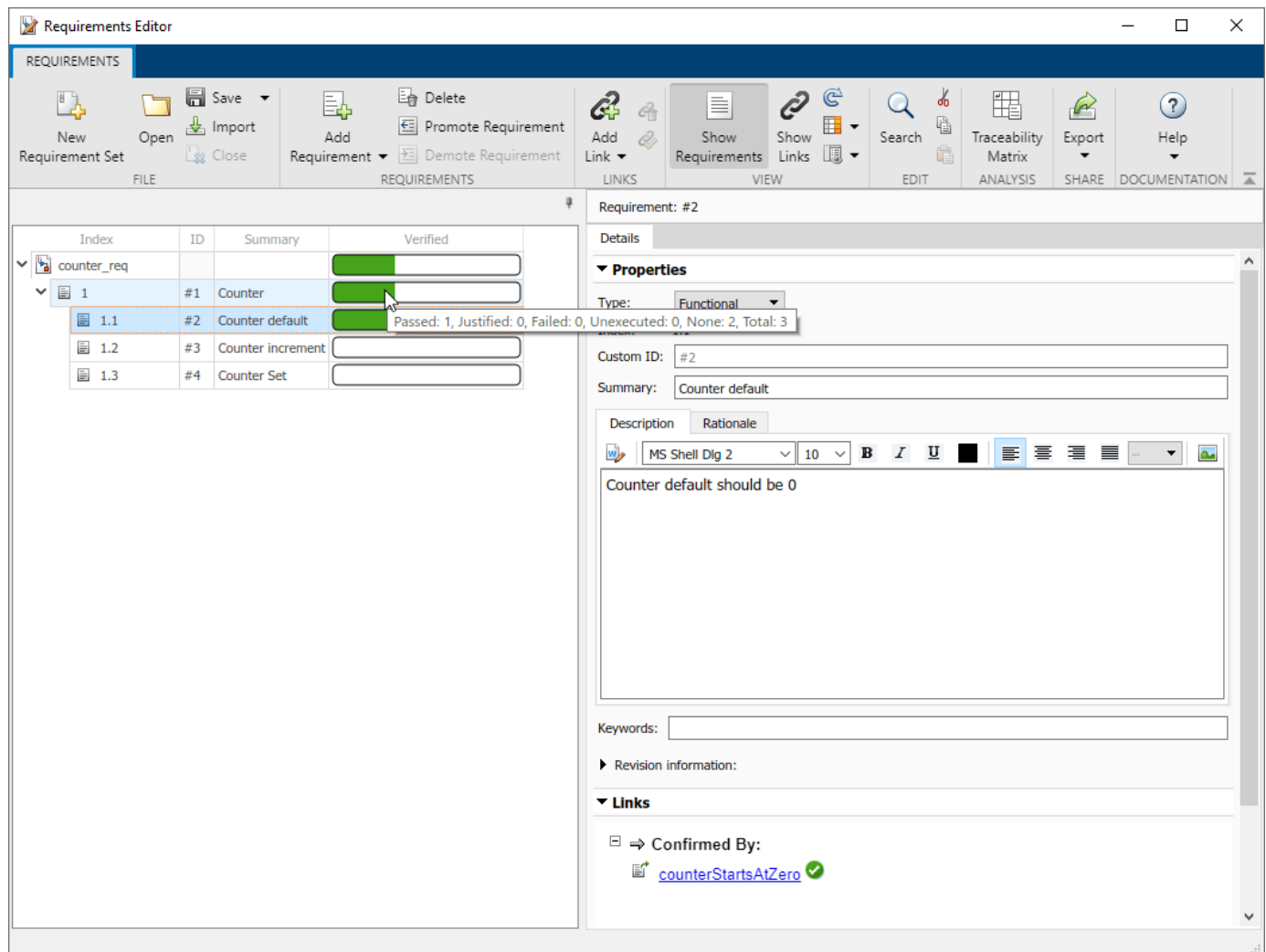
The Requirements Editor shows the verification status for entire requirements set that are passed or failed.



The verification status for the requirements for the counterStartsAtZero is fully verified. Open the Requirements Editor to see the verification status:

```
reqSet = slreq.open('counter_req.slreq');
```

The verification status shows that out of three tests, one test passed. Click Refresh to see the verification status for the requirements in the Requirements Editor.



Unregister the link type.

```
rmi unregister linktype_mymscripttap;
```

See Also

More About

- “Include Results from External Sources in Verification Status” on page 4-28
- “Linking to a Test Script” on page 4-20

Integrating Results from an External Result File

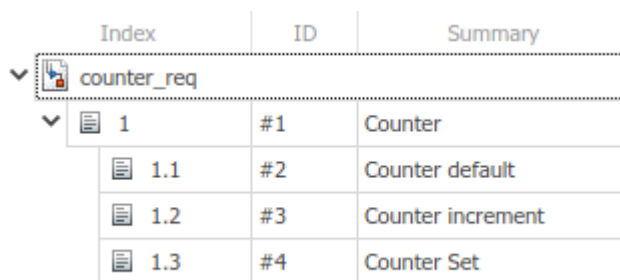
In this example, you link a requirement to a result file in Excel® Format. The verification status in the **Requirements Editor** reflects the test results. This example performs the steps described in “Linking to a Result File” on page 4-31. To run this example, click **Open Example** and run it. This example uses:

- A requirements set file named `counter_req.slreqx`.
- A test results file named `results.xlsx`. This file contains a test case named `counterSetsValue`.

Register the Link Type

Before creating the links, you need to register the link type from the requirements set file. Open the requirements file `counter_req.slreqx` in the **Requirements Editor**:

```
reqSet = slreq.open('counter_req.slreqx');
```



Index	ID	Summary
1	#1	Counter
1.1	#2	Counter default
1.2	#3	Counter increment
1.3	#4	Counter Set

Register the link type that is specific to the external results file. The domain registration needed for this example is `linktype_myexcelresults.m`. To register the custom link type `linktype_myexcelresults.m`, type:

```
rmi register linktype_myexcelresults;
```

The custom logic in the `GetResultFcn` function locates the result file that corresponds to the test case and fetches the results from that result file. For more information about `GetResultFcn`, see “Links and Link Types” on page 11-2. **Note:** If the register command returns any warning, then you must unregister the file and run the command again. To unregister the file, enter `rmi unregister linktype_myexcelresults`.

Create the Link

Make the struct containing properties of the external result. To create the link, at the command prompt, enter:

```
externalSource.id = 'counterSetsValue';
externalSource.artifact = 'results.xlsx';
externalSource.domain = 'linktype_myexcelresults';
```

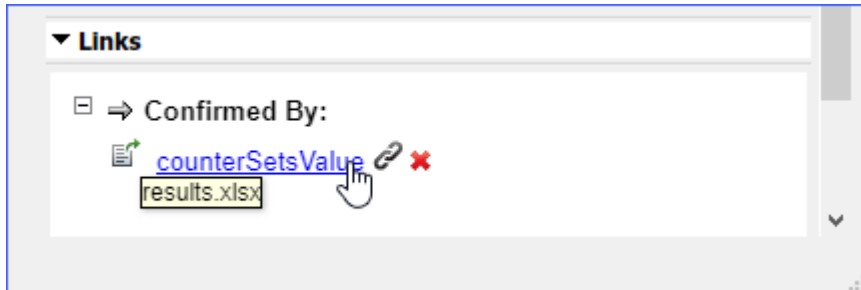
The requirement related to the link has its SID set to 4. To find the requirement related to the link, enter:

```
requirement = reqSet.find('Type', 'Requirement', 'SID', 4);
```

To create the link, enter:

```
link = slreq.createLink(requirement, externalSource);
```

This command creates the link between the test case `counterSetsValue` and the requirement with the SID of 4. In **Requirements Editor**, the link appears in the right pane under **Links**.



View the Verification Status

To view the verification status, you need to first update the verification status for the requirement set. At the MATLAB® command prompt, type:

```
reqSet.updateVerificationStatus;
```

To see the verification status column in the **Requirements Editor**, select **Columns > Verification Status**. After the update, fetch the verification status for the requirement:

```
status = reqSet.getVerificationStatus
```

```
status = struct with fields:
    total: 3
    passed: 1
    failed: 0
    unexecuted: 0
    justified: 0
    none: 2
```

Open the **Requirements Editor** to see the verification status:

```
reqSet = slreq.open('counter_req.slreqx');
```

The **Requirements Editor** shows the verification status for each requirement in the requirement set.

Index	ID	Summary	Verified
counter_req			
1	#1	Counter	
1.1	#2	Counter default	
1.2	#3	Counter increment	
1.3	#4	Counter Set	

Keywords:

▶ Revision information:

▼ **Links**

⇒ Confirmed By:

[counterSetsValue](#)

Passed

The verification status for requirements for the counterSetsValue is fully verified.

Index	ID	Summary	Verified
counter_req			
1	#1	Counter	
1.1	#2	Counter default	Passed: 1, Justified: 0, Failed: 0, Unexecuted: 0, None 2, Total: 3
1.2	#3	Counter increment	
1.3	#4	Counter Set	

Keywords:

▶ Revision information:

▼ **Links**

⇒ Confirmed By:

[counterSetsValue](#)

The verification status shows that out of three tests, one test passed. Click **Refresh** to see the verification status for the requirements in the **Requirements Editor**.

Cleanup

Unregister the linktype.

```
rmi unregister linktype_myexcelresults;
```

See Also

More About

- “Include Results from External Sources in Verification Status” on page 4-28
- “Linking to a Result File” on page 4-31

Integrating Results from a Custom Authored MUnit Script as a Test

In this example, you integrate the results from a MATLAB® xml Unit test by linking to a test script. The verification status in the **Requirements Editor** reflects the test results. To run this example, click **Open Example** and run it. This example uses:

- A requirements set file named `counter_req.slreqx`.
- A xml Unit test file named `myMUnitResults.xml`. This file contains a test case named `testCounterStartsAtZero`.

Step 1: Register the Link Type

Before creating the links, you need to register the link type from the requirements set file. Open the requirements file `counter_req.slreqx` in the **Requirements Editor**.

```
reqSet = slreq.open('counter_req.slreqx');
```

Index	ID	Summary
counter_req		
1	#1	Counter
1.1	#2	Counter default
1.2	#3	Counter increment
1.3	#4	Counter Set

Register the link type that is specific to the MUnit test file. The domain registration needed for this example is `linktype_mymljunitresults.m`. To register the custom linktype `linktype_mymljunitresults.m`, type:

```
rmi register linktype_mymljunitresults;
```

The custom logic in the `GetResultFcn` function locates the result file that corresponds to the test case and fetches the results from that `.xml` file. For more information about `GetResultFcn`, see “Links and Link Types” on page 11-2. The test is run with a customized test runner using XML Plugin producing a JUnit output. The XML Plugin class creates a plugin that writes test results to a file called `myMUnitResults.xml`.

Note: If the register command returns any warning, then you must unregister the file and run the command again. To unregister the file, enter `rmi unregister myMUnitResults.xml`

Section 2: Create the Link

Make the struct containing properties of the external test. To create the link, at the command prompt, enter:

```
externalSource.id = 'testCounterStartsAtZero';
externalSource.artifact = 'counterTests.m';
externalSource.domain = 'linktype_mymljunitresults';
```

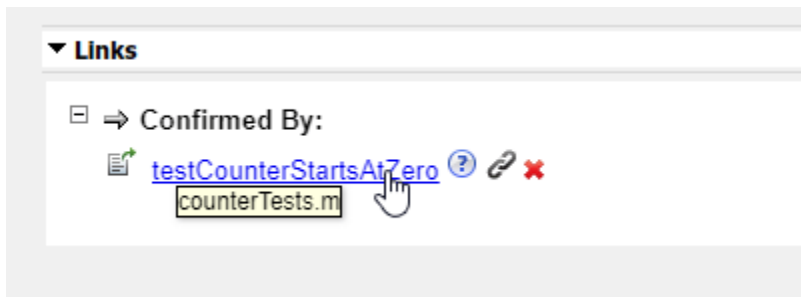
The requirement related to the link has its SID set to 2. To find the requirement related to the link, enter:

```
requirement = reqSet.find('Type', 'Requirement', 'SID', 2);
```

To create the link, enter:

```
link = slreq.createLink(requirement, externalSource);
```

This command creates the link between the test case `testCounterStartsAtZero` and the requirement with the SID of 2. In **Requirements Editor**, the link appears in the right pane, under **Links**.



Section 3: View the Verification Status

To view the verification status, you need to first update the verification status for the requirement set. At the MATLAB command prompt, type:

```
reqSet.updateVerificationStatus;
```

To see the verification status column in the **Requirements Editor**, ensure that **Columns > Verification Status** is selected. After the update, fetch the verification status for the requirement:

```
status = reqSet.getVerificationStatus
```

```
status = struct with fields:
    total: 3
    passed: 0
    failed: 0
    unexecuted: 1
    justified: 0
    none: 2
```

The **Requirements Editor** shows the verification status for entire requirements set that are passed or failed.

Index	ID	Summary	Verified
counter_req			<div style="width: 100%; background-color: green;"></div>
1	#1	Counter	<div style="width: 100%; background-color: green;"></div>
1.1	#2	Counter default	<div style="width: 100%; background-color: green;"></div>
1.2	#3	Counter increment	<div style="width: 0%; background-color: green;"></div>
1.3	#4	Counter Set	<div style="width: 0%; background-color: green;"></div>

Requirement: #2

Summary: Counter default

Description: Counter default should be 0

Keywords:

Revision information:

Links

Confirmed By: testCounterStartsAtZero (Passed)

Comments

The verification status for the requirements for the `testCounterStartsAtZero` is fully verified. Open the **Requirements Editor** to see the verification status:

```
reqSet = slreq.open('counter_req.slreqx');
```

The **Requirements Editor** shows the verification status for each requirement in the requirement set. The verification status for requirements for the `counterSetsValue` is fully verified.

Index	ID	Summary	Verified
counter_req			<div style="width: 100%; background-color: green;"></div>
1	#1	Counter	<div style="width: 100%; background-color: green;"></div>
1.1	#2	Counter default	<div style="width: 100%; background-color: green;"></div>
1.2	#3	Counter increment	<div style="width: 0%; background-color: green;"></div>
1.3	#4	Counter Set	<div style="width: 0%; background-color: green;"></div>

Requirement: #2

Details

Keywords:

Revision information:

SID: 2

Revision: 1

Passed: 1, Justified: 0, Failed: 0, Unexecuted: 0, None: 2, Total: 3

The verification status shows that out of three tests, one test passed. Click **Refresh** to see the verification status for the requirements in the **Requirements Editor**.

Unregister the link type.

```
rmi unregister linktype_mymljunitresults;
```

See Also

More About

- “Include Results from External Sources in Verification Status” on page 4-28
- “Integrating Results from a MATLAB Unit Test Case” on page 4-26

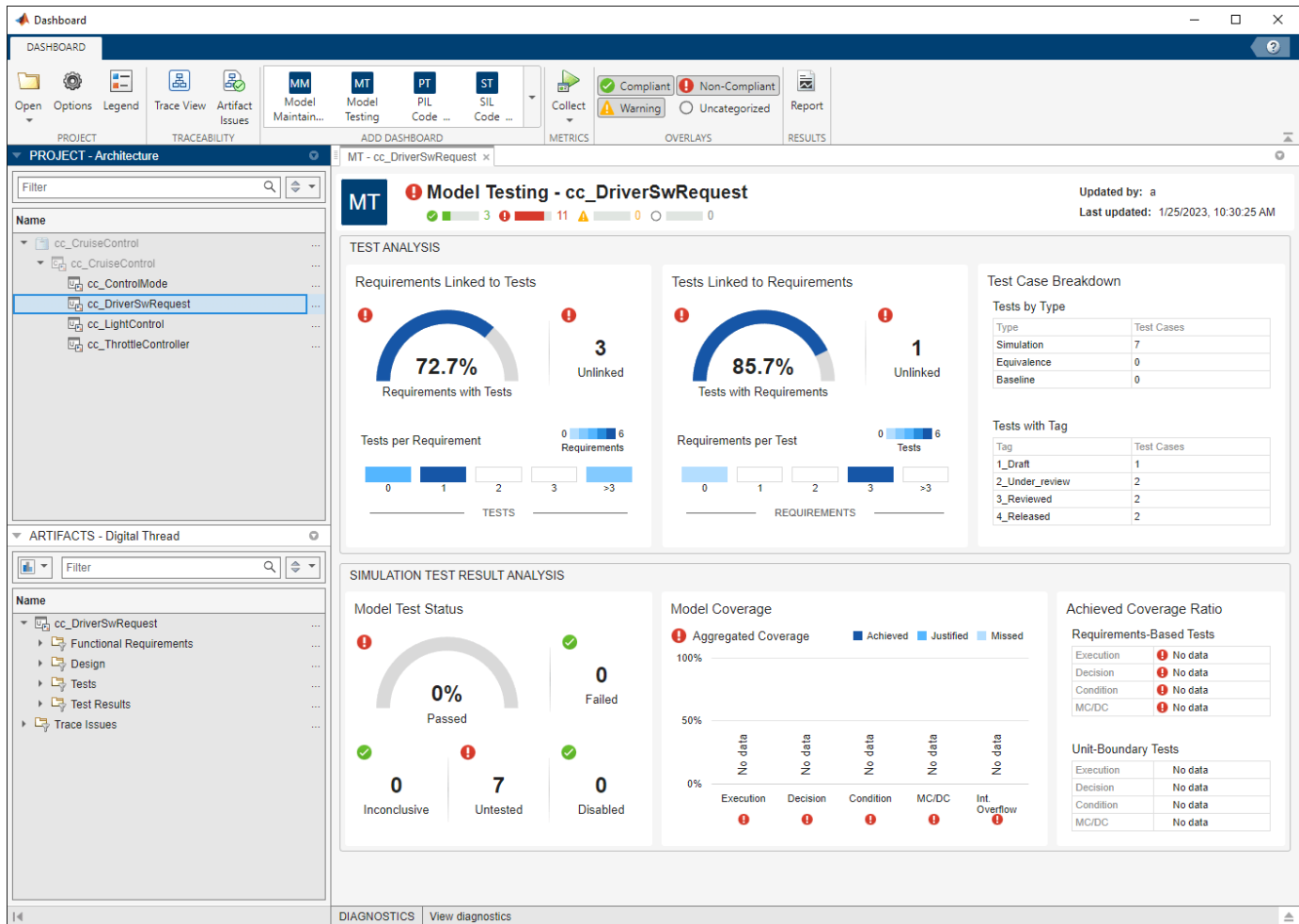
Fix Requirements-Based Testing Issues

This example shows how to address common traceability issues in model requirements and tests by using the Model Testing Dashboard. The dashboard analyzes the testing artifacts in a project and reports metric data on quality and completeness measurements such as traceability and coverage, which reflect guidelines in industry-recognized software development standards, such as ISO 26262 and DO-178C. The dashboard widgets summarize the data so that you can track your requirements-based testing progress and fix the gaps that the dashboard highlights. You can click the widgets to open tables with detailed information, where you can find and fix the testing artifacts that do not meet the corresponding standards.

Collect Metrics for the Testing Artifacts in a Project

The dashboard displays testing data for a model and the artifacts that the unit traces to within a project. For this example, open the project and collect metric data for the artifacts.

- 1 Open the project that contains the models and testing artifacts. For this example, in the MATLAB® Command Window, enter `dashboardCCProjectStart("incomplete")`.
- 2 Open the Dashboard window. To open the Model Testing Dashboard: on the **Project** tab, click **Model Testing Dashboard** or enter `modelTestingDashboard` at the command line.
- 3 In the **Project** panel, the dashboard organizes unit models under the component models that contain them in the model hierarchy. View the metric results for the unit `cc_DriverSwRequest`. In the **Project** panel, click the name of the unit, **cc_DriverSwRequest**. When you initially select **cc_DriverSwRequest**, the dashboard collects the metric results for uncollected metrics and populates the widgets with the data for the unit.



Link a Requirement to its Implementation in a Model

The **Artifacts** panel shows artifacts such as requirements, tests, and test results that trace to the unit selected in the **Project** panel.

In the **Artifacts** panel, the **Trace Issues** folder shows artifacts that do not trace to unit models in the project. The **Trace Issues** folder contains subfolders for:

- **Unexpected Implementation Links** — Requirement links of **Type Implements** for a requirement of **Type Container** or **Type Informational**. The dashboard does not expect these links to be of **Type Implements** because container requirements and informational requirements do not contribute to the Implementation and Verification status of the requirement set that they are in. If a requirement is not meant to be implemented, you can change the link type. For example, you can change a requirement of **Type Informational** to have a link of **Type Related to**.
- **Unresolved and Unsupported Links** — Requirement links which are broken or not supported by the dashboard. For example, if a model block implements a requirement, but you delete the model block, the requirement link is now unresolved. The Model Testing Dashboard does not support traceability analysis for some artifacts and some links. If you expect a link to trace to a unit and it does not, see the troubleshooting solutions in “Resolve Missing Artifacts, Links, and Results” (Simulink Check).

- **Untraced Tests** — Tests that execute on models or subsystems that are not on the project path.
- **Untraced Results** — Results that the dashboard can no longer trace to a test. For example, if a test produces results, but you delete the test, the results can no longer be traced to the test.


Address Testing Traceability Issues

The widgets in the **Test Analysis** section of the Model Testing Dashboard show data about the unit requirements, tests for the unit, and links between them. The widgets indicate if there are gaps in testing and traceability for the implemented requirements.

Link Requirements and Tests

For the unit `cc_DriverSwRequest`, the **Tests Linked to Requirements** section shows that some of the tests are missing links to requirements in the model.

To see detailed information about the missing links, in the **Tests Linked to Requirements** section, click the widget **Unlinked**. The dashboard opens the **Metric Details** for the widget with a table of metric values and hyperlinks to each related artifact. The table shows the tests that are implemented in the unit, but do not have links to requirements. The table is filtered to show only tests that are missing links to requirements.



Metric Details - Tests linked to requirements

Metric that determines if each test case or test iteration for the model is linked to at least one requirement in the project.

Artifact	Source	Requirement Link Status
Detect long decrement	cc_DriverSwRequest_Tests.mldatx	Missing linked requirements

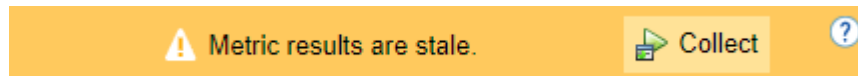
The test `Detect long decrement` is missing linked requirements.

- 1 In the **Artifact** column of the table, point to **Detect long decrement**. The tooltip shows that the test **Detect long decrement** is in the test suite **Unit test for DriverSwRequest**, in the test file **cc_DriverSwRequest_Tests**.
- 2 Click **Detect long decrement** to open the test in the Test Manager. For this example, the test needs to link to three requirements that already exist in the project. If there were not already requirements, you could add a requirement by using the Requirements Editor.
- 3 Open the software requirements in the Requirements Editor. In the **Artifacts** panel of the Dashboard window, expand the folder **Functional Requirements > Implemented** and double-click the requirement file **cc_SoftwareReqs.slreqx**.
- 4 View the software requirements in the container with the summary **Driver Switch Request Handling**. Expand **cc_SoftwareReqs > Driver Switch Request Handling**.
- 5 Select multiple software requirements. Hold down the **Ctrl** key as you click **Output request mode**, **Avoid repeating commands**, and **Long Increment/Decrement Switch recognition**. Keep these requirements selected in the Requirements Editor.
- 6 In the Test Manager, expand the **Requirements** section for the test `Detect long decrement`. Click the arrow next to the **Add** button and select **Link to Selected Requirement**. The traceability link indicates that the test `Detect long decrement` verifies the three requirements **Output request mode**, **Avoid repeating commands**, and **Long Increment/Decrement Switch recognition**.

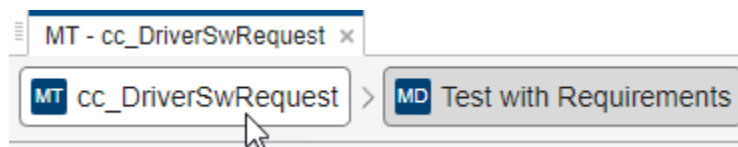
- 7 The metric results in the dashboard reflect only the saved artifact files. To save the test suite `cc_DriverSwRequest_Tests.mldatx`, in the **Test Browser**, right-click `cc_DriverSwRequest_Tests` and click **Save**.

Refresh Metric Results in the Dashboard

The dashboard detects that the metric results are now stale and shows a warning banner at the top of the dashboard.



- 1 Click the **Collect** button on the warning banner to re-collect the metric data so that the dashboard reflects the traceability link between the test and requirements.
- 2 View the updated dashboard widgets by returning to the **Model Testing** results. At the top of the dashboard, there is a breadcrumb trail from the **Metric Details** back to the **Model Testing** results. Click the breadcrumb button for `cc_DriverSwRequest` to return to the **Model Testing** results for the unit.



The **Tests Linked to Requirements** section shows that there are no unlinked tests. The **Requirements Linked to Tests** section shows that there are 3 unlinked requirements. Typically, before running the tests, you investigate and address these testing traceability issues by adding tests and linking them to the requirements. For this example, leave the unlinked artifacts and continue to the next step of running the tests.

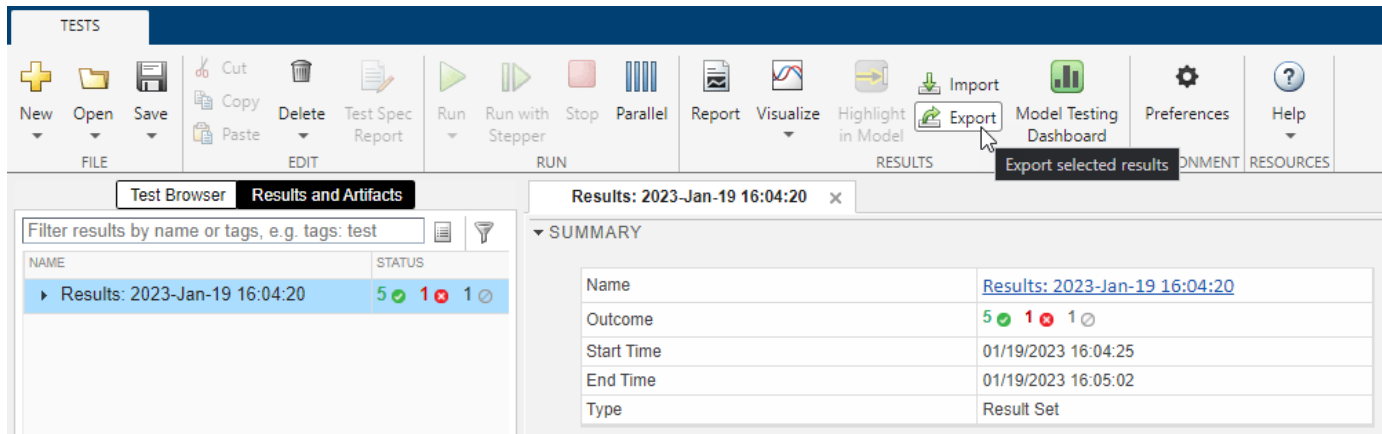
Test the Model and Analyze Failures and Gaps

After you create and link unit tests that verify the requirements, run the tests to check that the functionality of the model meets the requirements. To see a summary of the test results and coverage measurements, use the widgets in the **Simulation Test Result Analysis** section of the dashboard. The widgets help show testing failures and gaps. Use the metric results to analyze the underlying artifacts and to address the issues.

Perform Unit Testing

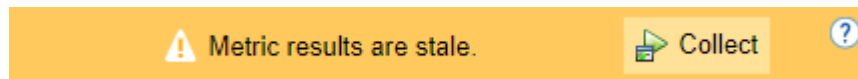
Run the tests for the model by using the Test Manager. Save the test results in your project and review them in the Model Testing Dashboard.

- 1 Open the unit tests for the model in the Test Manager. In the Model Testing Dashboard, in the **Artifacts** panel, expand the folder **Tests > Unit Tests** and double-click the test file `cc_DriverSwRequest_Tests.mldatx`.
- 2 In the Test Manager, click **Run**.
- 3 Select the results in the **Results and Artifacts** pane.
- 4 Save the test results as a file in the project. On the **Tests** tab, in the **Results** section, click **Export**. Name the results file `Results1.mldatx` and save the file under the project root folder.



The Model Testing Dashboard detects the results and automatically updates the **Artifacts** panel to include the new test results for the unit in the subfolder **Test Results > Model**.

The dashboard also detects that the metric results are now stale and shows a warning banner at the top of the dashboard.



The **Stale** icon **STALE** appears on the widgets in the **Simulation Test Result Analysis** section to indicate that they are showing stale data that does not include the changes.

Click the **Collect** button on the warning banner to re-collect the metric data and to update the stale widgets with data from the current artifacts.

Address Testing Failures and Gaps

For the unit `cc_DriverSwRequest`, the **Model Test Status** section of the dashboard indicates that one test failed and one test was disabled during the latest test run.

- 1 To view the disabled test, in the dashboard, click the **Disabled** widget. The table shows the disabled tests for the model.
- 2 Open the disabled test in the Test Manager. In the table, click the test artifact **Detect long decrement**.
- 3 Enable the test. In the **Test Browser**, right-click the test and click **Enabled**.
- 4 Re-run the test. In the **Test Browser**, right-click the test and click **Run** and save the test suite file.
- 5 View the updated number of disabled tests. In the dashboard, click the **Collect** button on the warning banner. Note that there are now zero disabled tests reported in the **Model Test Status** section of the dashboard.
- 6 View the failed test in the dashboard. Click the breadcrumb button for `cc_DriverSwRequest` to return to the **Model Testing** results and click the **Failed** widget.
- 7 Open the failed test in the Test Manager. In the table, click the test artifact **Detect set**.
- 8 Examine the test failure in the Test Manager. You can determine if you need to update the test or the model by using the test results and links to the model. For this example, instead of fixing the

failure, use the breadcrumbs in the dashboard to return to the **Model Testing** results and continue on to examine test coverage.

Check if the tests that you ran fully exercised the model design by using the coverage metrics. For this example, the **Model Coverage** section of the dashboard indicates that some conditions in the model were not covered. Place your cursor over the **Decision** bar in the widget to see what percent of condition coverage was achieved.

- 1 View details about the decision coverage by clicking one of the **Decision** bars. For this example, click the **Decision** bar for **Achieved** coverage.
- 2 In the table, expand the model artifact. The table shows the test results for the model and the results files that contains them. For this example, click on the hyperlink to the source file **Results1.mldatx** to open the results file in the Test Manager.
- 3 To see detailed coverage results, use the Test Manager to open the model in the Coverage perspective. In the Test Manager, in the **Aggregated Coverage Results** section, in the **Analyzed Model** column, click **cc_DriverSwRequest**.
- 4 Coverage highlighting on the model shows the points that were not covered by the tests. For this example, do not fix the missing coverage. For a point that is not covered in your project, you can add a test to cover it. You can find the requirement that is implemented by the model element or, if there is none, add a requirement for it. Then you can link the new test to the requirement. If the point should not be covered, you can justify the missing coverage by using a filter.

Once you have updated the unit tests to address failures and gaps in your project, run the tests and save the results. Then examine the results by collecting the metrics in the dashboard.

Iterative Requirements-Based Testing with the Model Testing Dashboard

In a project with many artifacts and traceability connections, you can monitor the status of the design and testing artifacts whenever there is a change to a file in the project. After you change an artifact, use the dashboard to check if there are downstream testing impacts by updating the tracing data and metric results. Use the **Metric Details** tables to find and fix the affected artifacts. Track your progress by updating the dashboard widgets until they show that the model testing quality meets the standards for the project.

Change Tracking and Team-Based Workflows

- “Requirements-Based Development in Projects” on page 5-2
- “Track Changes to Requirement Links” on page 5-3
- “Compare Requirement Sets” on page 5-10
- “Compare Link Sets” on page 5-11
- “Report Requirements Information” on page 5-12
- “Three-Way AutoMerge Solution for Requirement Set and Link Set” on page 5-15
- “Merge Requirement Set and Link Set Files” on page 5-17
- “Track Changes to Test Cases in Requirements Editor” on page 5-21
- “Track Changes to MATLAB Code Using Requirements Editor” on page 5-24
- “Verify Safety Requirements Linked to Test Steps Using Functional Requirements” on page 5-29
- “Publish and Save Printable Report of Comparison Results” on page 5-36

Requirements-Based Development in Projects

Projects help you organize and share files, and work with source control systems. Since requirements-based development commonly involves multiple contributors and multiple files, consider organizing your models, requirements, links, and tests in a project. For more information, see “Create Projects”.

Organizing Requirements, Models, and Tests

To facilitate multiple individuals working on a project in source control, consider the following:

- Store models, requirements, and tests in separate folders within a project.
- Add folders to the project path, so that link sources and destinations resolve when you open a requirement set or model.
- Use a source control tool, such as Git, to collaborate on projects and project files.
- When you link requirements to a model (or code, test, etc.) the traceability data file saves in the same folder as the model. Store traceability data files in a folder with the respective model, code, or test.
- Opening a requirement set in a project loads other requirement and link sets in the project.

This is a simple project with a model, several tests, and a requirement set.

Name	Git	Status	Classification
models	.	✓	
prohibitSimultaneousPress.slmx	●	✓	Design
prohibitSimultaneousPress.slx	●	✓	Design
requirements	.	✓	
functional_reqs.sreqx	●	✓	
tests	.	✓	
baseline_test.mldatx	●	✓	Test
baseline_test.slmx	●	✓	
baseline_test_data.xlsx	●	✓	
baseline_test_result_criteria.xlsx	●	✓	
simulation_tests.mldatx	●	✓	Test
simulation_tests.slmx	●	✓	
test_utilityfn.m	●	✓	Design

If your project includes shared library models, requirement sets, requirements links, and supporting files, you can create requirements links between your local models and shared requirements. You can also create requirements between your local requirements and shared models and supporting files. Shared library requirements data is integrated into your local requirements data.

You can refresh requirement link information by using the `sreq.refreshLinkDependencies` command.

Track Changes to Requirement Links

After you author or import requirements and create links between design elements and your requirements, Requirements Toolbox tracks the links and detects when linked requirements change. You can track change information from the **Requirements Editor** or in the Traceability Matrix and, then resolve change issues or clear changes that have no impact on the requirement status.

Enable Change Tracking for Requirement Links

To enable change tracking for requirement links:

- 1 Open the **Requirements Editor**. From your Simulink model, in the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Requirements Editor**. Alternatively, enter this command at the MATLAB command prompt.

```
slreq.editor
```

- 2 Open a requirement set.


- 3 On the **View** tab under **Information** , select **Change Information**.

Once you enable **Change Information**, this setting stays enabled even after you close the **Requirements Editor**.

Alternatively, you can enable change tracking for requirement links from the Requirements Perspective. Right-click an item in the Requirements Perspective and select **Change Information**.

Run Change Tracking Analysis

Requirements Toolbox does not perform change tracking analysis until you run the analysis. You can run the analysis in the **Requirements Editor** or Requirements Perspective.

A banner in the **Requirements Editor** or Requirements Perspective indicates when results are pending. To run the analysis, click **Analyze now** in the banner. Alternatively, click **Refresh** in the **Requirements Editor** or the refresh button  in the Requirements Perspective.

Change tracking analysis continuously runs in the background until you use `slreq.clear`.

Review Changes to Requirements, Test Objects, and MATLAB Code Lines

After you run change tracking analysis, you can use the Requirements Editor to review the changes.

- “Review Changes to Requirements” on page 5-3
- “Review Changes to Test Objects” on page 5-5
- “Review Changes to MATLAB Code Lines” on page 5-6

Review Changes to Requirements

You can link requirements to other types of items. For a full list of linkable items, see “Linkable Items” on page 3-32. When you change linked requirement, the **Requirements Editor** and

Traceability Matrix window and show a change issue. After you enable change tracking for requirements links, you can view the change issues associated with a particular requirement from the **Requirements Editor** or the Traceability Matrix window.

Note Requirements Toolbox provides change tracking information for unresolved links only if the linked requirement is valid. For more information on why a link might become unresolved, see “Unresolved Links” on page 3-40.

In the **Requirements Editor**, click **Show Requirements**. The linked requirements with changes are highlighted in red. When you select a requirement, the associated link is also highlighted in red in the right pane, under **Links**. To view the change issue, select a requirement and, under **Links**, point to the link, then click the link icon (🔗) to the right of the linked item.

The screenshot shows the Requirements Editor interface. The left pane displays a tree view of requirements under the folder 'crs_req_func_spec*'. The selected requirement is 'Set Switch Detection' (ID #8), which is highlighted in red. The right pane shows the details for 'Requirement: #12', including a list of links. The 'Links' section shows 'Derived from: Target Speed Increment' (with a link icon and a red 'x'), 'Implemented by: Enumerated Constant1', and 'Verified by: Increment button short'.

In the Traceability Matrix window, under **Highlight Missing Links** click **Highlight Changed Links** to highlight the row, column, and cell associated with the linked requirement. To view changes to the linked requirement, select the cell and, in the dialog box that appears, click the requirement hyperlink next to **Source** or **Destination**. To view the change issue, click the link hyperlink next to

Link. To learn more about using the Traceability Matrix window to find change issues, see “View and Clear Change Issues for Links” on page 3-14.

The screenshot shows the Traceability Matrix window with the following components:

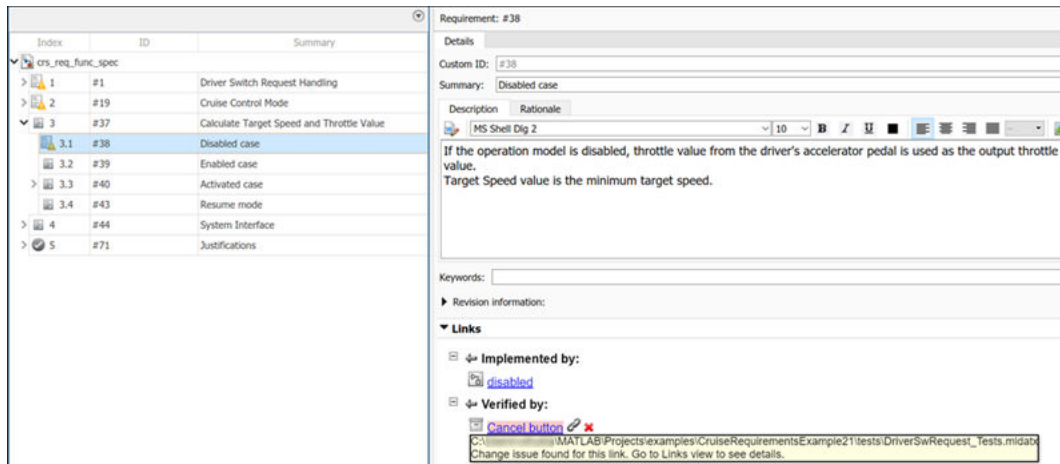
- HOME** tab selected.
- ARTIFACTS** section: Add, Configure Matrix.
- LINKS** section: Highlight Missing Links, Create Link, Remove Links, Clear Change Issue.
- VIEW** section: Update, Scope, Expand All, Collapse All.
- SHARE** section: Export.
- Filter Panel** on the left:
 - Top**: Type (Leaf Block, Subsystem), Link (Missing Links, Missing Expected Links).
 - Left**: Type (Container, Functional, Justification), Link (Missing Links), **Change Tracking**.
 - With**: Cell, Type.
 - Change Tracking**: Implements, With Change Issues.
- Requirement Set vs Simulink Model** window:
 - Artifacts: crs_controller, crs_req_func_spec.
 - Grid showing links between crs_controller and crs_req_func_spec.
 - Grid columns: crs_controller, DriverSwRequest, decrement, Enumerated Constant, Enumerated Constant2, Switch2.
 - Grid rows: crs_req_func_spec, Driver Switch Request Handling, Long Switch recognition, Set Switch Detection.
 - Link from crs_req_func_spec to Set Switch Detection is highlighted in blue.
- Tooltip** for the highlighted link:

Source	Switch2
Destination	Set Switch Detection
Link	#8: Set Switch Detection (Implement)

Review Changes to Test Objects

To create links between requirements and test objects, see “Link Test Cases to Requirements”. When you change Simulink Test test objects, the **Requirements Editor** and Traceability Matrix window highlight the link to indicate that the source object has changed.

In the **Requirements Editor**, click **Show Requirements**. The linked requirements show the change issues related to the source of the links. The editor indicates linked requirements that have change issues with the change issue icon.




When you link a test case to a requirement, the associated link is highlighted in red in the right pane, under the **Links** section. To view the change issue, select the link, then click **Change Information**.

You can clear the change issues for test cases from the Requirements Editor or the Traceability Matrix by:

- Compare requirement sets or link sets files: You can compare the two versions with and without the changes..
- Clear issues in the Requirements Editor: If a change has no impact, you can clear the change issue. If the change issue affects requirements or test cases, first resolve the discrepancy, then clear the issue.
- Clear issues using the Traceability Matrix window: In the Traceability Matrix, click **Highlight Missing Links > Highlight Changed Links** to highlight the row, column, and cell associated with the changed requirement. From the **Filter Panel**, select **Change Tracking > With Change Issues** to filter the test cases that have changed.

Review Changes to MATLAB Code Lines

When you change MATLAB code lines or MATLAB Function block, the Requirements Editor and Traceability Matrix window highlight the link to indicate that the source object has changed.

In the Requirements Editor, select Show Requirements from the **Requirements** pane. The linked requirements show the change issues related to the source of the links. The editor indicates linked requirements that have change issues with the change issue icon .

For more information on linking the requirements to MATLAB code, see “Verify Requirements with MATLAB Tests” on page 10-4. You can view the change issue in the MATLAB code when you:

- Add or remove lines.
- Insert or remove text from the existing lines.

Index	ID	Summary
> shortest_path_func...		
✓ shortest_path_tests...		
1	#1	Overview
2	#2	Test Cases
2.1	#22	Nominal Mode Tests
2.2	#15	Tests for invalid conditions
2.2.1	#18	Test with invalid startIdx > N
2.2.2	#14	Test with invalid startIdx < 1
2.2.3	#20	Test with invalid endIdx > N
2.2.4	#21	Test with invalid endIdx < 1
2.2.5	#11	Test a degenerate graph with...

▼ Properties

Type:

Index: 2.2.3

Custom ID:

Summary:

Description Rationale





B *I* U ■

Keywords:


▶ Revision information:


▼ Links

☐ ← Verified by:

 [function check_invalid_end_2\(testCase\)](#)   

function check_invalid_end_2(testCase) in graph_unit_tests.m
Change issue found for this link. Go to Links view to see details.

In the requirements view, you can see the highlighted link in the **Links** section along with the change icon  on the requirement in the requirement tree.

To view the change issue, click the  icon in the **Links** section. This directs you to the section of code with the changes. To clear the change issue, click the links in Links view and select **Clear Issue** under **Change Information**.

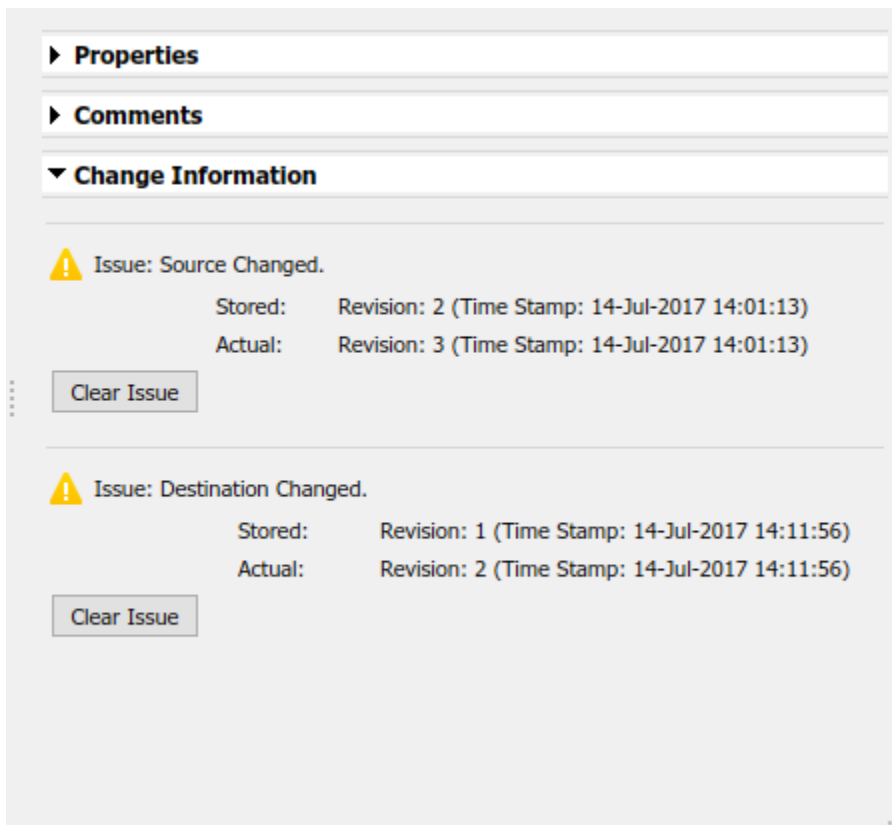
MATLAB code lines within MATLAB Function blocks in Simulink models are also tracked for change issues.

Resolve Change Issues

The **Requirements Editor** displays change information, including change issues, for each link. Click **Show Links** and, in the right pane, expand **Change Information**. Requirements Toolbox compares the stored timestamp and revision to the current timestamp and revision for the linked artifact. If you change the source or the requirement after you create the link, or after the last time you changed it, then the **Requirements Editor** indicates a change issue.

You can resolve change issues from the **Requirements Editor** or the Traceability Matrix. If a change has no impact, you can clear the change issue. The link change information is updated with the current timestamp and revision for the requirement. If the change issue affects the status of your requirements, you can change the model, the requirements, the test cases, or the links themselves to resolve the revision discrepancy, and then clear the issue.

In the **Requirements Editor**, links with change issues are highlighted in red when you select **Show Links**. To clear a change issue, select the link and, in the right pane, under **Change Information**, click **Clear Issue**.



In the Traceability Matrix, you can highlight links with change issues by selecting **Highlight Missing Links > Highlight Changed Links**. To clear the change issue, select the cell containing the link and, in the toolstrip, click **Clear Change Issue**.

Clear Change Issues for Multiple Links

You can clear change issues for multiple links at a time in the **Requirements Editor** or in the Traceability Matrix.

In the **Requirements Editor**, select multiple links by pressing **Shift** or **Ctrl** and clicking the links. Right-click one of the selected links and click **Clear Issue** from the context menu. To clear all change issues for an entire link set, select the link set and, in the right pane, under **Change Information**, click **Clear All**. You can also right-click the link set and select **Clear All Change Issues** from the context menu.

In the Traceability Matrix, select multiple cells by clicking and dragging, or pressing **Shift** or **Ctrl**, click the cells, and click **Clear Change Issue** in the toolstrip.

Add Comments to Links

When you resolve change issues, it is good practice to add a comment to the link describing the actions. Each link has a **Comments** property. When you clear a change issue in either the **Requirements Editor** or Traceability Matrix, a dialog box appears and you are prompted to add a comment.

To add an additional comment:

- 1 In the **Requirements Editor**, click **Show Links**.
- 2 Select the link.
- 3 In the right pane, under **Comments**, click **Add Comment**.

Manually Check for Using Links Change Tracking

Change tracking information is automatically updated in the **Requirements Editor**, but you can also manually refresh it. To refresh the change tracking information:

- In the **Requirements Editor**, click  **Refresh**.
- In the Traceability Matrix, click **Update**.

In the Traceability Matrix, you need to refresh change tracking information manually.

See Also

More About

- “Create a Project from a Model” (Simulink)
- “Track Requirement Links with a Traceability Matrix” on page 3-5
- “Track Changes to Test Cases in Requirements Editor” on page 5-21

Compare Requirement Sets

To compare differences between two requirement sets, use the “Compare Revisions” (Simulink) tool.

Compare two Requirements Toolbox requirement sets

If you have two versions of a Requirements Toolbox requirement set file, you can compare differences between the two files.

Select Two requirement set Files to Compare

- 1 In the **Current Folder** pane of MATLAB, or in the **Project Files View** of your project, select the first file for comparison.
- 2 In the **Current Folder** pane of MATLAB, or in the **Project Files View** of your project, press **Ctrl**, and then click the second file for comparison.
- 3 Right-click either file and select **Compare Selected Files/Folders**.

Select One File to Compare

- 1 In the **Current Folder** pane of MATLAB, right-click first file and select **Compare Against > Choose**.
- 2 Select the second file for comparison and select **Requirements Toolbox Comparison** as the **Comparison type**.

For more information, see “Review Changes in Project Files”.

The comparison tool shows the differences between the two requirement sets. The comparison shows which specific requirements in a requirement set changed and which fields of each requirement changed.

Note The comparison tool shows only changes in saved requirement sets. Changes that have occurred in memory but are not yet saved to file are not shown.

To view a requirements item in the **Requirements Editor**, highlight the requirements item and click **Highlight Now**. The requirements item from the right comparison pane opens in the **Requirements Editor**. If you select **Always Highlight**, the **Requirements Editor** opens to the selected requirements item whenever you click one.

Review Changes in Source-Controlled Files

If you use a separate change management tool to manage changes to your projects, you can compare differences with your source-controlled Requirements Toolbox files. For more information, see “Use Source Control with Projects”.

Compare Link Sets

If you have two versions of a Requirements Toolbox link set file, you can compare the differences between the two files.

Select Two Link Set Files to Compare

- 1 In the **Current Folder** pane of MATLAB, or in the **Project Files View** of your project, select the first file for comparison.
- 2 In the **Current Folder** pane of MATLAB, or in the **Project Files View** of your project, press **Ctrl**, and then click the second file for comparison.
- 3 Right-click either file and select **Compare Selected Files/Folders**.

Select One File to Compare

- 1 In the **Current Folder** pane of MATLAB, right-click the first file and select **Compare Against > Choose**.
- 2 Select the second file for comparison and select **Requirements Toolbox Comparison** as the **Comparison type**.

For more information, see “Review Changes in Project Files”.

The comparison tool shows the differences between the two `.slmx` link set files. The comparison shows which specific links in a link set changed and which fields of each link changed.

Note The comparison tool shows only changes in saved `.slmx` link sets. Changes that have occurred in memory but are not yet saved to file are not shown.

To view a link in the Links View of the **Requirements Editor**, highlight the link and click **Highlight Now**. The link from the right comparison pane opens in the Links View of the **Requirements Editor**. If you select **Always Highlight**, the **Requirements Editor** opens to the selected link item whenever you click one.

Report Requirements Information

To document your requirements for review, you can create a report for one or more requirement sets. You can select the requirements information to contain in the report, including:

- Navigable links to model entities and other requirements
- Requirements change and revision information
- Implementation and Verification status summaries

You can create reports in .docx (Microsoft Word), PDF and HTML formats. If you select multiple requirement sets for reporting, the information is contained in a single report.

You can create reports using the **Report Generation Options** dialog box or programmatically by using the `slreq.generateReport` function.

Report Generation Options

Title Page Options

Report title: Requirements Report

Report authors: jdoe

File

Folder: D:\

File Name: myReport.docx Select...

Included Requirement Sets

	Name	Path
<input checked="" type="checkbox"/>	crs_req	D:\SLRequirementsCruiseControlExample-master\S...
<input checked="" type="checkbox"/>	crs_req_func_spec	D:\SLRequirementsCruiseControlExample-master\S...

Unselect All

Report content

Table of Contents Implementation Status

Rationale Verification Status

Keywords Links

Custom Attributes

Revision information

Comments Change Information

Empty Sections

Group Links By:

Artifact Link Type

Generate Report Cancel Help

To create a report by using the Report Generation Options dialog box:

- 1 Right-click a requirement set in the **Requirements Editor** or Requirements Browser, and select **Generate Report**.

To create a report with multiple requirement sets, click **Export > Generate Report**.

The Report Generation Options dialog box opens.

- 2 Set the report file name and location by clicking the **Select** button next to the file name.
- 3 Select report content options.
- 4 Select requirement sets to include in the report. The dialog box displays requirement sets that are loaded in memory. To include a requirement set that does not appear in the list, first open the requirement set using the **Requirements Editor**.
- 5 Click **Generate Report**.



The Report Appendix provides summaries of all the change issues and requirement set artifacts that you create the report for.

Report Navigation Links



The requirements report contains links you can use to navigate to model items and other requirements. For example, this requirement is implemented by two model entities, and is derived from two requirements. Hold **Ctrl** and click a link to open the linked item.

Links

Artifact: crs_req.slreqx

Linked Item	Link Type
 Activating cruise control	← Derived from
 Deactivating cruise control	← Derived from

Artifact: crs_controller.slx

Linked Item	Link Type
 Switch2	← Implemented by
 Enumerated Constant2	← Implemented by

If you use `slreq.generateReport` to generate a report as a Microsoft Word document, you will need to manually update the Table of Contents. Open the report, select the contents, and press **F9**.

See Also

`slreq.generateReport` | `slreq.getReportOptions`

Three-Way AutoMerge Solution for Requirement Set and Link Set

If multiple users are working on the same set of requirement set and link set files in Git™, you can merge the changes into a single file by using the `mLAutoMerge`.

Configure Git Environment for AutoMerge

You can follow the process described in “Customize External Source Control to Use MATLAB for Diff and Merge” with Requirements Toolbox to merge changes in different branches in Git.

To use `mLAutoMerge` with the Git tool:

- 1 At the MATLAB command prompt, enter:


```
comparisons.ExternalSCMLink.setupGitConfig()
```
- 2 Create a project and add the project to Git. For more information, see “Add Existing Project to Source Control”.

Select and Merge Branches in Git

To select a branch and merge the changes:

- 1 Change the current folder to your Git repository folder.
- 2 Select **Branches** from the toolstrip.
- 3 From **Branches** drop-down list, select a branch from which you want to merge the changes.
- 4 Click **Merge** to merge from the selected branch.

After merging of a requirement set file is complete, a log file `<requirement_set_name>_merge_<timestamp>.log` is generated in the Git repository folder. The log file contains changes in the SID values of the requirements during merging of requirement set (`slreqx`) files.

Note If there are no conflicts in merging the branches, then merge modifies the target file. If the changes conflict, you must view and resolve the conflicts manually.

Limitations

- Git is the only supported source control tool.
- You must resolve the merge conflicts manually.
- When you **Move up** or **Move down** a requirement under its parent requirement, the change in order or sequence may not reflect in the merged requirement set.
- You cannot merge the requirement set files that contain images.

See Also

More About

- “Branch and Merge with Git”
- “Create Projects”

Merge Requirement Set and Link Set Files

This example explains how to merge changes from multiple requirement set and link set files.

Merge Files with No Conflicts

If you are editing a requirements file which is also concurrently modified by another user, you can get the changes from the other user using Git™ merge. If you want to merge the files, you first have to make sure you have Git and run the `ExternalSCMLink.setupGitConfig` command.

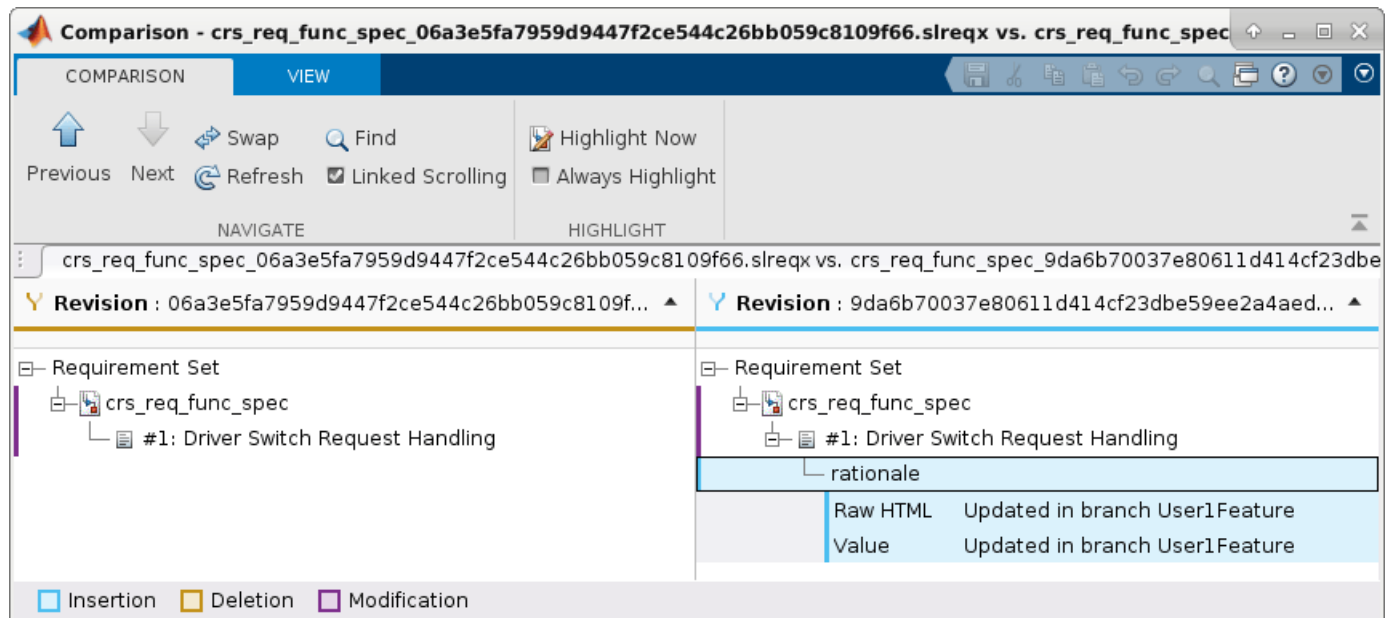
To merge a file without any conflict:

1. At the MATLAB® command prompt, enter:

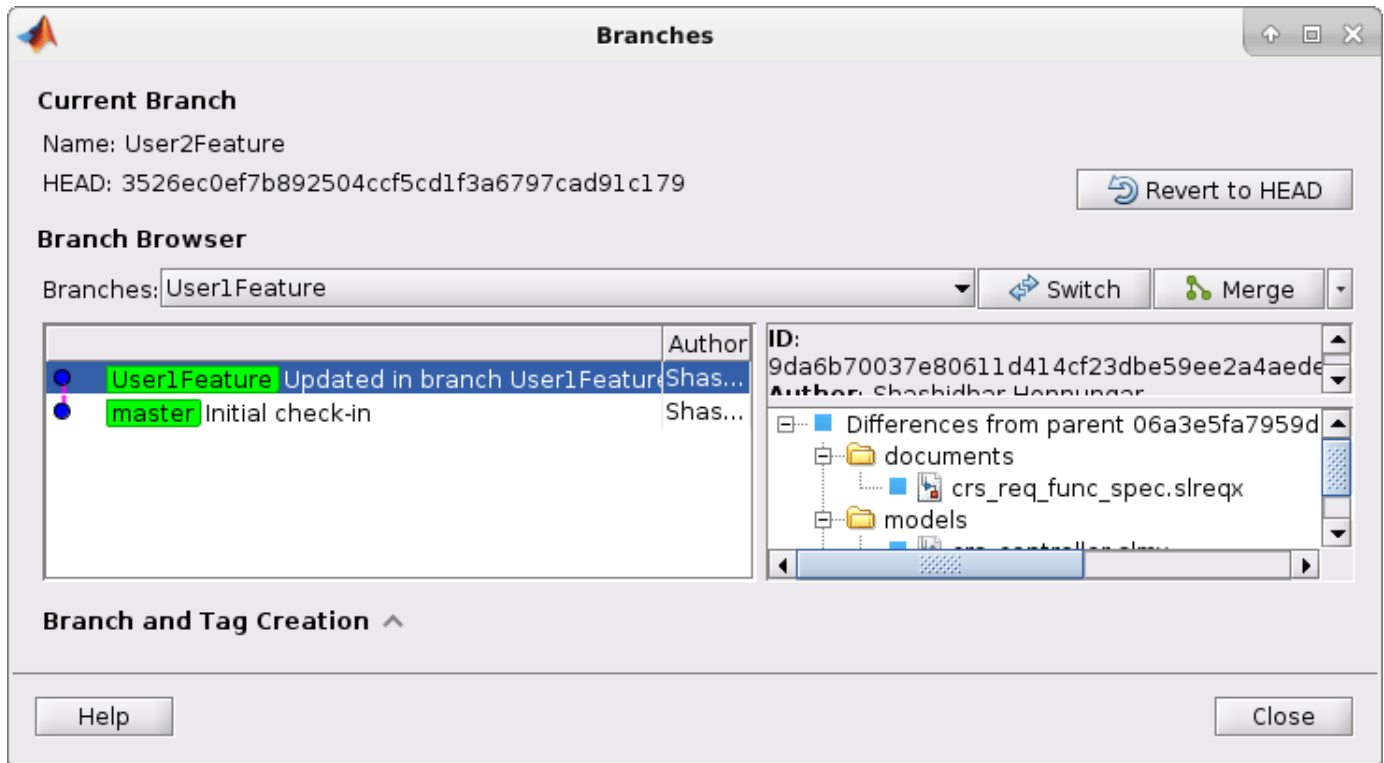
```
slreqCCMergeSetup
```

This helps you to set up two branches, `User1Feature` and `User2Feature`, where `User2Feature` is the current active branch.

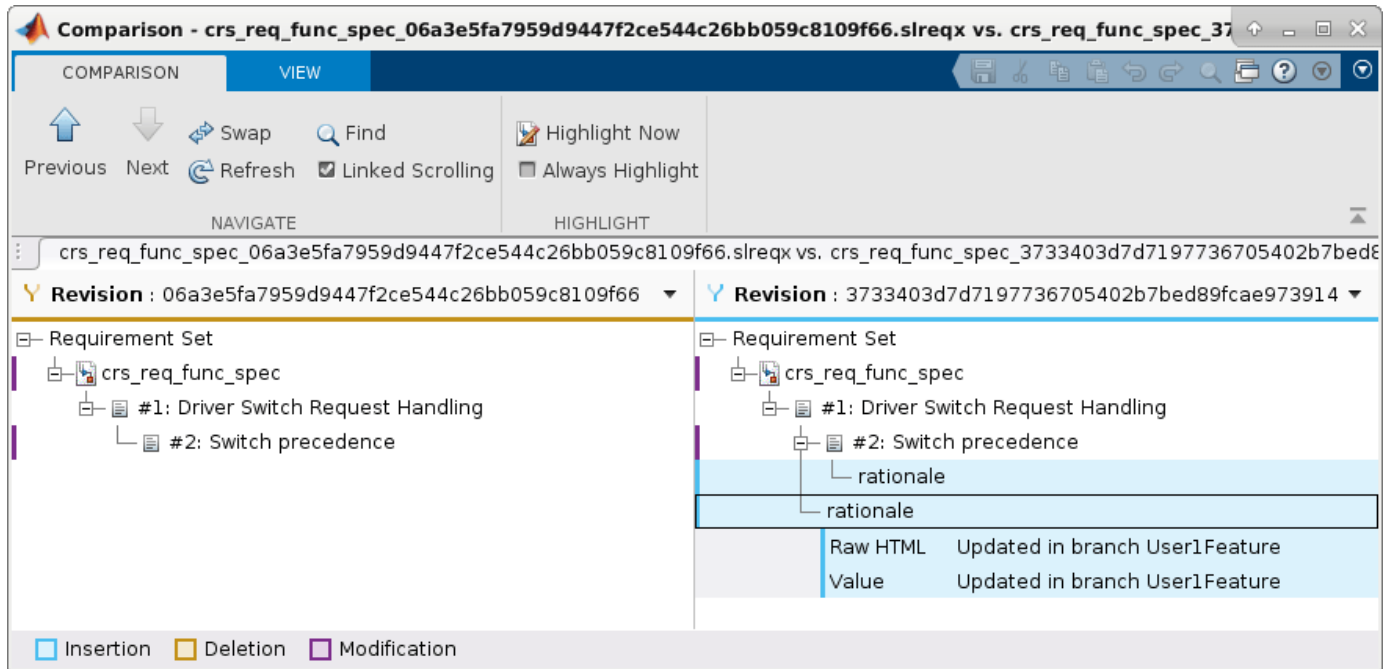
2. To inspect the changes in each branch, switch to that branch and right-click the file in the current folder browser and select **Compare To Ancestor**.



3. To merge changes from `User1Feature` branch to `User2Feature` branch, set `User2Feature` branch as the current branch and select `User1Feature` branch in **Branch** browser. Then click **Merge** to execute the merge operation.



4. To confirm if the changes are merged successfully into User2Feature branch, select the file in current folder browser and click **Compare to Ancestor**.



Merge Files with Conflicts

If the merged file has conflicts, you can view the file and resolve the conflicts manually. To resolve a merge conflict:

1. At the MATLAB® command prompt, enter:

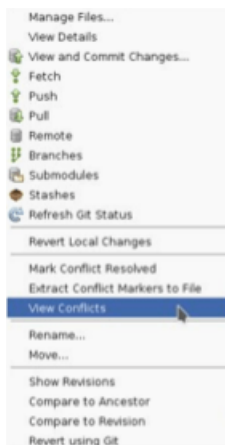
```
slreqCCMergeConflictSetup
```

This helps you to set up two branches, User1Feature and User2Feature, where User2Feature is the current active branch.

2. To inspect the changes in each branch, switch to that branch and right-click the file in the current folder browser and select **Compare To Ancestor**.

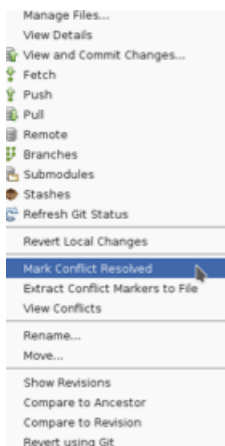
3. Select the **User1Feature** branch and click **Merge** to execute merge command. Observe that MATLAB reports a conflict.

4. The Merge tool automatically merges non-conflicting changes. To view the conflicting changes, right-click the file in current folder browser and click **View Conflicts**.



5. To manually resolve conflicts, open the requirement set in **Requirements Editor** and make the changes.

6. Right-click the file and select **Mark Conflicts Resolved**.



7. Click **Commit** to merge the changes.

8. Right-click the file and select **Compare to Ancestor** to observe the merged changes.

See Also

- “Three-Way AutoMerge Solution for Requirement Set and Link Set” on page 5-15

Track Changes to Test Cases in Requirements Editor

This example explains how to track changes to Simulink Test test cases in Requirements Editor. You make the changes to test cases associated with a controller model of an automobile cruise control system which is managed in a project. After you make changes to the test cases that are linked to the requirements, you track the changes in the Requirements Editor and clear the change issues by using Traceability Matrix window.

Open Test File

1. Open the project. At the MATLAB® command prompt, enter:

```
slreqCCProjectStart
```

2. Open **Requirements Editor**. In the **Apps** tab, under **Verification, Validation, and Test**, click **Requirements Editor**. Open the requirements set file `crs_req_func_spec`.

3. Open the Test Manager. In the **Apps** tab, click **Simulink Test**. In the **Test** tab, click **Simulink Test Manager**.

4. In the Test Manager, from the `tests` folder, open the `DriverSwRequest_Tests.mldatx` test file.

5. In the Test Browser pane, expand the test case hierarchy. The test file contains the test cases for several of the requirements in the `crs_controller` model. The `Enable Button` test case is linked to the requirement `Enable Switch Detection (ID 1.6)`. Similarly, the `Increment button hold` test case is linked to the requirement `Increment Long switch Detection (ID 1.8.2)`.

Make Changes to Test Cases

When you make changes to test cases linked to the requirements, the Requirements Editor highlights the corresponding link in the **Links** section. Follow these steps to make changes in the test cases:

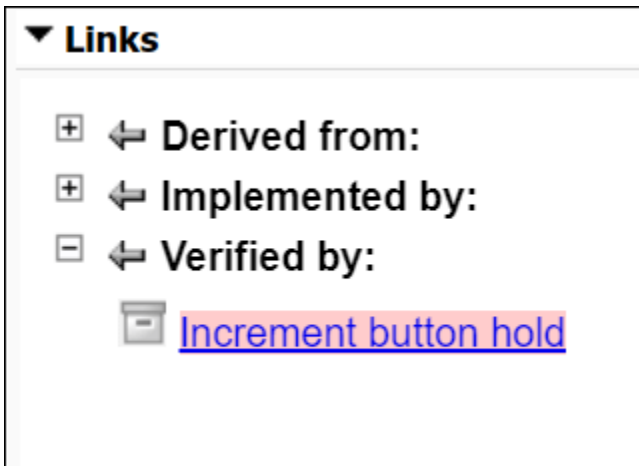
1. In the Test Manager, click the test case `Enable Button`, then expand the `CUSTOM CRITERIA` in the right pane. In the last line of code, change `test.verifyTrue(all(compOut))` to `test.verifyTrue(any(compOut))`.

2. Save the changes.

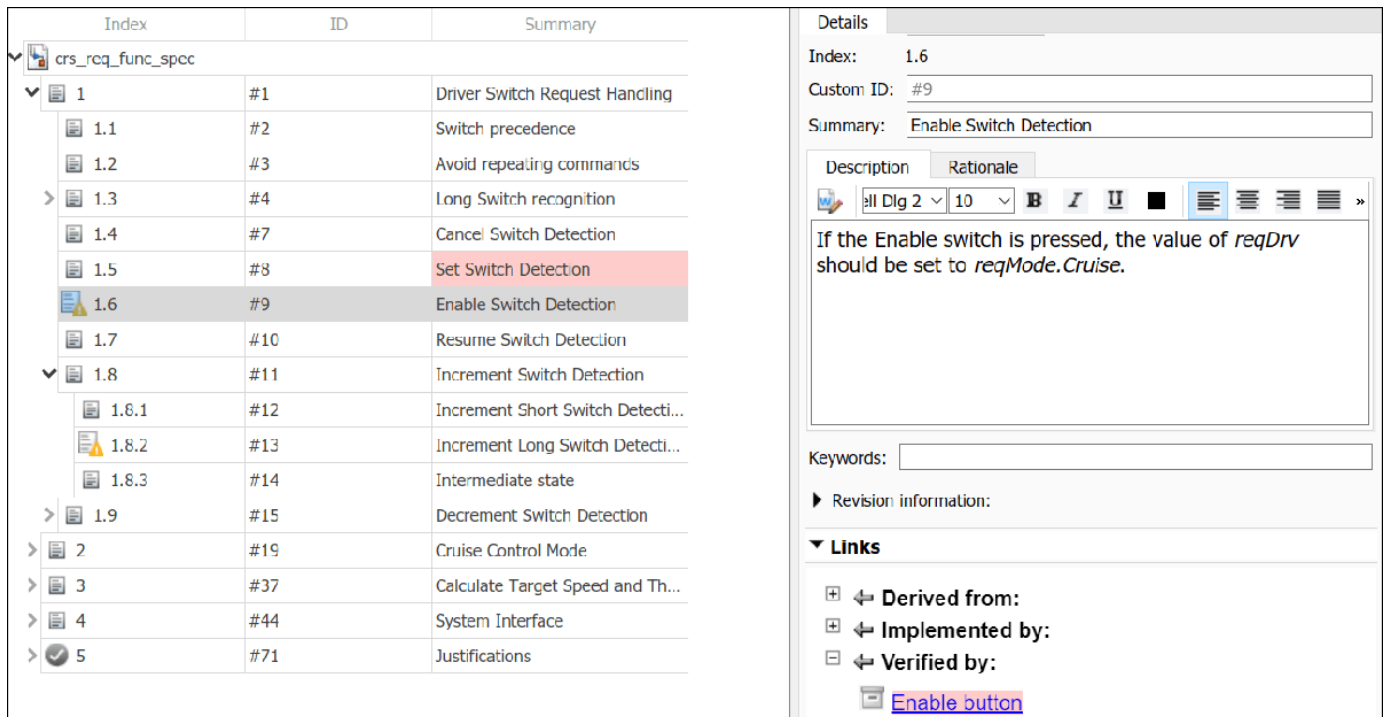
3. Similarly, click the `Increment button hold` test case, then expand the `SYSTEM UNDER TEST` section. Under `SIMUATION SETTINGS AND RELEASE OVERRIDES`, select and set the value **Stop Time** to 20.

4. Save the changes.

5. In the Requirements Editor, click requirement 1.6. The **Links** section shows the changed test objects in red.



6. Requirement 1.6 is highlighted because it links to a changed test case. Click requirement 1.6. In the right pane, in the **Links** section, the change issue icon indicates a change issue.



7. Under **Links**, select the link and then click on **Change Information** to view the change issue.

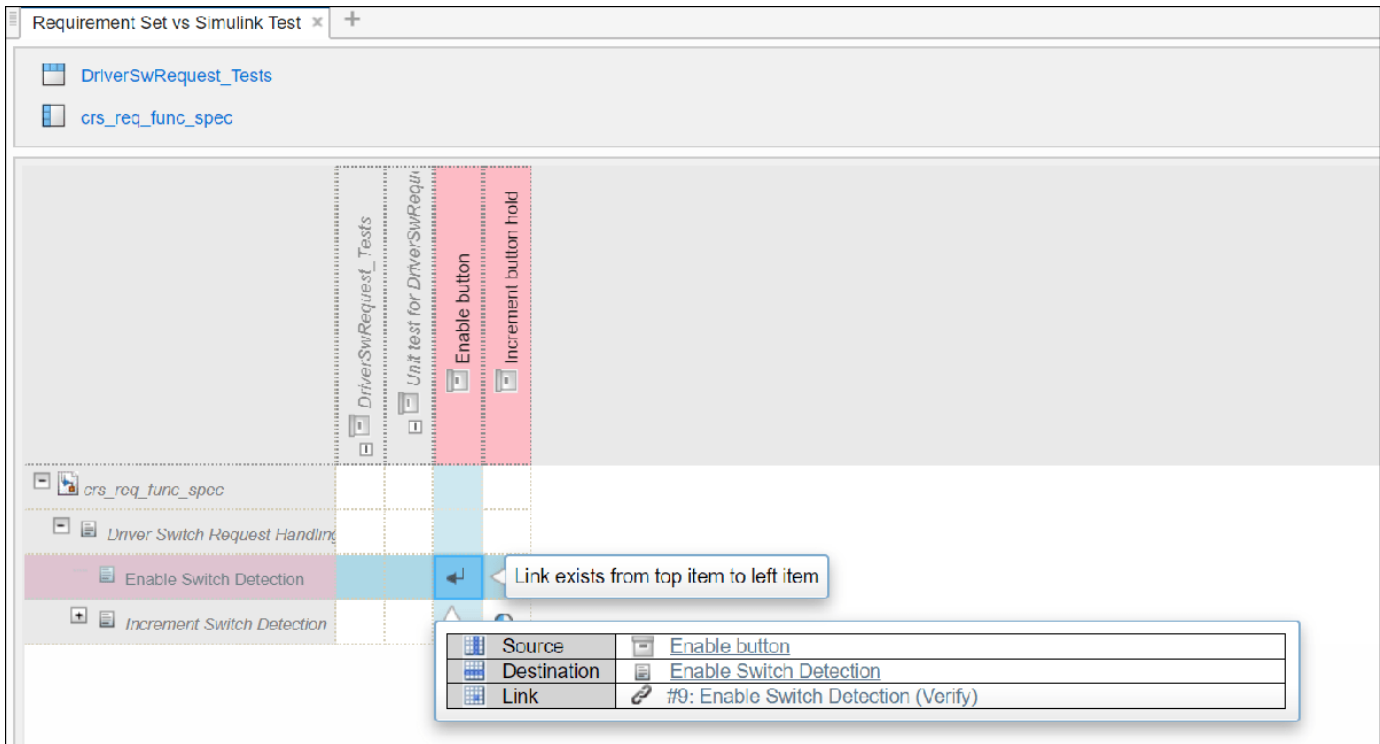
Clear Change Issues for test cases

To clear the change issues using the Traceability Matrix window:

1. Open **Traceability Matrix**. In the **Requirements Editor**, click **Traceability Matrix**.
2. In the Traceability Matrix window, click **Add**. In the Select Artifacts dialog, set **Left** to `crs_req_func_spec.s1reqx` and set **Top** to `DriverSwRequest_Tests.mldatx`. Then click **Generate Matrix**. The window generates a traceability matrix with the specified requirement set on the left and the test case file on the top.

3. To identify unlinked items and changes, click **Highlight Missing Links** and select **Highlight Changed Links** and **Show Changed Links Only**.

4. In the Filter Panel pane, under **Cell > Change Tracking**, click **With Change Issues**. The matrix highlights the row, column, and cell that corresponds to the link with a change issue.



5. To clear the change issue, select the cell containing the link and click **Clear Change Issue**. You can export the traceability matrix as an HTML report or as a MATLAB variable that contains the table data. Generate the HTML report by clicking **Export > Generate HTML Report**. Name and save the report.

See Also

- “Track Changes to Requirement Links” on page 5-3

See Also

More About

- “Track Changes to Requirement Links” on page 5-3
- “Track Requirement Links with a Traceability Matrix” on page 3-5

Track Changes to MATLAB Code Using Requirements Editor

This example uses the project `slreqShortestPathProjectStart` to show how to make changes to MATLAB® files and view the change issues against the requirements linked to the piece of MATLAB® code.

Open the project

```
slreqShortestPathProjectStart
```

The project contains:

- Requirement sets for functional (`shortest_path_func_reqs.slreqx`) and test requirements (`shortest_path_tests_reqs.slreqx`).
- A MATLAB® algorithm (`shortest_path.m`)
- MATLAB® unit tests (`graph_unit_tests.m`)
- Links from MATLAB® code lines to requirements, and `.slmx` files stored in the `src` and `tests` folders.

Open MATLAB® Function

The `shortest_path` function tests the validity of the function inputs and then uses the Dijkstra algorithm to calculate the number of edges in the shortest path between two nodes on a graph. Open the function.

```
open("shortest_path.m");
```

Open Requirement Files

Open the requirement sets in the Requirements Editor.

The `shortest_path_func_reqs` requirement set captures the functional behavior of the `shortest_path` function. Open the functional specification requirements file.

```
slreq.open("shortest_path_func_reqs.slreqx");
```

The `shortest_path_tests_reqs` requirement set contains test requirements that describe the functional behavior to be tested by a test case. The test requirements are derived from the functional requirements. There are test requirements for the nominal behavior and for the invalid conditions. Open the test specification requirements file.

```
slreq.open("shortest_path_tests_reqs.slreqx");
```

Index	ID	Summary
shortest_path_func...		
1	#1	Overview
2	#2	Functional behavior
2.1	#12	Nominal behavior
2.2	#8	Exceptional conditions
shortest_path_tests...		
1	#1	Overview
2	#2	Test Cases

Make Changes to MATLAB® File

For the purpose of this example, change the value of the error code returned by the `shortest_path.m` function from `-99` to `-44` whenever the input start index or the end index is greater than the number of nodes in the graph. In the `shortest_path.m` file, on line number 37, change the value of `ErrorCode` from `-99` to `-44`.

As you have changed the source, you also need to change the tests. Open `graph_unit_tests.m` file using this command.

```
open("graph_unit_tests.m")
```

In lines 16, 32, and 113, change the expected output value from `-99` to `-44` to match the source logic. Save the `shortest_path.m` and `graph_unit_tests.m` files.

Detect Change Issues

Click **Refresh** on the Requirements Editor toolbar. The linked requirements now show a new icon indicating that an incoming link to that requirement has a change issue.

The screenshot shows the Requirements Editor interface. On the left, a table lists requirements with columns for Index, ID, Summary, and Verified status. Requirement #18 is highlighted. On the right, the 'Requirement: #18' details pane is open, showing properties like Type (Functional), Index (2.2.1), and Summary (Test with invalid startIdx > N). The 'Links' section shows a link to the code section of graph_unit_test.m, with a tooltip indicating a change issue found for this link.

Index	ID	Summary	Verified
1	#1	Overview	
2	#2	Functional behavior	
2.1	#12	Nominal behavior	
2.2	#8	Exceptional conditions	
2.2.1	#9	Returns -9 for invalid ad...	
2.2.2	#10	Returns -19 if the start n...	
2.2.3	#11	Returns -29 if end node ...	
2.2.4	#6	Returns -99 if startIdx o...	
2.2.5	#7	Returns -199 if startIdx ...	
2.2.6	#5	Returns 0 if startIdx =-...	
2.2.7	#4	Returns -1 if no path fro...	
shortest_path_tests_reqs			
1	#1	Overview	
2	#2	Test Cases	
2.1	#22	Nominal Mode Tests	
2.2	#15	Tests for invalid conditio...	
2.2.1	#18	Test with invalid startIdx...	
2.2.2	#14	Test with invalid startIdx...	
2.2.3	#20	Test with invalid endIdx ...	
2.2.4	#21	Test with invalid endIdx ...	
2.2.5	#11	Test a degenerate graph...	

See Change in Link Details for Requirement

Navigate to the link in the **Links** pane to see that the Requirements Editor has detected a change issue from the MATLAB® code that you changed.

Select requirement 2.2.1 in `shortest_path_tests_reqs.slreqx` and click the red symbol in the **Links** section. It will navigate you to the code section of `graph_unit_test.m` and highlight it.

```
function check_invalid_start_2(testCase)
    adjMatrix = graph_unit_tests.graph_straight_sec();
    startIdx = 12;
    endIdx = 2;
    expOut = -44;
    verify_path_length(testCase, adjMatrix, startIdx, endIdx, expOut, 'Invalid start index, idx>NodeCnt');
end
```

See Change in Links View for Changed Link

From **Requirements** pane, select **Show Links** to navigate from the requirements view to the links view. Click the highlighted links to see the changed source along with the updated revision information under the **Change Information** section.

Label	Source	Type	Destination
shortest_path.slmx	Changed source: 1/11		Changed destination: 0/11
link #2	% Self-loop path is alw...	Implements	Returns 0 if startIdx == endIdx
link #3	if nodeIdx == endIdx ...	Implements	Returns -1 if no path from star...
link #4	pathLength = distance(no...	Implements	Returns the number of edges i...
link #5	% Check the validity of th...	Implements	Returns -9 for invalid adjacenc...
link #6	% Check the validity of th...	Implements	Returns -19 if the start node is...
link #7	% Check the validity of th...	Implements	Returns -29 if end node is enc...
link #8	% Start or end node is to...	Implements	Returns -99 if startIdx or endI...
link #9	% Start or end node is to...	Implements	Returns -199 if startIdx or end...
link #10	function out = isNodeVali...	Implements	Returns -19 if the start node is...
link #11	function out = isNodeVali...	Implements	Returns -29 if end node is enc...
link #12	function out = isAdjMatrix...	Implements	Returns -9 for invalid adjacenc...
graph_unit_tests.s...	Changed source: 5/26		Changed destination: 0/26
link #1	classdef graph_unit_tests ...	Related to	Overview
link #2	function check_invalid_idx...	Verifies	Test a degenerate graph witho...
link #3	function check_invalid_sta...	Verifies	Test with invalid startIdx <1
link #4	function check_invalid_sta...	Verifies	Test with invalid startIdx > N
link #5	function check_invalid_en...	Verifies	Test with invalid endIdx > N
link #6	function check_invalid_en...	Verifies	Test with invalid endIdx < 1
link #7	function check_edgeless_s...	Verifies	Test a graph starting from a n...
link #8	function check edgeless e...	Verifies	Test a graph ending on a node...

▼ Properties

Filepath: C:\Users\vshukja\MATLAB\Projects\examples\ShortestPath9\tests\graph_unit_tests.slmx
Artifact: C:\Users\vshukja\MATLAB\Projects\examples\ShortestPath9\tests\graph_unit_tests.m
Revision: 24
Created by: baldrich
Created on: 01-Oct-2021 19:18:04
Modified by: vshukja
Modified on: 20-Jun-2022 11:42:00
Description: Requirements data file, stores named bookmarks and outgoing link information.

▼ Change Information

Total number of links: 26
Total links with changed Source: 5
Total links with changed Destination: 0

Clear All

► Custom Attribute Registries

Run Tests

Optionally, you can run tests to confirm that the changed code is correct. Right-click the `shortest_path_tests_reqs.slmx` requirement set and click **Run Tests**. The result shows that all tests for this change have passed.

Index	ID	Summary	Verified
1	#1	Overview	
2	#2	Functional behavior	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.1	#12	Nominal behavior	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2	#8	Exceptional conditions	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2.1	#9	Returns -9 for invalid ad...	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2.2	#10	Returns -19 if the start n...	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2.3	#11	Returns -29 if end node ...	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2.4	#6	Returns -99 if startIdx o...	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2.5	#7	Returns -199 if startIdx ...	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2.6	#5	Returns 0 if startIdx = =...	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2.7	#4	Returns -1 if no path fro...	<div style="width: 100%; height: 10px; background-color: green;"></div>
shortest_path_tests_reqs			<div style="width: 100%; height: 10px; background-color: green;"></div>
1	#1	Overview	
2	#2	Test Cases	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.1	#22	Nominal Mode Tests	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2	#15	Tests for invalid conditio...	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2.1	#18	Test with invalid startIdx...	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2.2	#14	Test with invalid startIdx...	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2.3	#20	Test with invalid endIdx ...	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2.4	#21	Test with invalid endIdx ...	<div style="width: 100%; height: 10px; background-color: green;"></div>
2.2.5	#11	Test a degenerate graph...	<div style="width: 100%; height: 10px; background-color: green;"></div>

▼ Properties

Type: Functional
Index: 2.2.1
Custom ID: #18
Summary: Test with invalid startIdx > N

Description Rationale

Arial 10

Keywords:

Revision information:

▼ Links

Verified by:

function check_invalid_start_2(testCase) ✓

Clear Change Issue

From Requirements pane, make sure that **Show Links** is selected in order to clear the change issue for the requirement sets. Select `shortest_path_tests_reqs.slmx` and click **Clear issue** under **Change Information** section. This clears the change issue in the source code of the `shortest_path.m` file.

Since there are multiple links from the `graph_unit_tests.slmx` test file, you can right-click the `graph_unit_tests.slmx` link set and select **Clear all change issues** to clear all the change issues at once. Save both linkset files after clearing the change issue.

Click **Refresh** in the Requirements Editor to see that the warning icon on the requirements does not show anymore.

Cleanup

Clear the open requirement sets and link sets.

```
slreq.clear;
```

See Also

More About

- “Track Changes to Requirement Links” on page 5-3

Verify Safety Requirements Linked to Test Steps Using Functional Requirements

This example shows you how to verify safety requirements linked to test sequences or assessment steps in the Requirements Editor. You first simulate the test cases associated with the functional requirements and then verify the safety requirements linked to verify statements in the test assessment block. This example uses a Cruise Control Model project.

Open the Project

This project contains:

- Requirement sets for safety and functional requirements (`crs_req_safety_spec.slreqx` and `crs_req_func_spec.slreqx`, respectively)
- A Simulink model (`crs_controller`)
- A test harness for the model (`crs_controller_Harness1`)
- A test assessment block
- Test cases (`crs_controller_tests`)
- Links from test assessment steps to requirements, and link set files stored in the `src` and `tests` folders

Follow these steps:

1. Open the project. The project includes the model and supporting files.

```
slreqCCProjectStart
```

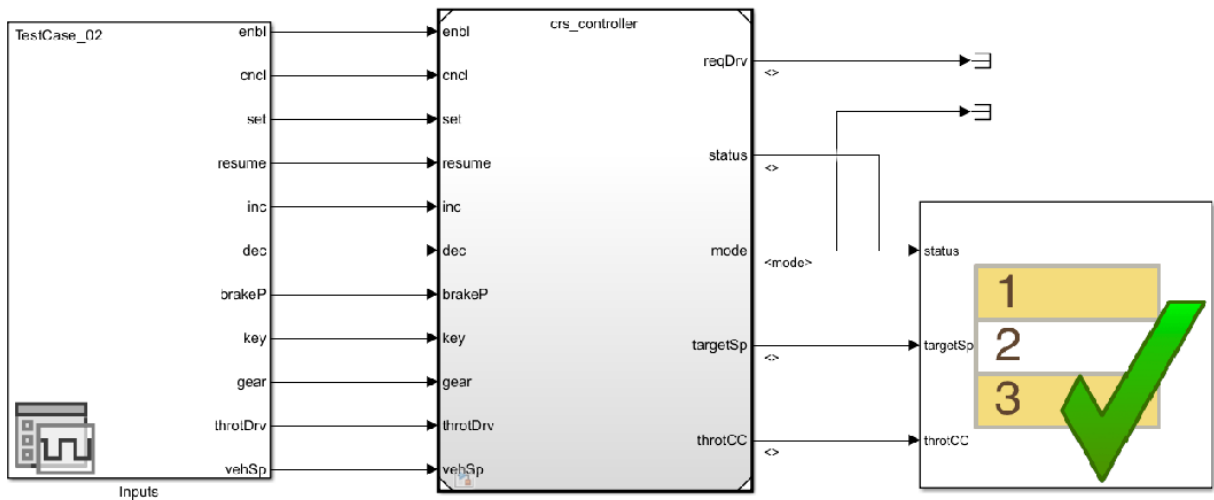
2. Open the safety requirements for the `crs_controller` model.

```
slreq.open("crs_req_safety_spec.slreqx");
```

3. Open the Simulink model and the harness for the model.

```
open_system("crs_controller");  
sltest.harness.open("crs_controller", "crs_controller_Harness1");
```

crs_controller_Harness1



4. In the crs_controller_Harness1 harness model, double-click the **Test Assessment** block to open the test assessments. Each test step in a test assessment is linked to requirements from the crs_req_safety_spec.slreqx requirement set.

Observe Requirements Linked to Test Steps

1. To view the requirements associated with the test step, right-click the step and then click **Requirements**. Click Maximum Throttle Value to highlight the step in the test assessments and the requirement in the Requirements Editor.

Step	Transition	Next Step	Description
<pre> : Assessments_WhenActivated % verify Target Speed verify(targetSp >= 40); verify(targetSp <= 100); % verify Throttle values verify(throtCC >= 0); verify(throtCC <= 100); </pre>	1. status == false	Asses... ▼	
<pre> Assessments_WhenNotAc % verify Target Speed verify(targetSp >= 0); verify(targetSp <= 100); % verify Throttle values verify(throtCC >= 0); verify(throtCC <= 100); </pre>			<ul style="list-style-type: none"> 1. "Maximum Throttle Value" 2. "Minimum Throttle Value" 3. "Maximum Target Speed" 4. "Minimum Target Speed"
	<ul style="list-style-type: none"> Add step before Add step after Add sub-step Delete step 		<ul style="list-style-type: none"> Link to Selection in Requirements Browser Link to Selection in MATLAB Editor Link to Current Test Case Link to Selection in Word Link to Selection in Excel
	<ul style="list-style-type: none"> Cut text Copy text Paste text 		<ul style="list-style-type: none"> Select for Linking with Simulink Add Link to Selected Object(s)
	<ul style="list-style-type: none"> Indent step Outdent step 		<ul style="list-style-type: none"> Open Outgoing Links dialog ... Delete All Outgoing Links ...
	<ul style="list-style-type: none"> Requirements <input type="checkbox"/> When decomposition <input type="checkbox"/> Break while executing step 		<ul style="list-style-type: none"> Copy URL to Clipboard

2. In the **Links** section of Requirements Editor, observe the link coming from the test step. The link type of the requirements and links show as **Verified by**.

Index	ID	Summary	Verified																
<div style="display: flex; align-items: flex-start;"> <div style="margin-right: 10px;"> ✓ crs_req_safety_spec </div> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">#1</td> <td>Maximum Throttle Value</td> <td style="background-color: #e0e0e0;"></td> </tr> <tr style="background-color: #e0e0e0;"> <td style="text-align: center;">2</td> <td style="text-align: center;">#2</td> <td>Minimum Throttle Value</td> <td style="background-color: #e0e0e0;"></td> </tr> <tr> <td style="text-align: center;">3</td> <td style="text-align: center;">#3</td> <td>Maximum Target Speed</td> <td style="background-color: #e0e0e0;"></td> </tr> <tr> <td style="text-align: center;">4</td> <td style="text-align: center;">#4</td> <td>Minimum Target Speed</td> <td style="background-color: #e0e0e0;"></td> </tr> </table> </div>				1	#1	Maximum Throttle Value		2	#2	Minimum Throttle Value		3	#3	Maximum Target Speed		4	#4	Minimum Target Speed	
1	#1	Maximum Throttle Value																	
2	#2	Minimum Throttle Value																	
3	#3	Maximum Target Speed																	
4	#4	Minimum Target Speed																	
<div style="display: flex; align-items: flex-start;"> <div style="margin-right: 10px;"> > crs_req_func_spec </div> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;"></td> <td style="width: 10%;"></td> <td style="width: 80%;"></td> <td style="width: 10%;"></td> </tr> </table> </div>																			

▼ Properties

Type: Functional

Index: 2

Custom ID: #2

Summary: Minimum Throttle Value

Description Rationale

MS Shell Dlg 2

Minimum Throttle value must not be less than 0.

Keywords:

▶ Revision information:

▼ Links

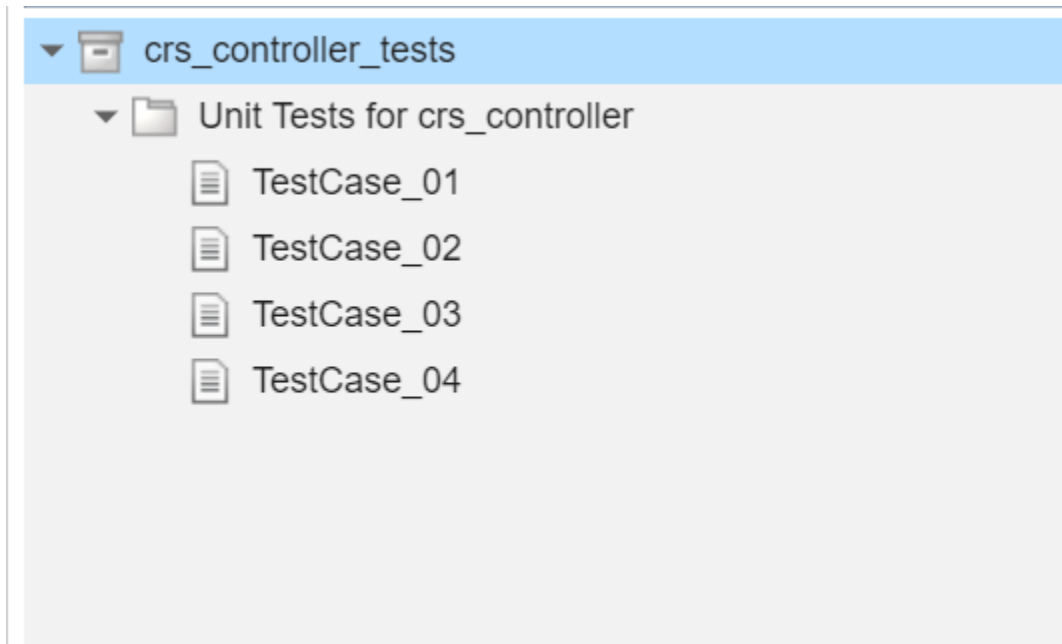
☰ ← Verified by:

- [Assessments_WhenActivated](#) ?
- [Assessments_WhenNotActivated](#) ?

Run Tests for Functional Requirements

The `crs_controller_tests` file contains the test cases related to the functional requirements. Complete the following steps to test the functional requirements.

1. To open the Test Manager, in the Test Harness window, click **Simulink Test Manager**.
2. In the Test Manager, click **Open Test File** that will direct you to `crs_controller_tests` test file under **Tests** folder.



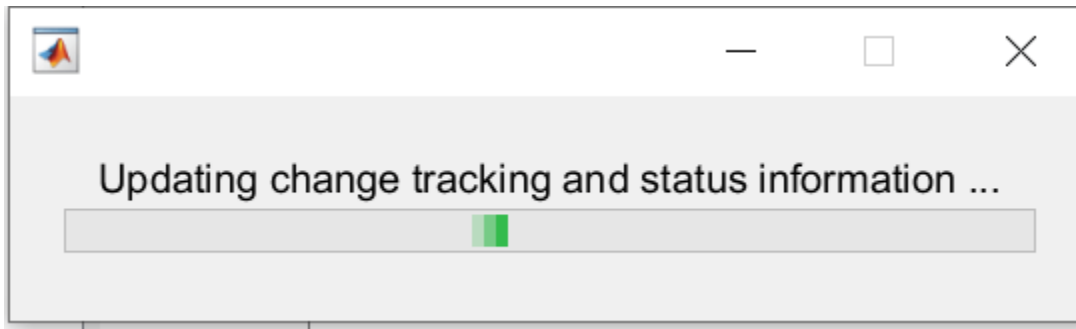
3. To run the tests, in the **Test Browser** tab, right-click `crs_controller_tests` and click **Run**. Simulink Test runs these tests and shows that all the tests have passed. You can also view the status of the verify statements associated with the test steps in the **Results and Artifacts** tab.

▼ Results: 2022-Aug-08 13:27:44	4 ✓
▼ Unit Tests for crs_controller	4 ✓
▶ TestCase_01	✓
▶ TestCase_02	✓
▶ TestCase_03	✓
▶ TestCase_04	✓

View Verification Status

You can now see if the tests ran in the **Test Manager** have been verified in the Requirements Editor.

1. In the **View** tab in the Requirements Editor, click **Refresh**. This updates the verification status of the requirements linked to the test steps.



2. Go to the Requirements Editor and click **Refresh**. Observe the verification status of all the linked requirements are passed. The unit test `Unit_Tests_for_crs_controller` ran for the functional requirements verified the safety requirements `crs_req_safety_spec.slreqx` in the Requirements Editor.

Index	ID	Summary	Verified
✓ crs_req_safety_spec			
1	#1	Maximum Throttle Value	
2	#2	Minimum Throttle Value	
3	#3	Maximum Target Speed	
4	#4	Minimum Target Speed	
✓ crs_req_func_spec			
1	#1	Driver Switch Request H...	
2	#19	Cruise Control Mode	
3	#37	Calculate Target Speed a...	
4	#44	System Interface	
5	#71	Justifications	

Summary: Calculate Target Speed and Throttle value

Description Rationale

Calculate the target speed and the throttle value.

Input:

- Operation mode
- Vehicle speed
- Throttle value from the driver's accelerator pedal

Keywords:

Revision information:

Links

- Implemented by:
 - [TargetSpeedThrottle](#)
- Verified by:
 - [Unit Tests for crs_controller](#)
 - Passed

Note: You can also run default simulation for the harness and verify safety requirements by viewing Simulation results in **Simulink Data Inspector**. The simulation, however will not run all the tests associated with the test file. Requirements Editor, on the other hand, fetches and assimilates verification data from **Simulink Data Inspector** runs. If you do not want certain runs to be considered in the final metric in the Requirements Editor, you can run and delete tests from the **Simulink Data Inspector**.

Cleanup

Clear the open requirement sets and close the current project.

```
slreq.clear();
bdclose('all');
slproject.closeCurrentProject();
```

Related Topics

“Track Changes to Requirement Links” on page 5-3

“Track Changes to Test Cases in Requirements Editor” on page 5-21

Publish and Save Printable Report of Comparison Results

You can analyze the comparison results of requirement or link set files by publishing the comparison report from the **Comparison** tool or from the MATLAB command line.

Publish Report from Comparison Tool

To save a printable version of the comparison report for requirement or link set files from the tool:

- 1 On the **Comparison** tab, click **Publish** and select one of these formats: **Publish to HTML**, **Publish to Word**, or **Publish to PDF**.

The Save dialog box opens, where you can save a printable version of the comparison report for requirement or link set files.

- 2 By default, the file name for all the file types is specified as `file1_file2` and location is current folder. You can change the file name and location.

By default, the report shows only the changed requirements or links. To include all requirements in the report, clear the **Show Only Changed** option on the **Filters** tab in the toolstrip.

Publish Report from Command Line

To publish and save the comparison report from the command line, use the `visdiff` function. You can manipulate the comparison report at the command line by specifying an output argument.

- 1 Compare two requirement sets `crs_req_func_spec_01.slreqx` and `crs_req_func_spec_02.slreqx` and return a comparison object.

```
comparison = visdiff('crs_req_func_spec_01.slreqx','crs_req_func_spec_02.slreqx');
```

- 2 Use the `publish` function to publish the report. By default, the function publishes the report in the HTML format and names it `file1_file2.html`.

```
file = publish(comparison);  
web(file)
```

To save the report in a different format and by a different name, use the `format` and `Name` arguments. For example, to save the report as a PDF file named `myreport`, use this syntax:

```
file = publish(comparison,"format","PDF","Name","myreport");  
web(file)
```

Alternatively, you can specify the name-value pairs in an `options` structure:

`publish(comparison,options)`. For more information on the available name-value pairs for the `publish` function, see `visdiff`.

See Also

“Compare Requirement Sets Using Comparison Tool” on page 12-50 | `visdiff`

Requirements Management Interface Setup

- “Configure Requirements Toolbox for Interaction with Microsoft Office and IBM DOORS” on page 6-2
- “Requirements Link Storage” on page 6-4
- “Supported Requirements Document Types” on page 6-8
- “Requirements Settings” on page 6-10
- “Migrating Requirements Management Interface Data to Requirements Toolbox” on page 6-16

Configure Requirements Toolbox for Interaction with Microsoft Office and IBM DOORS

Requirements Toolbox communicates with external tools such as Microsoft Office, IBM Rational DOORS, and IBM DOORS Next so that you can import requirements and establish links between requirements and Model-Based Design items such as Simulink model elements and tests.

You can configure MATLAB and Simulink to:

- Use ActiveX® controls for navigation from Microsoft Office documents to Simulink models (Windows only).
- Use Requirements Toolbox with IBM Rational DOORS software (Windows only).
- Use Requirements Toolbox with IBM DOORS Next web server.

Configure Requirements Toolbox for Microsoft Office

When you work with older requirements documents that include ActiveX controls inserted by previous versions of Simulink, register ActiveX controls. More recent Simulink versions use HTTP hyperlinks to navigate from Microsoft Office to Simulink.

- 1 Run MATLAB as an administrator.
- 2 At the command prompt, enter:

```
rmi setup
```
- 3 Press Y to register the current MATLAB installation as an ActiveX Automation Server.

Configure Requirements Toolbox for IBM Rational DOORS

You must configure your IBM Rational DOORS installation to communicate with MATLAB.

- 1 Run MATLAB as an administrator.
- 2 At the command prompt, enter:

```
rmi setup doors
```
- 3 Press Y to complete the ActiveX Automation Server setup.
- 4 Verify the path to your IBM Rational DOORS installation. The setup utility will list the DOORS client installations found on your system. You can select a file path from the list or use the option to manually enter the path to the folder.
- 5 If the DOORS installation was not detected in the previous step, press 2 to enter the installation folder.

Tip If Requirements Toolbox still does not communicate after performing this setup, try the setup process described in “Configure Requirements Toolbox for IBM Rational DOORS Software” on page 8-2.

Configure an IBM DOORS Next Server for Integration with Requirements Toolbox

To interface with IBM DOORS Next, you must configure the server for integration with Requirements Toolbox by configuring custom extensions.

Additionally, at the start of each MATLAB session, you must configure your session to interface with IBM DOORS Next by using `slreq.dngConfigure`. For more information, see “Configure IBM DOORS Next Session” on page 1-35.

Configure the IBM DOORS Next Server for Custom Extensions

In order to integrate your IBM DOORS Next server with Requirements Toolbox, you must configure the server for custom extensions and enable dropins. Additionally, you can install the MathWorks Requirements Toolbox widget, which enables you to propagate selection information from IBM DOORS Next to Requirements Toolbox.

- 1 In the Windows File Explorer, navigate to the folder `toolbox\slrequirements\slrequirements\resources` in your MATLAB installation.
- 2 Copy the `mwWidgetForDNG` folder into the `extensions` subfolder of your IBM DOORS Next installation. The location of this folder depends on your server version.
- 3 Configure the DOORS Next server for custom extensions and enable dropins, and then restart the server. For more information, see *Hosting extensions* on the IBM website.
- 4 After copying the `mwWidgetForDNG` folder to your server, add the **MathWorks Requirements Toolbox** widget to the **Mini Dashboard** in DOORS Next. In the **Mini Dashboard**, select **Add Widget > Add OpenSocial Gadget**.
- 5 Specify the URL to `dngsllink_config.xml` that corresponds to the `extensions/mwWidgetForDNG` subfolder in your server installation folder.

For example, if you have Liberty server installed on Windows, the `extensions` subfolder may be located in: `C:/Program Files/IBM/JazzTeamServer/server`. The corresponding URL for adding the widget will be: `https://JAZZSERVERNAME:9443/extensions/mwWidgetForDNG/dngsllink_config.xml`.

- 6 Click **Add Widget**. The **Mini Dashboard** now displays the widget.

See Also

`slreq.dngConfigure`

More About

- “Configure Requirements Toolbox for IBM Rational DOORS Software” on page 8-2

External Websites

- [Hosting extensions](#)

Requirements Link Storage

When you create a link from a Model-Based Design item to a requirement, Requirements Toolbox stores the link information in an external SLMX file with a name that combines the source artifact base name and the source artifact extension, separated by a tilde. For more information, see “Link Storage” on page 3-31.

When you create a link from a Simulink model to a requirement, you can store the links internally to the model or as an external file. External storage does not modify your model when you create or modify requirements links.

To specify the requirements link storage setting:

- 1 Open the Requirements Settings. In the **Apps** tab, click **Requirements Viewer**. In the **Requirements Viewer** tab, click **Link Settings**.
- 2 In the Requirements Settings dialog box, select the **Storage** tab.
- 3 Under **Default storage location for traceability data**:
 - To enable internal storage, select **Store internally (embedded in Simulink diagram file)**.
 - To enable external storage, select **Store externally (in a separate *.slmx file)**.

This setting applies immediately, and applies to new models and existing models that do not contain requirements links.

If you open a model that already has requirements links, the RMI uses the storage mechanism you used previously with that model, regardless of what your default storage setting is.

When links are stored with the model (internal storage), the time stamp and version number of the model changes every time you modify your requirements links.

Save Requirements Links in External Storage

The Requirements Management Interface (RMI) stores externally stored requirements links in a file whose name is based on the model file. Because of this, before you create requirements links to be stored in an external file, you must save the model with a value file name.

You add, modify, and, delete requirements links in external storage the same way you do when the requirements links are stored in the model file. The main difference is when you change externally stored links, the model file does not change. The asterisk in the title bar of the model window that indicates a model has unsaved changes does not appear when you change requirements links. However, when you close the model, the RMI asks if you want to save the requirements links modifications.

There are several ways to save requirements links that are stored in an external file, as listed in the following table.

Select...	To...
In the Apps tab, click Requirements Manager . In the Requirements tab, click Save All .	Save the requirements links in an external file using a file name that you specify. The model itself is not saved.

Select...	To...
In the Apps tab, click Requirements Manager . In the Requirements tab, click Save Links Only .	Save the requirements links in an external file using the default file name, <i>model_name~mdl.slmx</i> , or to the previously specified file. The model itself is not saved.
In the Simulation tab, click Save .	Save the current requirements links to an external file named <i>model_name~mdl.slmx</i> , or to the previously specified file. Model changes are also saved.
In the Simulation tab, Save > Save As	Rename and save the model and the external requirements links. The external file is saved as <i>new_model_name~mdl.slmx</i> .

Load Requirements Links from External Storage

RMI attempts to load internally stored model requirements links from an *.slmx* file — either the default file or a previously specified file. If no *.slmx* file is found, RMI does not display requirements links.

Your links may be stored in an external file. To load links:

- 1 In the **Apps** tab, click **Requirements Viewer**.
- 2 In the **Requirements Viewer** tab, click **Load Links**.
- 3 Select the file from which to load the requirements links.
- 4 Click **Open** to load the links from the selected file.

Save changes to your links before loading links from another file.

Move Internally Stored Requirements Links to External Storage

If you have a model with requirements links that are stored with the model, you can move those links to an external file. When you move internally stored links to a file, the RMI deletes the internal links data from the model file and saves the model. From this point on, the data exists only in the external file.

- 1 Open the model that contains internally stored requirements links.
- 2 In the **Apps** tab, open **Requirements Manager**.
- 3 In the **Requirements** tab, ensure **Layout > Requirements Browser** is selected.
- 4 In the **Requirements** pane, in the **View** drop-down menu, select **Links**.
- 5 In the **Requirements** tab, click **Link Settings > Save Links As Link Set File**.
- 6 Choose a file name for the new external *.slmx* file and click **OK**.

Move Externally Stored Requirements Links to the Model File

If you have a model with requirements links that are stored in an external file, you can move those links to the model file.

- 1 Open the model that has externally stored requirements links.
- 2 Make sure the right set of requirements links are loaded from the external file.
- 3 In the **Apps** tab, open **Requirements Manager**.
- 4 In the **Requirements** tab, in the **Requirements** pane, select Links from the **View** drop-down.
- 5 In the **Requirements** tab, select **Link Settings > Save Links in Model File**.

An asterisk appears next to the model name in the title bar of the model window indicating that your model now has unsaved changes.

- 6 Save the model with the requirements links.

From this point on, the RMI stores requirements links internally, in the model file. When you add, modify, or delete links, the changes are stored with the model, even if the **Default storage location for requirements links data** option is set to **Store externally (in a separate *.slmx file)**.

External Storage

The first time you create links to requirements in a Simulink model, the RMI uses your designated storage preference. When you reopen the model, the RMI loads the internally stored links, or the links from the external file, as long as the file exists with the same name and location as when you last saved the links.

The RMI allows you to save your links file as a different name or in a different folder. However, when you start with the links file in a nondefault location, you must manually load those links into the model. After you load those links, the RMI associates that model with that file and loads the links automatically when you load this model next time.

As you work with your model, the RMI stores links using the same storage as the existing links. For example, if you open a model that has internally stored requirements links, new links are also stored internally. This is true even if your preference is set to external storage.

Requirements links must be stored either with the model or in an external file. You cannot mix internal and external storage within a given model.

To see an example of the external storage capability using a Simulink model, at the command line, enter:

```
slvndemo_powerwindow_external
```

Guidelines for External Storage of Requirements Links

Follow these guidelines when storing requirements links in an external file.

- When sharing models, use the default name and location.

By default, external requirements are stored in a file named *model_name*.slmx in the same folder as the model. If you give your model to others to review the requirements traceability, give the reviewer both the model and .slmx files. That way, when you load the model, the RMI automatically loads the links file.

- Do not rename the model outside of Simulink.

If you need to re-save the model with a new name or in a different location, in the **Simulation** tab, click **Save As**. Selecting this option causes the RMI to re-save the corresponding `.slmx` file using the model name and in the same location as the model.

- Be aware of unsaved requirements changes.

If you create new requirements links that are stored externally, your model does not indicate that it has unsaved changes, because the model file itself has not changed. You can explicitly save the links, or, when you close the model, the RMI prompts you to save the requirements links. When you save the model, the RMI saves the links in the external file.

Copying Model Objects and their Linked Requirements

When you copy Simulink and Stateflow objects, their associated requirements links are duplicated by default. Alternatively, you can choose to duplicate requirements links only when the links are highlighted in the Simulink model by following this process:

- 1 In the **Apps** tab, open **Requirements Manager**.
- 2 In the **Requirements** tab, ensure **Layout > Requirements Browser** is selected.
- 3 In the **Requirements** pane, in the **View** drop-down menu, select **Links**.
- 4 In the **Requirements** tab, click **Link Settings > Default Link Storage**.
- 5 Select **Duplicate links only when model requirements are highlighted**.

Alternatively, you can navigate to **Apps** and open **Requirements Viewer**, then click **Link Settings** to view the same setting.

If you select **Duplicate links only when model requirements are highlighted**, your links will be duplicated when you copy model objects and, in the **Requirements** or **Requirements Viewer** tab, the **Highlight links** button is selected. If you don't want to duplicate links when copying model objects, ensure that **Highlight links** is not selected.

To change this setting programmatically, see `rmipref` and its preference “`DuplicateOnCopy`”.

Supported Requirements Document Types

The Requirements Management Interface (RMI) supports linking with external documents of the types listed in the table below. For each supported requirements document type, the table lists the options for requirements locations within the document.

If you would like to implement linking with a requirements document of a type that is not listed in the table below, you can register a custom requirements document type with the RMI. For more information, see “Create a Custom Requirements Link Type” on page 11-8.

Requirements Document Type	Location Options
Microsoft Word 2003 or later	<ul style="list-style-type: none"> • Named item — A bookmark name. The RMI links to the location of that bookmark in the document. The most stable location identifier because the link is maintained when the target content is modified or moved. • Search text — A search string. The RMI links to the first occurrence of that string in the document. This search is not case sensitive. • Page/item number — A page number. The RMI links to the top of the specified page.
Excel 2003 or later	<ul style="list-style-type: none"> • Named item — A named range of cells. The RMI links to that named item in the workbook. The most stable location identifier because the link is maintained when the target content is modified or moved. • Search text — A search string. The RMI links to the first occurrence of that string in the workbook. This search is not case sensitive. • Sheet range — A cell location in a workbook: <ul style="list-style-type: none"> • Cell number (A1, C13) • Range of cells (C5:D7) • Range of cells on another worksheet (Sheet1!A1:B4) <p>The RMI links to that cell or cells.</p>
IBM Rational DOORS	Page/item number — The unique numeric ID of the target DOORS object. The RMI links to that object.
Text	<ul style="list-style-type: none"> • Search text — A search string. The RMI links to the first occurrence of that string within the document. This search is not case sensitive. • Line number — A line number. The RMI links to the beginning of that line.
HTML	<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> ...contents... </pre> <p>in the Location field, enter <code>valve_timing</code> or, from the document index, choose the anchor name.</p> <p>Select the Document Index tab in the “Outgoing Links Editor” on page 11-6 to see available anchors in an HTML file.</p>

Requirements Document Type	Location Options
Web browser URL	<p>The RMI can link to a URL location. In the Document field, type the URL string. When you click the link, the document opens in a Web browser:</p> <ul style="list-style-type: none">• Named item — An anchor name. The RMI links to that location on the Web page at that URL.
PDF	<p>Navigation will open a PDF document but will not scroll to a specific page or bookmark.</p> <p>The RMI cannot create a document index of bookmarks in PDF files.</p>

Requirements Settings

You can manage your RMI preferences in the Requirements Settings dialog box. These settings are global and not associated with a particular model. To open the Requirements Settings dialog box, in the **Apps** tab, click **Requirements Viewer**. In the **Requirements Viewer** tab, click **Link Settings**.

In this dialog box, you can select the:

- **Storage** tab to set the default way in which the RMI stores requirements links in a model. For storage information, see “Requirements Link Storage” on page 6-4.
- **Selection Linking** tab to set the options for linking to the active selection in a supported document. For setting information, see “Selection Linking Tab” on page 6-10.
- **Filters** tab to set the options for filtering requirements in a model. For filtering information, see “Configure Requirements Filtering” on page 6-15.
- **Report** tab to customize the requirements report without using the Report Generator. For setting information, see “Customize Requirements Report Using the RMI Settings” on page 12-20.

Selection Linking Tab

In the Requirements Settings dialog box, on the **Selection Linking** tab, use the following options for linking to the active selection in a supported document.

Options	Description
For linking to the active selection within an external document:	
Enabled applications	Enable selection-based linking shortcuts to Microsoft Word, Excel, or DOORS applications.
Document file reference	Select type of file reference. For information on what settings to use, see “Document Path Storage” on page 12-36.
Apply this keyword to new links	Enter text to attach to the links you create. For more information about user keywords, see “Filter Requirements with User Keywords” on page 6-11.
When creating selection-based links:	
Modify destination for bidirectional linking	Creates links both to and from selected link destination.
Store absolute path to model file	Select to store the absolute path to the Simulink model file.
Use custom bitmap for navigation controls in documents	Select and browse for your bitmap. You can use your own bitmap file to control the appearance of navigation links in your document.
Use ActiveX buttons in Word and Excel (backward compatibility)	Select to use legacy ActiveX controls to create links in Microsoft Word and Excel applications. By default, if not selected, you create URL-based links.

Filter Requirements with User Keywords

- “User Keywords and Requirements Filtering” on page 6-11
- “Apply a User Keyword to a Requirement” on page 6-11
- “Filter, Highlight, and Report with User Keywords” on page 6-12
- “Apply User Keywords During Selection-Based Linking” on page 6-14
- “Configure Requirements Filtering” on page 6-15

User Keywords and Requirements Filtering

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

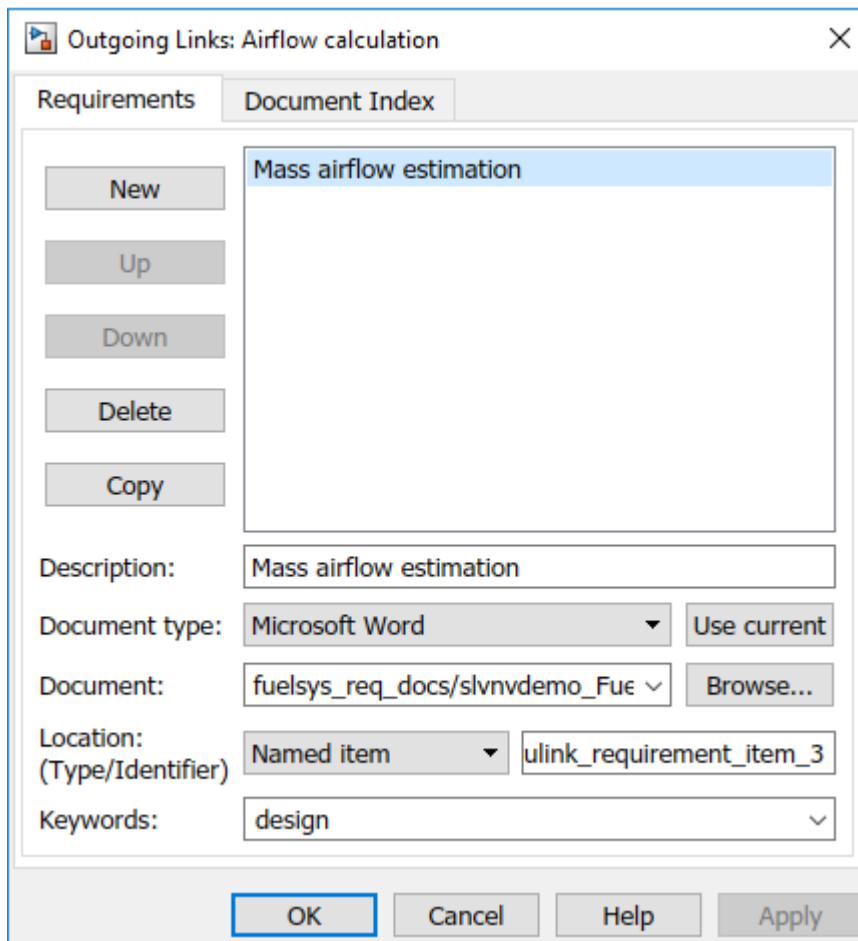
- Highlight or report only those requirements that have a specific user keyword.
- Highlight or report only those requirements that have one of several user keywords.
- Do not highlight and report requirements that have a specific user keyword.

Apply a User Keyword to a Requirement

To apply one or more user keywords to a newly created requirement:

- 1 Open the example model:
`slvnvdemo_fuelsys_officereq`
- 2 Open the fuel rate controller subsystem.
- 3 To open the requirements document, right-click the Airflow calculation subsystem and select **Requirements > Open Outgoing Links dialog**.

The Requirements Traceability Link Editor opens with the details about the requirement that you created.



- 4 In the **Keywords** 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 keyword `test` to specify a second user keyword for this requirement.

User keywords:

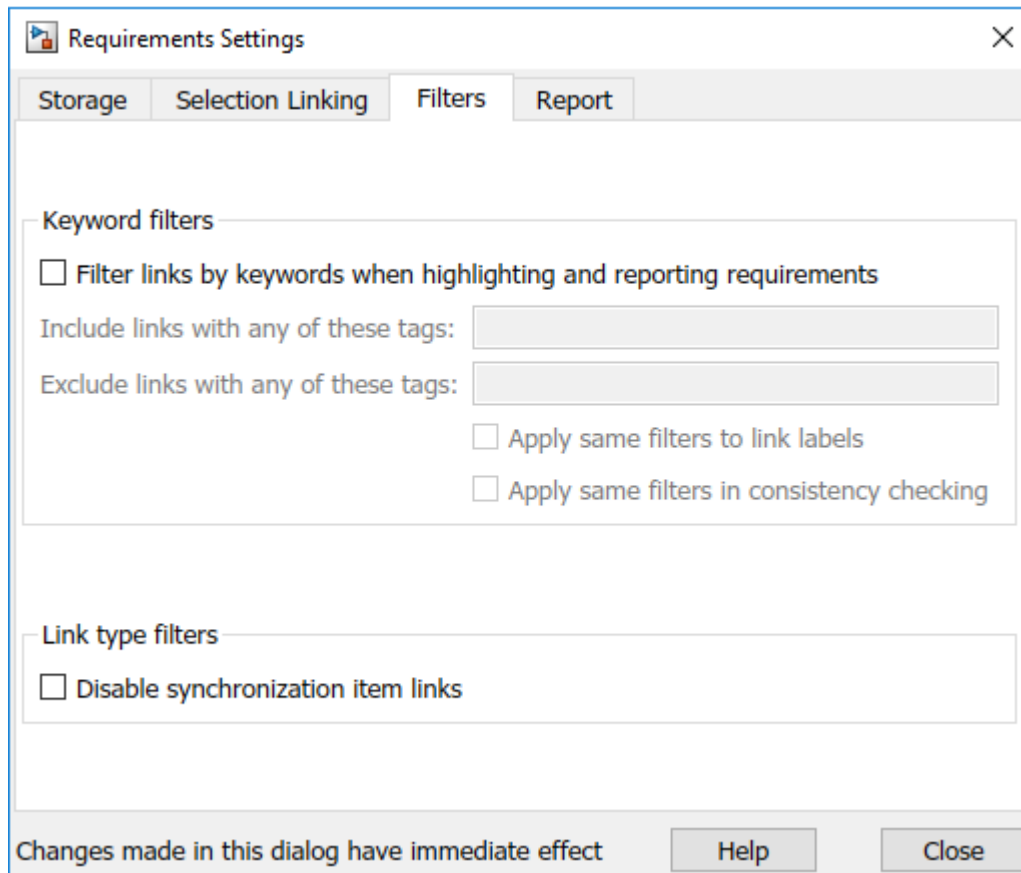
- Are not case sensitive.
- Can consist of multiple words. For example, if you enter `design requirement`, the entire phrase constitutes the user keyword. Separate user keywords with commas.

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

Filter, Highlight, and Report with User Keywords

The `slvnvdemo_fuelsys_officereq` model includes several requirements with the user keyword `design`. This section describes how to highlight only those model objects that have the user keyword, `test`.

- 1 Remove highlighting from the `slvnvdemo_fuelsys_officereq` model. In the **Apps** tab, click **Requirements**. In the **Requirements** tab, click **Highlight Links**.
- 2 Select **Link SettingsLinking Options**.
- 3 In the Requirements Settings dialog box, click the **Filters** tab.



- 4 To enable filtering with user keywords, click the **Filter links by user keywords when highlighting and reporting requirements** option.
- 5 To include only those requirements that have the user keyword, `test`, enter `test` in the **Include links with any of these keywords** field.
- 6 Click **Close**.
- 7 In the **Requirements** tab, click **Highlight Links**.

The RMI highlights only those model objects whose requirements have the user keyword `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 keywords** field, delete `test`. In the **Exclude links with any of these keywords** field, add `test`.

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

- 10 In the **Requirements** tab, select **Share > Generate Model Traceability Report**.

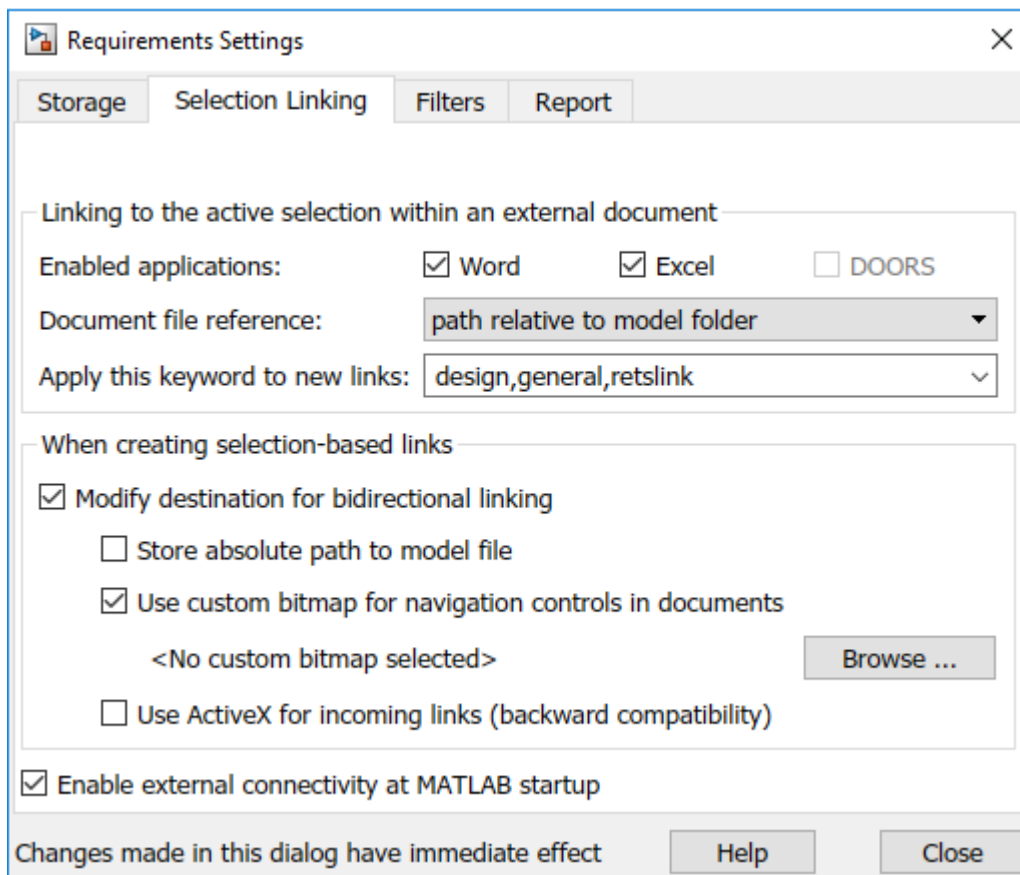
The report does not include information about objects whose requirements have the `test` user keyword.

Apply User Keywords During Selection-Based Linking

When creating a succession of requirements links, you can apply the same user keywords 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 keywords to apply to requirements:

- 1 In the **Requirements Viewer** tab, click **Link Settings**.
- 2 Select the **Selection Linking** tab.

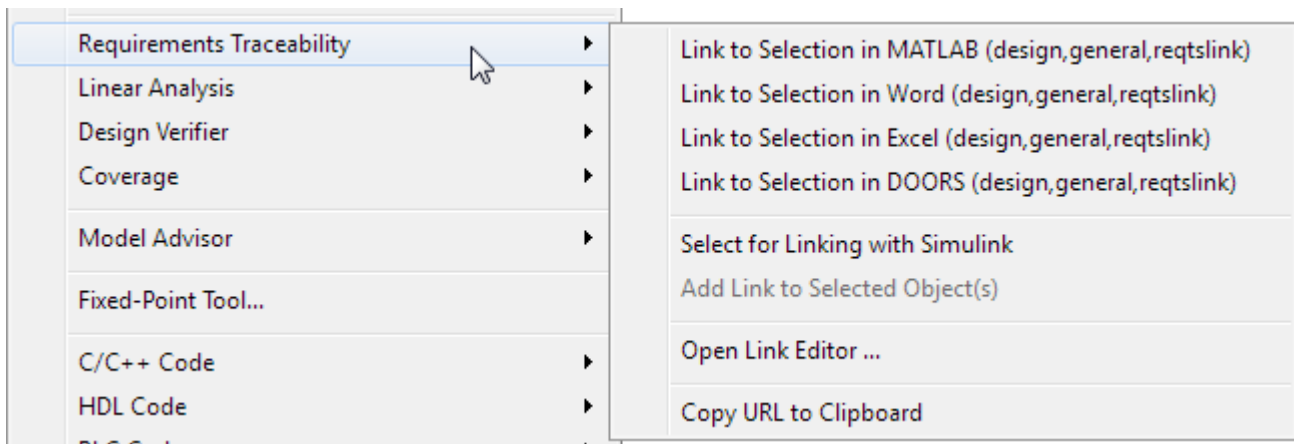


- 3 In the **Apply this keyword to new links** field, enter one or more user keywords, separated by commas.

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

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

The selection-based linking options specify which user keywords the RMI applies to the link that you create. In the following example, you can apply the user keywords `design`, `general`, and `reqtslink` to the link that you create to your selected text.



Configure Requirements Filtering

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

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

Migrating Requirements Management Interface Data to Requirements Toolbox

This example demonstrates the basic steps to update Requirements Management Interface (RMI) links to the format used by the **Requirements Editor**, and the Requirements Browser in the model canvas. Legacy RMI data consists of traceability link information, stored in a separate .req file, or embedded in a Simulink® model.

With Requirements Toolbox™, you can view requirements and links in the model canvas, while preserving existing links from your design elements to external documents. Additionally, you can create requirements, and establish relationships between requirements, model entities, and test cases.

This example uses Windows®.

Workflow

In this example:

- 1 You start with a model that has links to requirements in external documents.
- 2 You create a new requirement set.
- 3 You import the requirements from the external documents, creating requirements in the set that reference the external documents.
- 4 You update the model link destinations to the imported requirements.

Create a Requirement Set

Open the **Requirements Editor**.

```
slreq.editor
```

Create a new requirement set:

- 1 In the **Requirements Editor** toolbar, click **New Requirement Set**.
- 2 Enter a filename, such as FuelSysRequirements. Save the requirement set.

Import Requirements from External Documents

FuelSysRequirements is the requirements set. The requirements reference the content in the external documents:

- FuelSysDesignDescription.docx
- FuelSysRequirementsSpecification.docx
- FuelSysTestScenarios.xlsx

1. To import, right-click FuelSysRequirements in the **Index** and select **Import As Read-only References**.
2. In the Importing Requirements dialog box, select Microsoft Word Document for the **Document type**.

3. For **Document location**, browse for the FuelSysDesignDescription.docx file in the working folder. If the file is already open, you can select it from the drop-down list.

4. Select options:

- Select **Rich text (include graphics and tables)**.
- Select **Use bookmarks to identify items and serve as custom IDs**. This preserves links to existing document bookmarks in the new requirement set.

Importing Requirements: FuelSysRequirements.slreqx

Source document

Document type: Microsoft Word Document Use current

Document location: ies\slrequirements\FuelSysDesignDescription.docx Browse

Content

Plain text

Rich text (include graphics and tables)

Requirement Identification

Content is imported to match the document outline of section headings.

Within a section you can identify additional items:

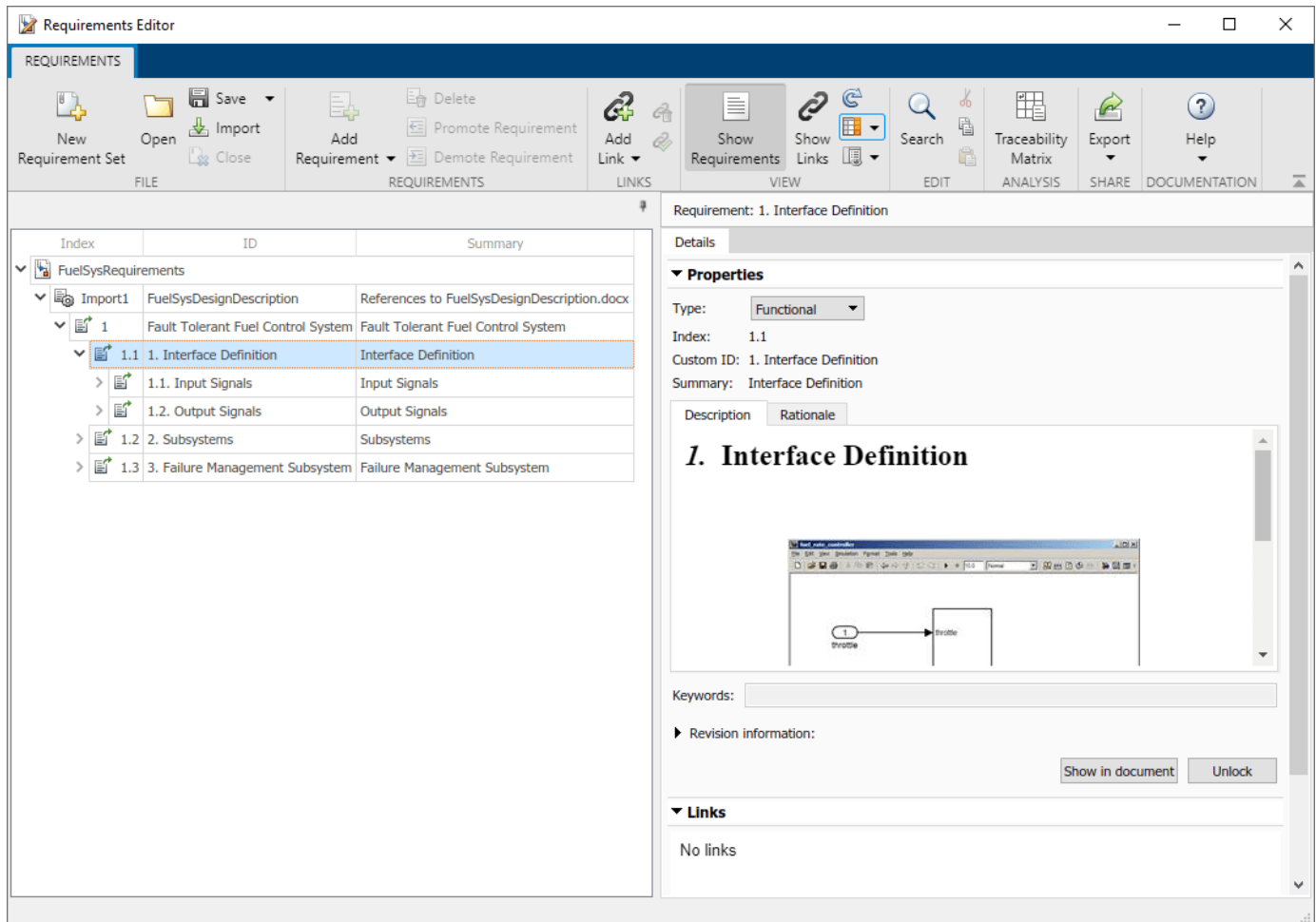
Use bookmarks to identify items and serve as custom IDs Preview

Identify items by occurrences of search pattern (REGEXP)

Ignore outline numbers in section headers

Import Cancel Help

4. Click **Import**. The new requirement set appears in the **Requirements Editor**.



5. You can navigate to the document. In the **Properties** pane, click **Show in document**.

1. Interface Definition

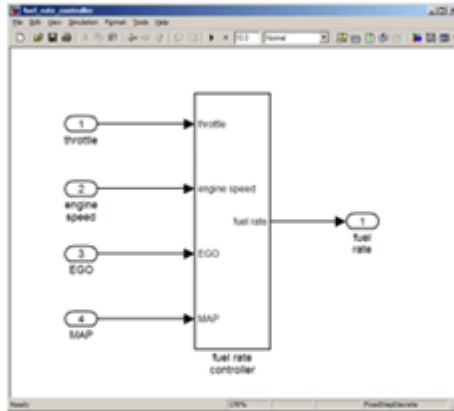


Image 1: High level block diagram of the System

1.1. Input Signals

1.1.1. Throttle Sensor

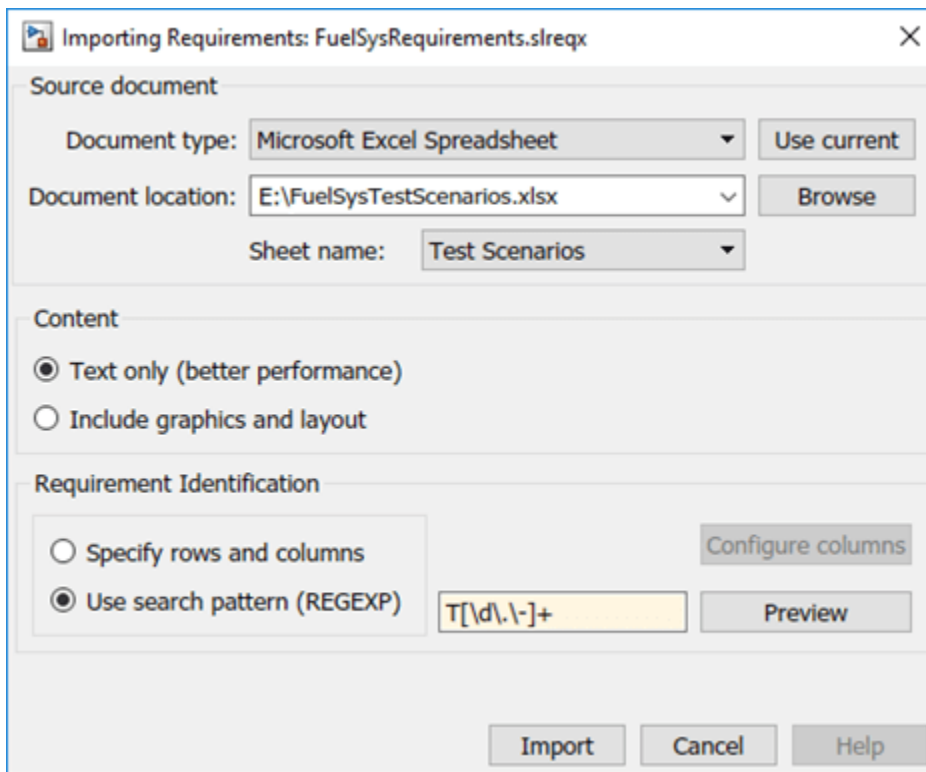
Now, import requirements from a Microsoft® Excel® document. When importing from Excel, you specify which columns to import. You can map the columns to either **Summary**, **Keywords**, or **Custom Attribute** fields in the imported data. You can also locate specific ranges in tables by specifying a regular expression pattern of requirements identifiers.

1. Open the FuelSysTestScenarios.xlsx file from the working folder.

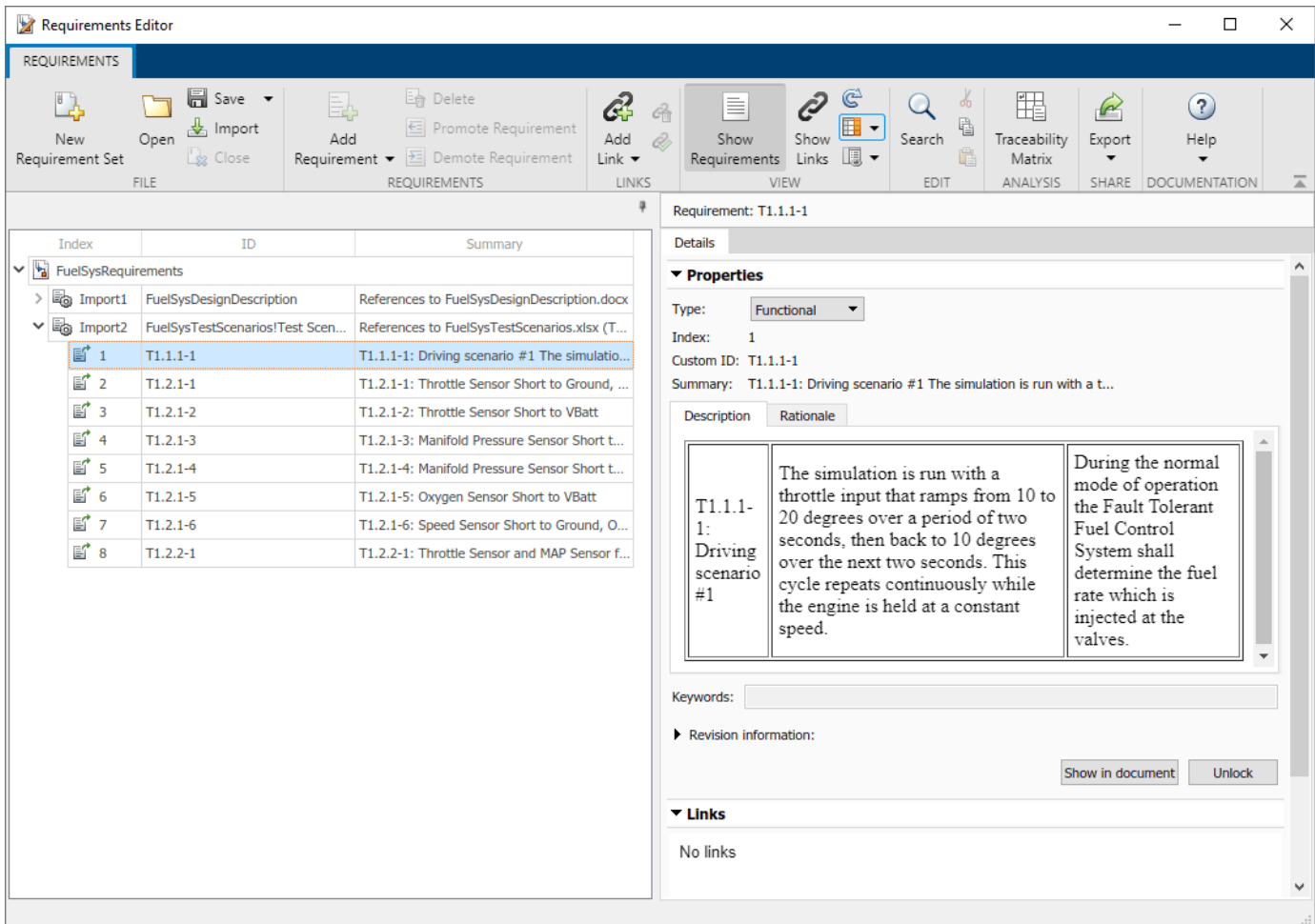
	A	B	C	D	E
1	Test Scenarios for the Fault Tolerant Fuel Control System				
2	<i>This Test Scenario Document provides a more detailed description on how the System is going to be verified against the high level requirements</i>				
3	<i>Typically such a document is a part of a more elaborate Test Plan.</i>				
4					
5	Date: October 7, 2009				
6	Version: 1.0				
7					
8	Requirements	Test Scenario	Test Description	Expected Result	
9	1.1 Normal Mode of Operation				
10		Normal operation		During the normal mode of operation the Fault Tolerant Fuel Control System shall determine the fuel rate which is injected at the valves.	
11		1.1.1 Stoichiometric mixture ratio		The stoichiometric mixture target shall have the value of 14.6 for the duration of the test scenario except for the duration of the warmup.	
12		1.1.2 Oxygen Sensor (EGO) 1.1.2.1 Oxygen sensor during warmup	T1.1.1-1: Driving scenario #1	The simulation is run with a throttle input that ramps from 10 to 20 degrees over a period of two seconds, then back to 10 degrees over the next two seconds. This cycle repeats continuously while the engine is held at a constant speed.	If the EGO sensor determines a high oxygen level present in

2. In the **Requirements Editor**, right-click FuelSysRequirements in the **Index** and select **Import As Read-only References**.

3. Configure the import settings as shown.



4. Click **Import**. A new top-level node contains references to the test scenario items in the Excel document.



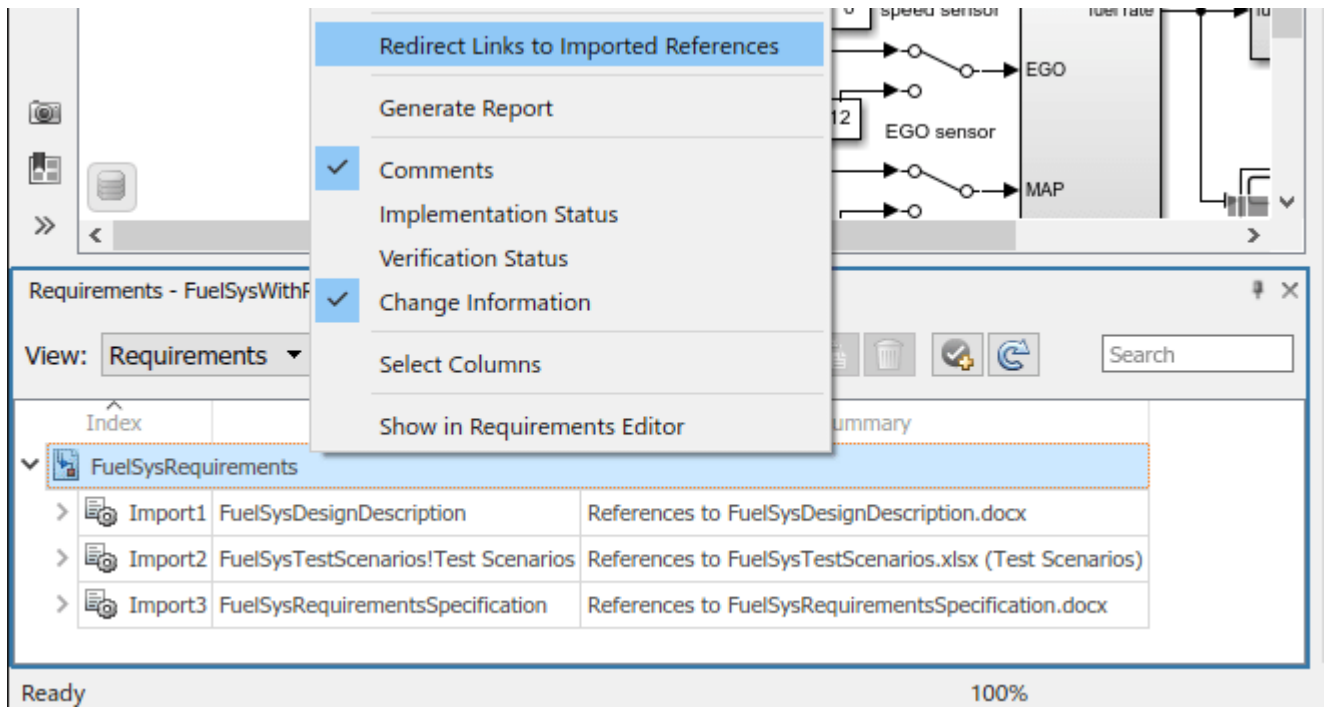
Repeat the import process for the file FuelSysRequirementsSpecification.docx.

Update Model Link Destinations

Update the model link destinations to the imported requirements. Open the FuelSysWithReqLinks model from the working folder.

```
open_system("FuelSysWithReqLinks.slx")
```

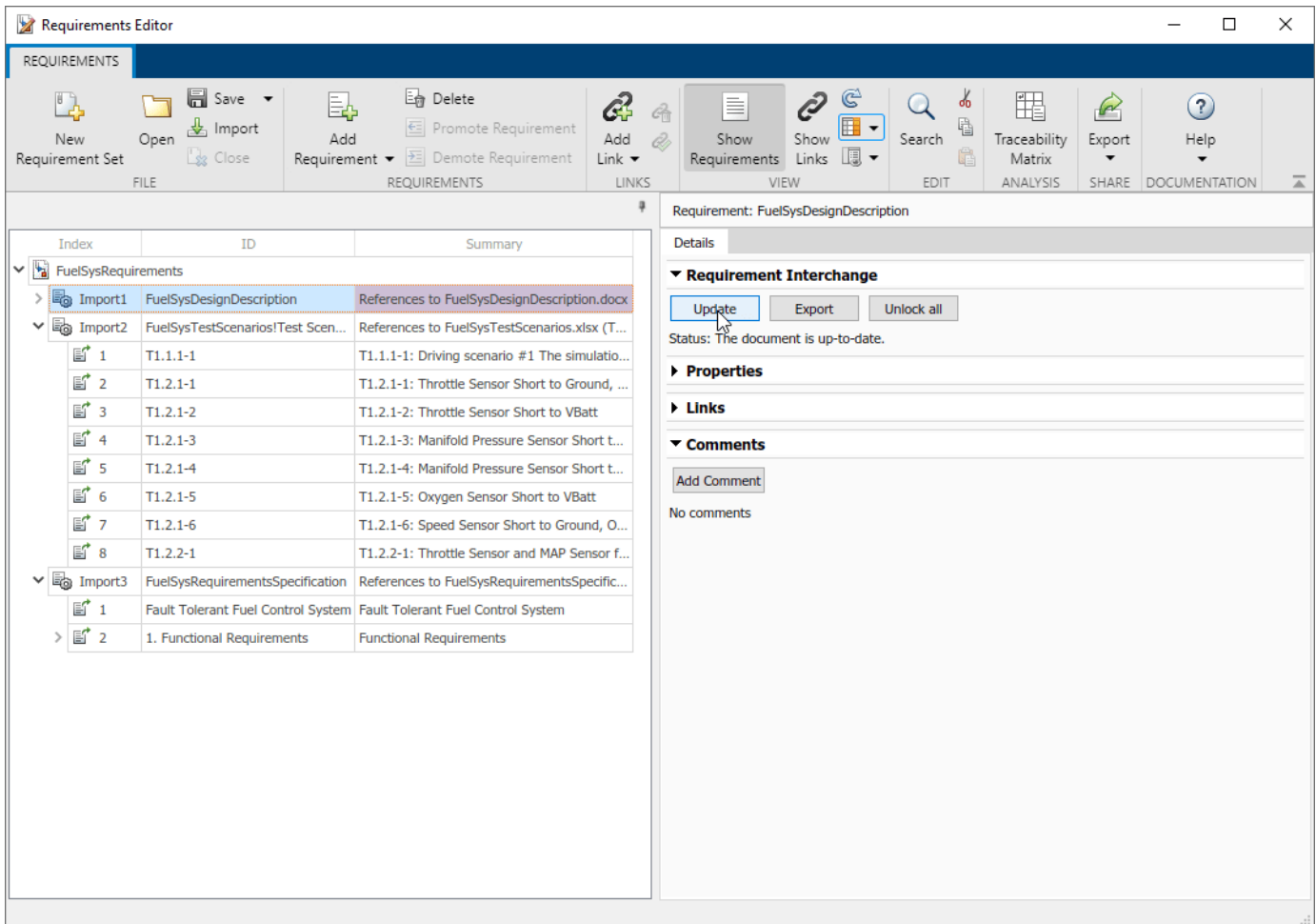
Enable the Requirements Perspective in the model. Click the icon at the lower-right of the model canvas and click the **Requirements** icon. The FuelSysRequirements set appears in the Requirements browser. Right-click the FuelSysRequirements line item and select **Redirect Links to Imported References**.



Update Requirements that Reference External Documents

If you change the requirement content in the external document, update the Requirement Set to reflect the latest version:

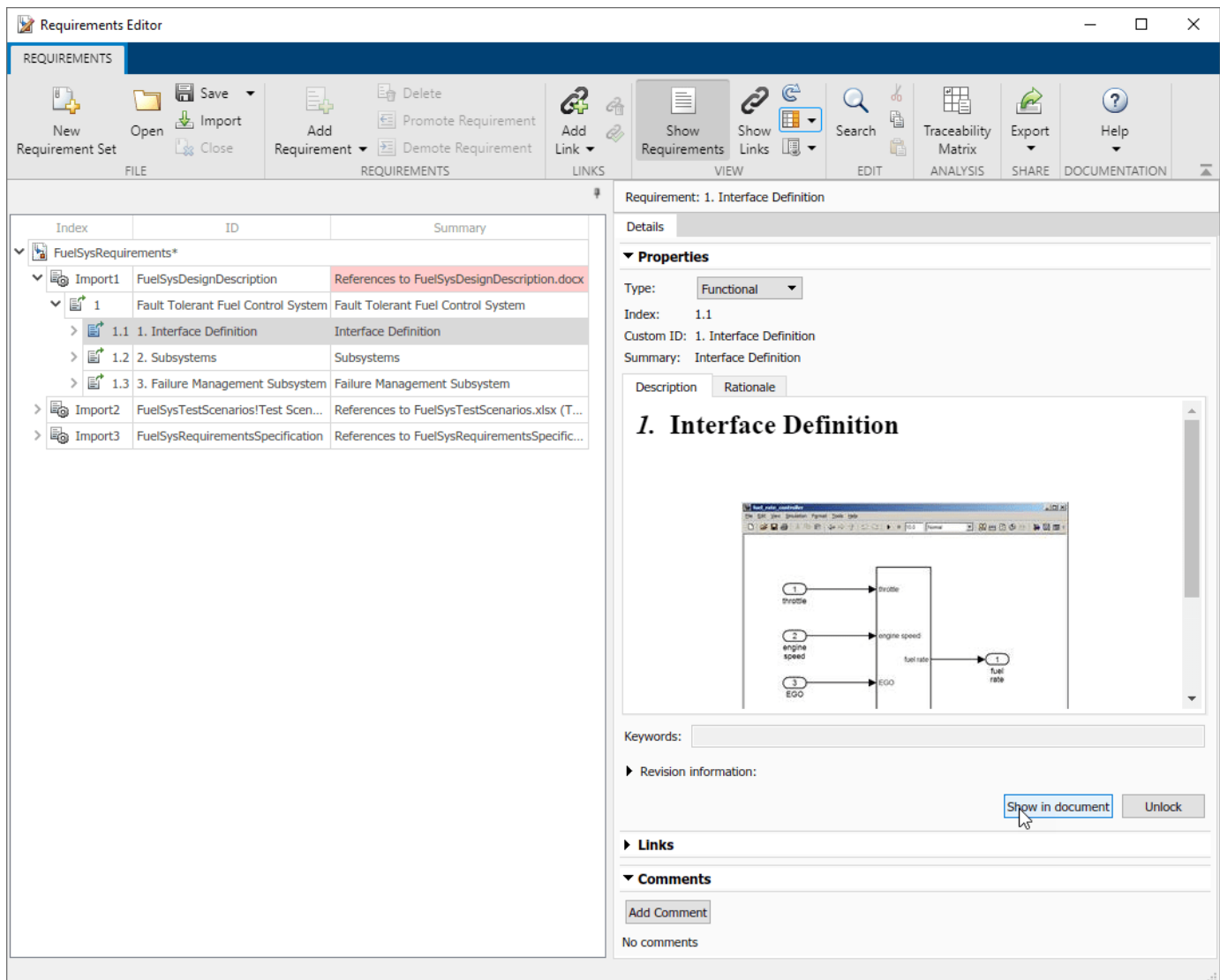
1. Select the top-level Import node in the **Requirements Editor**.
2. In the right pane, under **Requirement Interchange**, click the **Update** button.



Review the Imported References

Importing the three documents creates three top-level nodes in the left pane of **Requirements Editor**. Save the requirement set.

Expand the sub-trees and click on individual items to review the imported contents. Click the **Show in document** button for navigation to corresponding location in the original external document.



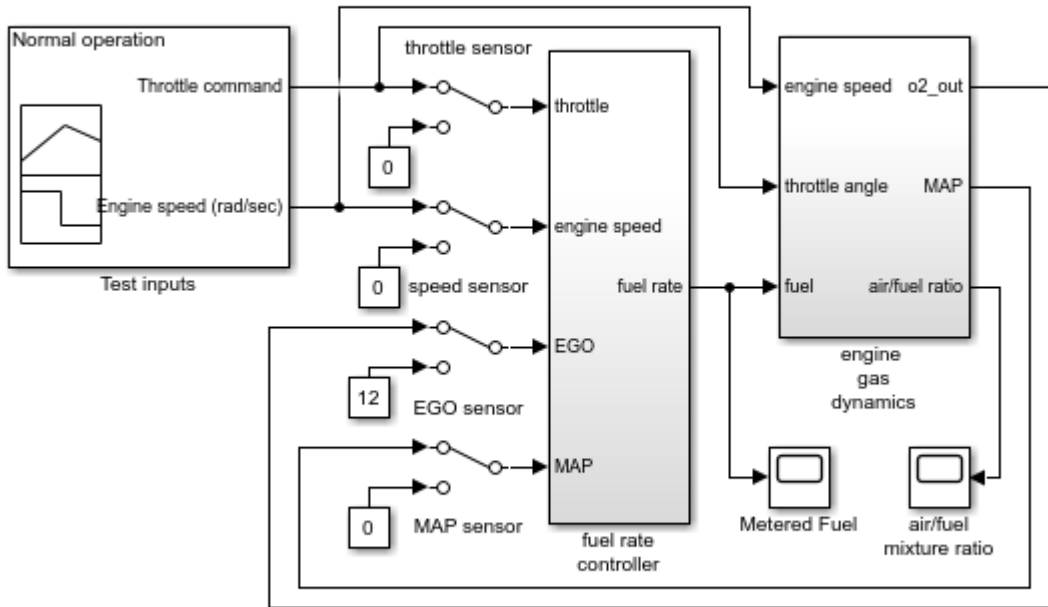
Load a Model with Links to Imported Documents

In this step, you load a model with existing links to imported documents. If you have models with RMI data in the Simulink® Verification and Validation™ format, opening those models with an available Requirements Toolbox license prompts you save the requirements data in the updated Requirements Toolbox format.

Clicking **Save now** creates a Link Set .slmx file.

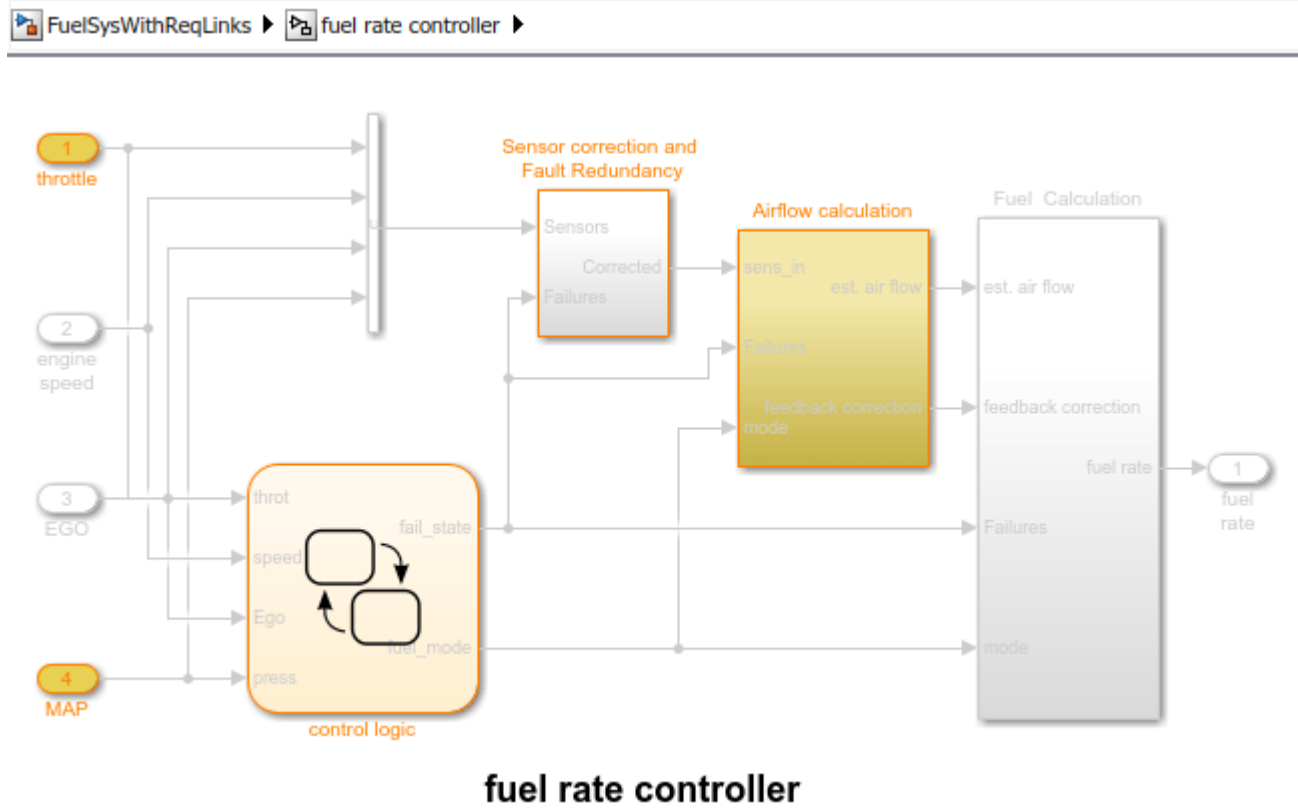
In this example, open the FuelSysWithReqLinks.slx model in the working folder. In the notification bar at the top of the canvas, click the **Save now** link to create a Link Set file FuelSysWithReqLinks.slmx.

Fault-Tolerant Fuel Control System

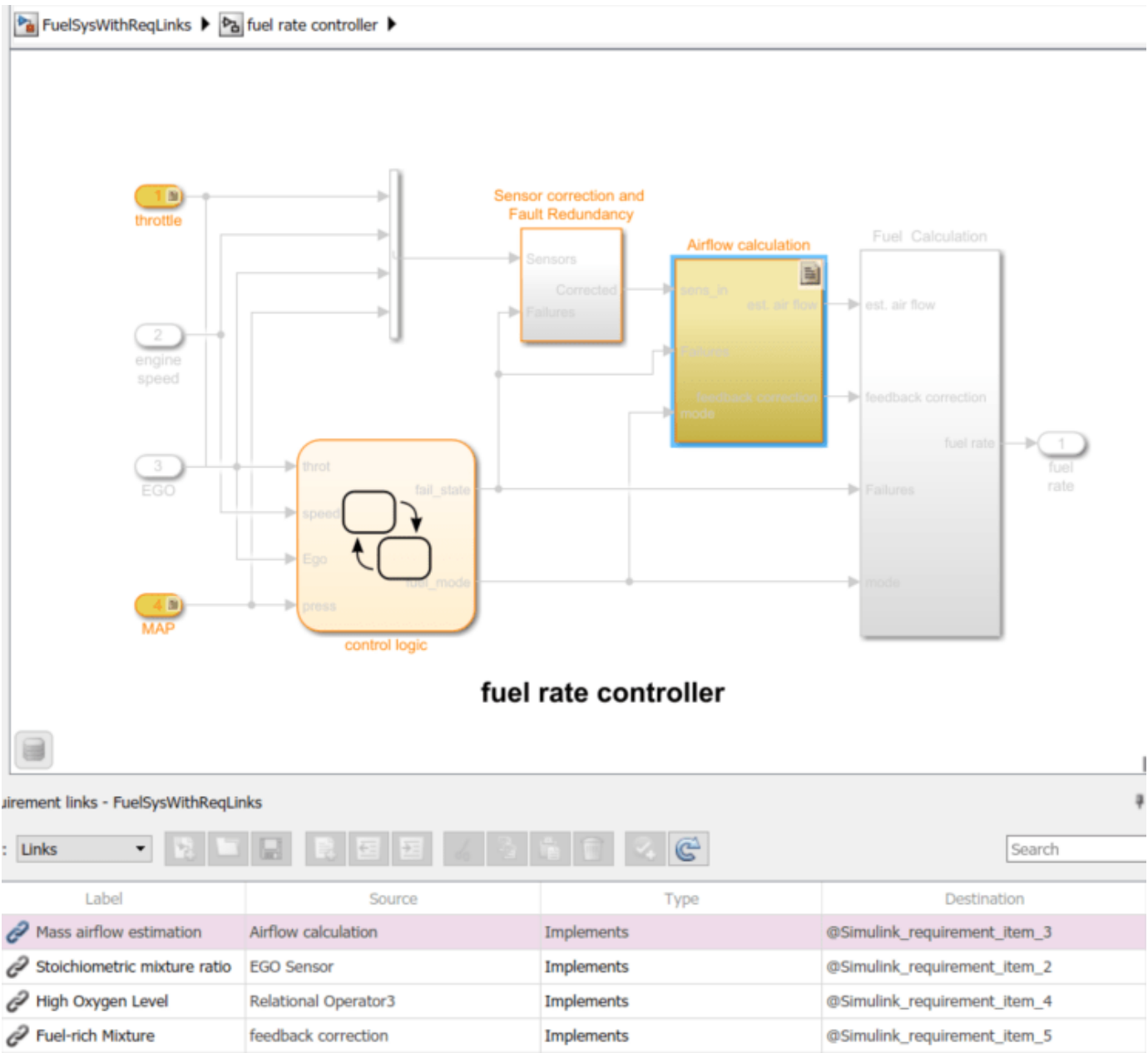


Copyright 1997-2010 The MathWorks, Inc.

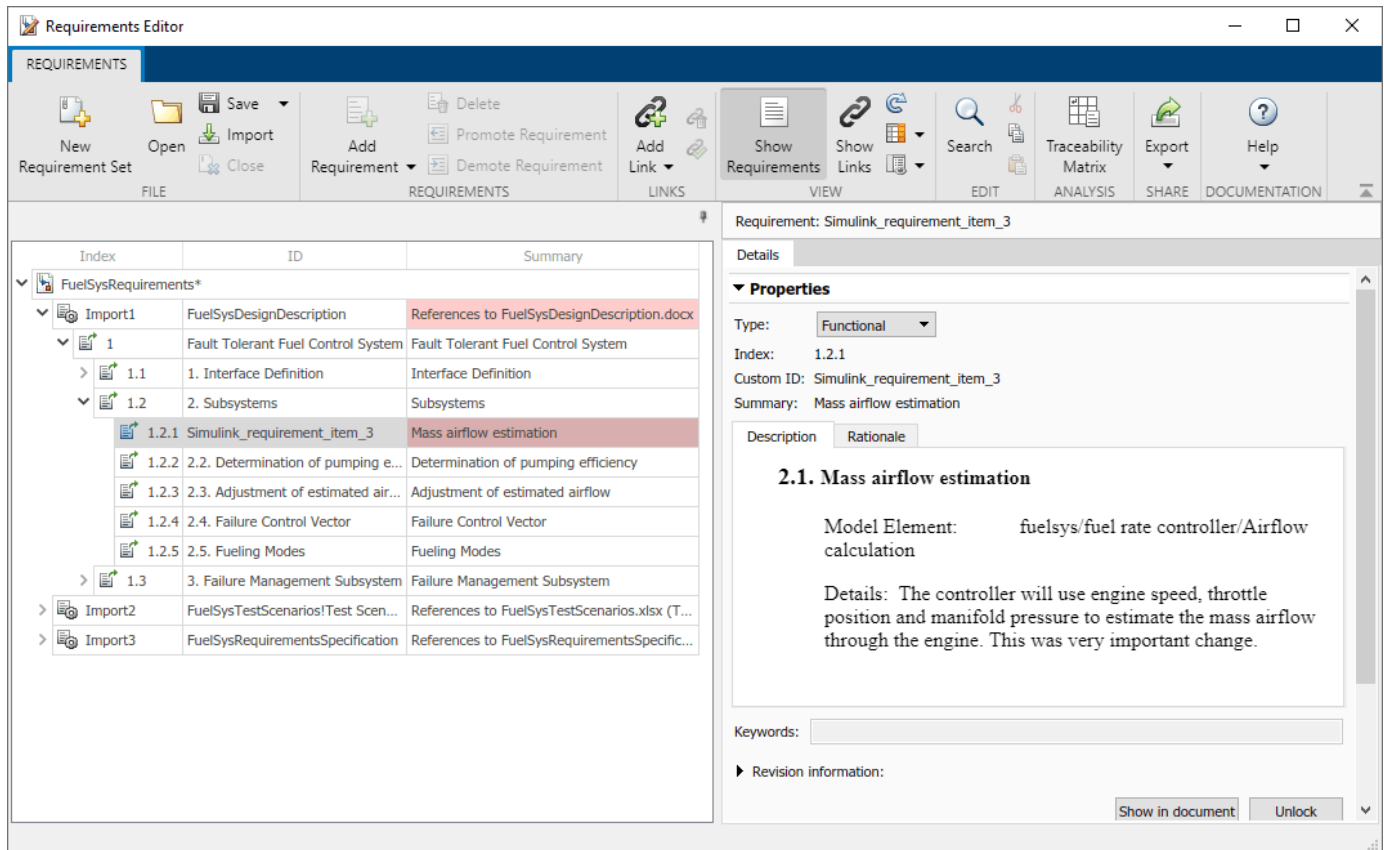
To highlight blocks with requirement links, in the **Requirements Viewer** tab, click the **Highlight Links** button.



To show the linked requirement, open the **Requirements Manager** tab and select a block.



If the **Requirements Editor** is open, the reference item is highlighted. You can review the contents of the requirement in the right pane.



The incoming link is displayed in the **Links** pane. With Requirements Toolbox, you do not need to insert navigation controls into Word and Excel documents to know where the links are. You can find the links in **Requirements Editor**.

A Requirements Toolbox license is needed to use the **Requirements Editor**. When corresponding references are not loaded in **Requirements Editor**, navigation will bring you to the original content in the external document, as in previous versions.

Microsoft Office Traceability

- “Link to Requirements in Microsoft Word Documents” on page 7-2
- “Link to Requirements in Microsoft Excel” on page 7-7
- “Navigate to Requirements in Microsoft Office Documents from Simulink” on page 7-11
- “Managing Requirements for Fault-Tolerant Fuel Control System (Microsoft Office)” on page 7-15

Link to Requirements in Microsoft Word Documents

With Requirements Toolbox, you can create direct links from linkable items on page 3-32 such as Simulink blocks or test cases to requirements in Microsoft Word. You can create a link to a selection in Microsoft Word, a named bookmark, or a section heading. You can only create direct links to requirements in Microsoft Word on Windows platforms.

Link a Requirement in Word to a Simulink Block

In this example, you will link requirements from a document in Microsoft® Word to a Simulink® block. Open the `slvndemo_fuelsys_officereq` model.

```
open_system('slvndemo_fuelsys_officereq');
```

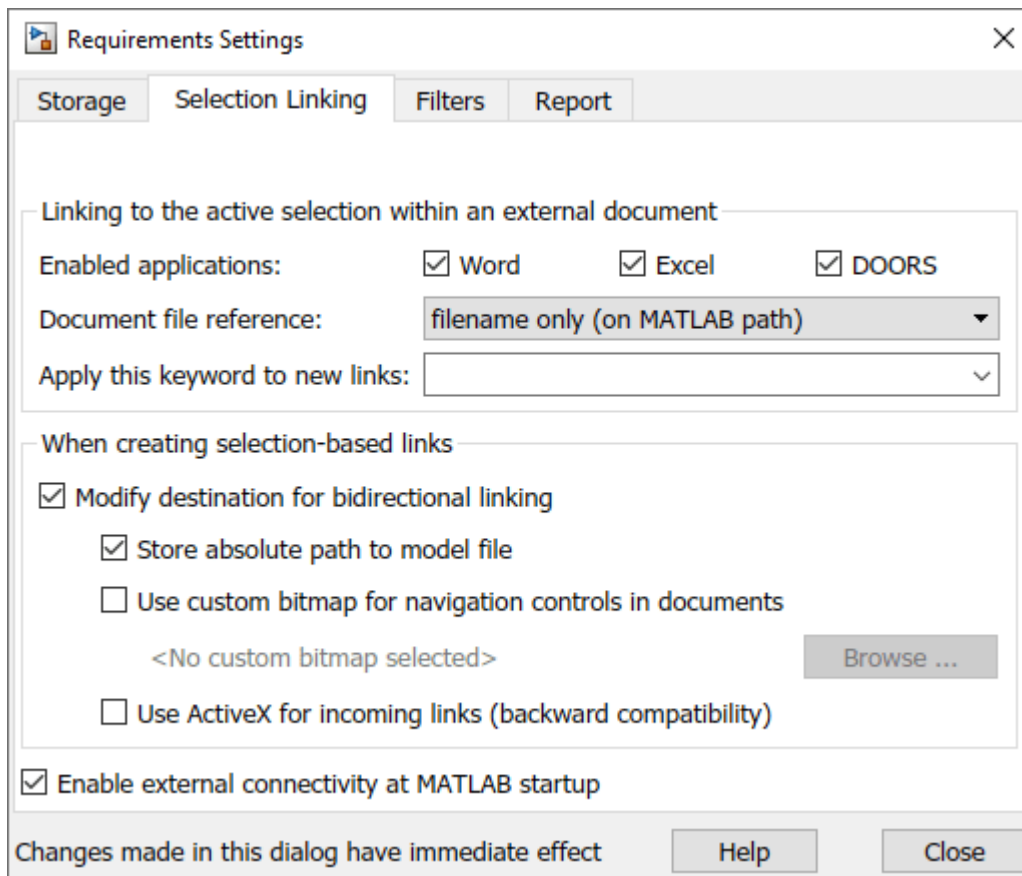
Open the `slvndemo_FuelSys_DesignDescription.docx` requirements document from the working directory, or at the MATLAB® command line by entering:

```
open('slvndemo_FuelSys_DesignDescription.docx')
```

Configure Selection Link Settings

First, ensure that Requirements Toolbox™ can link to Word documents and that bidirectional linking and external connectivity are enabled.


- 1 In Simulink, in the **Apps** tab, click **Requirements Manager**.
- 2 In the **Requirements** tab, ensure **Layout > Requirements Browser** is selected.
- 3 In the **Requirements** pane, in the **View** drop-down menu, select **Links**.
- 4 In the **Requirements** tab, click **Link Settings > Linking Options**. The Requirement Settings dialog appears.
- 5 Navigate to the **Selection Linking** tab.
- 6 Next to **Enabled applications** ensure that **Word** is selected.
- 7 In the **Document file reference** drop-down menu, select **filename only** (on MATLAB path).
- 8 Under **When creating selection-based links**, ensure that **Modify destination for bidirectional linking** and **Store absolute path to model file** are both selected. Make sure that **Use ActiveX for incoming links (backward compatibility)** is cleared.
- 9 Ensure that **Enable external connectivity at MATLAB startup** is selected.
- 10 Click **Close**.




Link to a Selection in Microsoft Word

Create a link from the selected text of the Determination of pumping efficiency requirement in Word to the Pumping Constant block:

- 1 In the slvnvdemo_FuelSys_DesignDescription Word document, find the section titled 2.2 Determination of pumping efficiency.
- 2 Select the header text.
- 3 In the slvnvdemo_fuelsys_officereq Simulink model, double-click the fuel rate controller subsystem to open it.
- 4 Double-click the Airflow calculation subsystem to open it.
- 5 Right-click the Pumping Constant block and click **Requirements > Link to Selection in**

Word. In Word, a bookmark is inserted with an automatically generated name. A link icon () is also inserted to navigate to the Simulink item associated with this requirement.

- 6 Navigate to your requirement in Word by right-clicking the Pumping Constant block, selecting **Requirements** and clicking the numbered requirement.

- 7 Navigate back to your Simulink block by clicking the link icon () in Word. If you don't want to add this link icon to your Word document when you create the link, clear **Modify destination for bidirectional linking** in the Requirements Settings window. However, the Word document is still modified when you create the link with this method because a bookmark is added.

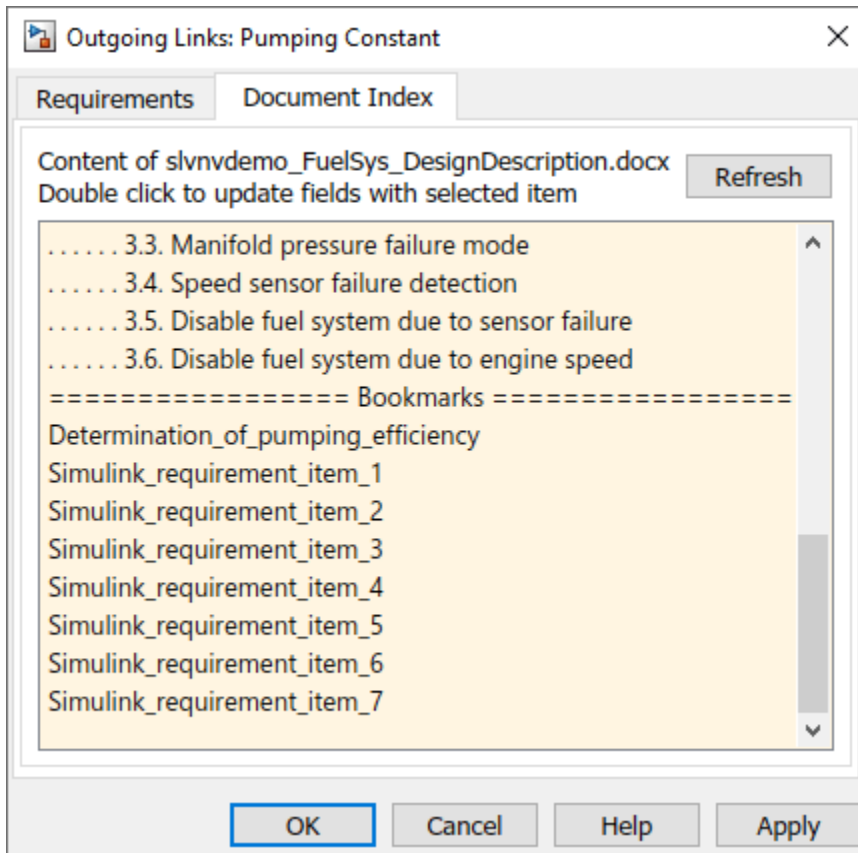
Create a Link to a Bookmark in a Microsoft Word Requirements Document

You can link from Simulink to an existing bookmark in your Word document. In Word, you can create bookmarks to each of your requirements with a meaningful name that represents the requirement content. When you create a link with this method, the requirements Word document is not modified and no Simulink navigation link is added to the Word document.

To add a bookmark to your Microsoft Word document, see [Add or delete bookmarks in a Word document or Outlook message on the Microsoft website](#).

Create a bookmark for the `Determination of pumping efficiency` requirement in Word, then link the `Pumping Constant` block to the bookmark:

- 1** In the `slvndemo_FuelSys_DesignDescription.docx` Word document, find the section titled `2.2 Determination of pumping efficiency`.
- 2** Create a bookmark with the name `Determination_of_pumping_efficiency`.
- 3** Save and close the Word document.
- 4** In the `slvndemo_fuelsys_officereq` Simulink model, double-click the `fuel_rate controller` subsystem to open it.
- 5** Double-click the `Airflow calculation` subsystem to open it.
- 6** Right-click the `Pumping Constant` block and select **Requirements > Open Outgoing Links dialog**.
- 7** In the Outgoing Links dialog, click **New**.
- 8** From the **Document type** drop-down, select `Microsoft Word`.
- 9** Next to the **Document** field, click **Browse** and select `slvndemo_FuelSys_DesignDescription.docx`. Click **Open**.
- 10** Select the **Document Index** tab. Scroll down to the bookmarks section and select the `Determination_of_pumping_efficiency` bookmark. If your bookmark does not appear, click **Refresh**. Click **Apply** and then click **OK** to create the link.



You can navigate to your requirement in Word by right-clicking the Pumping Constant block, selecting **Requirements**, and clicking the numbered requirement with the text `Determination_of_pumping_efficiency` in `slvnvdemo_FuelSys_DesignDescription.docx`.

Create a Link to a Heading in a Microsoft Word Requirements Document

You can create headings and subheadings in Microsoft Word, then link Simulink blocks to these headings. Similar to creating a link to a bookmark, creating a link to a heading allows you to give a link a meaningful name. If your requirements Word document already has section headings, then creating a link with this method does not modify the requirements document. However, you cannot navigate between Simulink and Word when you link to a heading.

If your Word document does not already have headings, see [Add a heading on the Microsoft website](#).

The `slvnvdemo_FuelSys_DesignDescription.docx` Word document already has headings for all of the requirements. Create a link from the `Determination of pumping efficiency` requirement in Word to the Pumping Constant block:

- 1 Close the `slvnvdemo_FuelSys_DesignDescription.docx` requirements Word document.
- 2 In the `slvnvdemo_fuelsys_officereq` Simulink model, double-click the fuel rate controller subsystem to open it.
- 3 Double-click the Airflow calculation subsystem to open it.
- 4 Right-click the Pumping Constant block and select **Requirements > Open Outgoing Links dialog**.

- 5** In the Outgoing Links dialog, click **New**.
- 6** From the **Document type** drop-down, select Microsoft Word.
- 7** Next to the **Document** field, click **Browse** and select slvnvdemo_FuelSys_DesignDescription.docx. Click **Open**.
- 8** Select the **Document Index** tab. Under Outline Headings, select 2.2 Determination of pumping efficiency. Click **Apply** and then click **OK** to create the link.
- 9** Navigate to your requirement in Word by right-clicking the Pumping Constant block, selecting **Requirements**, and clicking the numbered requirement with the text 2.2 Determination of pumping efficiency in slvnvdemo_FuelSys_DesignDescription.docx.

Cleanup

Clear the open requirement sets and link sets, and close the open models without saving changes.

```
slreq.clear;  
bdclose all;
```

See Also

More About

- “Create and Store Links” on page 3-31
- “Link to Requirements in Microsoft Excel” on page 7-7
- “Import Requirements from Microsoft Office Documents” on page 1-12
- “Import and Edit Requirements from a Microsoft Word Document” on page 1-95

Link to Requirements in Microsoft Excel

You can create direct links from requirements managed in Microsoft Excel to items in MATLAB and Simulink. You can use the links and backlinks inserted in Excel to navigate from the requirements to the design or to tests.

Alternatively, you can import requirements from Microsoft Excel into Requirements Toolbox. For more information, see “Differences Between Importing and Direct Linking” on page 1-11.

Note You can create direct links to requirements in Microsoft Excel only on Microsoft Windows platforms.

Link and Navigate to Requirements in Excel from Simulink blocks

This example shows how to link requirements authored in Microsoft Excel to Simulink blocks and how to navigate between the requirements and the design by using direct links and external backlinks.

Open the `slvnvdemo_fuelsys_officereq` model.

```
open_system("slvnvdemo_fuelsys_officereq");
```

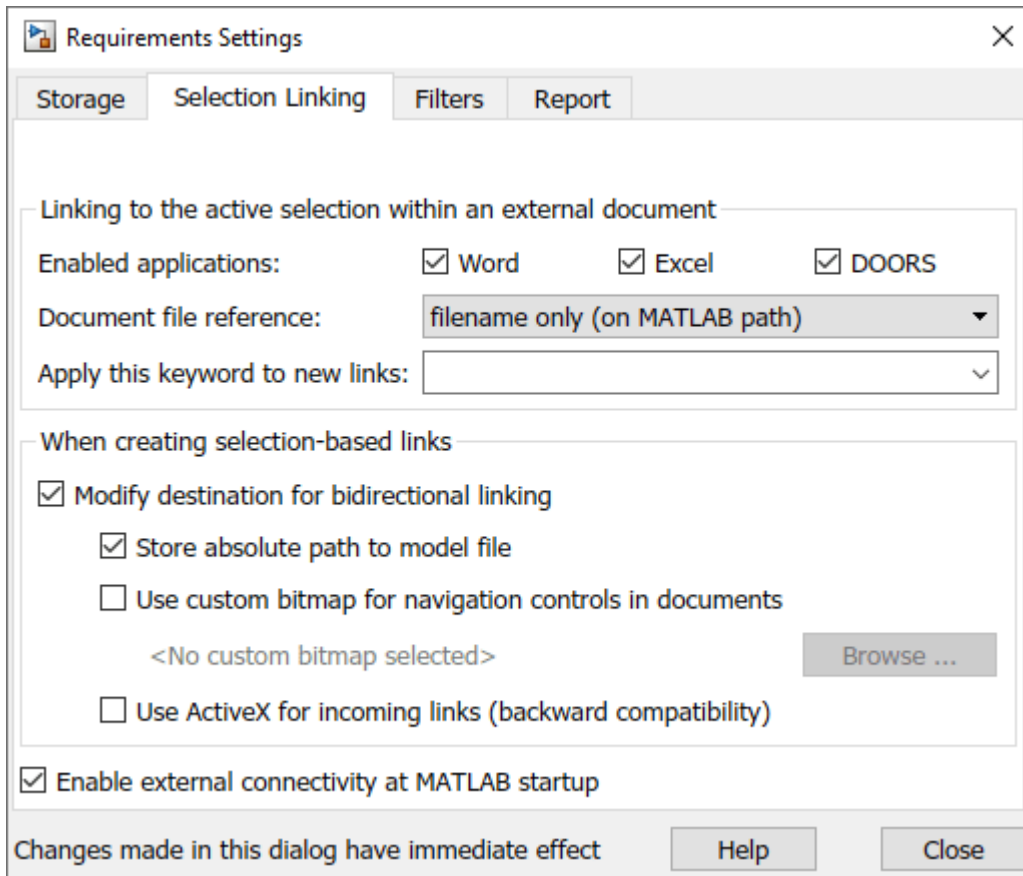
Open the `slvnvdemo_FuelSys_TestScenarios.xlsx` requirements document.

```
winopen("slvnvdemo_FuelSys_TestScenarios.xlsx")
```

Configure Selection Link Settings

First, ensure that Requirements Toolbox™ can link to Excel documents and that bidirectional linking and external connectivity are enabled.

- 1 In Simulink, in the **Apps** tab, click **Requirements Manager**.
- 2 In the **Requirements** tab, ensure **Layout > Requirements Browser** is selected.
- 3 In the **Requirements** pane, in the **View** drop-down menu, select **Links**.
- 4 In the **Requirements** tab, click **Link Settings > Linking Options**. The Requirement Settings dialog opens.
- 5 In the **Selection Linking** tab, next to **Enabled applications**, ensure that **Excel** is selected.
- 6 Ensure that **Document file reference** is set to **filename only** (on MATLAB path).
- 7 Under **When creating selection-based links**, select **Modify destination for bidirectional linking** and **Store absolute path to model file**. Ensure that **Use ActiveX for incoming links (backward compatibility)** is cleared.
- 8 Ensure that **Enable external connectivity at MATLAB startup** is selected.
- 9 Click **Close**.



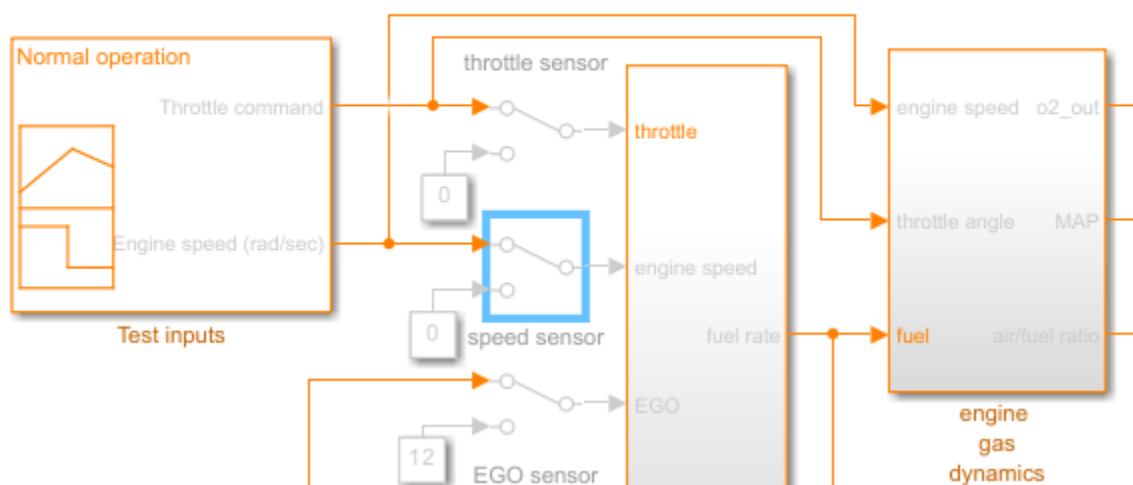
Link to a Selection in Microsoft Excel

In the `slvnvdemo_FuelSys_TestScenarios` Excel file, select the requirement 1.2.1.6 Speed Sensor Failure in cell B22.

	A	B	C
21		1.2.1.5 Oxygen sensor failure	T1.2.1-5: Oxygen Sensor Short to VBatt
22		1.2.1.6 Speed Sensor Failure	T1.2.1-6: Speed Sensor Short to Ground, Open Circuit
23	1.2.2 Multiple Sensor Failure		
24		1.2.2.1 Multiple Sensor Failure	T1.2.2-1: Throttle Sensor and MAP Sensor fail at the same time
25			
26			


At the top level of the `slvnvdemo_fuelsys_officereq` model, right-click the speed sensor block and select **Requirements > Link to selection in Excel** to create the link.

Fault-Tolerant Fuel Control System



Navigate Between Simulink and Excel

To navigate from the speed sensor block to the requirement in Excel, right-click the block and select **Requirements > 1. "1.2.1.6 Speed Sensor Failure"**.

To navigate from the requirement in Excel to the speed sensor block, click the backlink icon .

	A	B	C	
21		1.2.1.5 Oxygen sensor failure	T1.2.1-5: Oxygen Sensor Short to VBatt	During normal m sensor reading sl calibratable thre
22		1.2.1.6 Speed Sensor Failure	T1.2.1-6: Speed Sensor Short to Ground, Open Circuit	During normal m sensor reading sl
23	1.2.2 Multiple Sensor Failure		slrvvdemo_fuelsys_officereq/speed sensor (ManualSwitch)	
24		1.2.2.1 Multiple Sensor Failure	T1.2.2-1: Throttle Sensor and MAP Sensor fail at the same time	During normal o failure, the MAP
25				

If your Excel file does not contain a backlink, you can update the backlinks for the link set. For more information, see “Manage Navigation Backlinks in External Requirements Documents” on page 3-51.

See Also

More About

- “Create and Store Links” on page 3-31
- “Link to Requirements in Microsoft Word Documents” on page 7-2
- “Import Requirements from Microsoft Office Documents” on page 1-12
- “Import Requirements from a Microsoft Excel Document” on page 1-125

Navigate to Requirements in Microsoft Office Documents from Simulink

With Requirements Toolbox, you can capture, track, and manage requirements in Microsoft Word and Excel. When you create a link from a model object to a requirement RMI stores the link data in the model file. Using this link, you can navigate from the model object to its associated requirement. You can only create and navigate direct links to requirements in Microsoft Word and Excel on Windows platforms.

Enable Linking from Microsoft Office Documents to Simulink Objects

When you create a link to a requirement in a Microsoft Office document, you can insert a navigation object in the document. This navigation object serves as a link from the requirement to its associated model object. By default, the RMI does not insert navigation objects into requirements documents. You can change the settings to automatically insert the navigation objects when you create the link.

To enable linking from a Word or Excel document to the example model:

- 1 Open the example “Managing Requirements for Fault-Tolerant Fuel Control System (Microsoft Office)” on page 7-15.

```
openExample('slrequirements/ManageReqsForFaultTolerantFuelCtrlSysMicrosoftOffice07Example')
```

- 2 Open the model:

```
open_system("slvndemo_fuelsys_officereq")
```

Note You can modify requirements settings in the Requirements Settings dialog box. These settings are global and not specific to open models. Changes you make apply not only to open models, but also persist for models you subsequently open. For more information about these settings, see “Requirements Settings” on page 6-10.

- 3 In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, ensure **Layout > Requirements Browser** is selected. In the **Requirements** pane, in the **View** drop-down select **Links**. In the **Requirements** tab, select **Link Settings > Linking Options**.

The Requirements Settings dialog box opens.

- 4 On the **Selection Linking** tab of the Requirements Settings dialog box:

- Enable **Modify destination for bidirectional linking**.

When you select this option, every time you create a selection-based link from a Simulink object to a requirement, the RMI inserts a navigation object at the designated location in the requirements document.

- To specify one or more keywords to apply to the links that you create, in the **Apply this keyword to new links** field, enter the keyword names.

For more information about keywords, see “User Keywords and Requirements Filtering” on page 6-11.

- 5 Click **Close** to close the Requirements Settings dialog box. Keep the `slvndemo_fuelsys_officereq` model open.

Insert Navigation Objects in Microsoft Office Documents

Use selection-based linking to create a link from the `slvndemo_fuelsys_officereq` model to a requirements document. If you have configured the RMI as described in “Enable Linking from Microsoft Office Documents to Simulink Objects” on page 7-11, the RMI can insert a navigation object into the requirements document.

- 1 Open the Word 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 > Link to Selection in Word**.
- 5 The RMI inserts an URL-based link into the requirements document.

1.1.6. Throttle Sensor

Link to Multiple Model Objects

If you have several model objects that correspond to one requirement, you can link them to one requirement with a single navigation object. This eliminates the need to insert multiple navigation objects for a single requirement. The model objects must be available in the same file.


The workflow for linking multiple model objects to one Microsoft Word entry is as follows:

- 1 Make sure that the RMI is configured to insert navigation objects into requirements documents, as described in “Enable Linking from Microsoft Office Documents to Simulink Objects” on page 7-11.
- 2 Select the Word requirement to link to.
- 3 Select the model objects that need to link to that requirement.
- 4 Right-click one of the model objects and select **Requirements > Link to Selection in Word**.

A single navigation object is inserted at the selected requirement.

- 5 Navigate to the model by following the navigation object link in Word.

Customize Microsoft Office Navigation Objects

If the RMI is configured to modify destination for bidirectional linking, the RMI inserts a navigation object into your requirements document. This object looks like the icon for the Simulink software: 

Note In Microsoft Office requirements documents, following a navigation object link highlights the Simulink object that contains a bidirectional link to the associated requirement.

To use an icon of your own choosing for the navigation object:

- 1 In the **Requirements** tab, select **Link Settings > Linking Options**.
- 2 Select the **Selection Linking** tab.
- 3 Select **Modify destination for bidirectional linking**.

Selecting this option enables the **Use custom bitmap for navigation controls in documents** option.

- 4 Select **Use custom bitmap for navigation controls in documents**.
- 5 Click **Browse** to locate the file you want to use for the navigation objects.

For best results, use an icon file (.ico) or a small (16×16 or 32×32) bitmap image (.bmp) file for the navigation object. Other types of image files might give unpredictable results.

- 6 Select the desired file to use for navigation objects and click **Open**.
- 7 Close the Requirements Settings dialog box.

The next time you insert a navigation object into a requirements document, the RMI uses the file you selected.

Navigate Between Microsoft Office Requirement and Model

In “Insert Navigation Objects in Microsoft Office Documents” on page 7-12, you created a link between a Microsoft Office requirement and the Throttle & Manifold subsystem in the slvndemo_fuelsys_officereq example model. Navigate these links in both directions:

- 1 In the slvndemo_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**, follow the navigation object link.

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



Navigation from Microsoft Office requirements documents is not automatically enabled upon MATLAB startup. Navigation is enabled when you create a new requirements link or when you have enabled bidirectional linking as described in “Insert Navigation Objects in Microsoft Office Documents” on page 7-12.

Note You cannot navigate to requirements from Microsoft Word 2013 onwards when the document is open in read-only mode. Alternately, consider disabling the “Open e-mail attachments and other uneditable files in reading view” option in the Microsoft Word options or using editable documents.

When attempting navigation from requirements links with the  icon, if you get a “Server Not Found” or similar message, enter the command `rmi('httpLink')` to activate the internal MATLAB HTTP server.

Managing Requirements for Fault-Tolerant Fuel Control System (Microsoft Office)

Requirements Management Interface (RMI) provides tools for creating and reviewing links between Simulink® objects and requirements documents. This example illustrates linking model objects to Microsoft® Office Documents, navigation of these links, generating requirements report and maintaining consistency of links. See also “Working with IBM Rational DOORS 9 Requirements” on page 8-30 example for features specific to linking with requirements stored in IBM® Rational® DOORS®.

The included example model is linked to documents in Microsoft Office format. If only an earlier version of Microsoft Office is available to you, jump to Updating all links when documents are moved or renamed on page 7-28 for an example of how to adjust the example model to work with included earlier versions of documents. Direct links to Microsoft Office documents are only supported on Windows® platforms.

Open Example Model

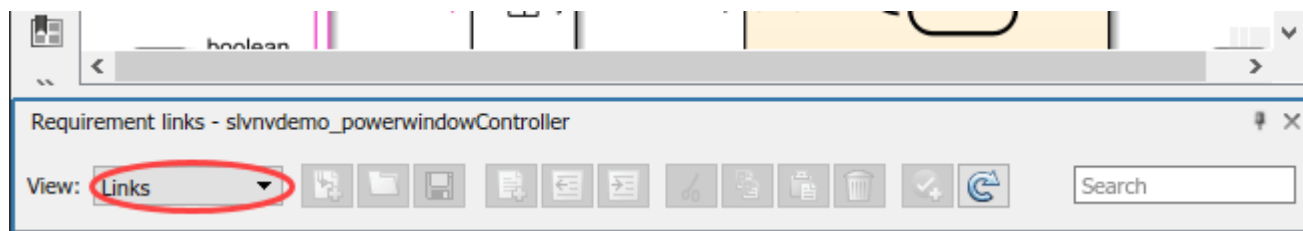
Requirements management features are demonstrated using an example model of a fault-tolerant fuel control system. You can open this model by evaluating the following code.

```
open_system('slvndemo_fuelsys_officereq');
```

Set Up Requirements Manager to Work with Links

- 1 In the **Apps** tab, open **Requirements Manager**.
- 2 In the **Requirements** tab, ensure **Layout > Requirements Browser** is selected.
- 3 In the **Requirements Browser**, in the **View** drop-down menu, select **Links**.

In this example, you will work exclusively in the **Requirements** tab and any references to toolbar buttons are in this tab.



Viewing Existing Requirements

This example starts with the model that only has a few requirements links. In the **Requirements** tab, click **Highlight Links** to highlight blocks with requirements links or evaluate the following code.

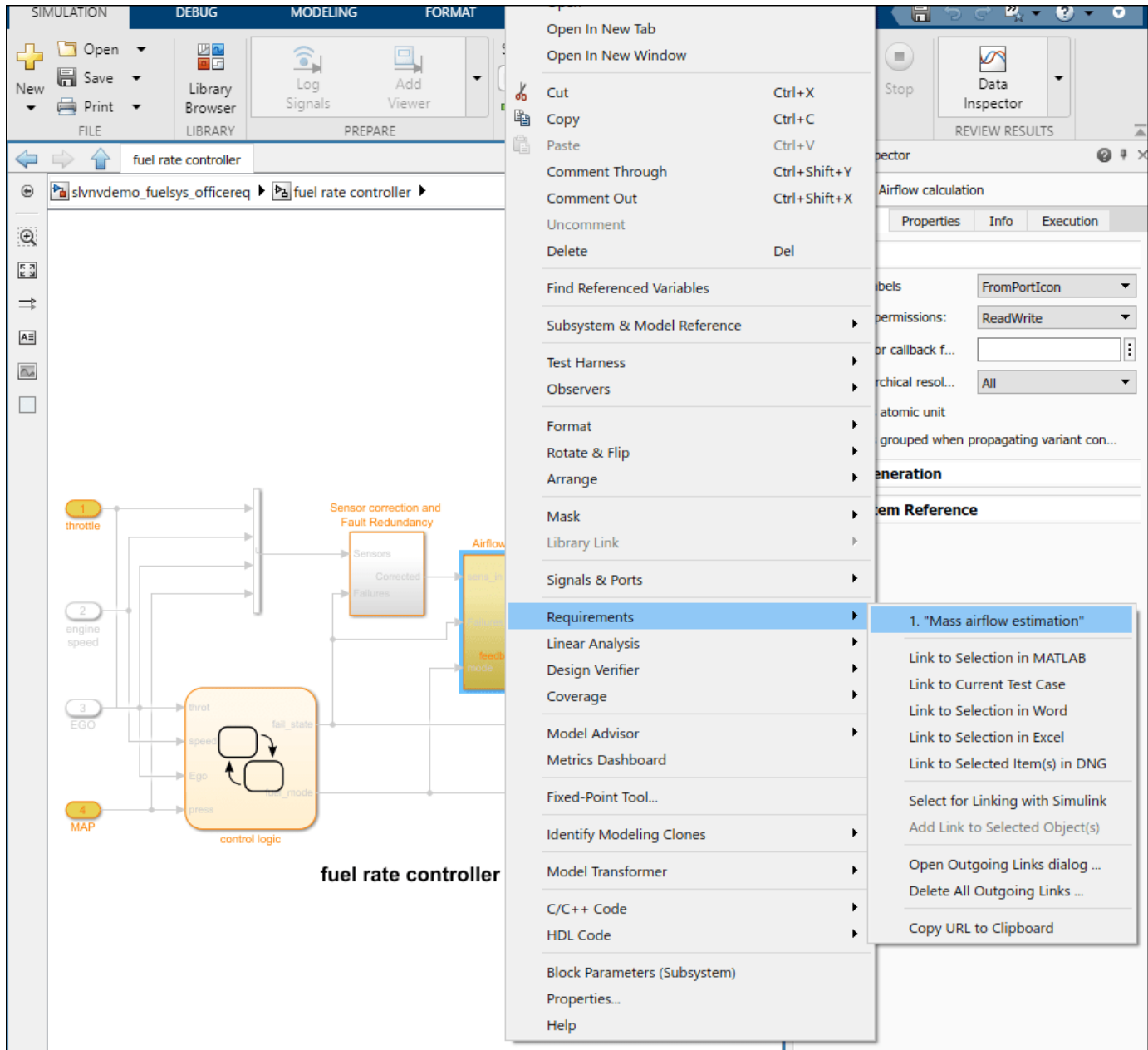
```
rmi('highlightModel', 'slvndemo_fuelsys_officereq');
```

Orange highlighting corresponds to objects with linked requirements. Empty-fill highlighting is for subsystems whose children have links to requirements. Double-click the fuel rate controller block to open the subsystem and review child objects with requirements, or evaluate the following.

```
open_system('slvndemo_fuelsys_officereq/fuel_rate_controller');
```

Navigate to Document

Right-click the Airflow calculation block in fuel rate controller subsystem and select **Requirements > Mass airflow estimation** in the context menu.



This opens the linked document and selects the target content. You can also evaluate the following code:

```
rmidemo_callback('view', ...
  'slvnvdemo_fuelsys_officereq/fuel rate controller/Airflow calculation',1)
```

Requirements Links in Stateflow Charts

Double-click the control logic chart block in the fuel rate controller subsystem to open the chart. If you can't find it, evaluate the following code.

```
rmdemo_callback('locate','slvndemo_fuelsys_officereq/fuel rate controller/control logic');
```

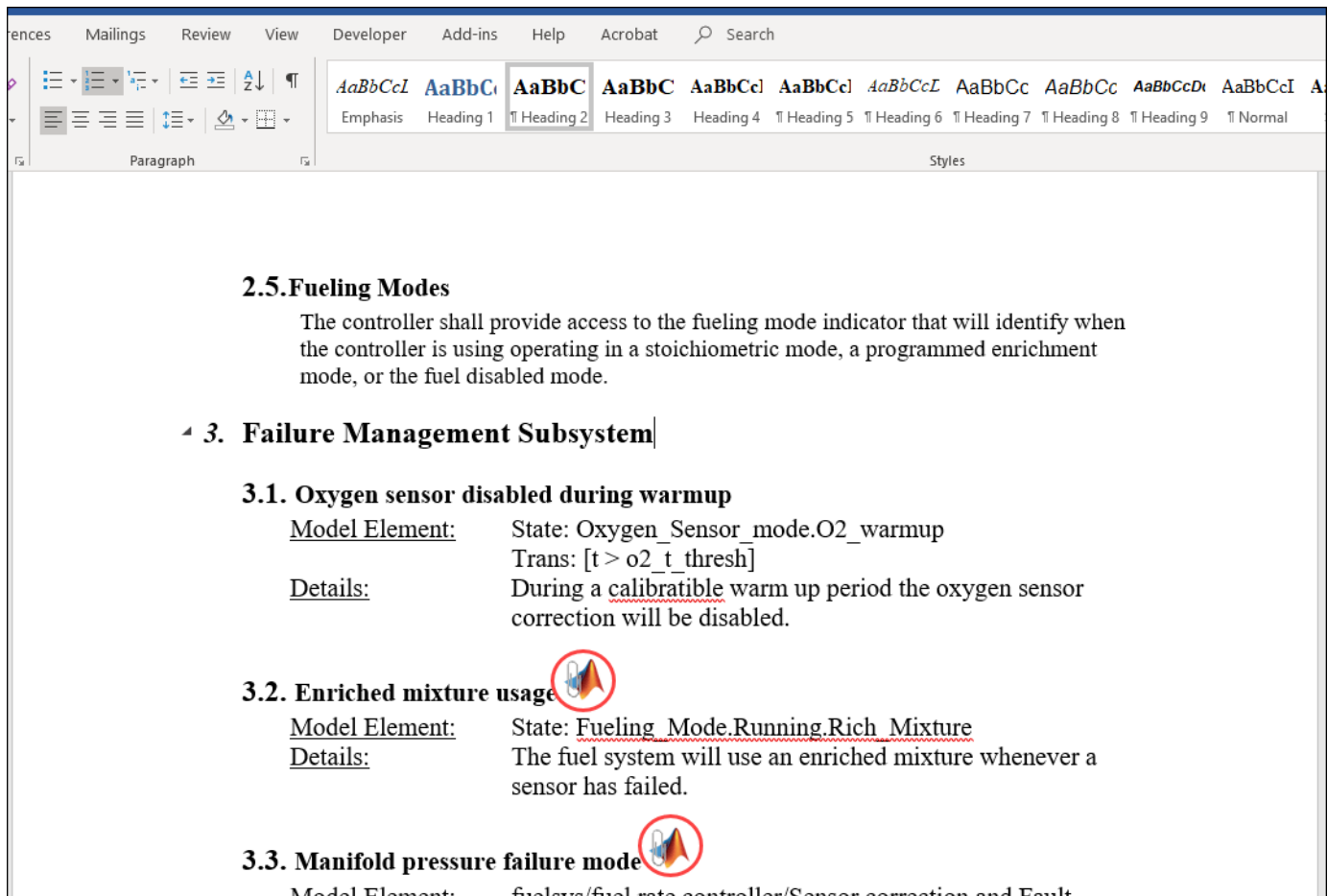
States and transitions linked to requirements are highlighted. Right-click the Rich Mixture state, select **Requirements** and follow the link at the top to view related documentation. Alternatively, evaluate the following code:

```
rmdemo_callback('view', ...
'slvndemo_fuelsys_officereq/fuel rate controller/control logic:26',1)
```

Navigate from Requirements Document to Model Objects

In the slvndemo_Fuelsys_DesignDescription.docx from the previous step, find section **3.3 Manifold pressure failure mode** in the document and double-click the Simulink icon at the end of subheader. This displays a relevant Simulink subsystem diagram with the target object highlighted. Close all model windows and repeat navigation from the document.

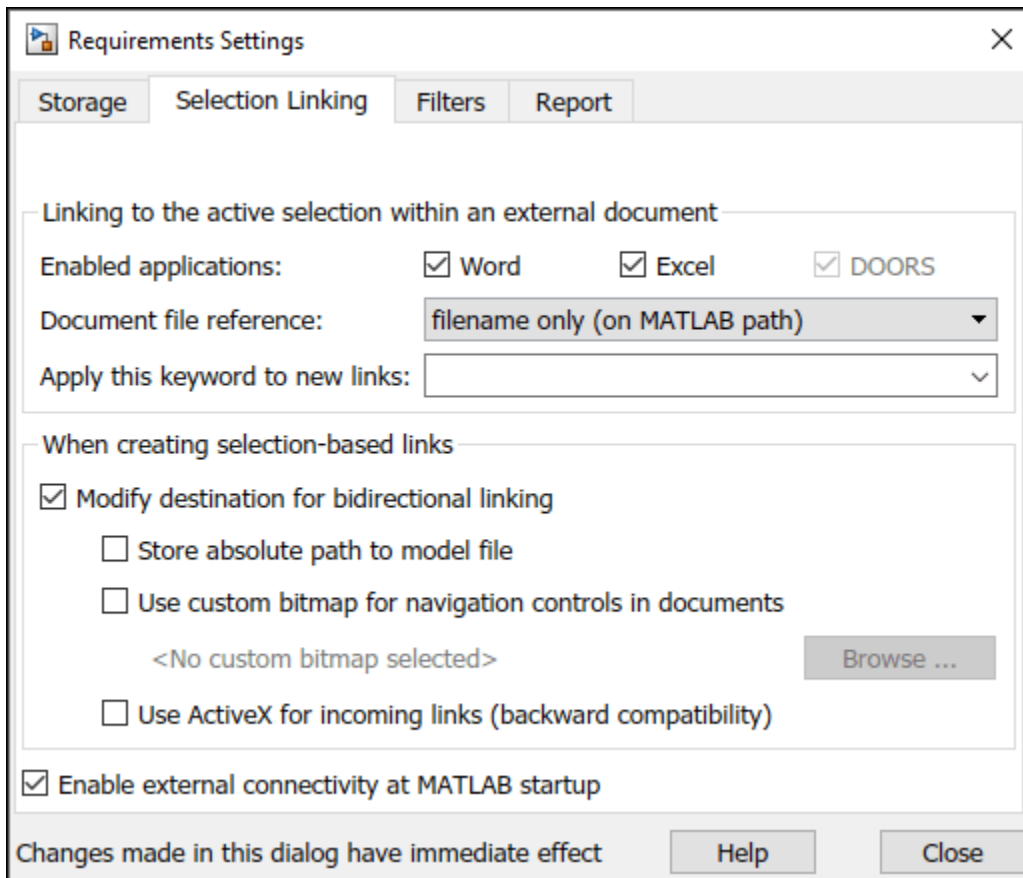
Diagrams or charts are opened as necessary as long as model file can be located.



Creating New Links

To configure your settings for creating bidirectional links, do the following:

- In the Simulink model, in the **Requirements** tab, select **Link Settings > Linking Options**.
- In the dialog box that appears, make sure that **Modify destination for bidirectional linking** is checked.

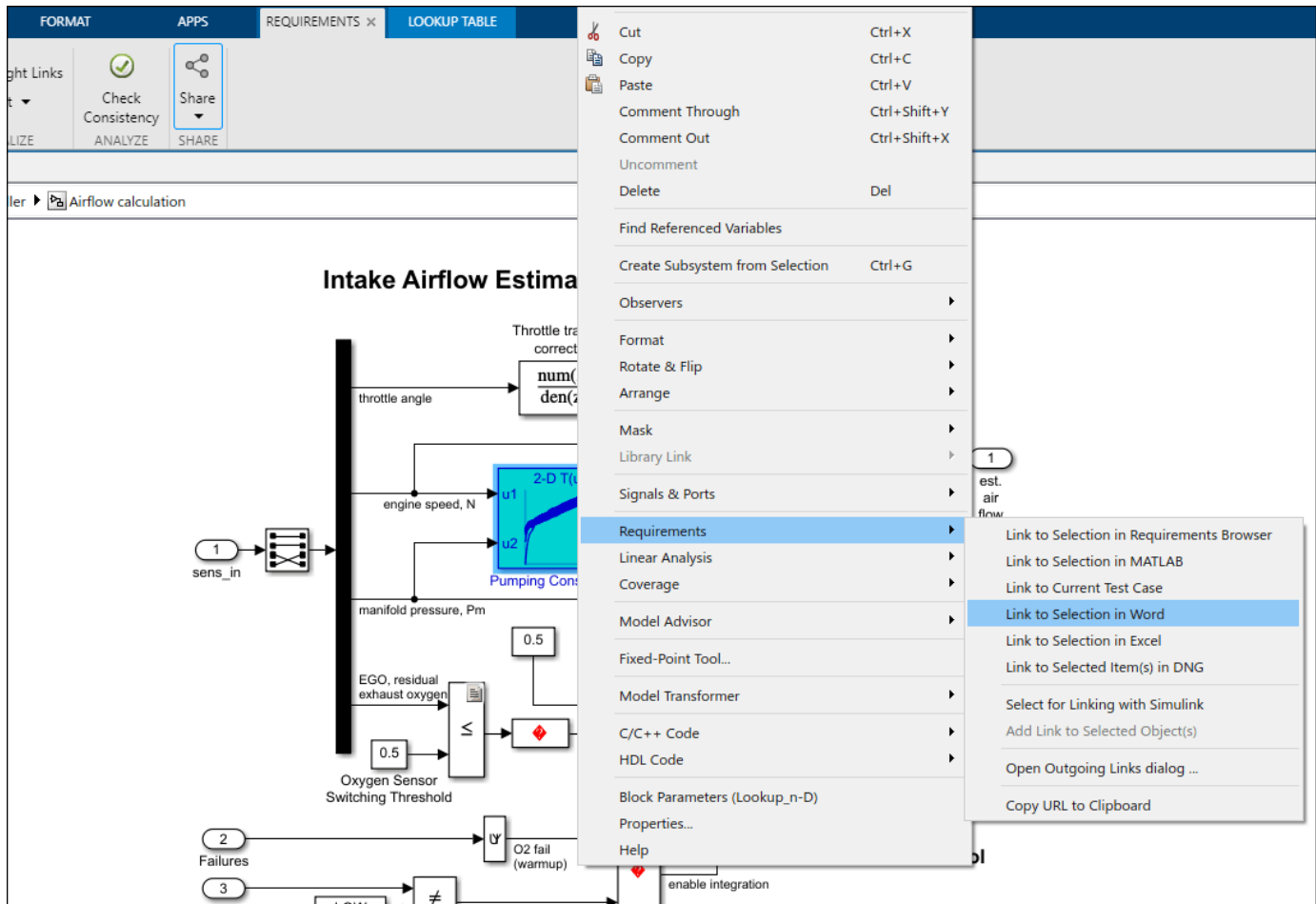


Now create new links similar to the ones you've just navigated. Note: Microsoft Word will not allow you to create the link when the document is *ReadOnly*. For the next step of this example, consider saving your own local copy of the document and using it instead of the installed document.

- In the `slvndemo_FuelSys_DesignDescription.docx` find section **2.2 Determination of pumping efficiency**.
- Select the entire header with a mouse.
- Right-click the Pumping Constant block in the Airflow calculation subsystem. If you can't find it, evaluate the following:

```
rmidemo_callback('locate', ['slvndemo_fuelsys_officereq/fuel rate controller/' ...
    'Airflow calculation/Pumping Constant']);
```

Select **Requirements > Link to Selection in Word** to create a link.





Right-click Pumping Constant block again. You should now see the newly created link at the top of the context menu. Click it to navigate to the target in section 2.2 of slvndemo_FuelSys_DesignDescription.docx.

Requirements Links in Signal Builder Blocks

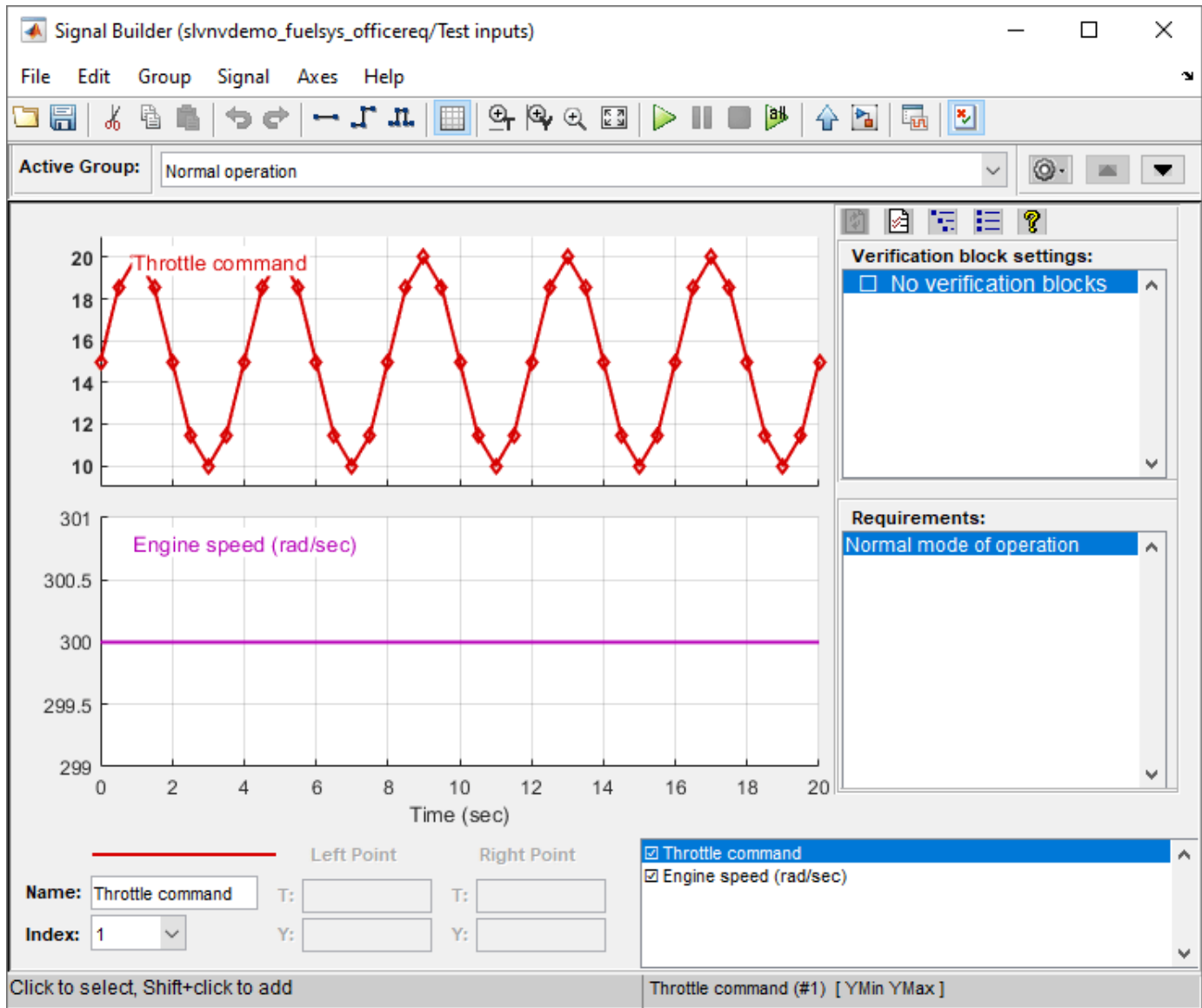
Signal links are attached to individual groups of signals, not to the Signal Builder block as a whole. Use this sort of links for test cases that are defined as Signal Builder groups.

Double-click the Test inputs Signal Builder block to see configured groups of signals. Normal operation signals are periodically depressed accelerator pedal and constant engine RPM. Navigate to the Test inputs block by evaluating the following code.

```
rmdemo_callback('locate', 'slvndemo_fuelsys_officereq/Test inputs');
```

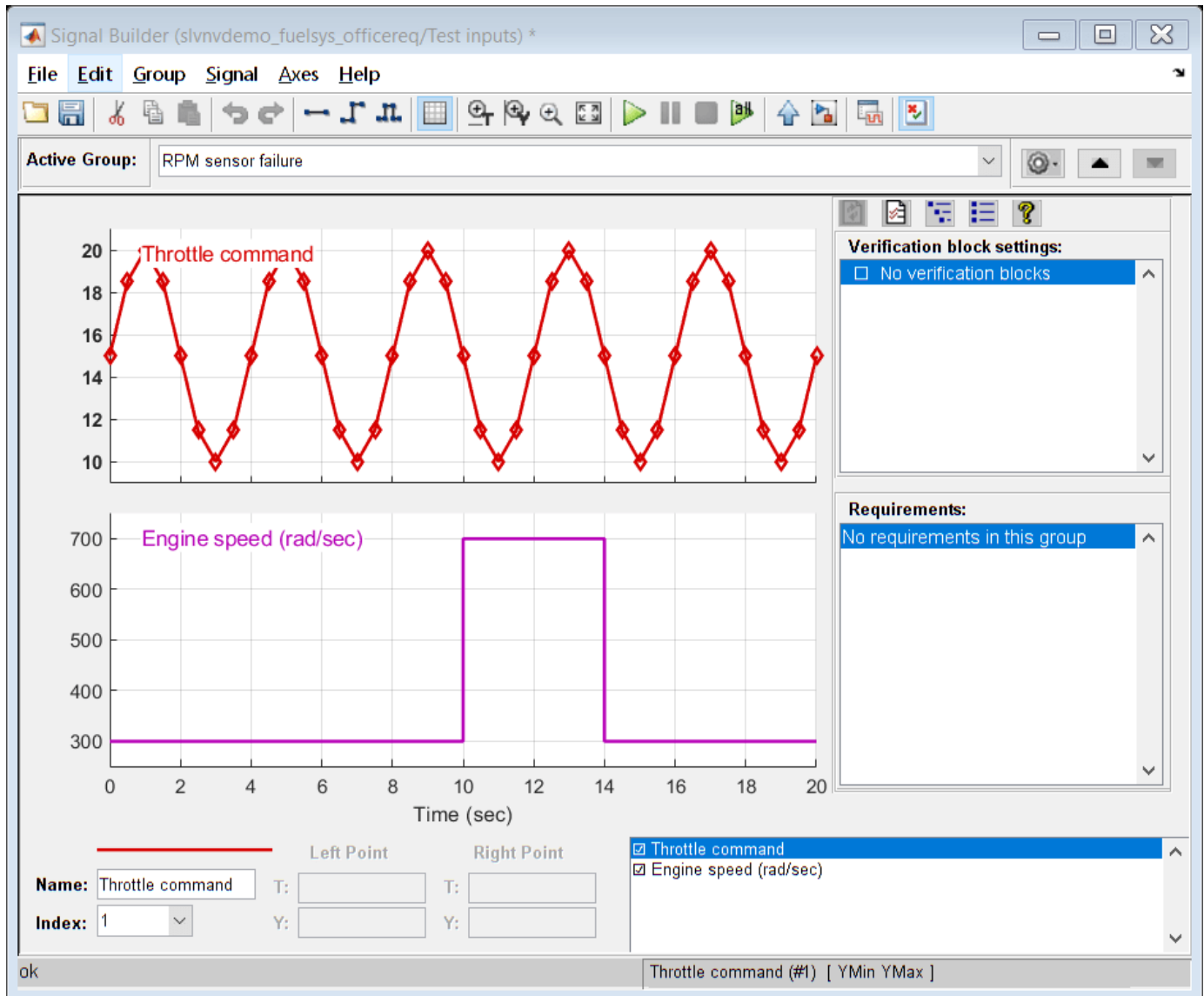
- Click the **Show verification settings** button  at the end of toolbar to display the Verification panel.
- If you do not see the **Requirements** panel below the **Verification block settings**, click the **Requirements display**  button at the top of the panel.
- Right-click the link label under **Requirements** and select **View** to open the related requirements data, this time in a Microsoft Excel document. The Simulink icon in the linked cell allows navigation back to this signal group. Alternatively, evaluate the following code:

```
rmdemo_callback('view','slvndemo_fuelsys_officereq/Test inputs',1)
```

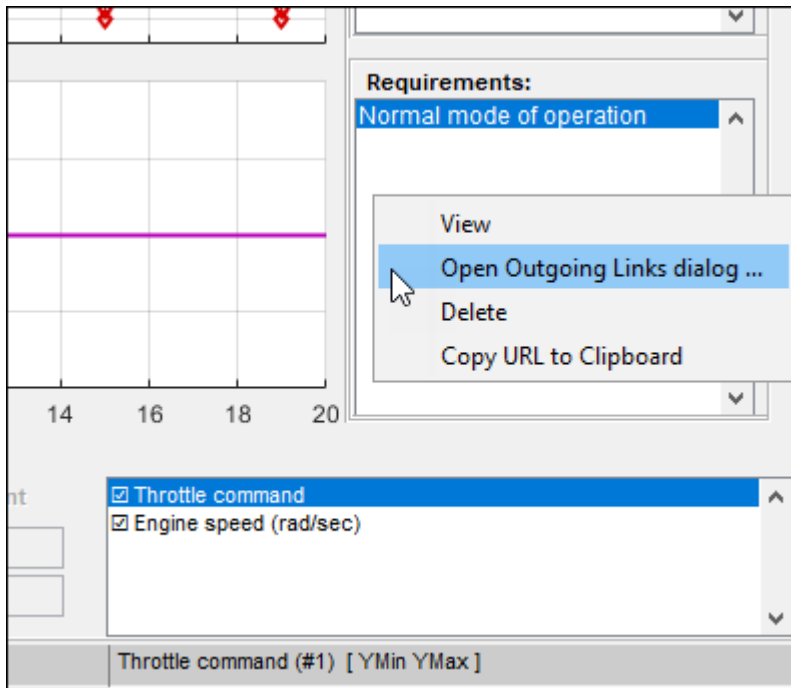


A transient RPM instability is modeled by a rectangular pulse on **Engine speed** data in the second group of signals:

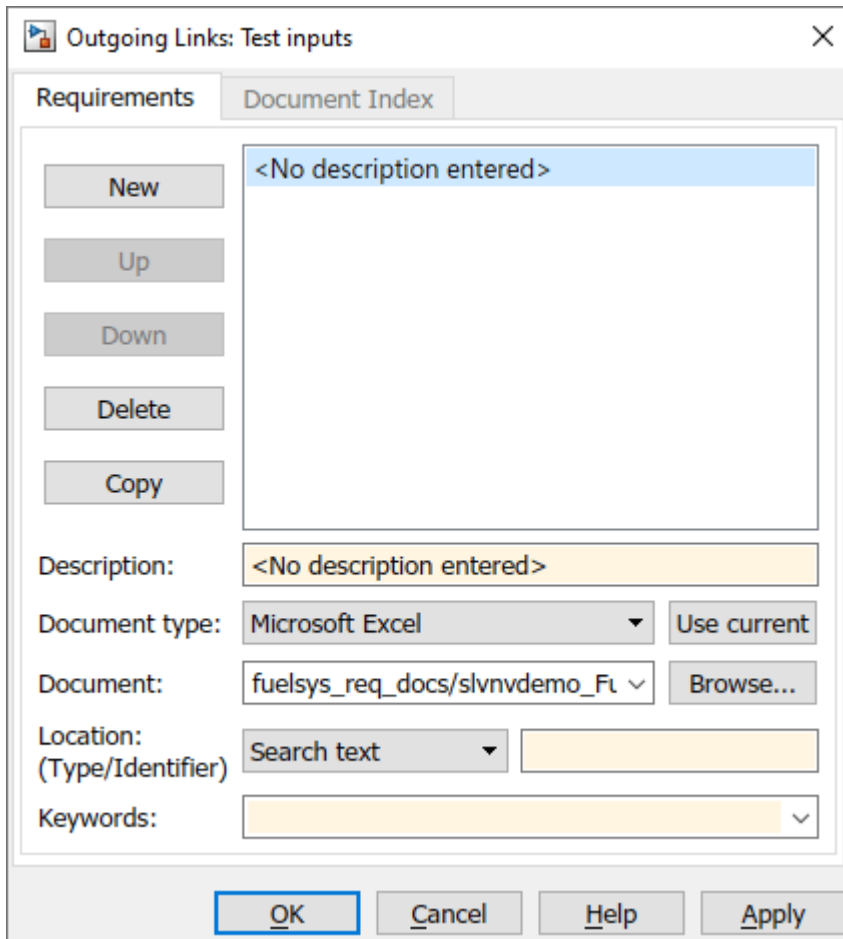
```
rmdemo_callback('signalgroup','slvndemo_fuelsys_officereq/Test inputs',2)
```

Suppose you need to link RPM sensor failure signal group to a different cell range in your Excel file. Select this signal group in the drop-down list, right-click in the empty **Requirements** and select **Open Outgoing Links dialog** from the context menu to open a dialog box.



The simplest way to add a link is to click **Browse** to find the Excel file. Then open the Excel file and select the cells you want to link to. In the **Outgoing Links** menu, click **Use Current** and a link will be created to your current selection.



- Right-click the new label under **Requirements** area and select **View** to navigate to see the target cell in **TestScenarios** file.

Generating Requirements Report

In the **Requirements** tab, click **Share > Generate Model Traceability Report** to automatically generate a report on all requirements links in the model, or evaluate the following code.

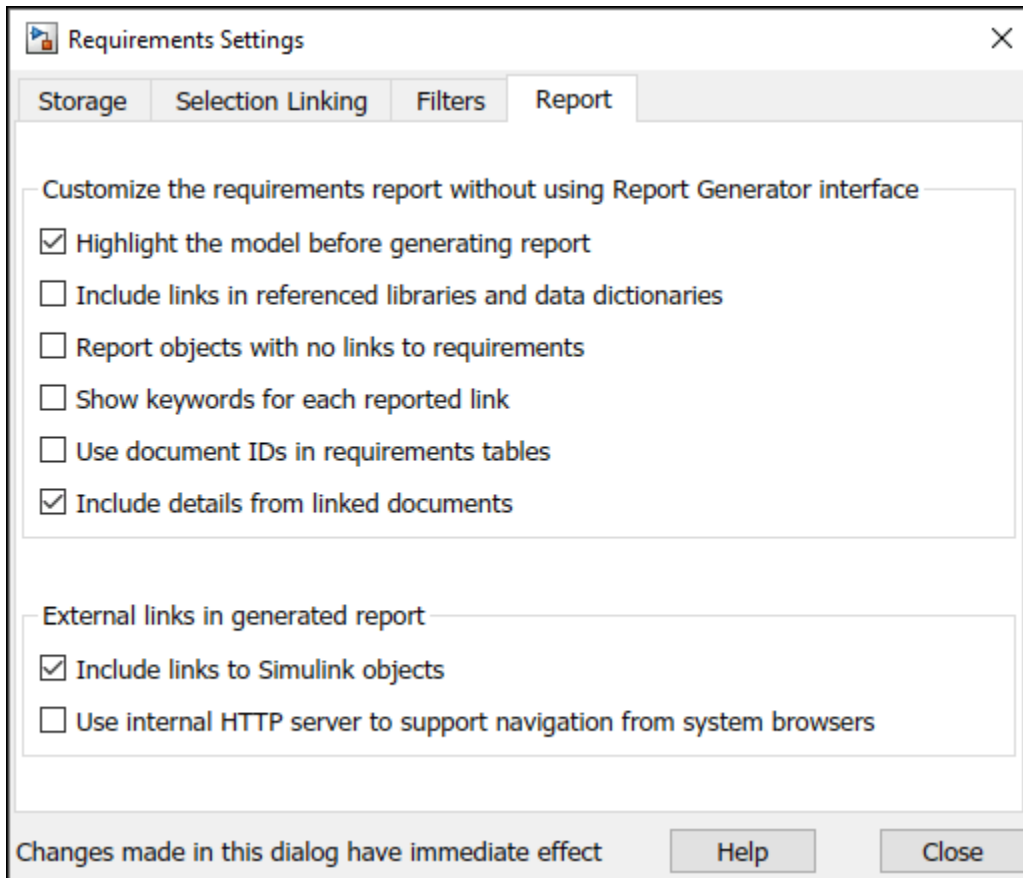
```
rmdemo_callback('report', 'slvndemo_fuelsys_officereq')
```

The default report is generated according to the template that is included with the product.

The Report Generator interface provides total control over the content of generated reports, including the creation of entirely new templates. It can be accessed by evaluating the following:

```
setedit('requirements')
```

A subset of options is also accessed in the **Requirements** tab, under **Share > Report Options**. For example, you may want to disable **Highlight the model before generating report** checkbox if the resulting report will be printed in black-and-white or viewed via projector, or you may want to include lists of objects that have no links to requirements.

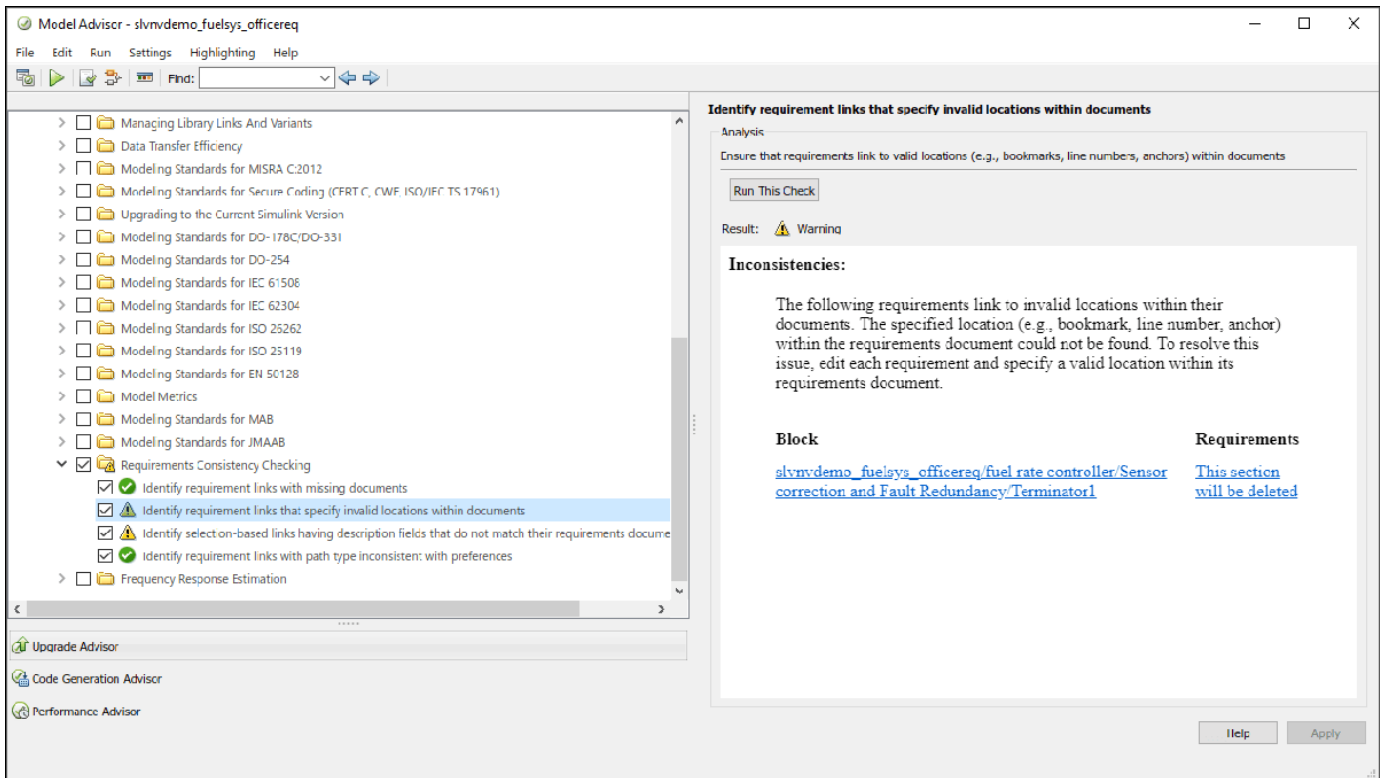


Requirements Consistency Checking

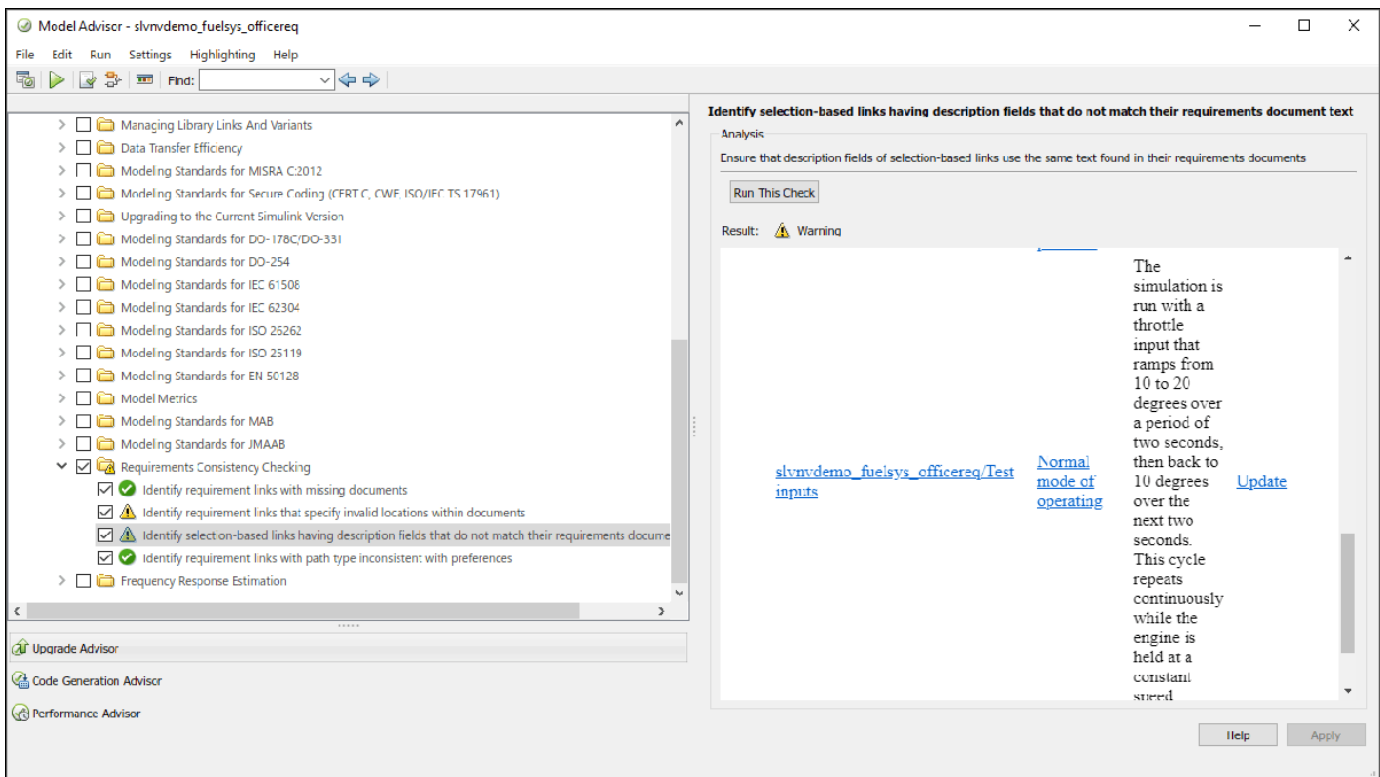
Use Model Advisor to automatically detect and fix inconsistencies in requirements links data. Click **Check Consistency** in the **Requirements** tab to open Model Advisor with only the RMI check points activated. The links are checked for missing documents, unmatched locations in documents, unmatched labels for selection-based links, and inconsistent path information. You can also evaluate the following to open Model Advisor:

```
rmidemo_callback('check','slvndemo_fuelsys_officereq')
```

Click the **Run Selected Checks** button to verify the consistency of links in your model. RMI will automatically open linked documents and check for consistency of stored data. When done, click individual check items to view the results in the right-side panel. In this example, one of the links points to invalid location in a document:



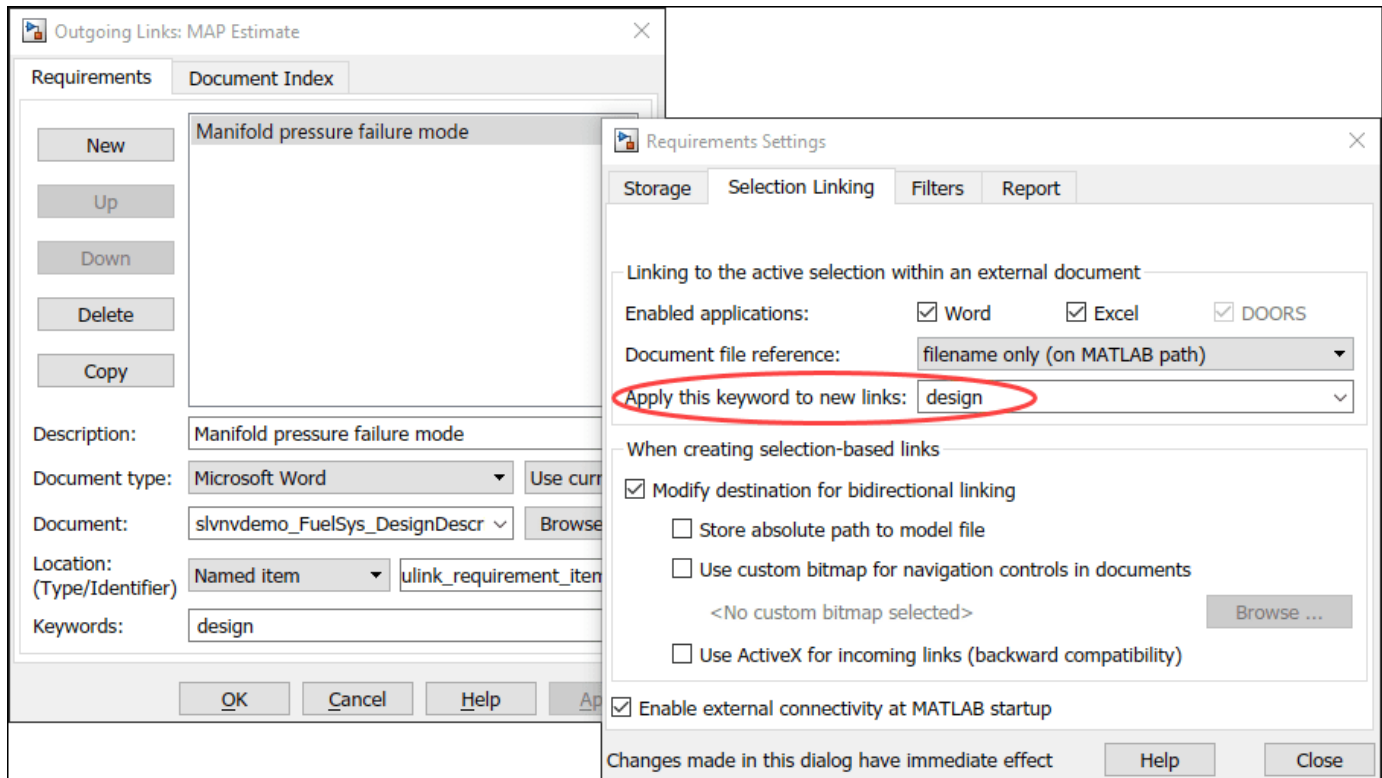
Another link has a label that does not match the original selection when the link was created:



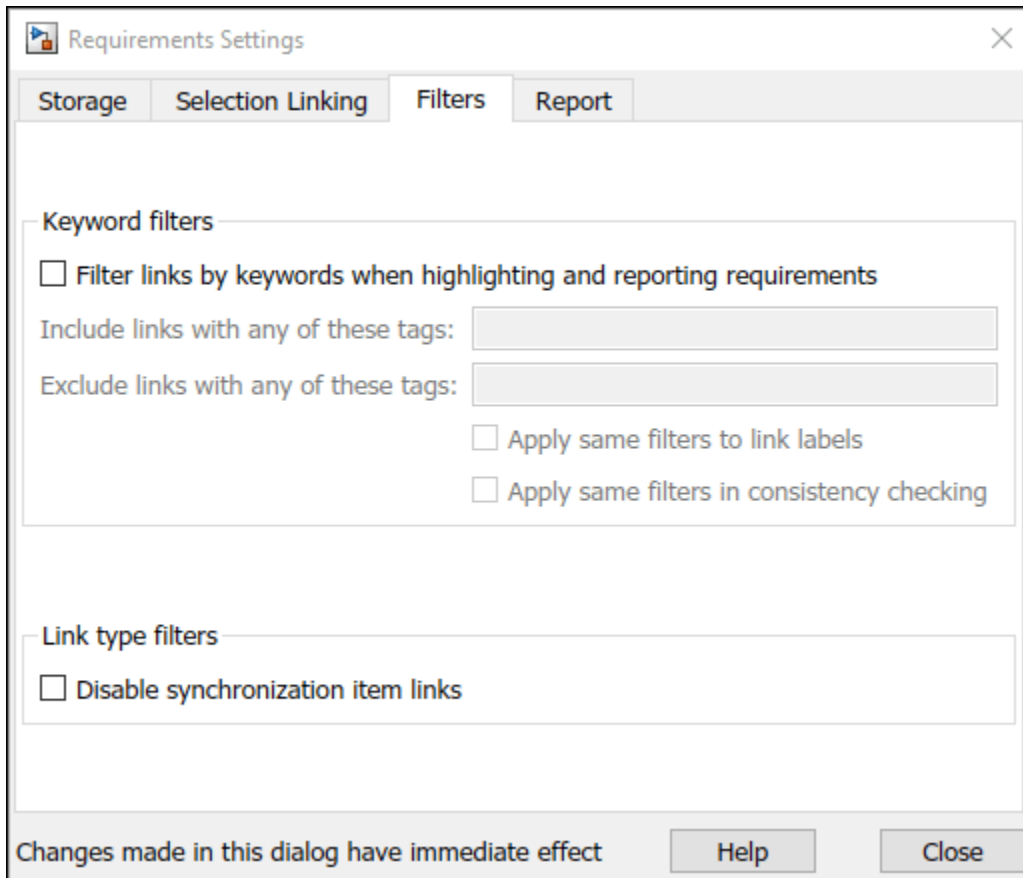
Click **Fix** or **Update** in Model Advisor report to automatically resolve reported inconsistencies. Rerun the checks to ensure reported problem is resolved.

Filtering Requirements on User Tag Property

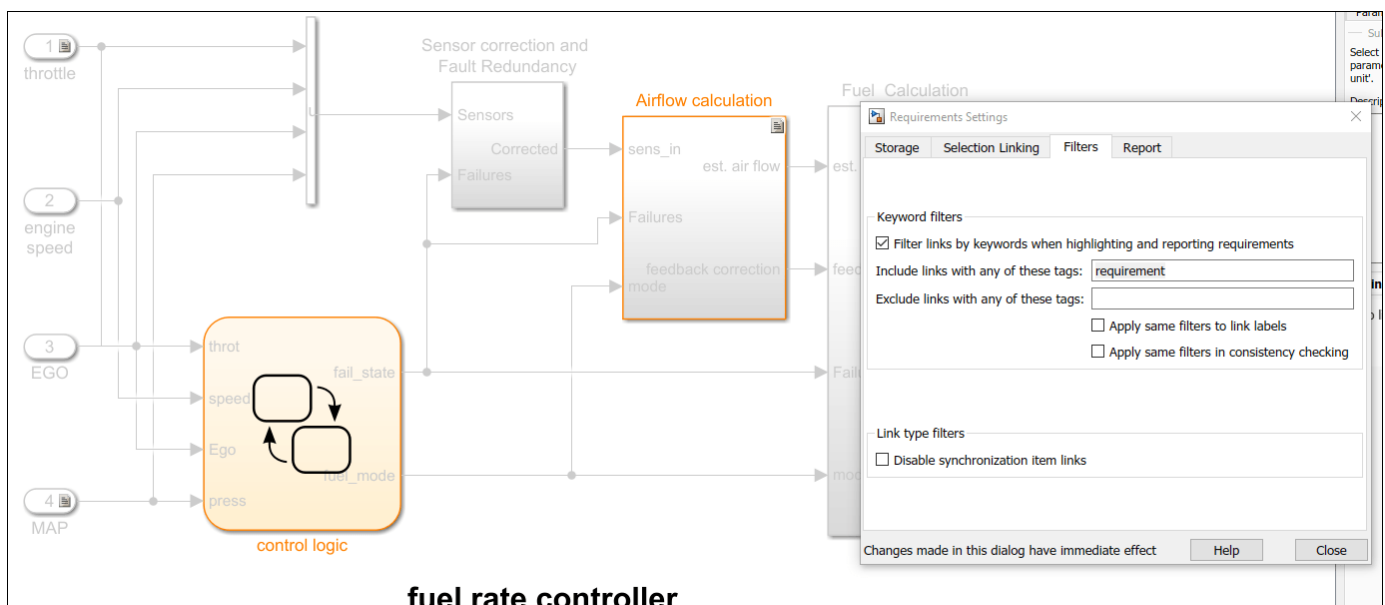
Requirements links in Simulink support an optional **User Tag** property that can store any comma-separated string values. Use these tags to distinguish between different types of links, for example, functional requirements links, design description links or testing details links. You can specify the tags when creating new links, or later via **Open Outgoing Links dialog...** dialog box.



You can later use these tags to focus your work on a subset of links, or to automatically strip a subset of links from the model. This is controlled via **Filters** tab of the **Requirements Settings** menu, opened by clicking **Link Settings > Linking Options** in the **Requirements** tab.



When model requirements are highlighted, modifying the filter setting updates the view to only show the matching requirements links. Requirements links in this example model are tagged with one of the following: "design", "requirement", "test". The view adjusts accordingly when you modify filter settings. For example, you can highlight only links that are tagged "requirement".



If you generate a report with **User Tag** filters enabled, your report content is filtered accordingly. This may be useful to focus your report on a particular subset of links.

If you run consistency checking with **User Tag** filters enabled, only links that match the given filter settings are checked. Use this to target your consistency checking to a required subset of links.

Updating All Links When Requirements Documents Are Moved or Renamed

It happens sometimes that the documents need to be renamed or moved after links were created. Use `rmidocrename` command-line utility to simultaneously adjust all links in the model.

help `rmidocrename`

```
rmidocrename - (Not recommended) Update model requirements document paths and file names.  
rmidocrename(MODEL_HANDLE, OLD_PATH, NEW_PATH)  
rmidocrename(MODEL_NAME, OLD_PATH, NEW_PATH)
```

Using `rmidocrename` is not recommended. Use `slreq.LinkSet.updateDocUri` instead.

```
rmidocrename(MODEL_HANDLE, OLD_PATH, NEW_PATH) collectively  
updates the links from a Simulink(R) 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 file name or path or  
a fragment of file name or path.  
NEW_PATH is a string that contains the new file name, path or fragment.
```

```
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 name string.
```

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.

`rmidocrename` displays the number of links modified.

Examples:

```
For the current Simulink(R) model, update all links to requirements  
files whose names contain the string 'project_0220', replacing  
with 'project_0221':  
rmidocrename(gcs, 'project_0220', 'project_0221');
```

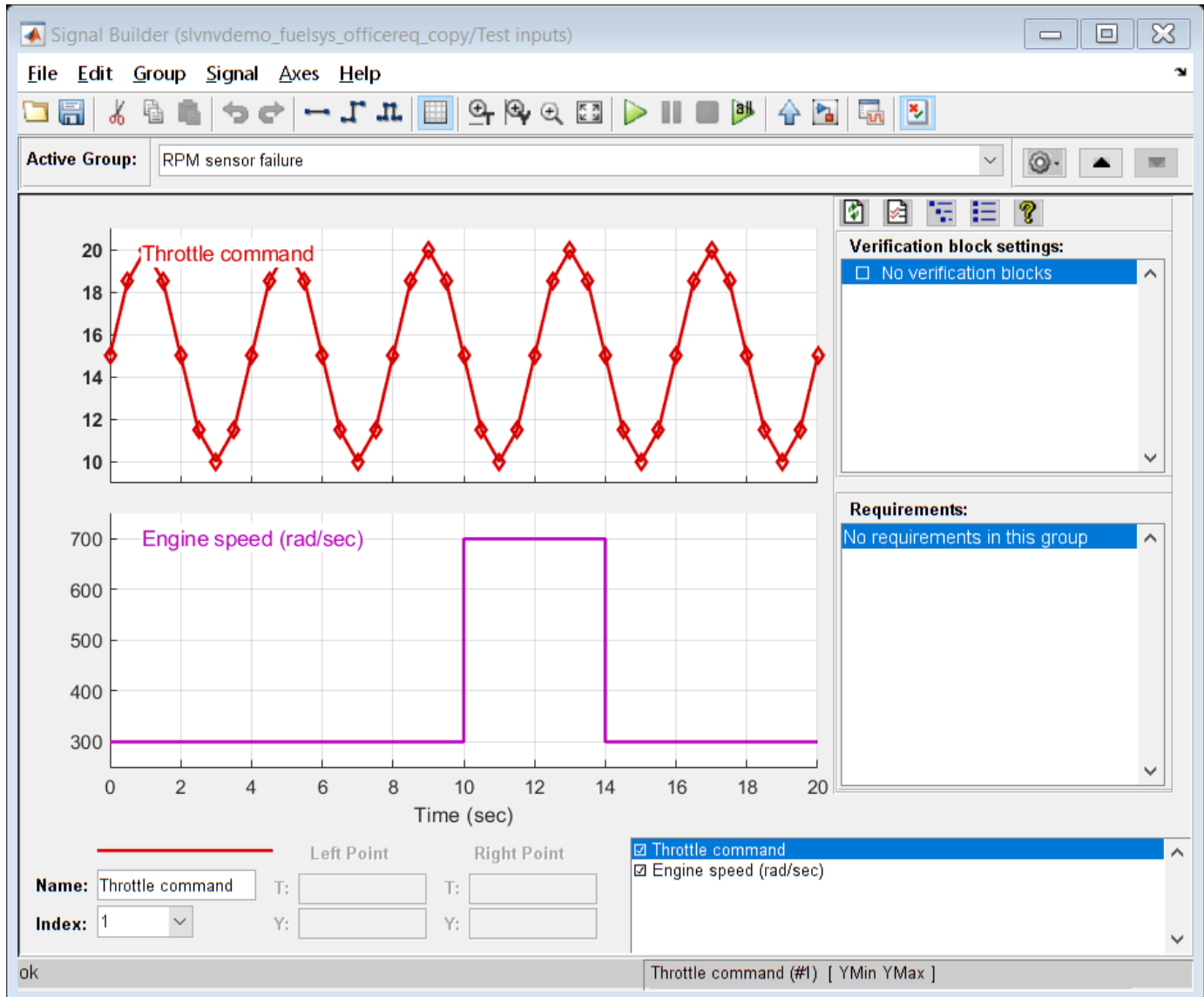
```
For the model whose handle is 3.0012, update links after all  
documents were moved from C:\My Documents to D:\Documents  
rmidocrename(3.0012, 'C:\My Documents', 'D:\Documents');
```

See also `slreq.LinkSet/updateDocUri`, `rmi`, `rmitag`

Documentation for `rmidocrename`

- Create a writable copy of the example model. For example, open the original system and save as `slvndemo_fuelsys_officereq_copy`.


```
open_system('slvnvdemo_fuelsys_officereq')
save_system('slvnvdemo_fuelsys_officereq','slvnvdemo_fuelsys_officereq_copy.slx')
```



- Highlight requirements that are tagged "requirement". These links point to slvnvdemo_FuelSys_RequirementsSpecification.docx and you need them to point to corresponding locations in Microsoft Word 2003 version of the same document.

```
rmdemo_callback('filter','slvnvdemo_fuelsys_officereq_copy','requirement')
```

- Evaluate the following code to redirect the links to the .doc document:
rmidocrename(gcs,'Specification.docx','Specification.doc')
- Partial matching in document name is performed; you do not need to specify the full name for the document, as long as the given pattern is specific enough to avoid unwanted changes. For example, replacing **.doc** with **.docx** would go wrong because **.docx** becomes **.docx**. RMI responds with the following message: "Processed 16 objects with requirements, 7 out of 18 links were

modified". Only objects with matching requirements were processed due to the current User Tag filter setting.

- Try navigating the highlighted links. For example, navigate from `Relational Operator` under `Airflow calculation`. Microsoft Word 2003 version of the document opens to a correct location.

```
rmidemo_callback('locate', ['slvndemo_fuelsys_officereq_copy/fuel rate controller/' ...
    'Airflow calculation/Relational Operator3'], 1);
```

Insert Navigation Controls into Documents to Match One-way Links

If you previously created one-way links from Simulink objects to documents, and later need to enable navigation from locations in documents back to the corresponding objects in Simulink, use the `rmiref.insertRefs` utility to insert matching "return" links into the requirements document.

One of the documents included with this example, `slvndemo_FuelSys_DesignDescription.doc` does not have Simulink navigation controls. It does have the bookmarks in locations that correspond to links tagged **design** in the example model.

- Reuse the writable copy of the model from the previous section.

```
open_system('slvndemo_fuelsys_officereq_copy')
```

- Evaluate the following code to redirect matching links from `slvndemo_FuelSys_DesignDescription.docx` to `slvndemo_FuelSys_DesignDescription.doc`. The following message appears in the command window: Processed 16 objects with requirements, 8 out of 16 links were modified.


```
rmidocrename('slvndemo_fuelsys_officereq_copy', 'Description.docx', 'Description.doc');
```
- Navigate one of the links. For example, right-click the `Airflow calculation` subsystem block, right-click and select **Requirements** and then click the linked requirement. This brings up `slvndemo_FuelSys_DesignDescription.doc`, which does not have Simulink navigation controls. If you can't find the block, evaluate the following code.

```
rmidemo_callback('locate', ['slvndemo_fuelsys_officereq_copy/fuel rate controller/' ...
    'Airflow calculation']);
```

- Run `rmiref.insertRefs('slvndemo_fuelsys_officereq_copy', 'word')` to insert document-to-model navigation controls into `slvndemo_FuelSys_DesignDescription.doc`. You see the Simulink navigation icons inserted into the document. You can now use these icons to navigate to Simulink objects in the `slvndemo_fuelsys_officereq_copy` model.

It is not possible to insert navigation controls if the specified location bookmark is missing in the document. In this example the document was not saved after the last link was created. The following warning appears in the command window: **The named item "Simulink_requirement_item_7" could not be located in the bookmarks or section headings.**

When the link in the model does not specify a location, the navigation control is inserted at the top of the document.

Remove All Simulink Reference Buttons from the Document

When Simulink navigation controls are inserted into an Microsoft Office Document, you do not necessarily have to save the document before navigating to Simulink. This allows you to temporarily insert navigation objects when needed. If you saved the document with navigation buttons inserted

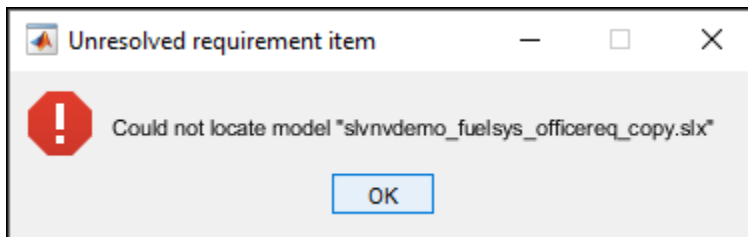
and now need to go back to a clean document, use the `rmiref.removeRefs` utility to remove the buttons. For example, perform the following steps to remove the buttons inserted in the previous step of this example:

- Make sure `slvndemo_FuelSys_DesignDescription.doc` is open and is your current Microsoft Word document.
- Run `rmiref.removeRefs('word')` to remove the buttons. RMI prompts for confirmation in command window.

Repair Links from Documents to Simulink

Simulink navigation controls embedded in requirements documents may get outdated when Simulink models are modified or moved. This results in broken links. Use the `rmiref.checkDoc` utility to detect and repair links from external documents to Simulink. In this example you will fix one broken link in `slvndemo_FuelSys_DesignDescription.docx` document.

- Run which `slvndemo_fuelsys_officereq_copy` and remove `slvndemo_fuelsys_officereq_copy.slx` from MATLAB path if exists.
- Open `slvndemo_FuelSys_DesignDescription.docx` and try to navigate a link at the very bottom of the document. The following error dialog appears:



- Run `rmiref.checkDoc('slvndemo_FuelSys_DesignDescription.docx')` to check the document for broken links. RMI highlights detected problems in the document and displays an HTML report.

Web Browser - Requirements Linking Report for "slvndemo_FuelSys_DesignDescription.docx"

Requirements Linking Report for "slvndemo_FuelSys_DesignDescription.docx" × +

Location: file:///fs-57-ah/vmgr%24/home07/ahoward/Documents/MATLAB/Examples/slrequirements-ex81896326/slvndemo_FuelSys_DesignDesc

Requirements Linking Report for "slvndemo_FuelSys_DesignDescription.docx"

Generated on 01-May-2020 16:9 [\[Refresh\]](#)

Document Name	slvndemo_FuelSys_DesignDescription.docx
Document Location	//fs-57-ah/vmgr\$/home07/ahoward/Documents/MATLAB/Examples/slrequirements-ex81896326
Last Saved	01-May-2020 16:09:14
Total links in document	7
Unique model paths	2
1 broken links:	
References with unresolved models	1 item

References with Unresolved Models - 1 unique problem in 1 link

Document content	Target model
This link is intentionally broken by removing slvndemo_fuelsys_officereq_copy.slx from MATLAB path:	slvndemo_fuelsys_officereq_copy/fuel rate controller (SubSystem)

Verified functional links in this document - 6 links

Document content	Target in Simulink
Manifold pressure failure mode	slvndemo_fuelsys_officereq/.../Sensor correction and Fault Redundancy/MAP Estimate (SubSystem)
Enriched mixture usage	slvndemo_fuelsys_officereq/.../control logic/Rich_Mixture (State)
Speed sensor failure detection	slvndemo_fuelsys_officereq/.../control logic/[speed==0 & press < zero_thresh]/Fail.INC (Transition)
Throttle Sensor	slvndemo_fuelsys_officereq/engine gas dynamics/fuel (Inport)
Manifold Absolute Pressure Sensor	slvndemo_fuelsys_officereq/fuel rate controller/Airflow calculation (SubSystem)
Mass airflow estimation	

All functional links are listed at the bottom of the report. You can navigate from this report to linked objects in Simulink (**Target in Simulink** column) and to the target locations in the document (**Document content** column).

Red font in the report highlights problems that require attention. In this case there is one broken link that references an unresolved model name. Repair the links before moving on.

Cleanup

Clear the open requirement sets and link set. Close all open Simulink models. Clear the link keyword filter.

```
slreq.clear;  
bdclose('all');  
rmipref('FilterRequireTags','');
```


Requirements Traceability with IBM Rational DOORS

- “Configure Requirements Toolbox for IBM Rational DOORS Software” on page 8-2
- “Link and Trace Requirements with IBM DOORS Next” on page 8-4
- “Navigate to Requirements in IBM Rational DOORS Databases from Simulink” on page 8-15
- “Synchronize Simulink Models with IBM Rational DOORS Databases by using Surrogate Modules” on page 8-19
- “Working with IBM Rational DOORS 9 Requirements” on page 8-30
- “Managing Requirements for Fault-Tolerant Fuel Control System (IBM Rational DOORS)” on page 8-39

Configure Requirements Toolbox for IBM Rational DOORS Software

Requirements Toolbox communicates with IBM Rational DOORS so that you can import requirements and establish links between requirements and Model-Based Design items such as Simulink model elements and tests. After you install or update MATLAB, Simulink, or IBM Rational DOORS, you must configure MATLAB to communicate with Rational DOORS. You can only integrate Requirements Toolbox and Rational DOORS on Windows platforms.

First perform the setup described in “Configure Requirements Toolbox for Interaction with Microsoft Office and IBM DOORS” on page 6-2. If Requirements Toolbox and IBM Rational DOORS communicate after performing that setup, you do not need to perform the setup described here.

Manually Install Additional Files for DOORS Software

The setup script automatically copies the required DOORS files to the installation folders. However, the script might 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.

You can also manually install the required files into the specified folders, as described in the following steps:

- 1 If the DOORS software is running, close the application.
- 2 Copy the following files from *matlabroot*\toolbox\shared\reqmgt\dxl to the *<doors_install_dir>*\lib\dxl\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\shared\reqmgt\dxl to the *<doors_install_dir>*\lib\dxl\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\dxl\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\dxl\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 using the following MATLAB command.

```
rmi setup
```

Address DXL Errors

If you try to synchronize your Simulink model to a DOORS project without configuring Requirements Toolbox for use with DOORS, you might 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 steps in “Configure Requirements Toolbox for Interaction with Microsoft Office and IBM DOORS” on page 6-2, and restart the DOORS software.

See Also

More About

- “Configure Requirements Toolbox for Interaction with Microsoft Office and IBM DOORS” on page 6-2
- “Import Requirements from IBM Rational DOORS” on page 1-42

Link and Trace Requirements with IBM DOORS Next

You can link and trace Simulink model elements and supported Model-Based Design artifacts to requirements in IBM DOORS Next (formerly known as IBM DOORS Next Generation or DNG) by importing DOORS Next requirements or by using direct linking.

If you import requirements from IBM DOORS Next to Requirements Toolbox, the imported requirements become `slreq.Reference` objects, which are called referenced requirements. When requirements change in DOORS Next, you can update the referenced requirements. The imported referenced requirements contribute to the implementation status, verification status, and change tracking. For more information, see “Import Requirements from IBM DOORS Next” on page 1-35. You can then link MATLAB or Simulink Model-Based Design artifacts or other linkable items on page 3-32 with the referenced requirements in the Simulink canvas or **Requirements Editor**. You can also navigate from the referenced requirement in Requirements Toolbox to the original requirement in DOORS Next.

If you use direct linking to link the requirements, then you create a link between the requirements in MATLAB or Simulink and the DOORS Next artifacts. You can establish traceability links and navigate directly from MATLAB or Simulink Model-Based Design artifacts to DOORS Next requirements. Direct linking does not require you to create additional files, as opposed to importing, which stores the requirements in an `.slreqx` file. However, the linking process requires additional setup steps, and the IBM DOORS Next requirements are not covered by Requirements Toolbox analyses, such as implementation status, verification status, and change tracking.

With either linking method, you can insert backlinks in DOORS Next, which are links that allow you to navigate from the requirement in DOORS Next to the artifact in MATLAB or Simulink.

Configure IBM DOORS Next Session

To interface with IBM DOORS Next, you must configure MATLAB every session.

- 1 At the MATLAB command prompt, enter:

```
slreq.dngConfigure
```

- 2 In the DOORS Server dialog box, provide the DOORS Next server address, port number, and service root as they appear in the web browser when accessing DOORS Next. Click **OK**.

If you do not see a port number in the web browser, enter the default value of 443.

- 3 In the Server Login Name and Server Login Password dialog box, enter your user name, then click **OK**. Enter your password, then press **Enter**.
- 4 In the DOORS Project dialog box, select the project and, if applicable, the configuration context. If your configuration context is not listed in the **Select configuration stream or changeset** list, load additional configurations by selecting `<more>`. To display global configurations in the configurations list, at the MATLAB command line, enter:

```
rmipref("OslcUseGlobalConfig",true);
```

For more information, see `rmipref` and “Specifying and Updating the IBM DOORS Next Configuration” on page 8-12.

MATLAB then tests the connection by opening a window in your browser. If the connection is successful, the MATLAB Connector Test dialog box appears with a confirmation message. Click **OK**. If the dialog does not appear or if an error appears after you enter `slreq.dngConfigure`, see the Tips for using `slreq.dngConfigure`.

Note If you plan to create direct links to requirements in IBM DOORS Next, you must leave the test connection browser window open, because this instance of the web browser is authenticated to communicate with MATLAB. You can re-open the test connection browser window by copying and pasting this address in the browser address bar: `https://localhost:31515/matlab/oslc/inboundTest`. Use this browser to select requirements in your IBM DOORS Next project to create direct links.

Linking with Referenced Requirements

Use this approach when you want to link requirements in MATLAB or Simulink and track the implementation, verification, and change tracking in Simulink.

First, import requirements by selecting a DOORS Next module or by creating a query. For more information, see “Import Requirements from IBM DOORS Next” on page 1-35.

After you import DOORS Next requirements into a requirement set, you can link these referenced requirements the same way you link other `slreq.Reference` objects. For example, you can open a Simulink model or test, select a model element or test case, and then create a link to the selected referenced requirement in the **Requirements Editor**. You can also create a link without leaving the Simulink model by clicking and dragging a requirement from the Requirements Perspective. For more information, see:

- “Link Requirements to Model Artifacts”
- “Link Test Cases to Requirements”
- “Link Blocks and Requirements” on page 3-2

If you update the requirements in DOORS Next after importing them to Requirements Toolbox, you can update the requirement set to reflect the changes. For more information, see “Update Referenced Requirements” on page 1-39. If the update changes or removes a referenced requirement that has links, the links have a change issue. For more information, see “Track Changes to Requirement Links” on page 5-3.

After you create a link, you can edit the link type in the **Requirements Editor**. For more information, see “Link Types” on page 3-34.

Inserting Backlinks in DOORS Next

When you import requirements from DOORS Next from a module and create links to the imported referenced requirements from items in MATLAB or Simulink, you can manually insert backlinks in the DOORS Next module:

- 1 Open the **Requirements Editor**. At the MATLAB command prompt, enter:

```
slreq.editor
```
- 2 In the **Requirements Editor**, click **Show Links** in the toolstrip to view the loaded link sets.
- 3 Select the link set that contains the links that do not have backlinks in your DOORS Next module. Right-click the link set and select **Update Backlinks**.
- 4 A dialog box displays the number of links that were checked for existing backlinks and the number of backlinks added. Click **OK**.

You can navigate to the original requirement by selecting the requirement in the **Requirements Editor** and clicking **Show in Document**.

When viewing DOORS Next items outside the module context, expand the **Links** pane, which displays backlinks to MATLAB or Simulink under **Link to**. When working in the module context, select the item. In the right pane, select **Selected Artifact**, then select **Artifact Links**. The backlinks are displayed under **Link to**.

Directly Linking DOORS Next Requirements

Use this approach when you prefer to link directly to requirements in DOORS Next. Direct links do not require importing requirements.

After the setup is complete, you can establish direct links either by right-clicking an item and using the context menu, or by using the Outgoing Links dialog.

Link to Selected Requirements by Using the Context Menu

When you link to requirements in DOORS Next by using the context menu, you can insert a backlink when the link is created. You can also create the link in the module context and in the specified stream or changeset. If you create the link in the module context and insert a backlink, the backlink is also inserted in the module context and in the specified stream or changeset. To read more about streams and changesets, see “Specifying and Updating the IBM DOORS Next Configuration” on page 8-12. Linking to requirements in DOORS Next by using the context menu requires additional setup in the **Requirements Editor**, and in your DOORS Next server.

To create links using the context menu, you first need to ensure that selection-based linking is enabled with IBM DOORS Next.

- 1 Open the **Requirements Editor** by entering the following at the MATLAB command prompt:
`s\req.editor`
- 2 In the **Links** section of the toolstrip, click **Preferences**.
- 3 In the Requirement Settings dialog, in the **Selection Linking** tab, next to **Enabled Applications**, ensure that **DOORS** is selected.

Install the MathWorks Requirements Toolbox widget in IBM DOORS Next and enable dropins. For more information, see “Configure an IBM DOORS Next Server for Integration with Requirements Toolbox” on page 6-2. To confirm the widget is operating, in your DOORS Next project, in the **Artifacts** tab, select an item and verify that the widget contents update as expected.

Tip Pin the **Mini Dashboard** to the page so that it is always visible and you know which selected ID is communicated to MATLAB.

The screenshot shows the IBM DOORS Next interface. At the top, there are tabs for 'Project Dashboard', 'Artifacts', 'Reviews', and 'Reports'. Below this is a 'Mini Dashboard' pane containing a 'MathWorks Requirements Toolbox' widget. The widget displays 'MATLAB session configuration' with 'Project: AMR Sample' and 'Context: AMR Sample Initial Stream'. A table shows a selected requirement with ID '713' and text 'The handheld unit shall have a mass no gr...'. Below the table is a link 'Query links from SL'. To the right, the 'Artifacts' pane shows a folder structure: 'AMR Sample' containing '00 Admin', '02 Referenc', and '01 Requireme'. A table of artifacts is shown with columns 'ID' and 'Name'. The row with ID '713' and name 'The handheld unit shall have a mass no greater than 2.25kg.' is highlighted in orange.

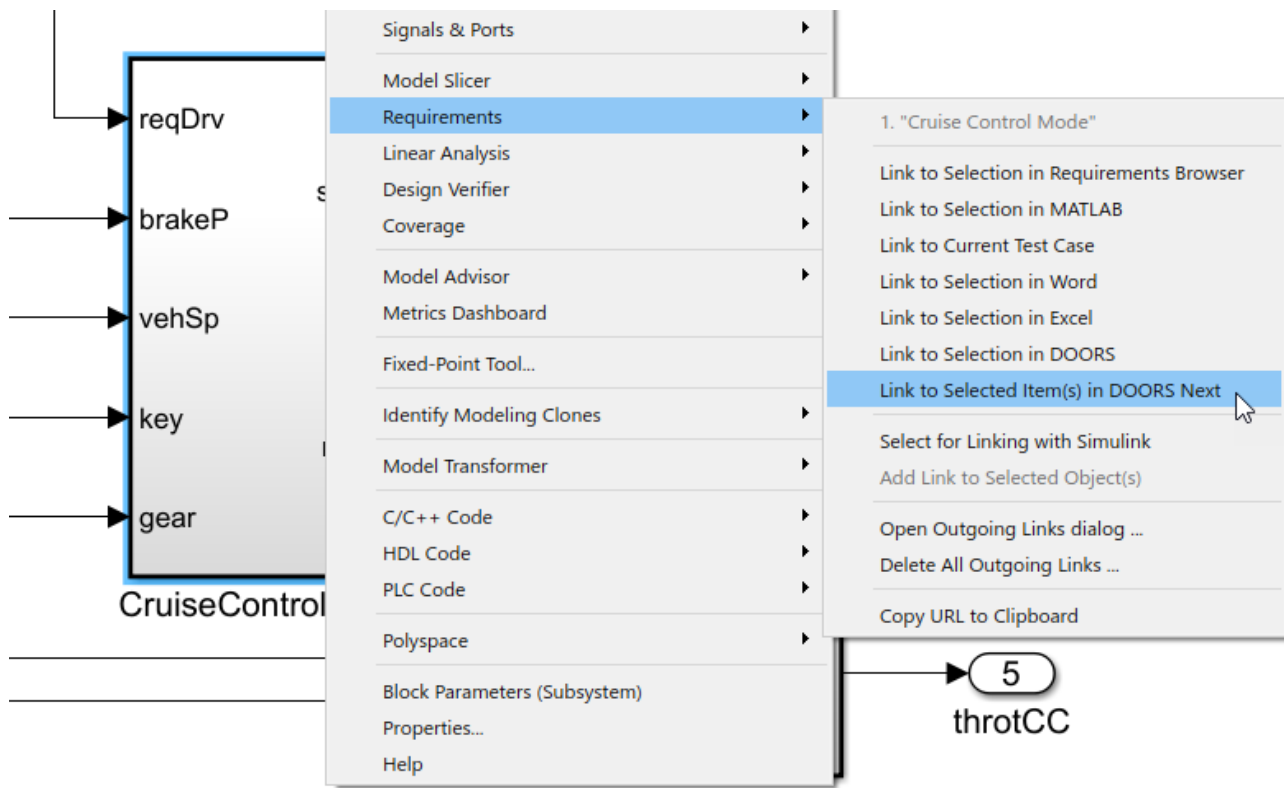
You can verify that MATLAB is receiving information about your selection in DOORS Next. At the MATLAB command prompt, enter:

```
oslc.selection
```

The returned number should correspond to the numeric ID of the selected item in the DOORS Next browser.

When the widget is operating, you can create links between Simulink linkable items and DOORS Next when you use the context menu:

- 1 In your DOORS Next project, select the **Artifacts** tab.
- 2 Select the requirements that you want to link to by selecting the check box next to the requirement. The requirements that you select are displayed in the MathWorks Requirements Toolbox widget in the **Mini Dashboard** pane.
- 3 In Simulink, right-click the Simulink model element that you want to link to the selected IBM DOORS Next requirements. Select **Requirements > Link to Selected Item(s) in DOORS Next** from the context menu.
- 4 The DOORS Link Target dialog appears. If the widget is functioning as expected, then the **Project Area** and **Requirement ID** fields are populated with information from your selection.
- 5 To create the link in the module context, select **Link in module context**. Then set the **Module context** to the module that the requirement belongs to.
- 6 To insert a backlink in DOORS Next, select **Insert backlink**. If the link is created in the module context, the backlink is also inserted in the module context.
- 7 Click **OK** to create the link and, if selected, insert the backlink.



To navigate to the linked requirement in DOORS Next, right-click the same Simulink model element and select **Requirements**. The link appears at the top of the context menu.

If the widget in IBM DOORS Next is unavailable or fails to communicate with MATLAB due to security restrictions, you can create the link without selecting a requirement in DOORS Next:

- 1 In Simulink, right-click the Simulink model element that you want to link to the selected IBM DOORS Next requirements. Select **Requirements > Link to Selected Item(s) in DOORS Next** from the context menu.
- 2 The DOORS Link Target dialog box appears, but no information is populated. Set the **Project Area** to the project that you want to work with.
- 3 In the **Requirement ID** field, enter the DOORS Next numeric ID of the requirement that you want to link to.
- 4 To create the link in the module context, select **Link in module context**. Then, set the **Module context** to the module that the requirement belongs to.

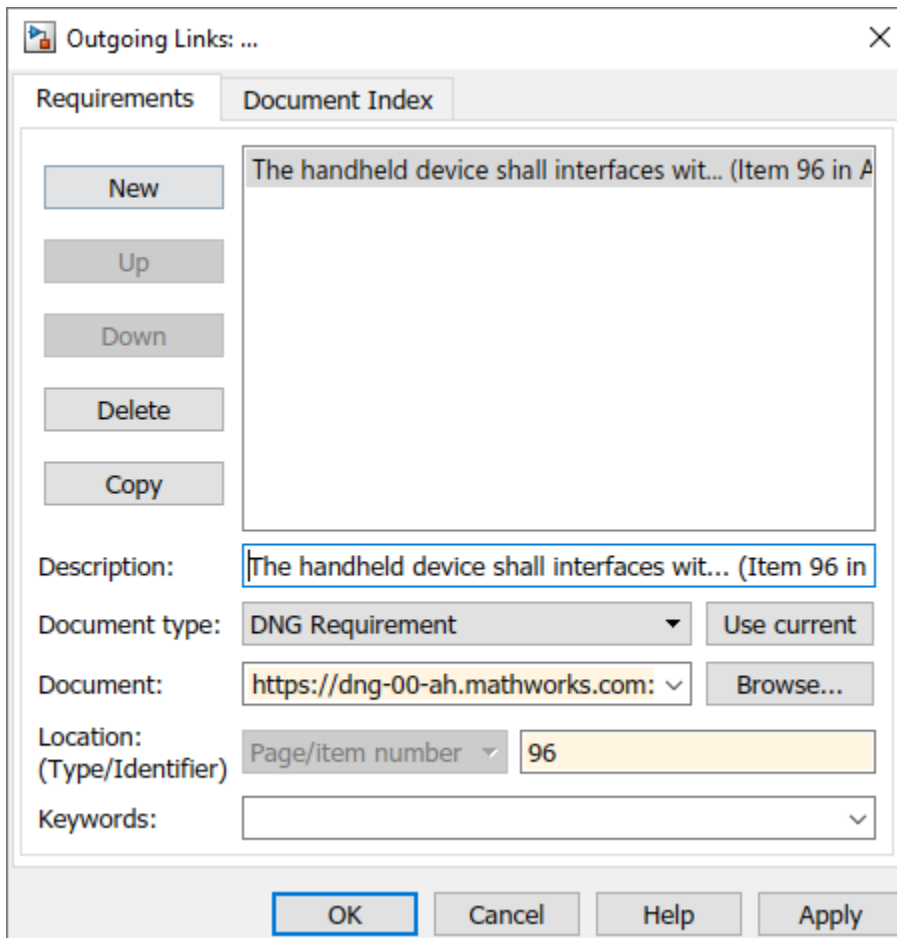
Note If you create a link to a requirement in the module context and then create more links to requirements in the same module, the links are created in the module context.

- 5 To insert a backlink in DOORS Next, select **Insert backlink**. If the link is created in the module context, the backlink is also inserted in the module context.
- 6 Click **OK** to create the link and, if selected, insert the backlink.

Link to Requirements by Using the Outgoing Links Dialog Box

Creating links by using the **Index** tab of the Outgoing Links dialog does not require communication between MATLAB and the system browser.

- 1 Right-click the Simulink model element that you want to link to IBM DOORS Next requirements.
- 2 Select **Requirements > Open Outgoing Links dialog**.
- 3 In the Outgoing Links dialog, click **New** and set **Document type** to IBM DOORS Next.
- 4 Click **Browse**. In the DOORS Project dialog box, select the project to work with and, depending on the project, select the configuration context. If your configuration context is not listed in the drop-down, load more configurations by selecting <more>.
- 5 The next step depends on whether you want to link to the requirement in the module context.
 - If you want to create links in the module context:
 - 1 Click the **Document Index** tab to see the list of module names.
 - 2 Double-click the module that you want to link to.
 - 3 When the list updates, select the requirement that you want to link to.
 - If your project doesn't have modules or if you don't want to create the link in the module context, enter the numeric ID of the DOORS Next link target requirement in the **Location** field.
- 6 To create the link, click **OK** or **Apply** to create the link.



When you create links using the Index tab in Outgoing Links dialog or by entering the ID in the dialog, the link is created without a backlink. After you create the links, you can insert backlinks in your DOORS Next project for requirements that are missing backlinks. See “Insert Missing Backlinks” on page 8-10.

Insert Missing Backlinks

If a requirement in your DOORS Next project does not contain a backlink because a backlink was not inserted when the link was created or because the backlink was deleted, you can insert missing backlinks:

- 1 Open the Simulink model or other artifact that contains direct links to requirements in DOORS Next.
- 2 Open the **Requirements Editor** by entering the following at the MATLAB command prompt:

```
slreq.editor
```
- 3 Select **Show Links** and select the link set containing the link that does not contain a backlink.
- 4 Right-click the link set and select **Update backlinks**. The Backlinks Checked dialog appears and displays the number of missing backlinks added.

Note When you insert missing backlinks with this method, backlinks are added for direct links in the link set where the destination project matches your currently configured DOORS Next project. If your link set contains links to other DOORS Next projects, these links will not be processed. You will need to re-run the **Update backlinks** procedure after re-configuring your MATLAB session for the other project to insert backlinks in the other project.

Each backlink in DOORS Next is independent of the link stored in Requirements Toolbox. If you later decide to delete the link in Simulink, the backlink will remain in DOORS Next until manually deleted. If you delete the backlink in DOORS Next, the change does not propagate to Requirements Toolbox.

Additionally, the backlinks in DOORS Next are visible to users of this configuration context, including users who do not have access to the Simulink source artifact.

To read more about updating backlinks, see “Manage Navigation Backlinks in External Requirements Documents” on page 3-51.

Navigate Between DOORS Next Requirements and Directly Linked Items

Once you've directly linked a linkable item on page 3-32 in MATLAB or Simulink to a DOORS Next requirement, you can navigate to the requirement from MATLAB by using the **Requirements Editor**.

- 1 Open the **Requirements Editor** by entering the following at the MATLAB command prompt:

```
slreq.editor
```
- 2 Select **Show Links** and select the link that you want to navigate.
- 3 In the right pane, under **Properties**, click the hyperlink next to **Destination** to navigate to the requirement in DOORS Next.

If you inserted backlinks in your DOORS Next project then you can navigate from the requirement in DOORS Next to the linked item in MATLAB or Simulink:

- 1 In your DOORS Next project, in the desired stream or changeset, select the **Artifacts** tab.
- 2 Select the desired requirement. If the requirement was linked in the module context, select the requirement in that module context.
- 3 In the right pane, ensure the **Selected Artifact** tab is selected.
- 4 In the right pane, select **Artifact Links**.
- 5 Under **Links to**, click the backlink to navigate to the linked item in MATLAB or Simulink.

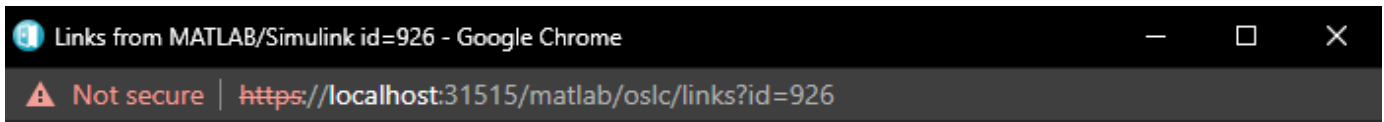
The screenshot displays the IBM DOORS Next interface for a project titled "744 Meter Interface Subsystem Requirements Specification". The main area shows a list of requirements with columns for ID and Contents. Requirement 926 is selected and highlighted in orange. The detailed view on the right shows the selected artifact: "926: <The meter interface unit shall take readings of pressure with a maximum interval of 1 second>". Below the artifact name, it indicates "No Tags Defined" and provides details about the module, description, component, team ownership, and creation date. A section titled "Artifact Links (1)" shows a link to "sampleTime in crs_controller.slx".

ID	Contents
926	The meter interface unit shall take readings of pressure with a maximum interval of 1 second
895	The meter interface unit shall measure pressure to an accuracy of +/- 2%
1022	The meter interface unit shall measure pressures between 0.5 bar and 10 bar
1100	The meter interface unit shall log a low pressure event if the pressure is below 1.8 bar
1095	The meter interface unit shall log a high pressure event if the pressure is above 6.5 bar
1086	- 3.1.2 detectLeak
811	The meter interface unit shall calculate a rolling average pressure over a period of 24 hours +/- 1 hour
1071	The meter interface unit shall calculate the average pressure for each hour of the day over a 7 day cycle
1090	The meter interface unit shall compare instantaneous readings to the historical average for the hour of the day
830	The meter interface unit shall compare the average for the hour of the day to the rolling average for the 24 hour period.
804	The meter interface unit shall log a leak if the instantaneous reading is more than 10% different from the historical average for the hour of the day

If you didn't insert backlinks in your DOORS Next project, you can use the MathWorks Requirements Toolbox widget to query links for a given requirement.

- 1 In your DOORS Next project, in the desired stream or changeset, select the **Artifacts** tab.
- 2 Select the desired requirement. Ensure that the MathWorks Requirements Toolbox widget updates to reflect the selected requirement.
- 3 In the MathWorks Requirements Toolbox widget, click **Query Links from SL**. The Links from MATLAB/Simulink window opens in the browser.

Note The linked artifact in MATLAB or Simulink must be loaded for the link to appear in the browser window.



Links from MATLAB/Simulink id=926

Source Artifact	Source Object	Linked Document	Version	Item
crs_controller.slx	sampleTime in crs_controller.slx	AMR with CM	AMR with CM Initial Stream	926

[Manage Linked Configurations](#)

- Click the hyperlink under **Source Object** to navigate to the item in MATLAB or Simulink.

Specifying and Updating the IBM DOORS Next Configuration

Projects with configuration management enabled in IBM DOORS Next support multiple branches called streams and changesets (which are also referred to as configurations). Requirements Toolbox enables you to update the outgoing link destination for an existing link in Simulink to the same requirement in a different stream or changeset.

Specifying the Configuration Stream or Changeset

Select the IBM DOORS Next project and the stream or changeset you want to work with. At the MATLAB command prompt, enter:

```
slreq.dngConfigure
```

For more information about the function, see `slreq.dngConfigure`.

Updating Stored Stream or Changeset by API

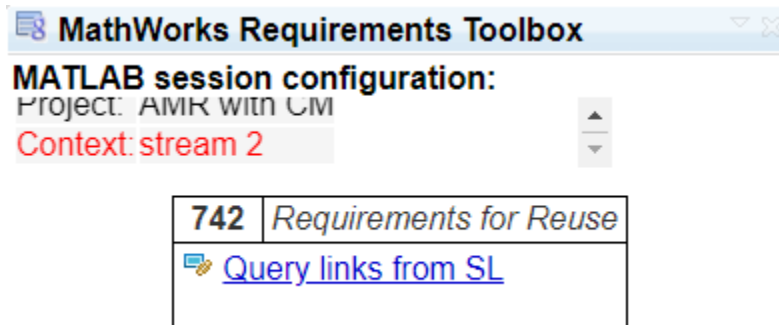
Requirements Toolbox provides functions to manage your DOORS Next requirements when your stream or changeset changes:

- Find the number of links in an `slreq.LinkSet` object to a specific DOORS Next stream or changeset with `slreq.dngCountLinks`.
- Query your DOORS Next project for known streams or changesets with `slreq.dngGetProjectConfig`.
- Identify streams or changesets linked in an `slreq.LinkSet` object with `slreq.dngGetUsedConfig`.
- Update existing links to point to a different stream or changeset of the DOORS Next requirements when the stream or changeset changes with `slreq.dngUpdateConfig`.

Using the MathWorks Requirements Toolbox Widget to Synchronize and Update Session Context

The MathWorks Requirements Toolbox widget displays information about the current configuration stream context in Requirements Toolbox. The widget indicates a mismatch between the active

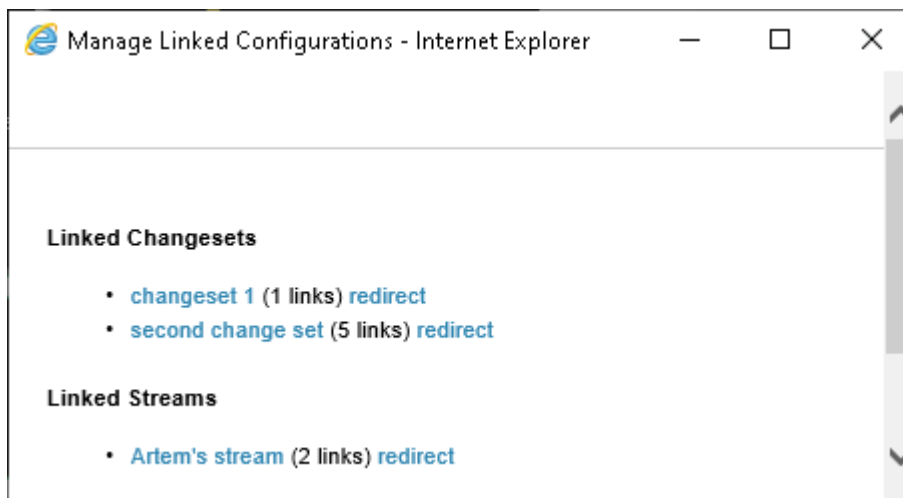
configuration stream contexts in Requirements Toolbox and in IBM DOORS Next by displaying and highlighting the active Requirements Toolbox configuration stream context in red:



To resolve a mismatch, click the red highlighted text in the widget and click **Update** in the DNG Configuration Context Mismatch dialog box. Alternatively, you can change the active configuration stream in IBM DOORS Next.

You can update the configuration context for existing links by either using the functions listed in “Updating Stored Stream or Changeset by API” on page 8-12 or by using the **Query Links from SL** hyperlink in the MathWorks Requirements Toolbox widget.

- 1 In DOORS Next, under the MathWorks Requirements Toolbox widget, click **Query Links from SL**. A new window opens in the browser with a summary of links for the selected requirements in DOORS Next.
- 2 Click the **Managed link configurations** hyperlink to display the report on DOORS Next links in the current MATLAB session and grouped by the target configuration context attribute.



- 3 Click **redirect** for the group of links that you want to associate with a different configuration context.
- 4 When the window updates, click the stream or changeset you want to associate with.
- 5 Locate one of the streams or changesets whose links you updated, and confirm that the link now brings you to the expected configuration of the linked requirement.

See Also

slreq.dngConfigure | **Requirements Editor**

More About

- “Import Requirements from IBM DOORS Next” on page 1-35
- “Manage Navigation Backlinks in External Requirements Documents” on page 3-51

Navigate to Requirements in IBM Rational DOORS Databases from Simulink

Enable Linking from IBM Rational DOORS Databases to Simulink Objects

By default, the RMI does not insert navigation objects into requirements documents. If you want to insert a navigation object into the requirements document when you create a link from a Simulink object to a requirement, you must change the RMI's settings. The following tutorial uses the `sldemo_fuelsys` example model to illustrate how to do this.

To enable linking from the DOORS database to the example model:

- 1 Open the model `sldemo_fuelsys`. At the MATLAB command line, enter:

```
openExample('simulink_automotive/ModelingAFaultTolerantFuelControlSystemExample')
```

Note You can modify requirements settings in the Requirements Settings dialog box. These settings are global and not specific to open models. Changes you make apply not only to open models, but also persist for models you subsequently open. For more information about these settings, see “Requirements Settings” on page 6-10.

- 2 In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, ensure that **Layout > Requirements Browser** is selected. In the **Requirements** pane, in the **View** drop-down, select **Links**. In the **Requirements** tab, select **Link Settings > Linking Options**.

The Requirements Settings dialog box opens.

- 3 Click the **Selection Linking** tab.
- 4 Select **Modify destination for bidirectional linking**.

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

- 5 Select **Store absolute path to model file**.

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

If you add requirements to a model that is not on the MATLAB path, you must select this option to enable linking from your requirements document to your model.

- 6 In the **Apply this keyword to new links** field, enter one or more user keywords to apply to the links that you create.

For more information about user keywords, see “User Keywords and Requirements Filtering” on page 6-11.

- 7 Click **Close** to close the Requirements Settings dialog box. Keep the `sldemo_fuelsys` model open.

Insert Navigation Objects into IBM Rational DOORS Requirements


When you enable **Modify destination for bidirectional linking** as described in “Enable Linking from IBM Rational DOORS Databases to Simulink Objects” on page 8-15, the RMI can insert a


navigation object into both the Simulink object and its associated DOORS requirement. This tutorial uses the `sldemo_fuelsys` example model to illustrate how to do this. For this tutorial, you also need a DOORS formal module that contains requirements.

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

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

- 5 Right-click the Simulink object (in this case, the `fuel_rate_control` subsystem) and select **Requirements > Link to Selection in DOORS**.

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

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

- 6 Close the model.

Note When you navigate to a DOORS requirement from outside the software, the DOORS module opens in read-only mode. If you want to modify the DOORS module, open the module using DOORS software.

Insert Navigation Objects to Multiple Simulink Objects

If you have several Simulink objects that correspond to one requirement, you can link them all to that requirement with a single navigation object. This eliminates the need to insert multiple navigation objects for a single requirement. The Simulink objects must be available in the same model diagram or Stateflow chart.

The workflow for linking multiple Simulink objects to one DOORS requirement is as follows:

- 1 Make sure that you have enabled **Modify destination for bidirectional linking**.
- 2 Select the DOORS requirement to link to.
- 3 Select the Simulink objects that need to link to that requirement.
- 4 Right-click one of the objects and select **Requirements Traceability > Link to Selection in DOORS**.

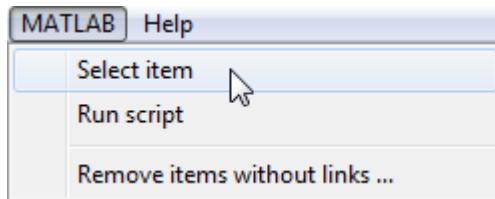
A single navigation object is inserted at the selected requirement.

- 5 Double-click the navigation object in DOORS to highlight the Simulink objects that are linked to that requirement.

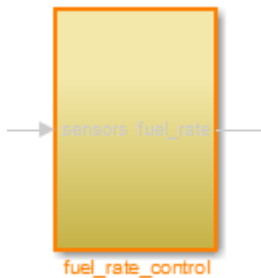
Navigate Between IBM Rational DOORS Requirement and Model Object

In “Insert Navigation Objects into IBM Rational DOORS Requirements” on page 8-15, you created a 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**.

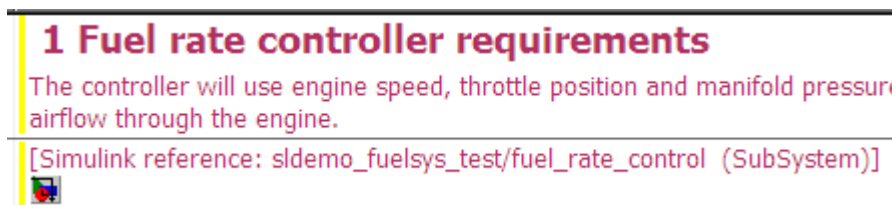


Your version of the `sldemo_fuelsys` model opens, with the `fuel_rate_control` subsystem highlighted.



- 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.



Why Add Navigation Objects to IBM Rational 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 Simulink objects to requirements managed by external applications, including the DOORS software.

When you create a link from a Simulink object to a DOORS requirement, the RMI stores the link data in Simulink. Using this link, you can navigate from the Simulink object to its associated requirement.

You can also configure the RMI to insert a navigation object in the DOORS database. This navigation object serves as a link from the DOORS requirement to its associated Simulink object.

To insert navigation objects into a DOORS database, you must have write access to the DOORS database.

Customize IBM Rational DOORS Navigation Objects

If the RMI is configured to modify the destination for bidirectional linking as described in “Enable Linking from IBM Rational DOORS Databases to Simulink Objects” on page 8-15, the RMI can insert a navigation object into your requirements document. This object looks like the icon for the Simulink software: 

Note In IBM Rational DOORS requirements documents, clicking a navigation object does not navigate back to your Simulink object. Select **MATLAB > Select object** to find the Simulink object that contains the requirements link.

To use an icon of your choosing for the navigation object:

- 1 In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, select **Link Settings > Linking Options**.
- 2 Select the **Selection Linking** tab.
- 3 Select **Modify destination for bidirectional linking**.

Selecting this option enables the **Use custom bitmap for navigation controls in documents** option.

- 4 Select **Use custom bitmap for navigation controls in documents**.
- 5 Click **Browse** to locate the file you want to use for the navigation objects.

For best results, use an icon file (.ico) or a small (16×16 or 32×32) bitmap image (.bmp) file for the navigation object. Other types of image files might give unpredictable results.

- 6 Select the desired file to use for navigation objects and click **Open**.
- 7 Close the Requirements Settings dialog box.

The next time you insert a navigation object into a requirements document, the RMI uses the file you selected.

Tip You can specify a custom template for labels of requirements links to DOORS objects. For more information, see the `rmi` command.

Synchronize Simulink Models with IBM Rational DOORS Databases by using Surrogate Modules

Synchronize 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.

- 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.

```
openExample('sf_car.slx')
```

- 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 Save the changes to the model.
- 6 In the `sf_car` model, navigate to the **Apps** tab and open the **Requirements Manager**.
- 7 In the **Requirements** tab, select **Share > Synchronize with DOORS**.

The DOORS synchronization settings dialog box opens.

- 8 For this tutorial, accept the default synchronization options.

The default option under **Extra mapping additionally to objects with links**, **None**, creates objects in the surrogate module only for the model and any model objects with links to DOORS requirements.

Note For more information about the synchronization options, see “Customize IBM Rational DOORS Synchronization” on page 8-23.

- 9 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, a formal module named `sf_car_doors`, 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 shows the IBM Rational DOORS interface with a table of blocks. The table has four columns: ID, Block Name, Block Type, and Block Deleted?. The blocks are as follows:

ID	Block Name	Block Type	Block Deleted?
1	1 sf_car_doors	Block Diagram	False
2	1.1 Engine	SubSystem	False
3	1.1.1 engine torque	Lookup_n-D	False
4	1.2 transmission	SubSystem	False

10 Save the surrogate module and the model.

Create Links Between Surrogate Module and Formal Module in an IBM Rational DOORS Database

The surrogate module is the interface between the DOORS formal module that contains your requirements and the Simulink model. To establish links between the surrogate module and the requirements module, copy the link information from the model to the surrogate module:

- 1 Open the sf_car_doors model.
- 2 In the **Requirements** tab, select **Share > Synchronize with DOORS**.
- 3 In the DOORS synchronization settings dialog box, select two options:
 - **Update links during synchronization**
 - **from Simulink to DOORS**.
- 4 Click **Synchronize**.

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 same IBM Rational DOORS interface as before, but with red triangles in the 'Block Type' column for the '1.1.1 engine torque' and '1.2 transmission' rows, indicating that links have been established.

ID	Block Name	Block Type	Block Deleted?
1	1 sf_car_doors	Block Diagram	False
2	1.1 Engine	SubSystem	False
3	1.1.1 engine torque	Lookup_n-D	False
4	1.2 transmission	SubSystem	False

Resynchronize IBM Rational DOORS 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 In the **Requirements** tab, select **Share > Synchronize with DOORS**.
- 3 In the DOORS synchronization settings dialog box, set the **Extra mapping additionally to objects with links** option to Complete - All blocks, subsystems, states, and transitions. Click **Synchronize**.

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

91	1.10.7 Tout	Outport	False
92	1.11 vehicle mph (yellow) & throttle %	Scope	False
93	1.12 vehicle mph (yellow) & throttle %1	Scope	False

- 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.

91	1.10.7 Tout	Outport	False
92	1.11 vehicle mph (yellow) & throttle %	Scope	False
93	1.12 vehicle mph (yellow) & throttle %1	Scope	True

- 5 Save the surrogate module.
- 6 Save the `sf_car_doors` model.

Navigate with the Surrogate Module

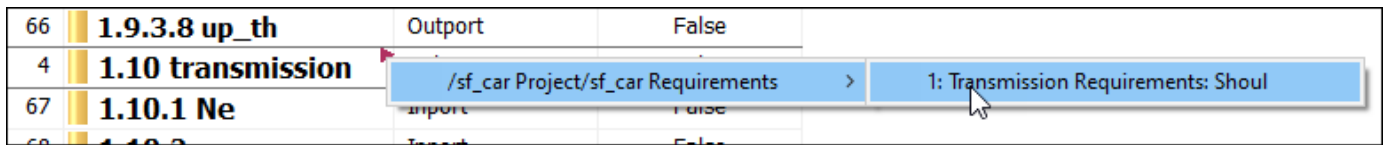
Navigate Between Requirements and the Surrogate Module in the DOORS Database

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.

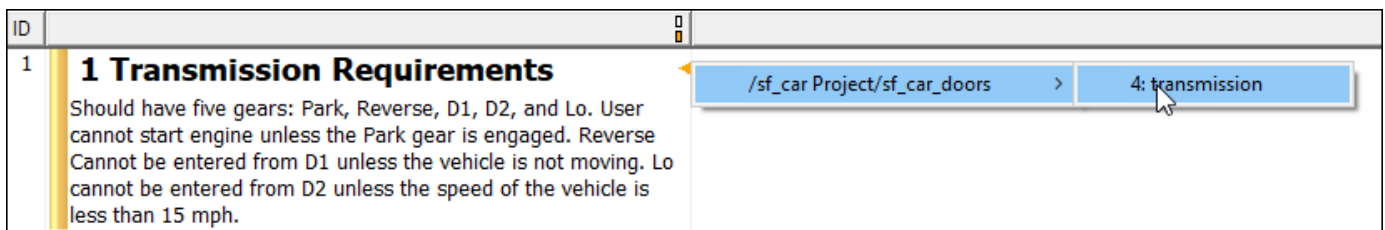


- 2 Select the requirement name.

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.

Navigate Between DOORS Requirements and the Simulink Module via the Surrogate Module

You can create links that allow you to navigate from Simulink objects to DOORS requirements and from DOORS requirements to the model. If you synchronize your model, the surrogate module serves as an intermediary for the navigation in both directions. The surrogate module allows you to navigate in both directions even if you remove the direct link from the model object to the DOORS formal module.

Navigate 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.

Navigate 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.

Customize IBM Rational DOORS Synchronization

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 verify the 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 “Document Path Storage” on page 12-36.
Extra mapping additionally to objects with links	Determines the completeness of the Simulink model representation in the DOORS surrogate module. None specifies synchronizing only those Simulink objects that have linked requirements, and their parent objects. For more information about these synchronization options, see “Customize the Level of Detail in Synchronization” on page 8-25.
Update links during synchronization	Specifies updating any unmatched links the RMI encounters during synchronization, as designated in the Copy unmatched links and Delete unmatched links options.

DOORS Settings Option	Description
Copy unmatched links	<p>During synchronization, selecting the following options has the following results:</p> <ul style="list-style-type: none"> • from Simulink to DOORS: For links between the model and the formal module, the RMI creates matching links between the DOORS surrogate and formal modules. • from DOORS to Simulink: For links between the DOORS surrogate and formal modules, the RMI creates matching links between the model and the DOORS modules.
Delete unmatched links	<p>During synchronization, selecting the following options has the following results:</p> <ul style="list-style-type: none"> • Remove unmatched in DOORS: For links between the formal and surrogate modules, when 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 from Simulink to DOORS option. • Remove unmatched in Simulink: For links between the model and the DOORS modules, when 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 from DOORS to Simulink option.
Save DOORS surrogate module	<p>After the synchronization, saves changes to the surrogate module and updates the version of the surrogate module in the DOORS database.</p>
Save Simulink model (recommended)	<p>After the synchronization, saves changes to the model. If you use a version control system, selecting this option changes the version of the model.</p>

Resynchronize a Model with a Different Surrogate Module

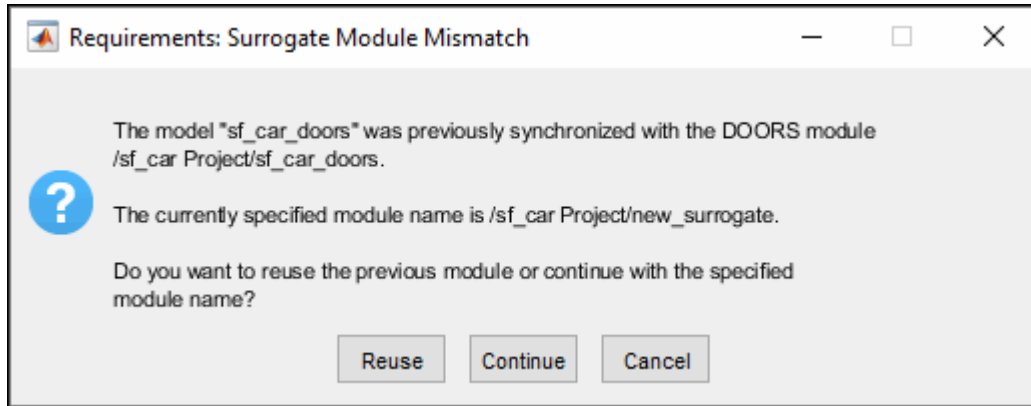
You can synchronize the same Simulink model with a new DOORS surrogate module. For example, you might want the surrogate module to contain only objects that have requirements to DOORS, rather than all objects in the model. In this case, you can change the synchronization options to reduce the level of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

Customize 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 “Synchronize a Simulink Model to Create a Surrogate Module” on page 8-19, 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.

Drop-Down List Option	Description
Complete - All blocks, subsystems, states and transitions	Copies <i>all</i> blocks, subsystems, states, and transitions to the surrogate module.

Resynchronize 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 “Synchronize a Simulink Model to Create a Surrogate Module” on page 8-19 and “Create Links Between Surrogate Module and Formal Module in an IBM Rational DOORS Database” on page 8-20.

- 1 Open the `sf_car_doors` model that you synchronized in “Synchronize a Simulink Model to Create a Surrogate Module” on page 8-19 and again in “Create Links Between Surrogate Module and Formal Module in an IBM Rational DOORS Database” on page 8-20.

- 2 In the **Requirements** tab, select **Share > 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 “Synchronize a Simulink Model to Create a Surrogate Module” on page 8-19.

For information about how the RMI resolves the path to the requirements document, see “Document Path Storage” on page 12-36.

- 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	1 sf_car_doors	Block Diagram	False
2	1.1 Engine	SubSystem	False
5	1.1.1 Ti	Inport	False
6	1.1.2 throttle	Inport	False
7	1.1.3 Integrator	Integrator	False
8	1.1.4 Sum	Sum	False
3	1.1.5 engine torque	Lookup_n-D	False
9	1.1.6 engine + impeller inertia	Gain	False
10	1.1.7 Ne	Outport	False
11	1.2 Mux	Mux	False
12	1.3 User Inputs:Passing Maneuver	Signal Group	False
13	1.4 User Inputs:Gradual Acceleration	Signal Group	False
14	1.5 User Inputs:Hard braking	Signal Group	False
15	1.6 User Inputs:Coasting	Signal Group	False
16	1.7 Vehicle	SubSystem	False
17	1.7.1 brake torque	Inport	False
18	1.7.2 transmission output torque	Inport	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 “Resynchronize to Include All Simulink Objects” on page 8-26:

- 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.

Synchronization with IBM Rational DOORS Surrogate Modules

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 DOORS model hierarchy.

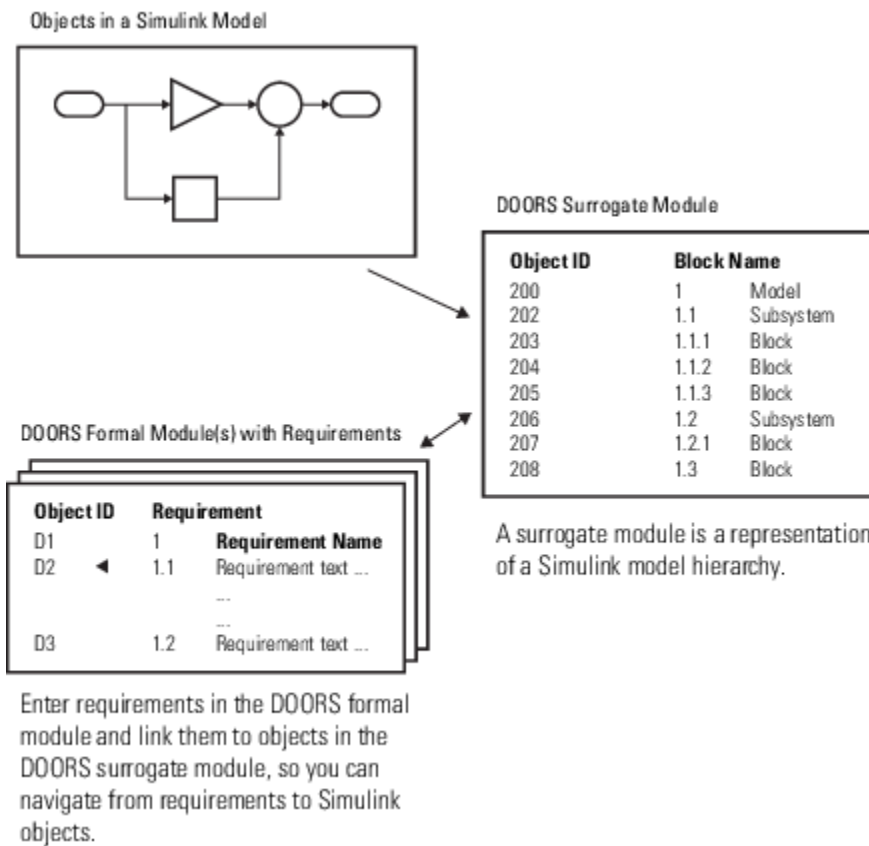
When you synchronize a model for the first time, the DOORS software creates a surrogate module. The surrogate module contains a representation of the model, depending on your synchronization settings. (To learn how to customize the links and level of detail in the synchronization, see “Customize IBM Rational DOORS Synchronization” on page 8-23.)

If you create or remove model objects or 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.



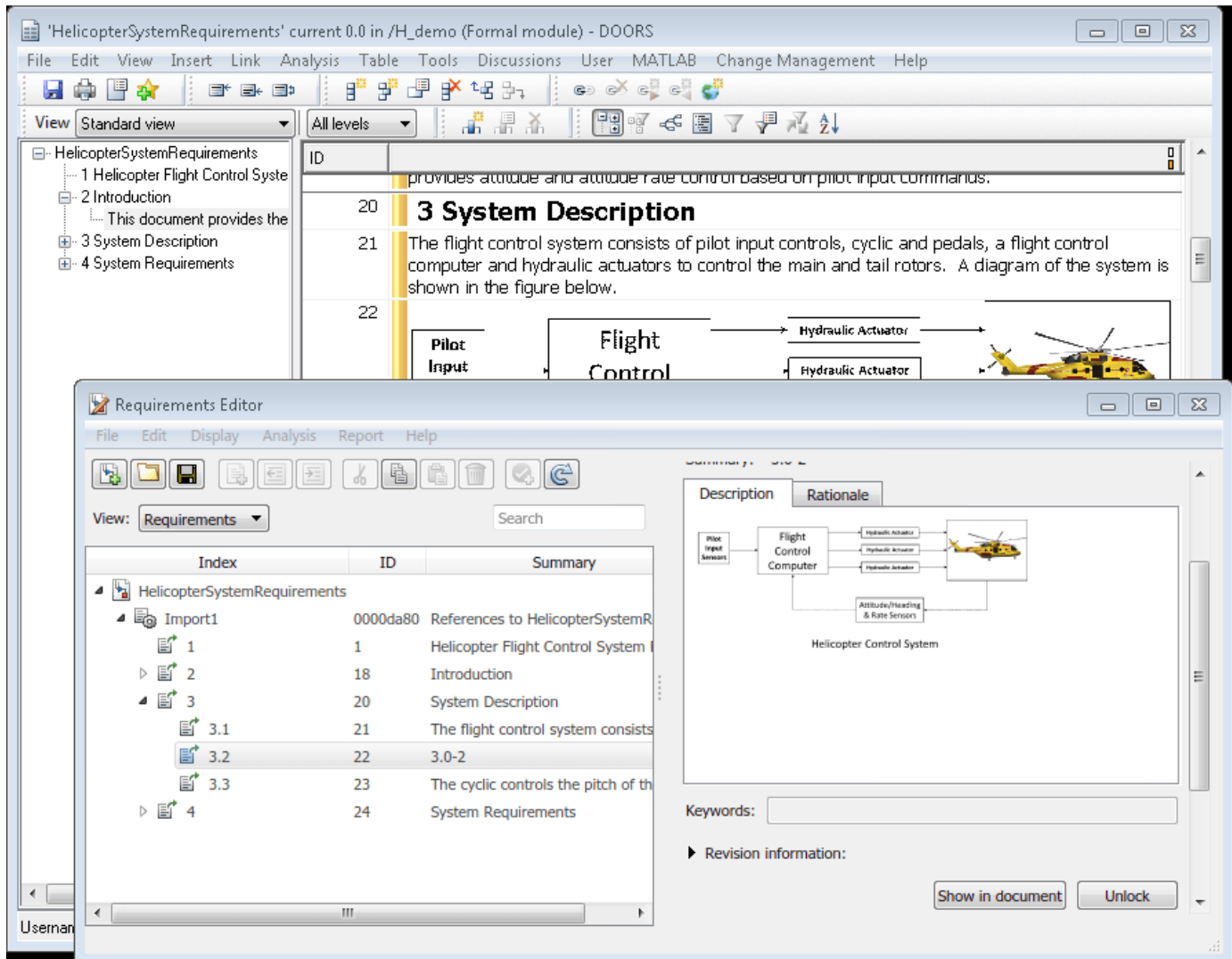
Advantages of Synchronizing Your Model with a Surrogate Module

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.

Working with IBM Rational DOORS 9 Requirements

How to import, link, and update requirements from IBM® Rational® DOORS® 9. Working with DOORS 9 is supported on Microsoft® Windows®.



Setup for IBM Rational DOORS

Configure the requirements management interface for interaction with IBM Rational DOORS by following the instructions in "Configure Requirements Toolbox for Interaction with Microsoft Office and IBM DOORS" on page 6-2.

Overview of Workflow with DOORS

You can import requirements from DOORS into the Simulink® environment, then establish traceability from your model to DOORS requirements through the imported references. Traceability is bi-directional. If DOORS requirements change, you can update the references in Requirements Toolbox™ while maintaining traceability. Additionally:

- You can establish traceability from MATLAB® and Simulink to DOORS without modifying DOORS Formal or Link modules.
- You can link between design, tests, and requirements without leaving the Simulink Editor.
- You can establish traceability from low-level requirements in Simulink to high-level requirements in DOORS.
- You can identify gaps in implementation and verification using metrics in Requirements Toolbox.
- Change detection and cross-domain traceability can be used to conduct change impact analysis.

If you have existing Simulink artifacts that are linked to DOORS with previous versions of the Requirements Management Interface, update your existing links. See the **Update Model Link Destinations** section in “Migrating Requirements Management Interface Data to Requirements Toolbox” on page 6-16.

Import a DOORS Module

You can import a DOORS requirements module or a subset of requirements from a module by using a filter. For more information, see “Import Requirements from IBM Rational DOORS” on page 1-42.

To navigate between the imported requirements references and DOORS:

- Select an imported requirements reference and click **Show in document** to navigate to DOORS.
- Select **MATLAB > Select Item** in DOORS to navigate to the imported requirements reference.

If your DOORS module has links between DOORS items, you may use additional commands to bring links into the requirements set. Also, if your DOORS module has links to Simulink models, use link synchronization to bring the links into the requirements set. See the section **Copying Link Information from DOORS to Simulink** in “Managing Requirements for Fault-Tolerant Fuel Control System (IBM Rational DOORS)” on page 8-39.

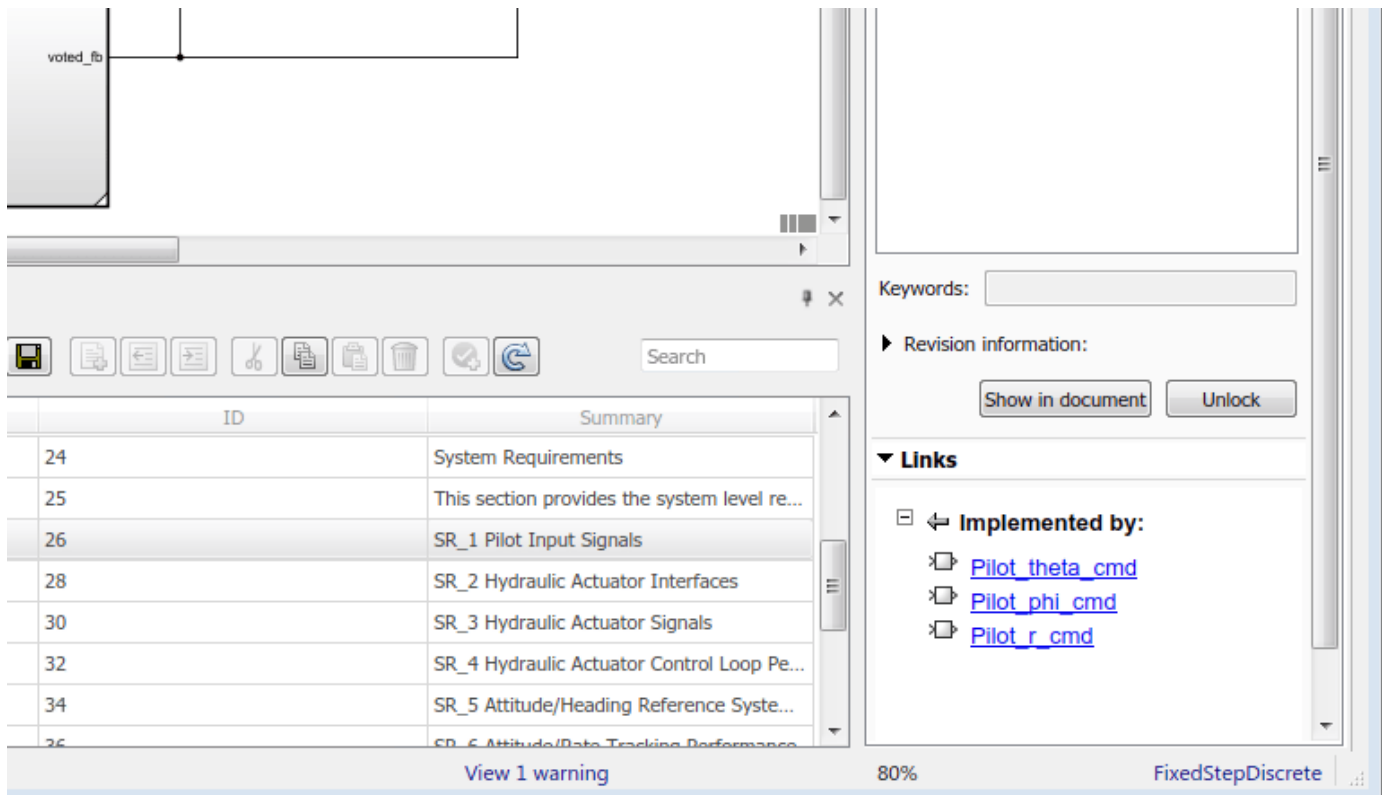
Before you import your DOORS module, be sure that you've added all desired requirements attributes. You cannot import additional attributes to Requirements Toolbox after the original import.

Link to Your Model

You can link imported requirements to Simulink blocks by dragging items from the Requirements Browser to items in your model. Open the Requirements Perspective in the model window by clicking the icon at the lower right of the window and selecting the **Requirements** tile.

When you open the Requirements Perspective and select a requirement, links are displayed in the Property Inspector under **Links**. You can:

- Navigate to linked artifacts outside the current model.
- Delete links by pointing to the link and clicking the red cross.
- Check and modify link properties by selecting Links from the **View** drop-down.



You can link imported requirements to entities such as test cases, MATLAB code, data dictionaries, and other requirements. For more information, see “Link Test Cases to Requirements” and “Working with IBM Rational DOORS 9 Requirements” on page 8-30.

Update Requirements to Reflect DOORS Changes

If the source requirements in DOORS change, you can update the imported references in Requirements Toolbox.

- Select the top-level node that corresponds to updated DOORS module.
- Click the **Update** button.

Follow the steps in “Update Imported Requirements” on page 1-89.

If you have added attributes to your DOORS module since the original import to Requirements Toolbox, the new attributes are not imported. If you want to import attributes from your DOORS module, be sure to add them before importing to a new requirement set in Requirements Toolbox.

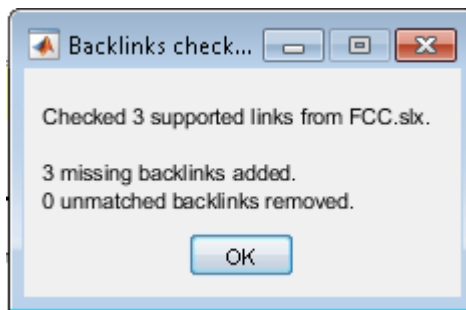
Synchronizing Links and Navigation from DOORS

You can bring traceability data into DOORS for easier navigation from original requirements to design and tests. To synchronize your Requirements Toolbox links into DOORS:

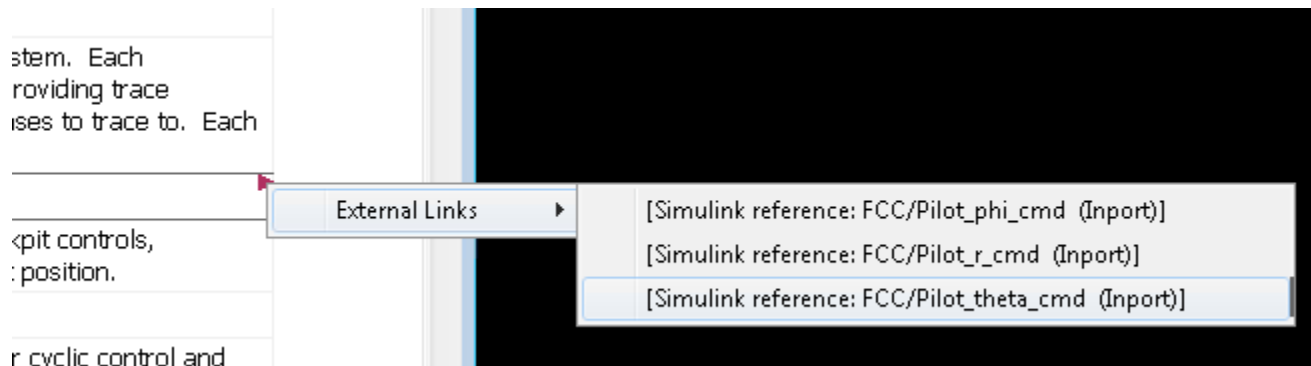
- Select **Links** from the **View** drop-down.
- Locate and right-click the Link Set that has new links.
- Select **Update Backlinks** shortcut at the bottom of context menu.

Requirements Toolbox analyzes outgoing links in the Link Set and checks for incoming links from applications that support **backlinks** insertion, including DOORS.

- Missing links are added to the external document. In DOORS, links appear as outgoing **External Links** and correspond to Simulink entities, such as a blocks or test cases in Simulink Test™.
- Linked documents are checked for stale links, where there is no matching link from Simulink to this external requirement.
- You can delete unmatched links from the DOORS module by confirming the prompt.
- A short report dialog is displayed on successful completion of **Update Backlinks** action:



After performing **Update Backlinks** step, review your linked requirements in DOORS module - you should see links to MATLAB or Simulink. You may see multiple links if same requirement is linked to multiple elements. Click the link in DOORS to navigate:

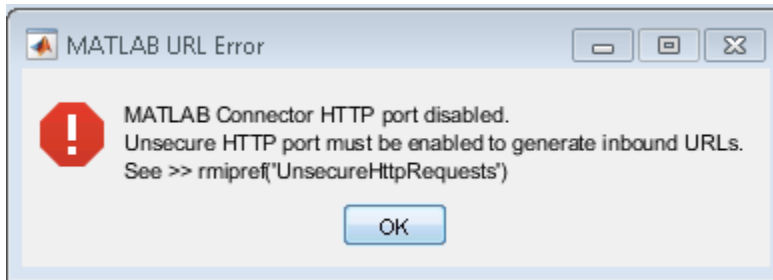


See “Manage Navigation Backlinks in External Requirements Documents” on page 3-51 for general information about managing links from external documents.

Embedded HTTP Connector

Navigation from external applications to MATLAB/Simulink relies on the built-in HTTP server in MATLAB. Requirements Toolbox will fail to insert a link in external application unless the MATLAB's built-in HTTP server is active on the correct port number.

If you see the following error popup when performing **Update Backlinks** action, this indicates that HTTP server is not in the correct state:



Use the `connector.port` command-line API to check the status of HTTP server, and use `rmi('httpLink')` API to activate the server if `connector.port` command returns 0.

```
>> rmi('httpLink')
>> connector.port

ans =

    31415

fx >>
```

Update Backlinks feature requires that HTTP server is activated for port 31415. If `connector.port` command returns a higher number, this indicates that port number 31415 was taken by some other process when this instance of MATLAB was started. You will need to:

- Save your work and quit all instances of MATLAB.
- Restart only one instance of MATLAB.
- Check HTTP server status by running `connector.port` command.
- If you get 0, rerun `rmi('httpLink')` command.
- Re-check using `connector.port` command - you should now see 31415 port activated.
- Re-open your MBD artifacts and retry **Update Backlinks** procedure.

Tracing to DOORS Module Baseline

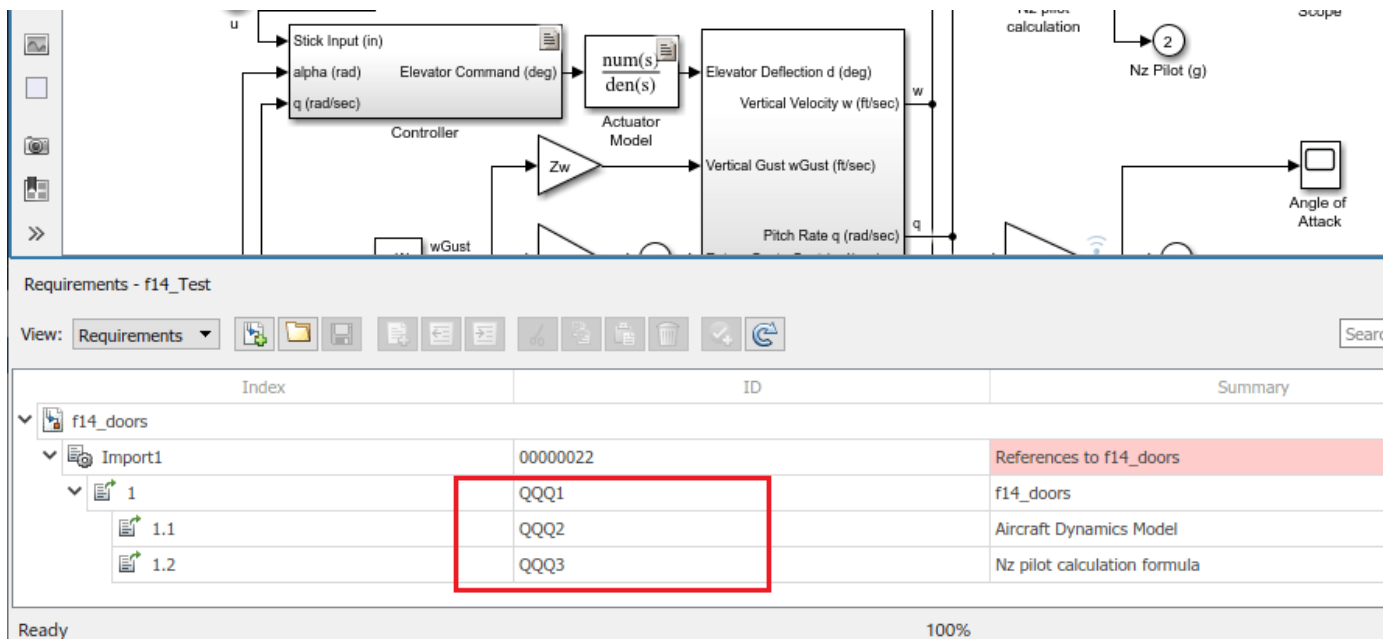
At some point after linking MBD artifacts with requirements in DOORS, you may have created Baselines for linked modules. By default, your links stored in Requirements Toolbox will still navigate to the current version of the linked modules. If you want to lock your design version to a baseline version of requirements, Requirements Toolbox allows you to specify a Baseline number for each DOORS module you are linking with. You can choose to configure the preferred DOORS baseline numbers for all linked artifacts in your current MATLAB session, or you can specify a different DOORS baseline number, for specified MBD artifacts.

- `slreq.cmConfigureVersion` is the command-line API that you use to specify your preferred DOORS baseline numbers.
- Use `slreq.cmGetVersion` command to check the configured DOORS baseline number for a given DOORS module.

- If you later created next version baselines for linked modules, and if you want navigation of previously stored links to target the later baseline, you rerun `slreq.cmConfigureVersion` command to specify the updated baseline number.
- Per-artifact values are stored with the corresponding Link Sets and will affect navigation for all users of same Link Set files.
- Global (session-scope) assignments are stored in user preferences. Your next MATLAB session on the same installation remembers your previously configured baseline numbers. If you shared your work with other users, each user will need to re-enter the same preferred baseline numbers. If needed, you can include the required configuration commands in your MATLAB startup script or in your Simulink Project startup script.

Repair Links to Previously Imported References After Module Prefix Changed in DOORS

When requirements change in DOORS, you perform the **Update** action to bring updated DOORS contents into previously imported Requirements Set. The process relies on matching DOORS object IDs with Custom IDs of previously imported items to determine which existing references need update, and which DOORS objects are new and require creation of new references in the Requirements Toolbox™ requirement set. Also, when updates received from DOORS do not include some Custom IDs that are present in the requirement set, the corresponding items are assumed to be deleted in DOORS, and will be cleaned-up from the requirement set. With this comes the following danger: if DOORS user has modified the module prefix in DOORS before performing the **Update** for the requirement set, none of the existing Custom IDs will match, because DOORS module prefix is a part of ID, and all IDs known on Requirements Toolbox side are based on the old prefix. **Update** process will remove all existing references and will then create new ones with Custom IDs that correspond to updated prefix in DOORS. If previously imported references were linked with design artifacts on Simulink side, all the links will be broken, because the originally linked references no longer exist. For example, if the original module prefix in DOORS was "KKK" and this was changed to "QQQ", you will see QQQ-based IDs in the Requirements Browser after performing **Update**,



Index	ID	Summary
<ul style="list-style-type: none"> f14_doors <ul style="list-style-type: none"> Import1 <ul style="list-style-type: none"> 1 <ul style="list-style-type: none"> 1.1 1.2 1.3 	0000022	References to f14_doors
	QQQ1	f14_doors
	QQQ2	Aircraft Dynamics Model
	QQQ3	Nz pilot calculation formula

... but the links will still point to KKK-based items as destinations. You will see orange warning triangles on all the links that got broken:

Requirement links - f14_Test

View: Links

Label	Source	Type	Destination
f14_Test.slmx	Changed source: 0/4		Changed destination: 2/4
00000022: References to f14_doors (f...	Actuator Model	Implements	00000022 References to f14_doors
KKK2: Aircraft Dynamics Model (f14_...)	Controller	Implements	f14_doors.slreqx:15
KKK3: Nz pilot calculation formula (f1...)	Controller	Implements	f14_doors.slreqx:16
00000022: References to f14_doors (f...)	Controller	Implements	00000022 References to f14_doors

Ready 100%

You can repair broken links by performing the following steps:

- 1 identify the original DOORS IDs in LinkSet data,
- 2 construct the expected updated DOORS IDs based on your knowledge of the original and current module prefix,
- 3 rely on reconstructed IDs to locate the matching Requirement Set entry for each broken link destination,
- 4 update each broken link to connect with the updated reference in Requirement set.

If an older copy of Requirement Set file is still available, you can collect the SID->CustomID mapping from it. But if you only have the updated version of the Requirement Set, and the links are already broken, you may be able to pull old DOORS IDs from the stored link labels (from `link.Description` values).

The following script demonstrates accomplishing this task for the case when all stored `link.Description` labels start with the DOORS ID. In our example the labels look like "KKK123: DOORS Object Text or Heading", and we assume that DOORS item with old ID "KKK123" now has DOORS ID "QQQ123".

```

1  function resolveLinksByLabelMatch(artifactName, reqSetName, oldPrefix, newPrefix)
2
3  -   countChecked = 0;
4  -   countMatched = 0;
5  -   countUpdated = 0;
6
7  -   % Make sure ReqSet and LinkSet are loaded
8  -   slreq.load(reqSetName); slreq.load(artifactName);
9
10 -   % Iterate all links and fix the ones with old IDs
11 -   % by reconnecting to updated requirement with matching new ID
12 -   linkSet = slreq.find('type', 'LinkSet', 'Name', artifactName);
13 -   reqSet = slreq.find('type', 'ReqSet', 'Name', reqSetName);
14 -   sources = linkSet.sources();
15 -   for i = 1:numel(sources)
16 -       links = slreq.outLinks(sources(i));
17 -       for j = 1:numel(links)
18 -           link = links(j);
19 -           countChecked = countChecked + 1;
20 -           if startsWith(link.Description, oldPrefix)
21 -               destInfo = link.getReferenceInfo();
22 -               if contains(destInfo.artifact, reqSetName)
23 -                   countMatched = countMatched + 1;
24 -                   oldDoorsId = strtok(link.Description, ':');
25 -                   newDoorsId = strrep(oldDoorsId, oldPrefix, newPrefix);
26 -                   req = reqSet.find('CustomId', newDoorsId);
27 -                   if ~isempty(req)
28 -                       link.setDestination(req);
29 -                       link.Description = strrep(link.Description, oldDoorsId, newDoorsId);
30 -                       countUpdated = countUpdated + 1;
31 -                   end
32 -               end
33 -           end
34 -       end
35 -   end
36
37 -   disp([num2str(countChecked) ' links checked, ' ...
38 -        num2str(countMatched) ' matched, ' ...
39 -        num2str(countUpdated) ' updated']);
40
41 -   if countUpdated > 0
42 -       disp('please review the changes and save the LinkSet');
43 -   end
44 - end

```

Run this script with four input arguments: LinkSet name, ReqSet name, old prefix, new prefix:

```
>> resolveLinksByLabelMatch('f14_Test', 'f14_doors', 'KKK', 'QQQ')
4 links checked, 2 matched, 2 updated
please review the changes and save the LinkSet.
fx >> |
```

Now all the links are resolved and labels are updated correctly:

The screenshot displays the IBM Rational DOORS interface. At the top, a model diagram shows a 'Controller' block with inputs 'Stick Input (in)', 'alpha (rad)', and 'q (rad/sec)'. It outputs 'Elevator Command (deg)'. This is connected to a 'Model' block, which outputs 'Vertical Gust wGust (ft/sec)' and 'Pitch Rate q (rad/sec)'. The 'Model' block is further connected to 'Nz pilot calculation' and 'Angle of Attack' blocks. A 'Requirement links - f14_Test' window is open at the bottom, showing a table of resolved links.

Label	Source	Type	Destination
f14_Test.slmx*	Changed source: 0/4		Changed destination: 4/4
00000022: References to f14_doors (f...	Actuator Model	Implements	00000022 References to f14_doors
QQQ2: Aircraft Dynamics Model (f14_...	Controller	Implements	QQQ2 Aircraft Dynamics Model
QQQ3: Nz pilot calculation formula (f1...	Controller	Implements	QQQ3 Nz pilot calculation formula
00000022: References to f14_doors (f...	Controller	Implements	00000022 References to f14_doors

See Also

Related Examples

- “Managing Requirements for Fault-Tolerant Fuel Control System (IBM Rational DOORS)” on page 8-39
- “Import Requirements from IBM Rational DOORS by Using the API” on page 1-119

More About

- “Configure Requirements Toolbox for IBM Rational DOORS Software” on page 8-2
- “Import Requirements from IBM Rational DOORS” on page 1-42

Managing Requirements for Fault-Tolerant Fuel Control System (IBM Rational DOORS)

The Requirements Management Interface (RMI) provides tools for creating and reviewing links between Simulink® objects and requirements documents. This example illustrates linking model objects to requirements stored in IBM® Rational® DOORS®. For more information using the RMI, see “Managing Requirements for Fault-Tolerant Fuel Control System (Microsoft Office)” on page 7-15.

Setup RMI for DOORS

Make sure your DOORS installation is configured for communication with RMI. Run MATLAB® as Administrator and execute `rmi('setup')`. If DOORS Client installation is detected, RMI will prompt to install the required API files. You only have to do this once after reinstalling either DOORS or MATLAB. For more information, see “Configure Requirements Toolbox for Interaction with Microsoft Office and IBM DOORS” on page 6-2.

Simulink Model and DOORS Modules Used in this Example

For the purposes of this example, an example model of a fault-tolerant fuel control system called `slvndemo_fuelsys_doorsreq.slx` is included. Use it for the exercises presented below.

Open the Simulink model manually or by evaluating the following code.

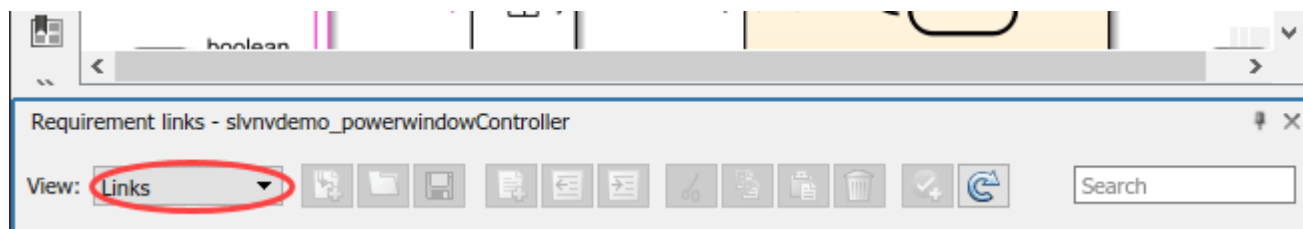
```
open_system('slvndemo_fuelsys_doorsreq');
```

You can use any temporary DOORS module for basic link creation exercises below, and you can use the included `DemoRMI.dpa` archive for a more advanced exercise of Surrogate Module Synchronization.

Set Up Requirements Manager to Work with Links

- 1 In the **Apps** tab, open **Requirements Manager**.
- 2 In the **Requirements** tab, ensure **Layout > Requirements Browser** is selected.
- 3 In the **Requirements Browser**, in the **View** drop-down menu, select **Links**.

In this example, you will work exclusively in the **Requirements** tab and any references to toolbar buttons are in this tab.



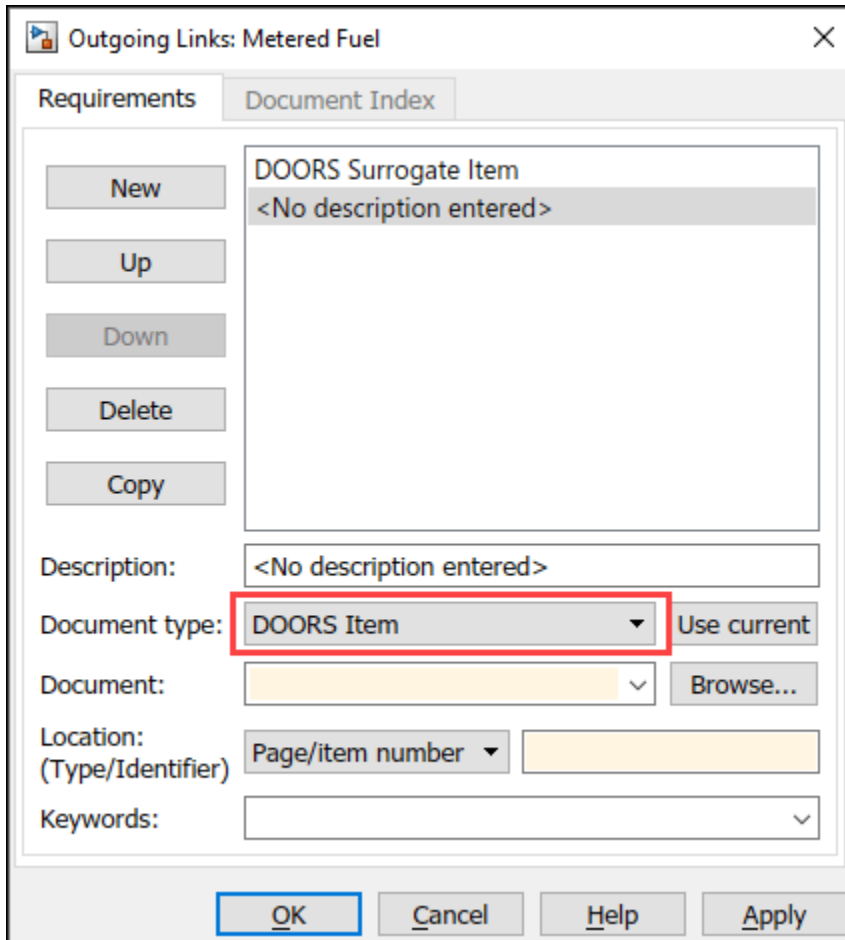
Linking via Outgoing Link Dialog

You can link a model object to requirements stored in a DOORS database (DOORS objects). You do not need to modify DOORS documents when creating links. The most hands-on way to create new links is via Outgoing Link dialog. This requires manually filling-in link attribute fields. See next subsection for an easier automated way.

- Navigate to the Metered Fuel Scope block.

```
rmdemo_callback('locate', 'slvndemo_fuelsys_doorsreq/Metered Fuel');
```

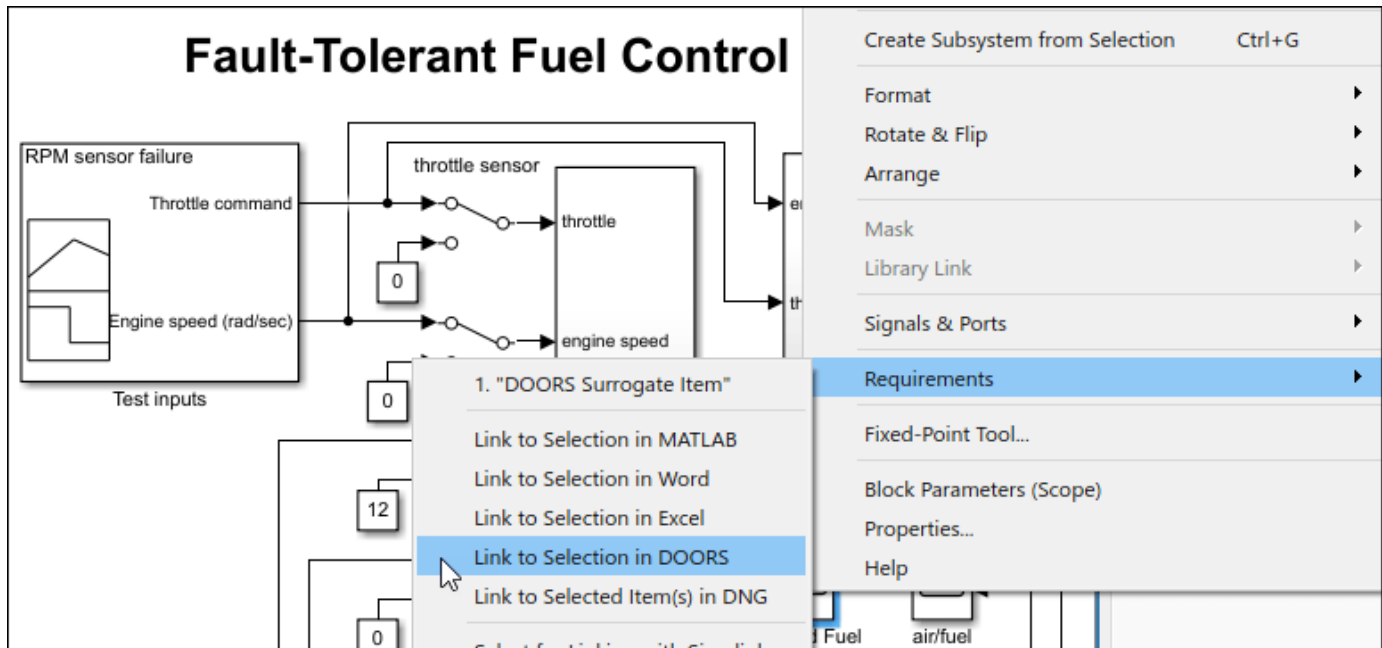
- Right-click the block and select **Requirements > Open Outgoing Links dialog...** from the context menu. The Outgoing Link dialog opens.
- Click **New** to create a new requirement.
- Set **Document type** to IBM DOORS.



- Specify a unique target module ID in the **Document** input field or use the **Browse** button to select the target module in DOORS database.
- Enter target object ID in the **Location Identifier** field, or use the **Document Index** tab to select the target object in a chosen module.
- Click **Apply** or **OK** to store the new requirement link.
- Right-click the same Simulink block again to see the new link label listed in the top portion of the context menu.

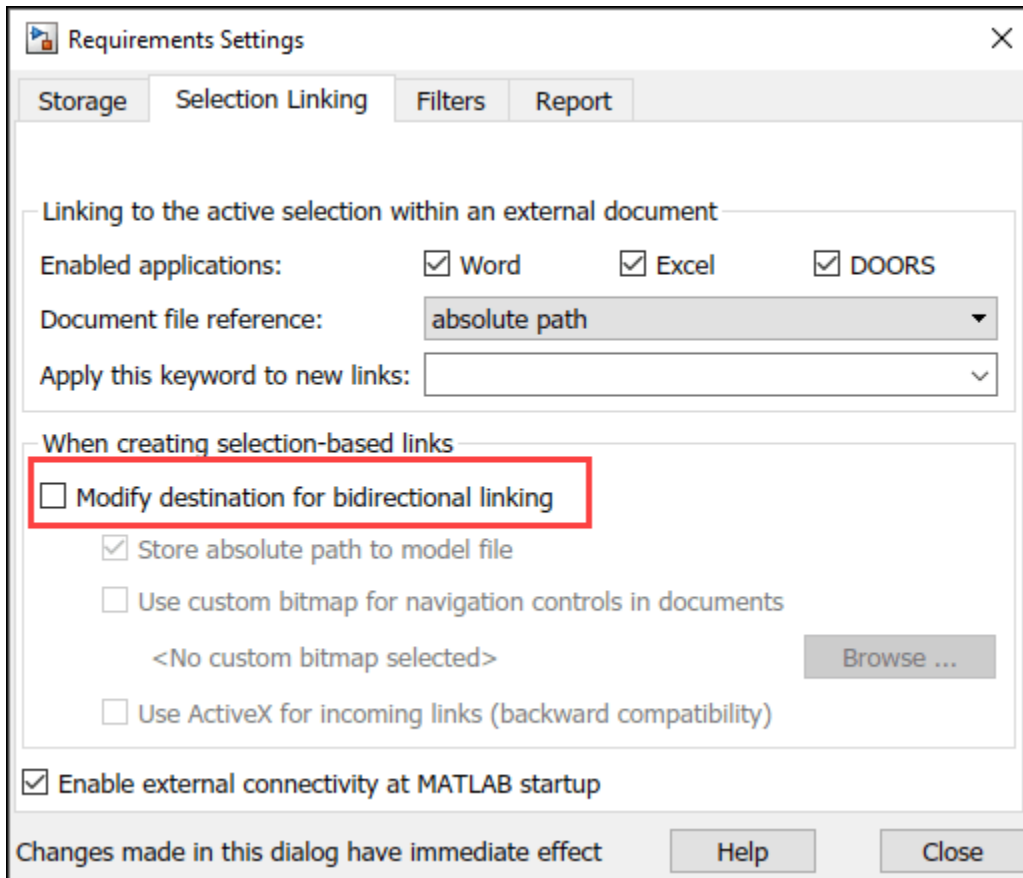
Linking via Context Menu Shortcuts

An easier way to establish new links is via Selection Linking Shortcuts. Right-click a block and select **Requirements > Link to Selection in DOORS**.



Links creation via context menu shortcuts do not require any manual input. Link target destination is determined by the current selection in DOORS, and the **Description** field is set to the corresponding Object Heading or to DOORS Object Text when there is no Heading. Because the **Description** is used for navigation shortcuts in context menus, number of characters limit applies.

RMI supports bidirectional linking with requirements in DOORS, but you will start with one-directional links. Disable bi-directional linking in the **Requirements** tab of the model by clicking **Link Settings > Linking Options** and unchecking **Modify destination for bidirectional linking** under the **When creating selection-based links**.



Alternatively, you can evaluate the following code.

```
rmipref('BiDirectionalLinking', false);
```

We will cover bidirectional linking later. Now try this out:

- Select any object in your test module in DOORS.
- Navigate to the `throttle sensor` block.

```
rmidemo_callback('locate', 'slvndemo_fuelsys_doorsreq/throttle sensor');
```

- Right click the block and select **Requirements > Link to Selection in DOORS** in the context menu to create a link.
- Right-click the `throttle sensor` block again and locate the link label at the top of **Requirements** context menu to confirm that the link was added. You may use Outgoing Link dialog later to adjust the description label or User Tag keywords.

Current Selection Linking via Outgoing Link Dialog

The **Use current** button in the Outgoing Link dialog box provides a combined approach:

- Right-click a block in the model and select **Requirements > Open Outgoing Links dialog...** from the context menu.
- Push the **New** button to add another link item.

- Select **DOORS Item** in the **Document type** drop-down box.
- In DOORS module window, click on the object that you want to link.
- Click the **Use current** button to automatically fill in all the input fields with the data from the current selected DOORS object.
- Adjust the **Description** as required.
- Save the changes by clicking **OK** or **Apply**.

You can also use the **Use current** button to redirect an existing link:

- Select the required new target object in DOORS.
- In Outgoing Link dialog, click on the list item you want to update.
- Click the **Use current** button to update link attributes.

Viewing and Navigating Links from Simulink to DOORS

You highlight and navigate DOORS links in the same way you do that with other types of links.

- In the **Requirements** tab, click **Highlight Links** to highlight all requirements in the example model. You can also evaluate the following to highlight the links.

```
rmi('highlightModel', 'slvndemo_fuelsys_doorsreq');
```

- Make sure DOORS is running and logged in.
- Right-click on one of the highlighted objects that you used to create new links in the previous section.
- Select **Requirements** from the context menu. The labels of the links you created should be visible at the top.
- Click on the link label. Your test module opens in DOORS with the correct object selected.

Be careful to only try this with the links you created. There are other links in the model that will not work just yet. We will cover fixing those links in sections below.

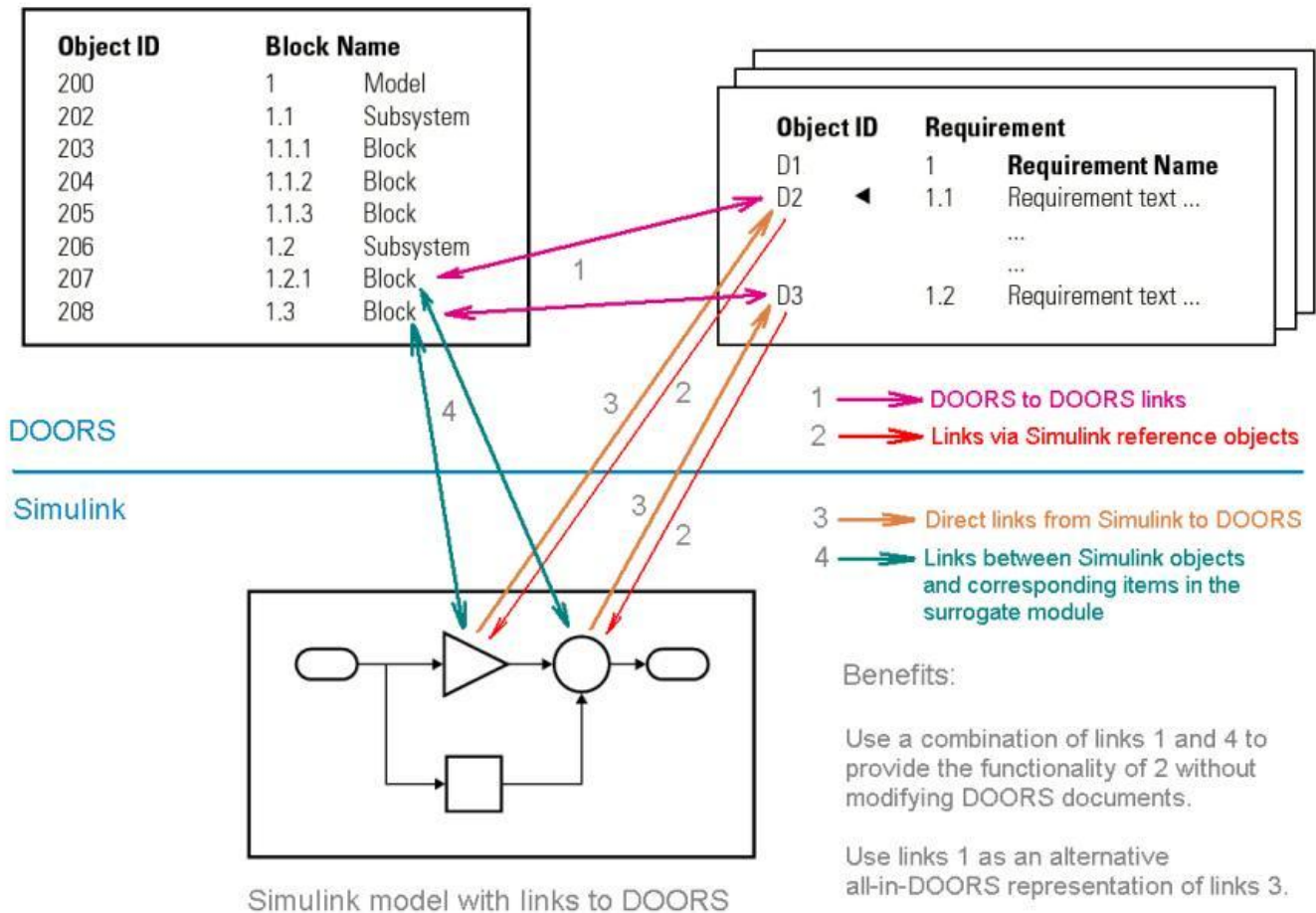
About Surrogate Modules and Synchronization

Surrogate module workflow is supported for DOORS to allow two-way linking without needing to modify DOORS requirements modules. The following picture illustrates the workflow.

Working with surrogate modules

Surrogate module in DOORS

Requirements documents in DOORS



A new formal DOORS module, referred to as a **surrogate module**, is automatically generated by Simulink to be used as a DOORS representation of the Simulink model. You can choose to map all the objects in your model, or only those with links to DOORS, or pick one of the intermediate options as discussed in the documentation.

You can create direct links to requirements in DOORS, as demonstrated in previous sections (marked 3 in the picture) and optional matching direct links from DOORS documents to Simulink objects, as demonstrated in the last section of this example (marked 2 in the picture).

Additionally, with the surrogate module present in DOORS, you can establish links within DOORS between the items in surrogate modules and requirements stored in DOORS (marked 1 in the picture), while navigation to and from Simulink is provided by surrogate item links (marked 4 in the picture).

Surrogate module workflow provides the following advantages:

- Bidirectional linking is possible without the need to modify documents in DOORS or the models in Simulink. All required information is stored in the surrogate modules and corresponding link modules.
- You can manage and analyze links in the DOORS environment without necessarily running Simulink, including using the native reporting capabilities of DOORS.

Below is an example screenshot of the autogenerated Surrogate module. Note that DOORS hierarchy mirrors the structure of the originating Simulink model, and DOORS object headers match Simulink object names:

The screenshot shows the DOORS interface for a surrogate module. The left pane displays a hierarchical tree of blocks, and the right pane displays a table of blocks. A red circle highlights the '1.12 fuel rate controller' block in the tree, and red arrows point from it to the corresponding row in the table.

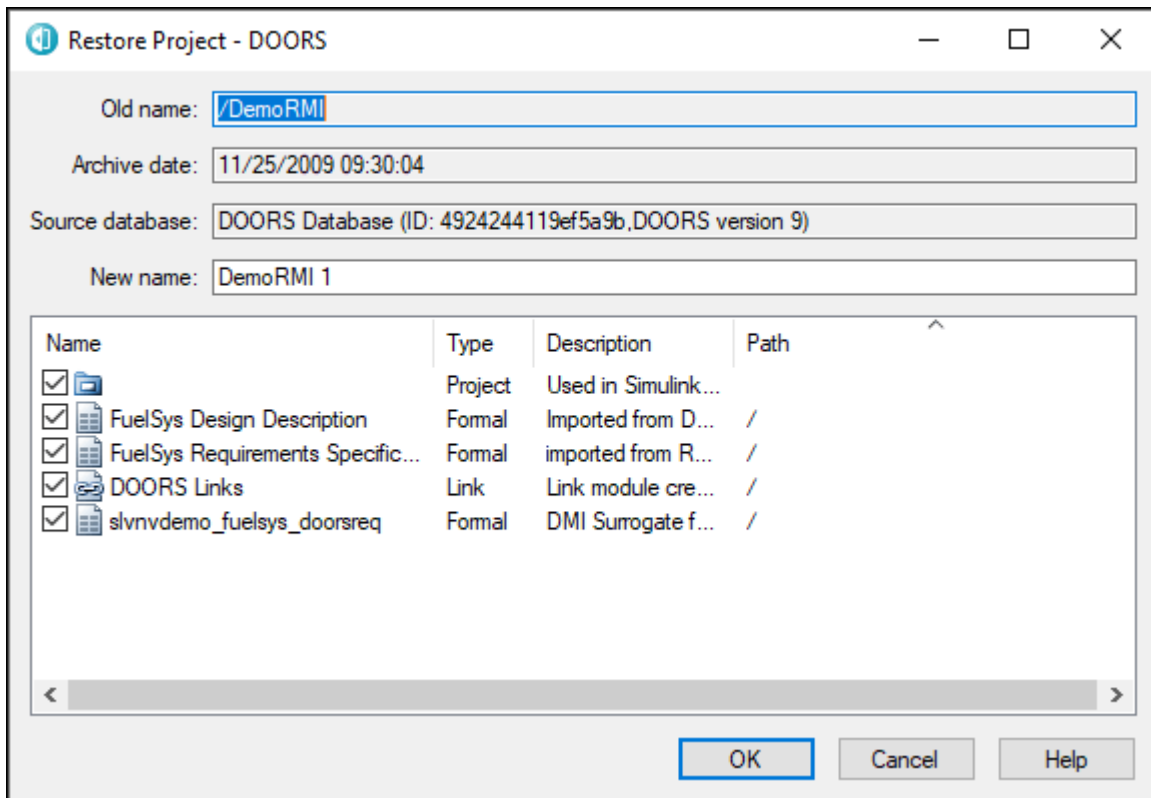
ID	Block Name	Block Type	Block Deleted?
52	1.11.5.6.11 f(theta)	Fcn	False
53	1.11.5.6.12 g(pratio)	Fcn	False
54	1.11.5.6.13 threshold = 0.5	Switch	False
55	1.11.5.6.14 Throttle Flow, mdot (g/s)	Outport	False
56	1.11.5.7 Mass Airflow Rate	Outport	False
57	1.11.5.8 MAP (bar)	Outport	False
58	1.11.6 o2_out	Outport	False
59	1.11.7 MAP	Outport	False
60	1.11.8 air/fuel ratio	Outport	False
61	1.12 fuel rate controller	SubSystem	False
62	1.12.1 throttle	Inport	False
63	1.12.2 engine speed	Inport	False
64	1.12.3 EGO	Inport	False
65	1.12.4 MAP	Inport	False
66	1.12.5 Airflow calculation	SubSystem	False
67	1.12.5.1 sens_in	Inport	False
68	1.12.5.2 Failures	Inport	False
69	1.12.5.3 mode	Inport	False
70	1.12.5.4 Constant	Constant	False

Synchronizing Your Simulink Model with a DOORS Database

Normally, you would navigate to the **Requirements** tab in your Simulink model and use **Share > Synchronize with DOORS** to create a new DOORS surrogate module for your Simulink model.

For the purposes of this example, an existing DOORS project is provided as an archive, including the surrogate module with links to other modules. To try out the interactive features of this example, restore the project into your DOORS database, and then re-synchronize the example model as explained below. Note that this archive was created in DOORS version 9.1 and may not work with earlier versions of DOORS.

- Use the **File > Restore** feature in DOORS and point it to **DemoRMI.dpa** archive provided in the present working directory. If you already have a project named **DemoRMI** in your DOORS database, DOORS appends a number to the project name. As shown in the screenshot below, the project includes one link module and three formal modules. One formal module is the DOORS surrogate for the `slvndemo_fuelsys_doorsreq` model; the other two are example modules produced by importing Microsoft® Word documents from “Managing Requirements for Fault-Tolerant Fuel Control System (Microsoft Office)” on page 7-15.



- Extract all the included modules and open the surrogate module.
- Note the red and orange link navigation triangles in two of the extracted modules. Right-click to navigate between modules. These links are preserved through the backup-restore procedure.

The top screenshot shows the 'FuelSys Requirements Specification' window. The table below is a representation of the content shown:

ID	Requirement Text
27	2.1 Normal Mode of Operation
28	During the normal mode of operation the Fault Tolerant Fuel Control System shall determine the fuel rate which is injected at the valves.
29	2.1.1 Stoichiometric mixture ratio
30	During normal model of operation the System shall maintain the stoichiometric mixture target ratio of 14.6.
31	2.1.2 Oxygen Sensor (EGO)
32	The System shall determine the amount of residual oxygen present in the exhaust gas (EGO) by reading the value of the EGO sensor. During a calibratable warm up period the oxygen sensor correction shall be disabled.

The bottom screenshot shows the 'slvnvdemo_fuelsys_doorsreq' surrogate module window. The table below is a representation of the content shown:

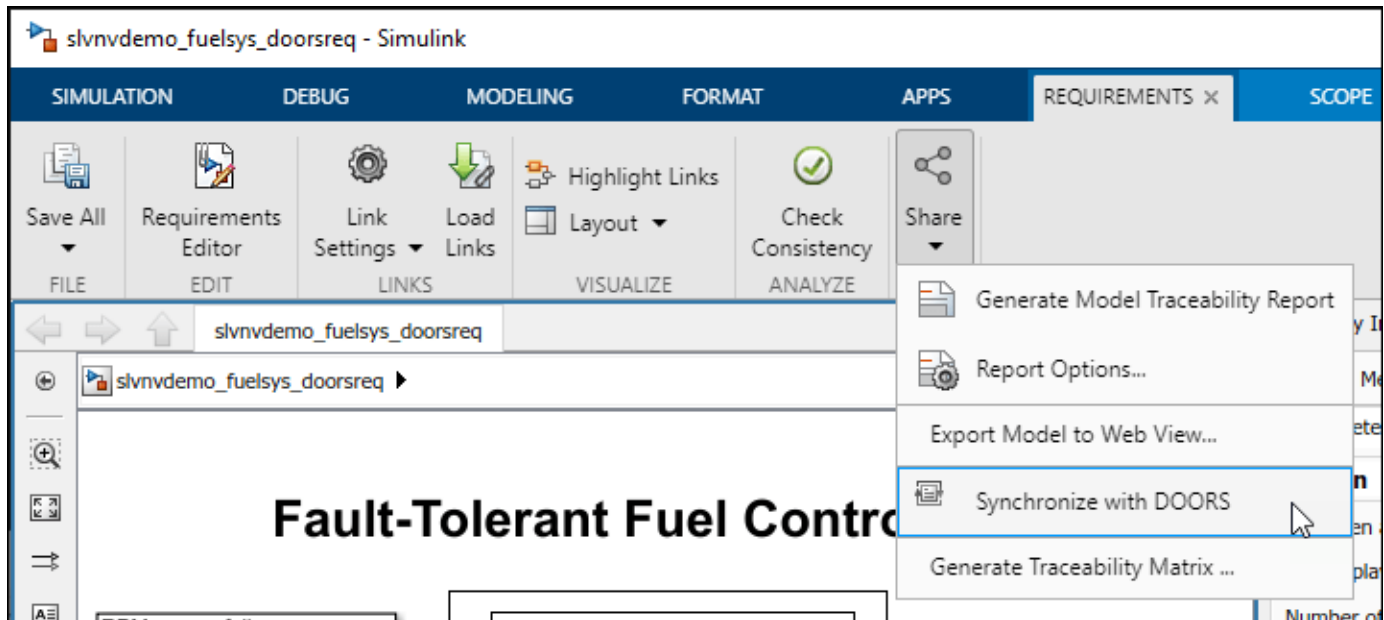
ID	Block Name	Block Type	Block Deleted?
13	1.11.1 engine speed	Import	False
14	1.11.2 throttle angle	Import	False
15	1.11.3 fuel	Import	False
16	1.11.4 Mixing & Combustion	SubSystem	False
17	1.11.4.1 fuel rate	Import	False
18	1.11.4.2 air flow	Import	False
19	1.11.4.3 Constant4	Constant	False
20	1.11.4.4 EGO Sensor	Fcn	False
21	1.11.4.5 MinMax	MinMax	False
22	1.11.4.6 Product	Product	False
23	1.11.4.7 system lag	SubSystem	False
24	1.11.4.8 o2_out	Output	False
25	1.11.4.9 air/fuel ratio	Output	False
26	1.11.5 Throttle & Manifold	SubSystem	False
27	1.11.5.1 Engine Speed, N	Import	False
28	1.11.5.2 Throttle Ang.	Import	False
29	1.11.5.3 Atmospheric Pressure, Pa (bar)	Constant	False
30	1.11.5.4 Intake Manifold	SubSystem	False
31	1.11.5.4.1 mdot Input (a/s)	Import	False

Try navigating from the extracted surrogate module to the corresponding object in Simulink from DOORS:

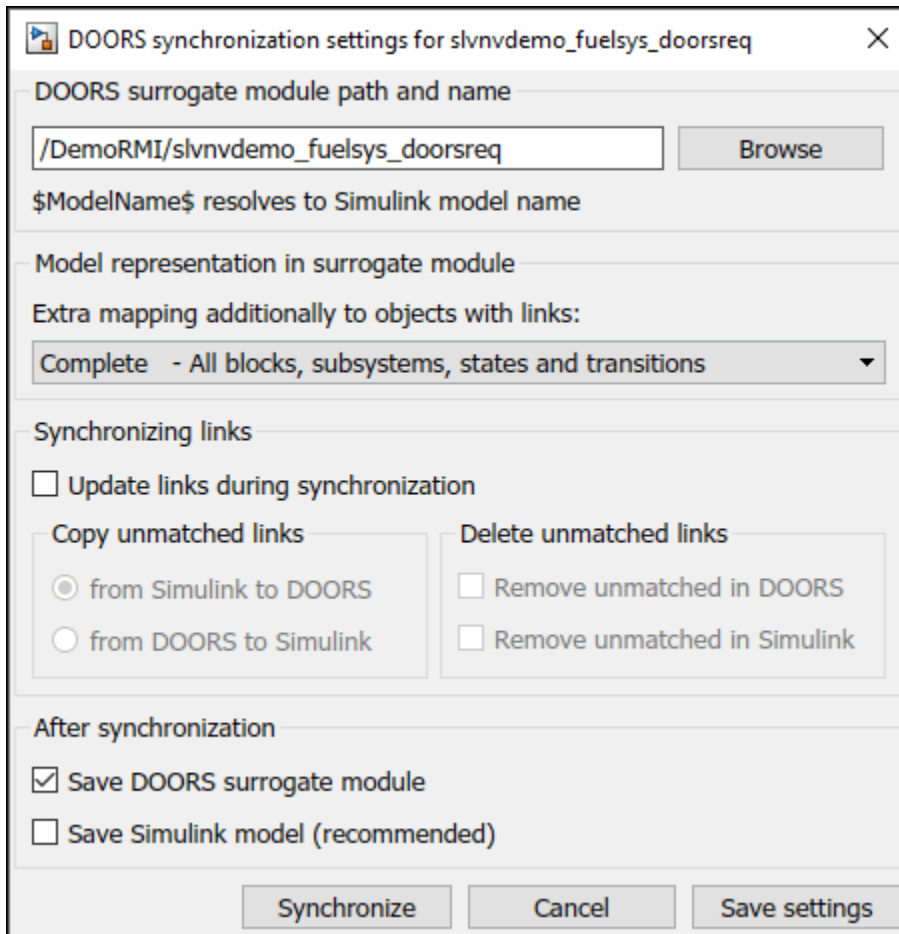
- Click **1.11.4.1 fuel rate** in the **slvnvdemo_fuelsys_doorsreq** surrogate module.
- In main menu of the module window, click **MATLAB > Select Item**. A correct subsystem diagram opens and the corresponding input is highlighted.

Navigation from Simulink objects to the surrogate module is broken, because the extracted modules have new numeric IDs in your DOORS database, trying to navigate **DOORS Surrogate Item** link on any object will produce an error. To repair **DOORS Surrogate Item** links on all objects in the `slvndemo_fuelsys_doorsreq` model after you have successfully restored the **DemoRMI** project, resynchronize the Simulink model with the restored instance of the surrogate.

- In the model window, select the **Requirements** tab and then click **Share > Synchronize with DOORS** to open a the Synchronization Settings dialog box.



- Enter the following settings, using the correct DOORS path for in the **DOORS surrogate module path and name** input field, depending on the location of the restored project, or simply make it a current project in DOORS and use "/" notation: enter `./slvndemo_fuelsys_doorsreq`.



- Do not enable the **Save Simulink model** checkbox at the bottom, you will not be able to save changes to example model unless you use a writable copy.
- Simulink might warn you about the previous synchronization path. Click **Continue** to proceed with the new path. You may get the following message in the command window: "No update needed for the surrogate module". Your restored surrogate module is correct as is.
- Retry navigation from any object in the model to corresponding DOORS object in the surrogate module by right clicking the Simulink block and selecting **Requirements > 1. "DOORS Surrogate Item"** on the context menu. This should now highlight the corresponding DOORS item in the surrogate module.

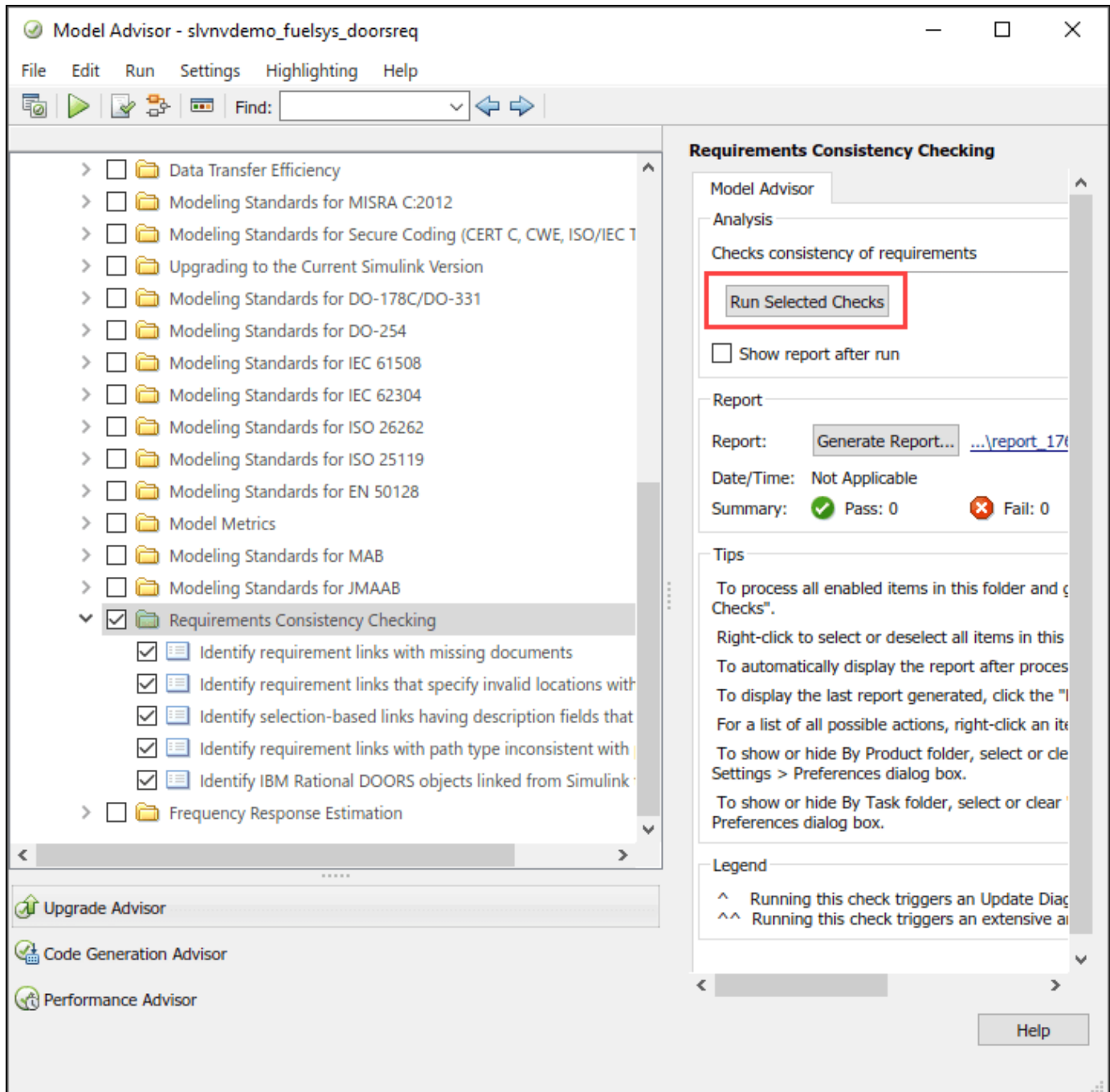
Using Model Advisor for RMI Consistency Checking

The example model comes with some pre-existing links to DOORS document, **FuelSys Design Description** module. Similarly to the original **DOORS Surrogate Item** links, these links are broken, because the restored copy of the module has a new ID in your local database. For example, right-click the **Airflow** calculation subsystem in the model and select "1.2.1 Mass Airflow estimation" from the **Requirements** context menu. This will produce an error message. Evaluate the following code to navigate to the **Airflow** calculation block.

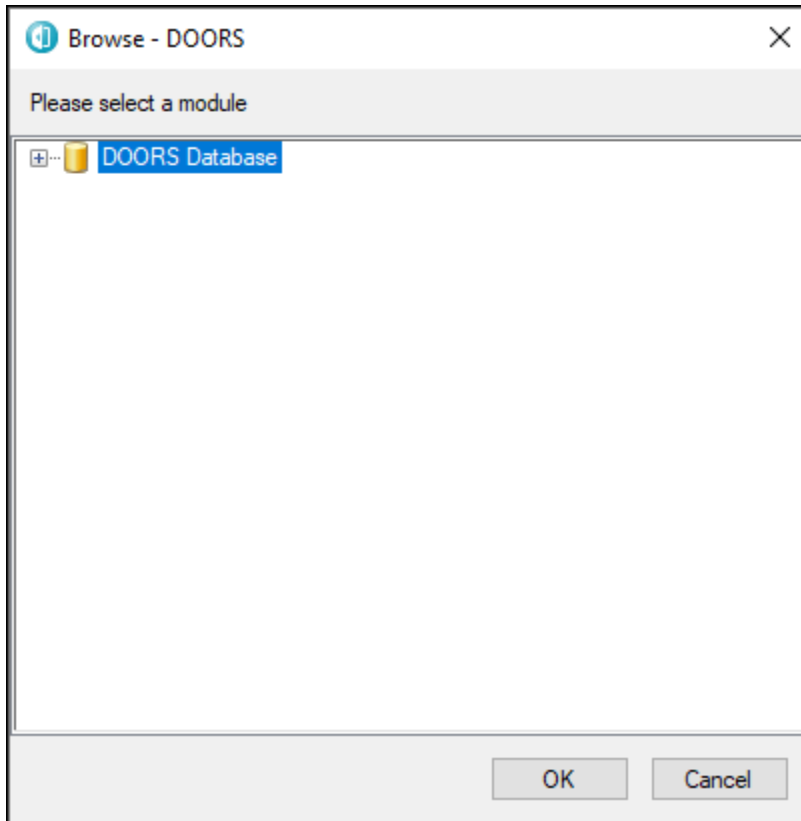
```
rmidemo_callback('locate', ['slvndemo_fuelsys_doorsreq/fuel_rate_controller/' ...
    'Airflow calculation']);
```

We will now fix these links using RMI consistency checking in Model Advisor.

- In the Simulink model window in the **Requirements** tab, click **Check Consistency** to bring up the Model Advisor graphical interface.
- Locate **Identify requirement links with missing documents** item under **Requirements consistency checking** and select it with a mouse.
- Click **Run This Check** button at the top-left of the right-hand panel. Blocks with broken links are listed. You can fix listed inconsistencies one-by-one or, in the **Model Advisor** pane you can use **Fix All** link at the bottom. We will use the **Fix All** shortcut, because we know that all broken links need to be redirected to the same restored copy of the original module.



- In the **Model Advisor** pane, click **Fix All** link at the bottom - DOORS database browser comes up.

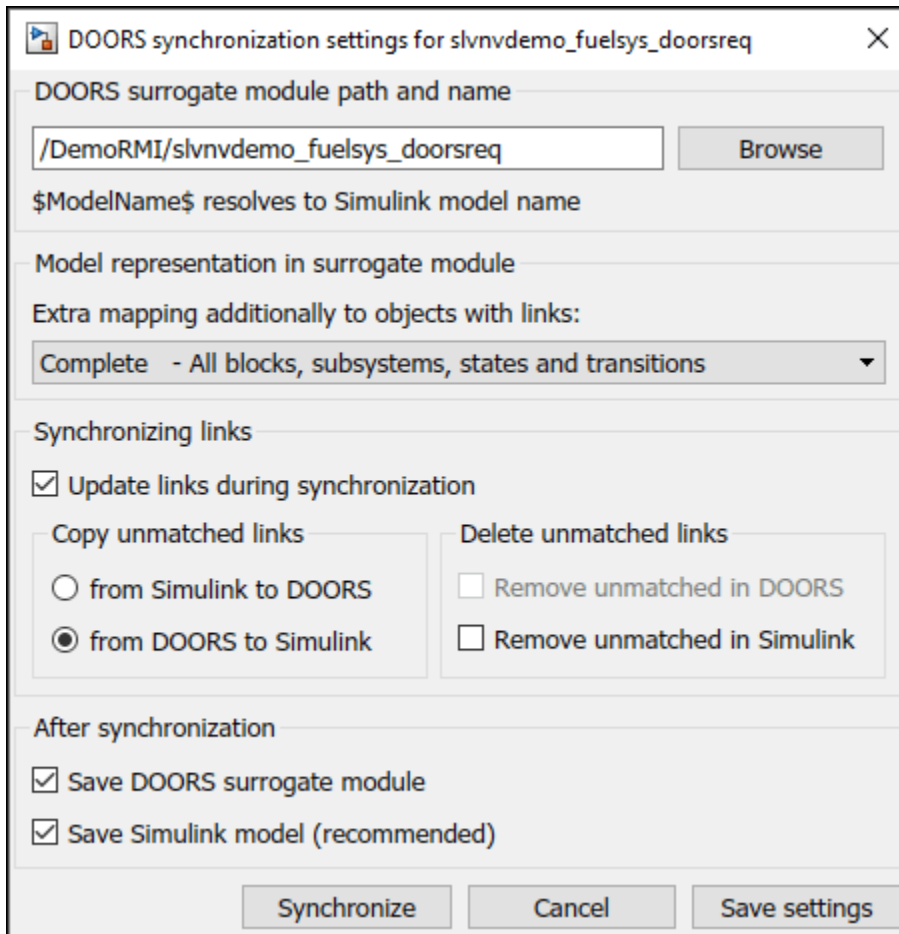


- Locate the restored **FuelSys Design Description** module in your database and select it with a mouse.
- Click **OK** to close DOORS database browser.
- Click **Run This Check** again. The check should now pass.
- Re-try navigation: right-click the **Airflow** calculation subsystem in the model and select **Requirements > "1.2.1 Mass Airflow estimation"** from the context menu. This will now highlight the correct object in one of the DOORS modules you restored from the included archive.

Copying Link Information from Simulink to DOORS

Now that your direct links from Simulink to DOORS are correct, you can use synchronization to copy link information into the DOORS database. Links will be duplicated in the DOORS project, where you can use native DOORS navigation, analysis and reporting tools. These links between the surrogate and other DOORS modules can even be reused with a new copy of the model.

- Re-open **Share > Synchronize with DOORS** dialog and configure the following settings. Make sure to disable the **Remove unmatched in DOORS** checkbox, because there are unmatched links in the restored project that you need later.



- Click **Synchronize** button at the bottom.
- Give it a couple of seconds and check the surrogate module in DOORS. It should now display more links - some that existed in the original restored project (links to the **FuelSys Requirements Specification** module), and some that were just copied from Simulink (links to the **FuelSys Design Description** module).
- Locate the **Airflow calculation** subsystem.

```
rmidemo_callback('locate', ['slvndemo_fuelsys_doorsreq/fuel rate controller/' ...
    'Airflow calculation']);
```

- Navigate to the corresponding surrogate object using the **Requirements > 1. "DOORS Surrogate Item"** on the context menu for this block.
- The new red triangle shows an outgoing link for **1.12.5 Airflow calculation** item in DOORS. Right-click to navigate this DOORS link - this brings you to item **1.2.1 Mass airflow estimation** in the **FuelSys Design Description** module.

Copying Link Information from DOORS to Simulink

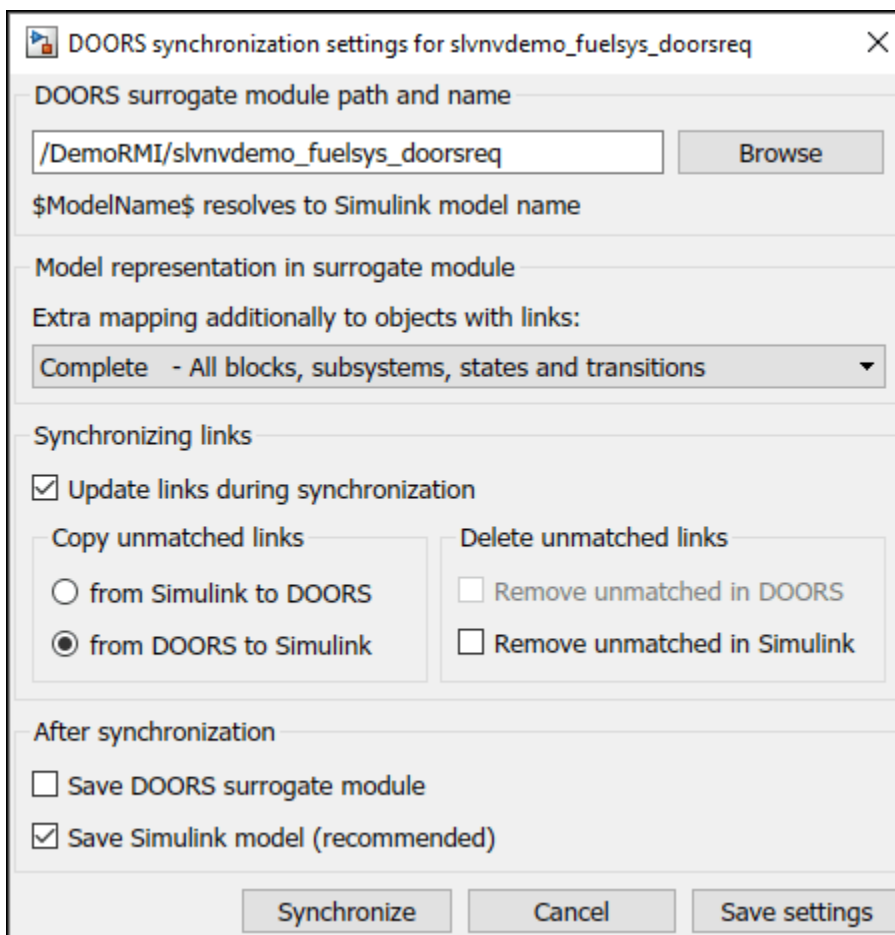
Synchronization via surrogate module provides a convenient way to propagate system requirements updates in DOORS to corresponding Simulink implementation elements. To demonstrate this workflow, the restored project contains DOORS links from the surrogate module to the **FuelSys Requirements Specifications** DOORS module that are not present in the Simulink model. In

DOORS, navigate back to the module that you restored in the section "Synchronizing Your Simulink Model with a DOORS Database".

- Starting in the **FuelSys Requirements Specification** module, locate **2.1 Normal Mode of Operation**.
- Use the DOORS link to navigate to the "1.11.3 fuel" item in the surrogate module by right-clicking the yellow link in the DOORS module, and clicking through on the context menu to navigate to "1.11.3 fuel" item in the **slvndemo_fuelsys_doorsreq** DOORS module.
- While "1.11.3 fuel" is selected, click **MATLAB > Select Item** in the surrogate module main menu to locate the corresponding source object in Simulink model.
- Right-click the located `fuel` input element in Simulink and check **Requirements** in the context menu. 1. "**DOORS Surrogate Item**" is the only available link: there are no links to documents.

To copy link information from DOORS to Simulink, re-synchronize with **Update links during synchronization** enabled, and select **from DOORS to Simulink**.

- Re-open the **Share > Synchronize with DOORS** dialog.
- Configure the following synchronization options:



It is now OK to enable **Remove unmatched in Simulink** checkbox. After the previous synchronization step, there are no unmatched links in Simulink.

Keep some diagrams open and highlighted to visualize changes when new links are added in Simulink.

- Click **Synchronize**. The surrogate module window may come up to the front, but there are no red markers, because there are no changes in DOORS.
- Navigate back to the fuel input in Simulink, or evaluate the following.

```
rmidemo_callback('locate','slvnvdemo_fuelsys_doorsreq/engine gas dynamics/fuel');
```

- Right-click and expand the **Requirements Traceability** section of the context menu. Notice the new link below the **DOORS Surrogate Item** link: "->2.1 Normal Mode of Operation". The arrow prefix indicates that this requirement was not created in Simulink but copied from DOORS.
- Click the new link to navigate to the corresponding requirement in DOORS - **2.1 Normal Mode of Operation** section opens in **FuelSys Requirements Specification** module.

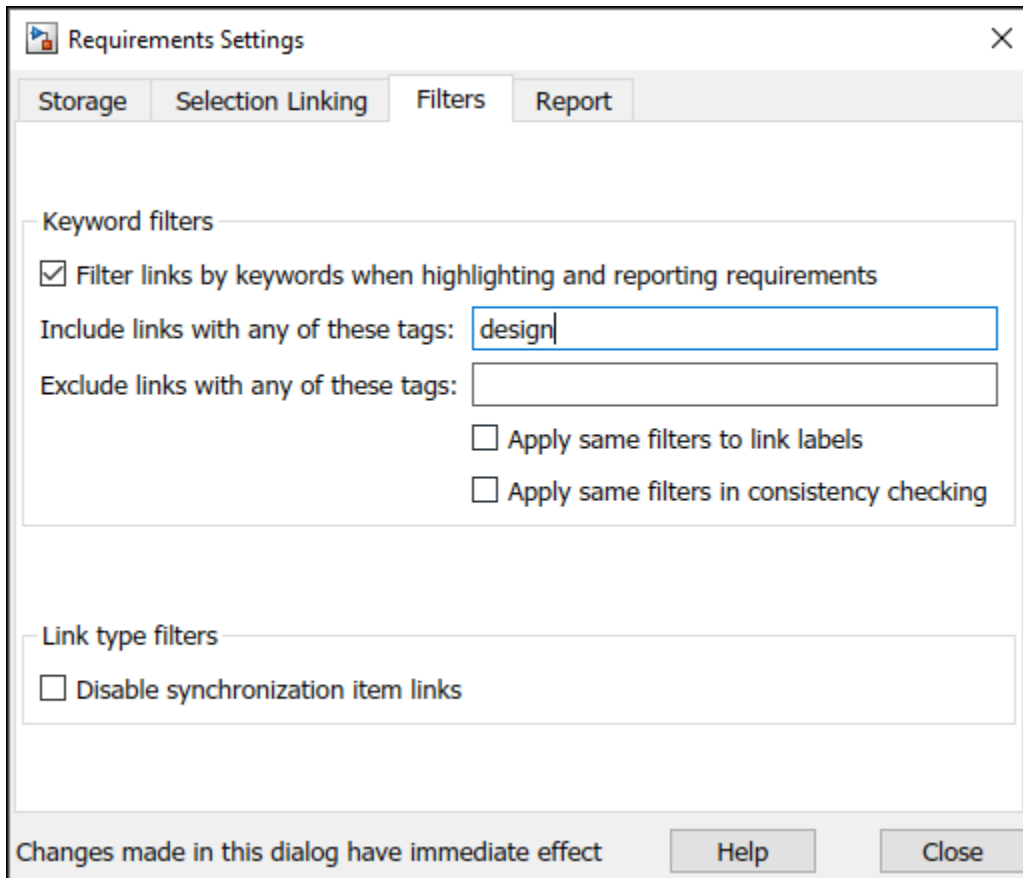
Review Your Changes Using User Tags

You now apply the user tag filter to confirm the changes you made to the model. All DOORS requirements that existed in the original version of the example model were tagged "design". You now use this fact to selectively highlight or hide these links:

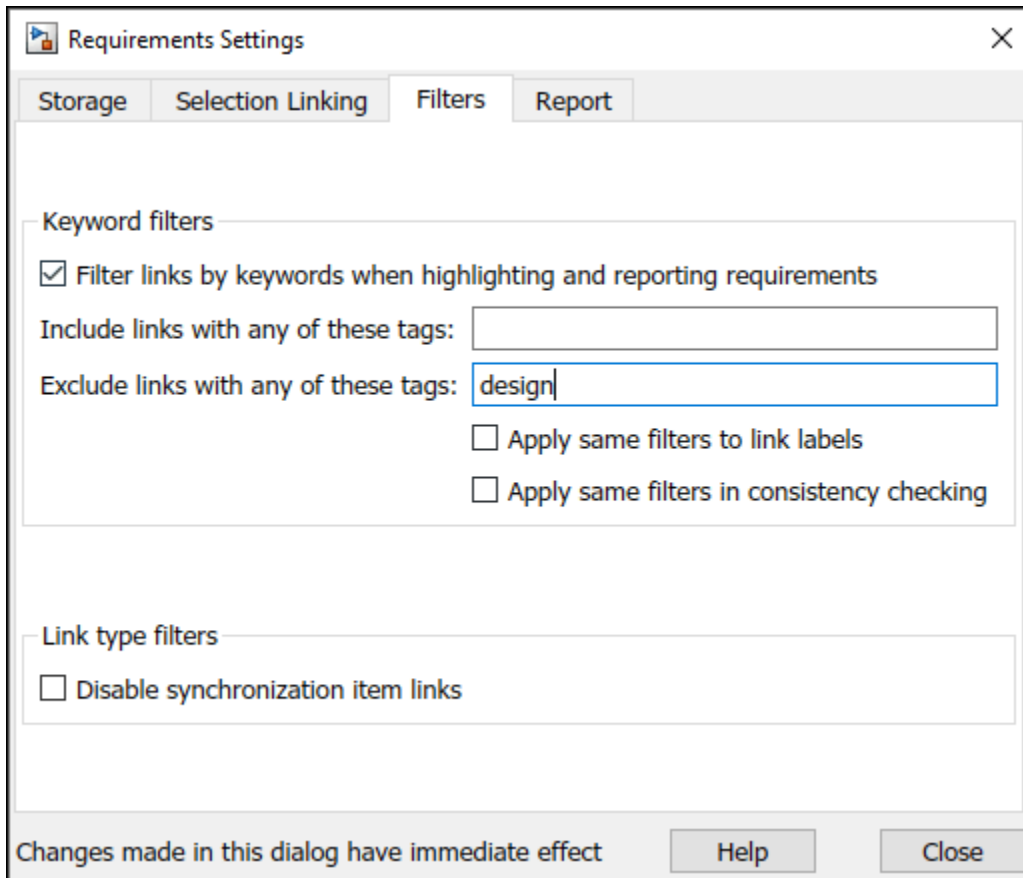
- Navigate back to the fuel rate controller subsystem and in the **Requirements** tab, click **Highlight Links**.

```
rmidemo_callback('open_highlight','slvnvdemo_fuelsys_doorsreq/fuel rate controller');
```

- In the **Requirements** tab, click **Link Settings > Linking Options** to open the **Requirements Settings** dialog.
- Navigate to the **Filters** tab and configure as shown below, checking the **Filter links by keywords when highlighting and reporting requirements** box and entering design in the **Include links with any of these tags** field.



- Check the highlighted objects in diagrams. These are the links that existed in the original model.
- Now modify the **Filters** settings as shown below to exclude "design" links:



- Check the Simulink model. The highlighting now points to links you have just copied from DOORS database.

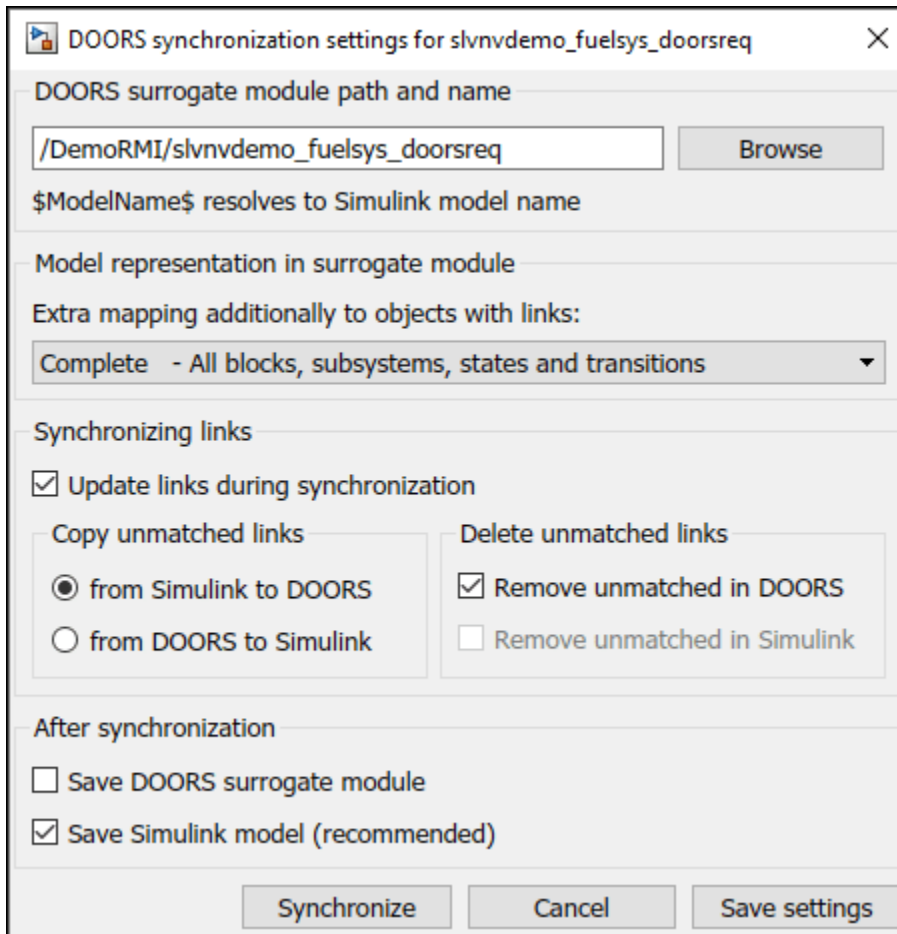
Removing Links in Simulink and DOORS

Synchronization also allows you to maintain consistency when links are removed. For example:

- Navigate to the fuel input again.

```
rmidemo_callback('locate', 'slvndemo_fuelsys_doorsreq/engine gas dynamics/fuel');
```

- Right-click, select **Requirements > Open Outgoing Links Dialog...**
- Select the "->2.1 Normal Mode of Operation" item in the dialog.
- Click **Delete** button to remove the item from the list.
- Click **OK** to apply the changes.
- Check the context menu again to confirm that the link is gone.
- Note that the link is still present in DOORS, connecting **1.11.3 fuel** in the surrogate module to "2.1 Normal Mode of Operation" in the **FuelSys Requirements Specification** module.
- Purge the removed link from DOORS by re-running synchronization with link updates option set to **Simulink to DOORS** and the **Remove unmatched in DOORS** checkbox enabled.



- Click **Synchronize**. Observe the link in DOORS disappear.

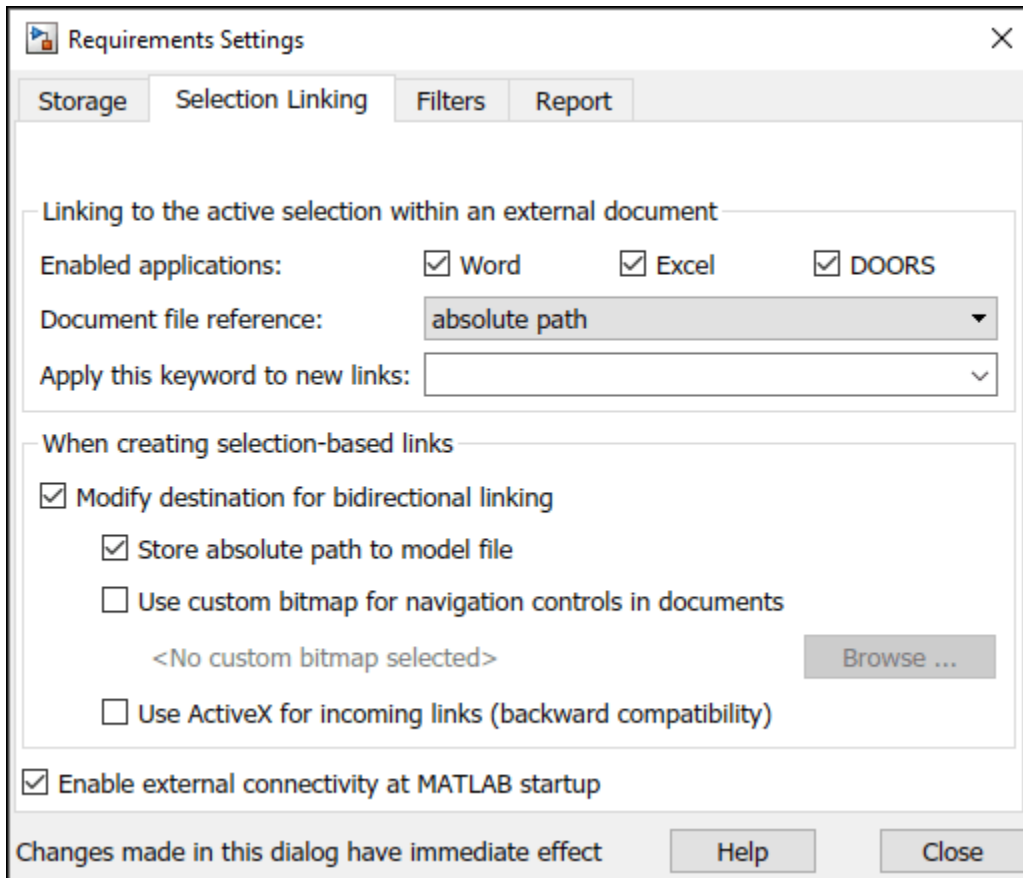
Similarly, when links are removed in DOORS and you need to propagate the changes to Simulink, rerun synchronization with the **DOORS to Simulink** option selected and **Remove unmatched in Simulink** checkbox enabled.

Optional Direct Links from DOORS to Simulink

When using selection linking with DOORS, you have an option to automatically insert reference objects into DOORS documents to enable direct navigation from DOORS to Simulink without the need for the surrogate module.

WARNING: The DOORS document is modified when you use this feature of RMI.

- In the **Requirements** tab, click **Link Settings > Linking Options** to open the **Requirements Settings** dialog.
- Enable the **Modify destination for bidirectional linking** checkbox.

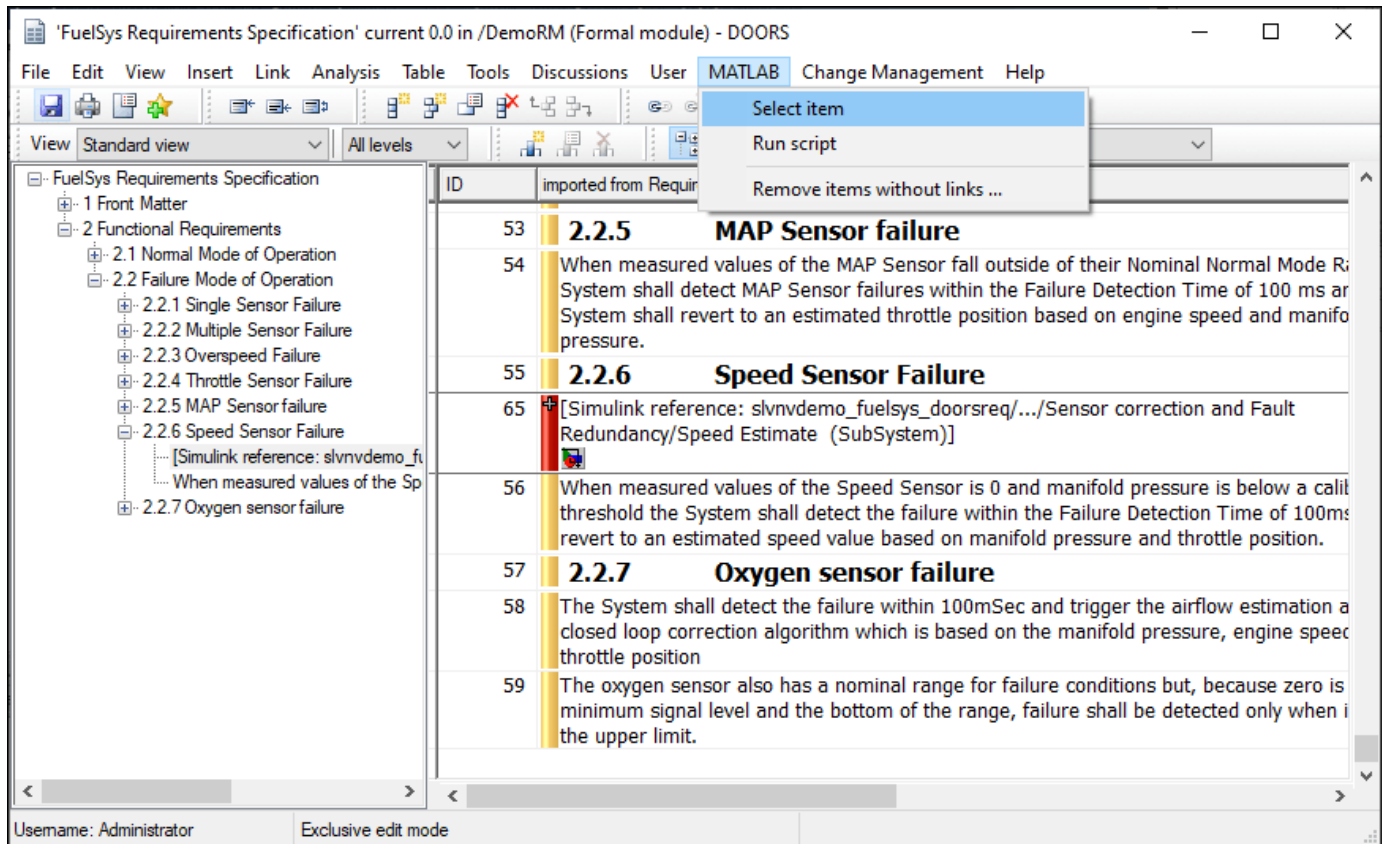


Now, when you use selection linking, Simulink creates navigation objects. There are two types of references to choose from. When **Use ActiveX for incoming links** option at the bottom of **Selection Linking** tab is ON, RMI will insert new DOORS objects with Simulink icon and destination object label as DOORS Object Text. With **Use ActiveX...** option OFF, RMI will insert External Link hyperlinks. For the following exercise, try both options and decide what works best for you.

- Locate and select "2.2.6 Speed Sensor Failure" in **FuelSys Requirements Specification** module.
- Locate the Speed Estimate block in the Simulink model.

```
rmidemo_callback('locate', ['slvndemo_fuelsys_doorsreq/fuel_rate_controller/' ...
    'Sensor correction and Fault Redundancy/Speed Estimate']);
```

- Right-click the block and select **Requirements > Link to Selection in DOORS**.
- Observe the new object inserted as the first child of the target object in DOORS.



- Click the just inserted navigation object in DOORS, or use **MATLAB > Select Item** from the main menu of the DOORS module window.
- When using External Link hyperlinks, navigate the MATLAB hyperlink in the expanded cascade of right-click context menu.
- The correct diagram opens in Simulink and the linked block is highlighted.

Note: You have just enabled navigation from DOORS to Simulink model without needing to save any changes in the model. Consider this workflow when modifications to models need to be avoided.

Normally, when the Simulink model is saved after creating links, two-way navigation is possible while bypassing the complexity of surrogate synchronization process. However, there is the disadvantage of cluttering DOORS documents with Simulink navigation objects.

To avoid making unintentional modifications to your DOORS documents, re-open the **Requirements Settings** dialog to the **Selection Linking** tab and disable **Modify destination for bidirectional linking** checkbox.

Cleanup

Cleanup commands. Clears open requirement sets without saving changes, and closes open models without saving changes.

```
slreq.clear;
bdclose all;
```


Simulink Traceability Between Model Objects

- “Link Model Objects” on page 9-2
- “Link Test Cases to Requirements Documents” on page 9-3
- “Link Simulink Data Dictionary Entries to Requirements” on page 9-7
- “Link Signal Builder Blocks to Requirements and Simulink Model Objects” on page 9-9
- “Requirements Links for Library Blocks and Reference Blocks” on page 9-13
- “Navigate to Requirements from Model” on page 9-16
- “Link to Requirements Modeled in Simulink” on page 9-18

Link Model Objects

Link Objects in the Same Model

You can create a requirements link from one model object to another model object:

- 1 Right-click the link destination model object and select **Requirements > Select for Linking with Simulink**.
- 2 Right-click the link source model object and select **Requirements > Add Link to Selected Object**.
- 3 Right-click the link source model object again and select **Requirements**. The new link appears at the top of the **Requirements** submenu.

Link Objects in Different Models

You can create links between objects in related models. This example shows how to link model objects in `slvndemo_powerwindow_controller` and `slvndemo_powerwindow`.

- 1 Open the `slvndemo_powerwindow_controller` and `slvndemo_powerwindow` models.
- 2 In the `slvndemo_powerwindow` model window, double-click the `power_window_control_system` subsystem. The `power_window_control_system` subsystem opens.
- 3 In the `slvndemo_powerwindow/power_window_control_system` subsystem window, right-click the `control` subsystem. Select **Requirements > Select for Linking with Simulink**.
- 4 In the `slvndemo_powerwindow_controller` model window, right-click the `control` subsystem. Select **Requirements > Add Link to Selected Object**.
- 5 Right-click the `slvndemo_powerwindow_controller/control` subsystem and select **Requirements**. The new RMI link appears at the top of the **Requirements** submenu.
- 6 To verify that the links were created, in the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Highlight Links**.

The blocks with requirements links are highlighted.

- 7 Close the `slvndemo_powerwindow_controller` and `slvndemo_powerwindow` models.

Link Test Cases to Requirements Documents

Since requirements specify behavior in response to particular conditions, you can build test cases (test inputs, expected outputs, and assessments) from the model requirements. Test cases reproduce specific conditions using test inputs, and assess the actual model output against the expected outputs. As you develop the model, build test files that check system behavior and link them to corresponding requirements. By defining these test cases in test files, you can periodically check your model and archive results to demonstrate model stability.

Establish Requirements Traceability for Testing

If you have a Simulink Test and a Requirements Toolbox license, you can link requirements to test harnesses, test sequences, and test cases. Before adding links, review “Supported Requirements Document Types” on page 6-8.

Requirements Traceability for Test Harnesses

When you edit requirements links to the component under test, the links immediately synchronize between the test harness and the main model. Other changes to the component under test, such as adding a block, synchronize when you close the test harness. If you add a block to the component under test, close and reopen the harness to update the main model before adding a requirement link.


To view items with requirements links, on the **Apps** tab, under Model Verification, Validation, and

Test, click **Requirements Manager**. In the **Requirements** tab, click **Highlight Links**  Highlight Links .

Requirements Traceability for Test Sequences

In test sequences, you can link to test steps. To create a link, first find the model item, test case, or location in the document you want to link to. Right-click the test step, select **Requirements**, and add a link or open the link editor.

To highlight or remove the highlighting from test steps that have requirements links, toggle the

requirements links highlighting button  in the Test Sequence Editor toolstrip. Highlighting test steps also highlights the model block diagram.

Requirements Traceability for Test Cases

If you use many test cases with a single test harness, link to each specific test case to distinguish which blocks and test steps apply to it. To link test steps or test harness blocks to test cases,

- 1 Open the test case in the Test Manager.
- 2 In the left pane, in the **Test Browser** tab, select the test case.
- 3 In Simulink in the **Apps** tab, click **Requirements Manager**.
- 4 To link a test case to a:
 - Simulink block, right-click the block and select **Requirements > Link to Current Test Case** from the context menu.
 - Test step, double-click the test sequence block in the test harness to open the Test Sequence Editor. Right-click the test step and select **Requirements > Link to Current Test Case** from the context menu.

Requirements Traceability Example

This example demonstrates adding requirements links to a test harness and test sequence. The model is a component of an autopilot roll control system. This example requires Simulink Test and Requirements Toolbox.

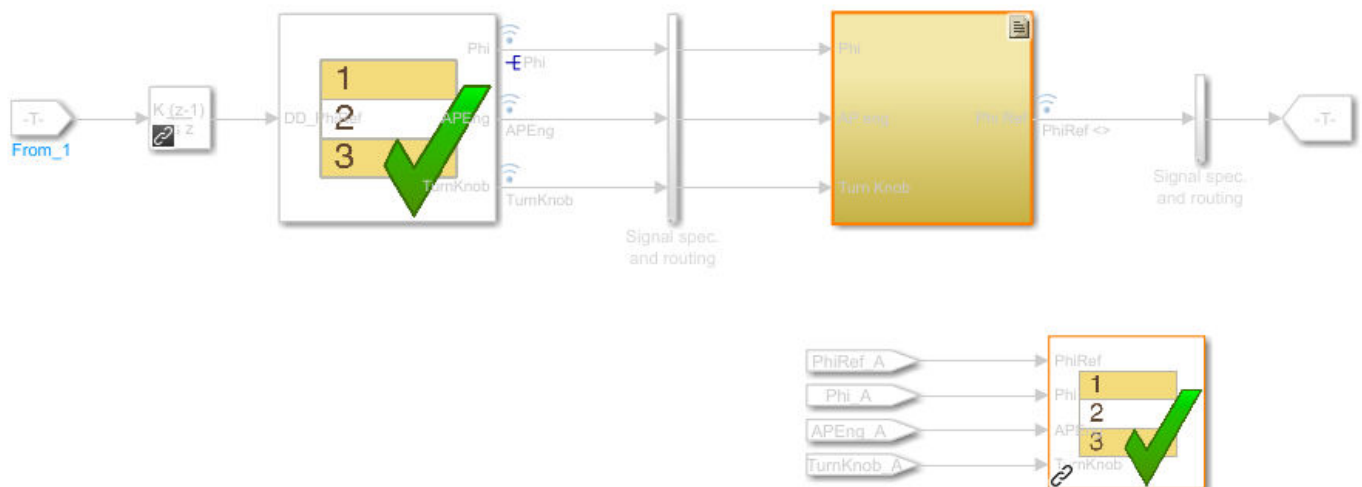
- 1 Open the model, the test file, and the harness.

```
openExample("simulinktest/ModelCoverageMATLABUnitExample", ...
    supportingFile="RollAutopilotMdlRef.slx")
openExample("simulinktest/ModelCoverageMATLABUnitExample", ...
    supportingFile="AutopilotTestFile.mldatx")
sltest.harness.open("RollAutopilotMdlRef/Roll Reference",...
    "RollReference_Requirement1_3")
```

- 2 In the test harness, on the **Apps** tab, under Model Verification, Validation, and Test, click

Requirements Manager. In the **Requirements** tab, click **Highlight Links** .

The test harness highlights the Test Sequence block, component under test, and Test Assessment block.



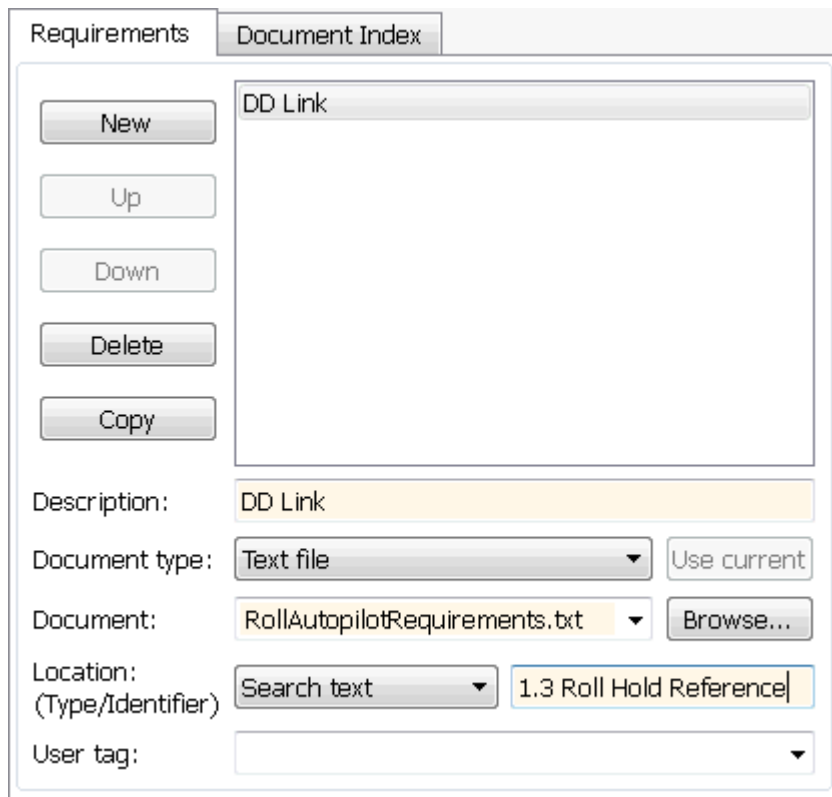
- 3 Add traceability to the Discrete Derivative block.

- a Right-click the Discrete Derivative block and select **Requirements > Open Outgoing Links dialog**.

- b In the **Requirements** tab, click **New**.

- c Enter the following to establish the link:

- Description: DD link
- Document type: Text File (legacy)
- Document: RollAutopilotRequirements.txt
- Location: 1.3 Roll Hold Reference



- d Click **OK**. The Discrete Derivative block highlights.
- 4 To trace to the requirements document, right-click the Discrete Derivative block, and select **Requirements > DD Link**. The requirements document opens in the editor and highlights the linked text.

1.3 Roll Hold Reference

```
Navigate to test harness using MATLAB command:
web('http://localhost:31415/matlab/feval/rmiobjnavigate?argu
```

REQUIREMENT

```
1.3.1 When roll hold mode becomes the active mode the roll hold
Navigate to test step using MATLAB command:
web('http://localhost:31415/matlab/feval/rmiobjnavigate?argu
```

```
1.3.1.1. The roll hold reference shall be set to zero if the act
Navigate to test step using MATLAB command:
web('http://localhost:31415/matlab/feval/rmiobjnavigate?argu
```

- 5 In the test harness, open the Test Sequence block. Add a requirements link that links the InitializeTest step to the test case.
- a In the Test Manager, in the left pane, in the **Test Browser** tab, select Requirement 1.3 Test.

- b In the test harness, double-click the test sequence block to open the Test Sequence Editor. Right-click the `InitializeTest` step and select **Requirements > Link to Current Test Case** from the context menu.

When the requirements link is added, the Test Sequence Editor highlights the step.

Step	Transition
<code>InitializeTest</code> <code>Phi = 0;</code> <code>APEng = false;</code> <code>TurnKnob = 0;</code> <code>% Initializes test sequence outputs</code>	1. <code>true</code>

See Also

“Requirements-Based Testing for Model Development” (Simulink Test) | “Link Test Cases to Requirements”

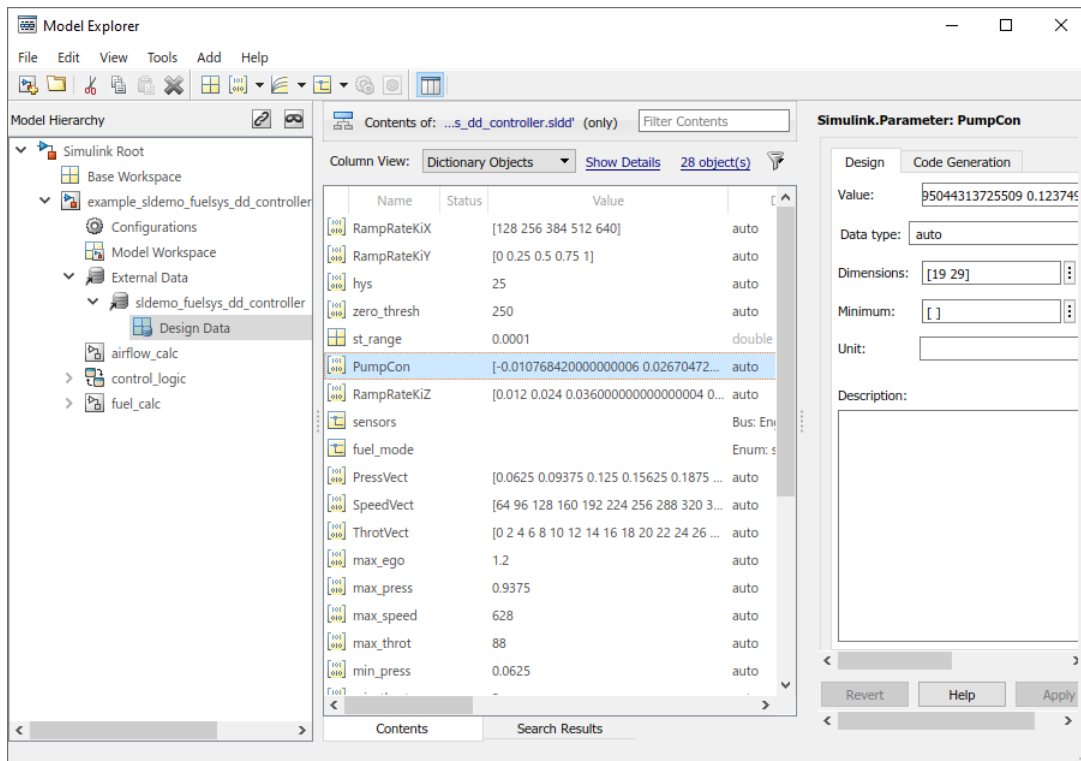
Link Simulink Data Dictionary Entries to Requirements

You can create requirements traceability links for entries in Simulink data dictionaries. The process is similar to linking for other model objects. In the Model Explorer, right-click a data dictionary entry, select **Requirements**, and choose one of the selection-based linking options. You can also use the Link Editor.

This example demonstrates linking to a data dictionary entry.

- 1 Open the `example_sldemo_fuelsys_dd_controller` model. At the MATLAB command line, enter:


```
openExample('simulink/IdentifyPerformanceSlowdownsUsingTheSimulinkProfilerExample')
open_system('example_sldemo_fuelsys_dd_controller')
close_system('profiling_example_fuelcontrol_model')
```
- 2 In the `example_sldemo_fuelsys_dd_controller` model, open the linked data dictionary. Click the model data badge in the bottom left corner of the model, then click the **External Data** link.
- 3 In the **Model Hierarchy** pane of the Model Explorer, under the **External Data** node, expand the `sldemo_fuelsys_dd_controller` data dictionary node.
- 4 Select **Design Data**.
- 5 Locate the `PumpCon` parameter.



- 6 In the `example_sldemo_fuelsys_dd_controller` model, open the `airflow_calc` subsystem and select the Pumping Constant lookup table.
- 7 In the Model Explorer, right-click the `PumpCon` parameter and select **Requirements > Link to Selection in Simulink** to create a link between the two items.

- 8 Check the link by right-clicking the PumpCon parameter and selecting **Requirements**, then select the navigation shortcut at the top of the **Requirements** submenu. Simulink highlights the lookup table.

Link Signal Builder Blocks to Requirements and Simulink Model Objects

This example shows how to create links from a signal group in a Signal Builder block to a requirements document and to a model object.

Note When you create links as described in this example, requirements link to individual signal groups, not with the entire Signal Builder block.

In this example, switch to a different active group in the drop-down list to link a requirement to another signal group.

Link Signal Editor Blocks to Requirements Documents

This example shows how to create links from a signal group in a Signal Editor block to a requirements document.

Open the `sf_car` model.

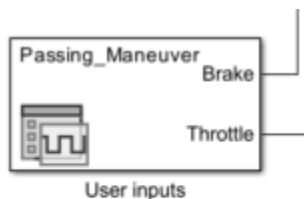
```
open_system("sf_car")
```

Set the Document File Reference Settings



- 1 In the **Apps** tab, open **Requirements Manager**.
- 2 In the **Requirements** tab, ensure **Layout > Requirements Browser** is selected.
- 3 In the **Requirements Browser**, in the **View** drop-down menu, select **Links**.
- 4 In the **Requirements** tab, select **Link Settings > Default Link Storage**.
- 5 Next to **Document file reference**, select the option from the list that suits your needs for your external requirements document. For more information, see “Document Path Storage” on page 12-36.

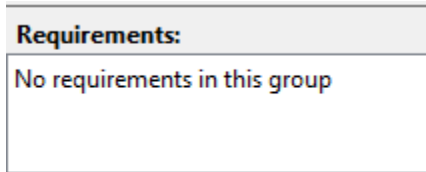
Create Links from the Signal Editor Block

In the `sf_car` model window, double-click the **User Inputs** block. The Signal Editor dialog box opens, displaying four groups of signals. The **Passing Maneuver** signal group is the current active group. The requirements link to the current active signal group.



*Double-click to
open the GUI
and select an
input maneuver*

At the far-right end of the toolbar, click the **Show verification settings** button . Ensure that the **Requirements display** button  is selected. The **Requirements** pane opens on the right-hand side of the Signal Editor dialog box.



Create the link to this signal group by using the Outgoing Links dialog.

- 1 In the Signal Editor window, in the **Requirements** pane, right-click and select **Open Outgoing Links dialog**. The Outgoing Links dialog opens.
- 2 Click **New**. In the **Description** field, enter `User input requirements`.
- 3 Click **Browse** and select your external requirements document, then click **Open**.
- 4 In the **Location** drop-down list, select **Search text** to link to specified text in the document.
- 5 Next to the **Location** drop-down list, enter `User Input Requirements` to create a link to that specified text in the requirements document.
- 6 Click **Apply** to create the link.
- 7 To verify that the link was created, in the `sf_car` model, select the User Inputs block, right-click, and select **Requirements**. The link to the new requirement is the option at the top of the submenu.

Clear the open requirement sets and link sets and close the model.

```
slreq.clear;
bdclose all;
```



Link Signal Builder Blocks to Model Objects

This example shows how to create links from a signal group in a Signal Builder block to a model object:

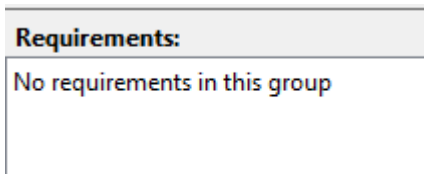
- 1 Open the `sf_car` model.


```
openExample("slrequirements/LinkSignalBuilderBlocksExample")
open_system("sf_car")
```
- 2 Open the `sf_car/shift_logic` chart.
- 3 Right-click `upshifting` and select **Requirements > Select for Linking with Simulink**.
- 4 In the `sf_car` model window, double-click the User Inputs block.

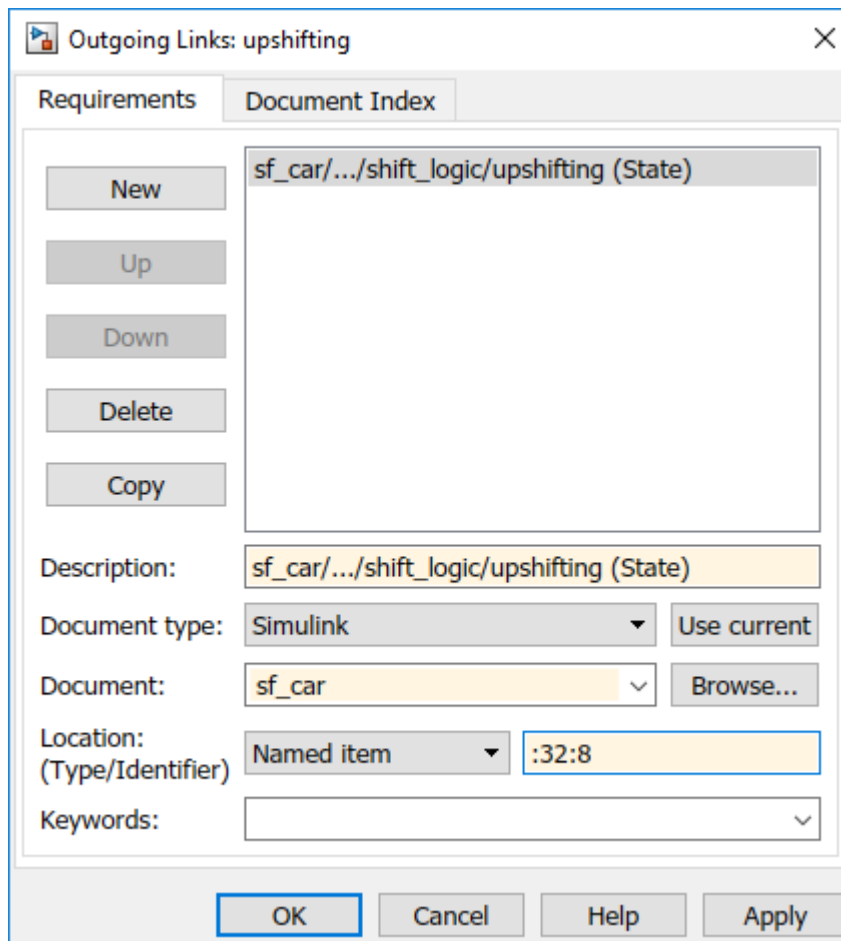
The Signal Builder dialog box opens, displaying four groups of signals. The Passing Maneuver signal group is the current active group. The RMI associates any requirements links that you add to the current active signal group.

- 5 In the Signal Builder window, in the **Active Group** list, select `Gradual Acceleration`.
- 6 At the far-right end of the toolbar, click the **Show verification settings** button . Ensure that the **Requirements display** button  is selected.

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



- 7 Place your cursor in the window, right-click, and select **Open Outgoing Links dialog**. The Outgoing Links dialog opens.
- 8 Click **New**. In the **Description** field, enter Upshifting.
- 9 In the **Document type** field, select Simulink Diagram. Click **Use current**. The software fills in the field with the **Location: (Type/Identifier)** information for upshifting.



- 10 Click **Apply** to create the link.
- 11 In the model window, select the User Inputs block, right-click, and select **Requirements**.

The link to the new requirement is the option at the top of the submenu.

- 12 To verify that the links were created, in the sf_car model window, in the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Highlight Links** to highlight the model objects with requirements.

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

13 Close the sf_car model.

See Also

More About

- “Create and Store Links” on page 3-31
- “Link Model Objects” on page 9-2
- “Link Test Cases to Requirements Documents” on page 9-3
- “Link Simulink Data Dictionary Entries to Requirements” on page 9-7

Requirements Links for Library Blocks and Reference Blocks

Introduction to Library Blocks and Reference Blocks

Simulink allows you to create your own block libraries. If you create a block library, you can reuse the functionality of a block, subsystem, or Stateflow atomic subchart in multiple models.

When you copy a library block to a Simulink model, the new block is called a reference block. You can create several instances of this library block in one or more models.

The reference block is linked to the library block using a library link. If you change a library block, any reference block that is linked to the library block is updated with those changes when you open or update the model that contains the reference block.

Note For more information about reference blocks and library links, see “Custom Libraries” (Simulink).

Library Blocks and Requirements

Library blocks themselves can have links to requirements. In addition, if a library block is a subsystem or atomic subchart, the objects inside the library blocks can have library links. You use the Requirements Management Interface (RMI) to create and manage requirements links in libraries and in models.

The following sections describe how to manage requirements links on and inside library blocks and reference blocks.

Copy Library Blocks with Requirements

When you copy a library subsystem or masked block to a model, you can highlight, view and navigate requirements links on the library block and on objects inside the library block. However, those links are not associated with that model. The links are stored with the library, not with the model.

You cannot add, modify, or delete requirements links on the library block from the context of the reference block. If you disable the link from the reference block to the library block, you can modify requirements on objects that are inside library blocks just as you can for other block attributes when a library link has been disabled.

Manage Requirements on Reference Blocks

You use the RMI to manage requirements links on a reference block just like any other model object. You can view and navigate both local and library requirements on a reference block.

- Locally created requirements links — Can be modified or deleted without changing the library block:
 - **Manifold absolute pressure sensor**
 - **Mass airflow estimation**
- Requirements links on the library block — Cannot be modified or deleted from the context of the reference block:

- **Speed sensor**
- **Throttle sensor**
- **Oxygen sensor**

Manage Requirements Inside Reference Blocks

If your library block is a subsystem or a Stateflow atomic subchart, you can create requirements links on objects *inside* the subsystem or subchart. If you disable the link from the reference block to the library, you can add, modify, or delete requirements links on objects inside a reference block. Once you have disabled the link, the RMI treats those links as locally created links.

After you make changes to requirements links on objects inside a reference block, you can resolve the link so that those changes are pushed to the library block. The next time you create an instance of that library block, the changes you made are copied to the new instance of the library block.

The workflow for creating a requirement link on an object inside a reference block is:

- 1 Within a library you have a subsystem S1. Drag S1 to a model, creating a new subsystem. This subsystem is the reference block.
- 2 Disable the library link between the reference block and the library block. Keep the library loaded while you disable the link to maintain RMI data. To disable the link, select the reference block, and in the **Subsystem** tab, click **Disable Link**.
- 3 Create a link from the object inside the reference block to the requirements document.

Note When linking to a requirement from inside a reference block, you can create links only in one direction: from the model to the requirements document. The RMI does not support inserting navigation objects into requirements documents for objects inside reference blocks.

- 4 Resolve the library link between the reference block and the library block:
 - a Select the reference block.
 - b In the **Subsystem** tab, click **Restore Link**.
 - c In the **Action** column, click **Push**.
 - d Click **OK** to resolve the link to the library block and push the newly added requirement to the object inside the library block.

When you resolve the library link between the library block and the subsystem, Simulink pushes the new requirement link to the library block S1. The following graphic shows the new link from inside the library block S1 to the requirement.

Note 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.

- 5 If you reuse library block S1, which now has an object with a requirement link, in another model, the new subsystem contains an object that links to that requirement.

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 links *only* to the library block.

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

When you use library block B1 in a model, you can navigate from the reference block to the requirement. However, the link from the requirement still points only to library block B1, not to the reference block.

As discussed in the previous section, you can create requirements links on objects inside instances of library block after disabling library links. However, the RMI prohibits you from creating a link from the requirements document to such an object because that link would become invalid when you restored the library link.

Navigate to Requirements from Model

Navigate from Model Object

You can navigate directly from a model object to that object's associated requirement. When you take these steps, the external requirements document opens in the application, with the requirements text highlighted.

- 1 Open the example 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.

Note If you are running a 64-bit version of MATLAB, when you navigate to a requirement in a PDF file, the file opens at the top of the page, not at the bookmark location.

Navigate from System Requirements Block

Sometimes you want to see all the requirements links at a given level of the model hierarchy. In such cases, you can insert a System Requirements block to collect all requirements links in a model or subsystem. The System Requirements block lists requirements links for the model or subsystem in which it resides; it does not list requirements links for model objects inside that model or subsystem, because those are at a different level of the model hierarchy.

In the following example, you insert a System Requirements block at the top level of the `slvndemo_fuelsys_officereq` model, and navigate to the requirements using the links inside the block.

- 1 Open the example model:
`slvndemo_fuelsys_officereq`
- 2 Enable **Model Highlighting** in the **Coverage** app.
- 3 Open the fuel rate controller subsystem.

The Airflow calculation subsystem has a requirements link.
- 4 Open the Airflow calculation subsystem.
- 5 In the Simulink toolstrip, click **Library Browser**.
- 6 In the **Libraries** tree view, select **Simulink Requirements**.

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

- 7 Drag a System Requirements block into the Airflow calculation subsystem.

The RMI software collects and displays any requirements links for that subsystem in the System Requirements block.

- 8** In the System Requirements block, double-click **1. “Mass airflow subsystem”**.

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

Link to Requirements Modeled in Simulink

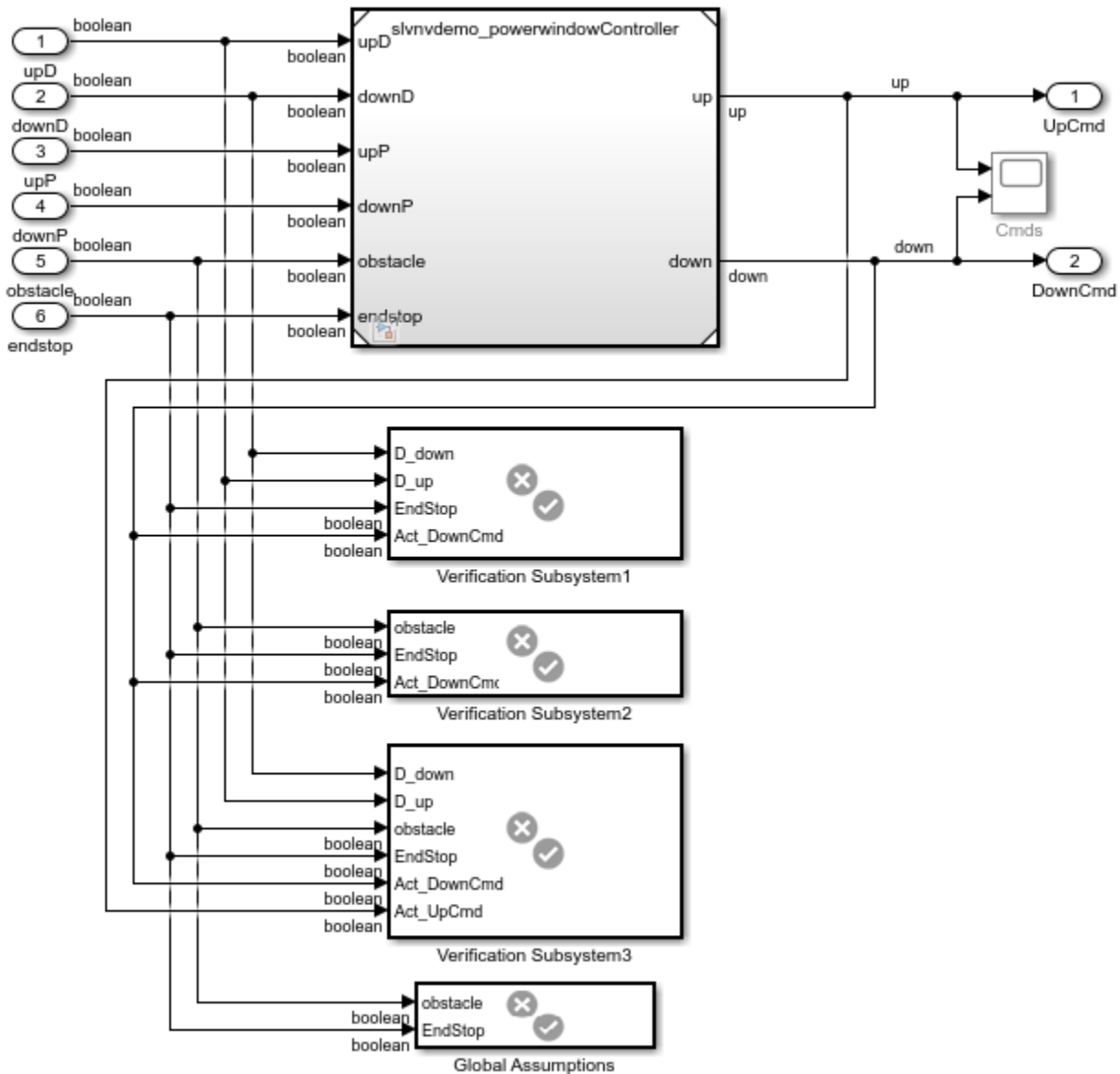
This example shows how to link between verification subsystems and models. You can use verification subsystems to model functional requirements and verify them in simulation. Traceability between the verification and implementation models allow you to summarize analysis and test results in the **Requirements Editor** .

The Verification and Design Models

At the command line, enter

```
open_system('slvndemo_powerwindow_vs')
```

Power Window Controller Temporal Property Specification



Copyright 1990-2010 The MathWorks, Inc.

The verification model specifies properties and requirements for `slvndemo_powerwindowController`. The verification subsystems include logic that verifies system behavior when an obstacle is detected:

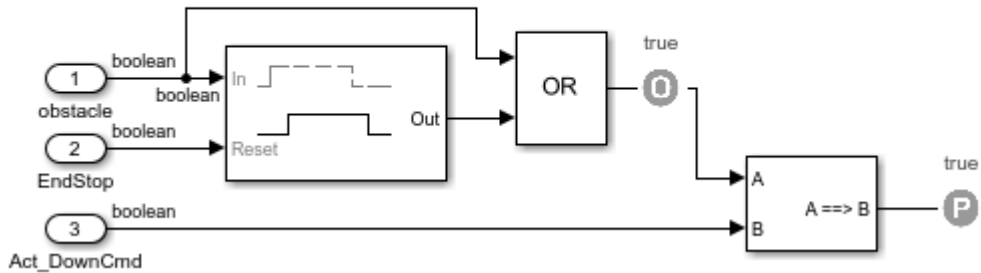
- **Obstacle Response:** When an obstacle is detected, the controller shall give the down command for 1 second.

The requirement is modeled in `Verification Subsystem2`.

```
open_system('slvndemo_powerwindow_vs/Verification Subsystem2')
```

Requirement:

Whenever an obstacle is detected, then the down command shall be given for 1 second.



- In the design model, the obstacle response is implemented in the emergencyDown state:

```

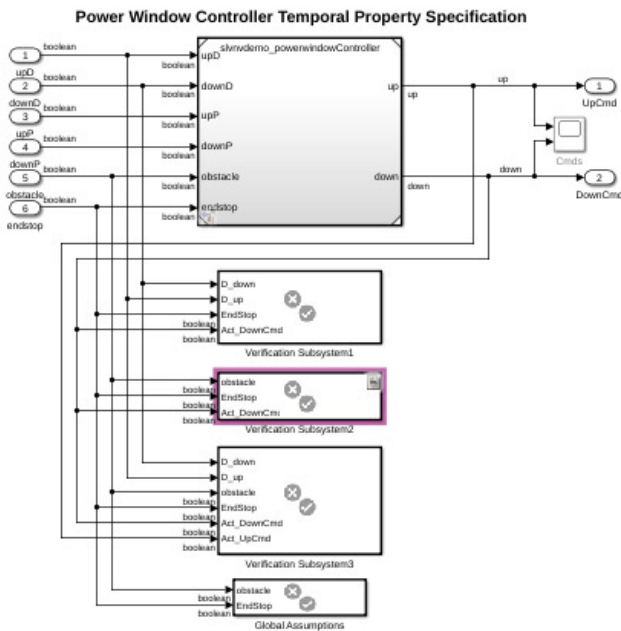
emergencyDown
entry:
moveUp = 0;
moveDown = 1;
    
```

Link from Verification to Design Model

Link from Verification Subsystem2 to the emergencyDown state:

- 1 Double-click on the Model block to open slvndemo_powerwindow.
- 2 In the control chart, right-click the emergencyDown state and select **Requirements > Select for Linking with Simulink**.
- 3 In the slvndemo_powerwindow_vs model, right click Verification Subsystem2 and select **Requirements > Add Link to Selected Object**.
- 4 In the slvndemo_powerwindow_vs model, open the **Requirements Manager** app. A badge appears on Verification Subsystem2, indicating a link, and the link appears in the Property Inspector.
- 5 Change the link type to **Verifies**. Next to the link in the Property Inspector, click the **Show in Links View** icon. Select the link in the table, then change the link property **Type** from Implements to Verifies.

slvndemo_powerwindow_vs ▶



Link: slvndemo_powerwindowController/control/e...

Details

▼ Properties

Source: [Verification Subsystem2](#)

Type: Verifies

Destination: [emergencyDown](#)

Description Rationale

slvndemo_powerwindowController/control/emergenc

Keywords:

Cleanup

These commands unload requirements sets and close open models.

```
slreq.clear; % Closes open requirements sets without saving changes
close_system('slvndemo_powerwindow_vs',0)
```


MATLAB Code Traceability

- “Requirements Traceability for MATLAB Code” on page 10-2
- “Associate Traceability Information with MATLAB Code Lines in Simulink” on page 10-9

Requirements Traceability for MATLAB Code

You can associate requirements with MATLAB code and plain-text external code, such as C code, by creating selection-based links with the **Requirements Editor** or by creating links programmatically at the MATLAB command line. You can also create links to MATLAB code in MATLAB Function blocks. You can verify requirements with MATLAB code by creating links to MATLAB unit tests and running the tests. You can then view and edit links to code in the MATLAB Editor or **Requirements Editor**.

Create Links to MATLAB Code or Plain-Text External Code

You can create links to MATLAB code or plain-text external code programmatically or by using the **Requirements Editor**.

When you create links to code, Requirements Toolbox creates `slreq.TextRange` objects that correspond to the selected lines. These `slreq.TextRange` objects are referred to as line ranges.

To create links programmatically, create the `slreq.TextRange` object, then use the object as a link source when you create the link. When you create multiple `slreq.TextRange` objects in the same MATLAB code file or plain-text external code file, the line numbers for the `slreq.TextRange` objects cannot overlap.

Create Links by Using the Requirements Editor

To create selection-based links to code by using the **Requirements Editor**:

- 1 In the MATLAB Editor, open the MATLAB code file or plain-text external code file.

Note You cannot create links to MATLAB code in MLX files.

- 2 Select the lines of code that you want to link.
- 3 In the **Requirements Editor**, load the requirement set that you want to link.
- 4 Select the requirement to link.
- 5 In the **Links** section, click **Add Link > Link from Selection in MATLAB Editor**.

Alternatively, in the MATLAB Editor, right-click the selected code range and select **Requirements > Link to Selection in Requirements Browser**.

Create Links Programmatically

Suppose that you want to programmatically create links to a MATLAB function called `myAdd`.

```
function y = myAdd(u,v)
y = u + v;
end
```

You want to link the function to these requirements:

myAddRequirements		
1	#1	Input u
2	#2	Input v
3	#3	Add u and v
4	#4	Output y

To create links at the MATLAB command line:

- 1 Use `slreq.createTextRange` to create an `slreq.TextRange` object that represents the lines of code that you want to link to.

```
lr = slreq.createTextRange("myAdd.m",2);
```

- 2 Use `slreq.find`, `find`, or `slreq.getCurrentObject` to get a handle to the requirement that you want to link.

```
req = slreq.find(Type="Requirement",Summary="Add u and v");
```

- 3 Use `slreq.createLink` to create the link.

```
myLink = slreq.createLink(lr,req);
```

Create Links to MATLAB Function Blocks

Use the **MATLAB Function Block Editor** to create links to lines of code in MATLAB Function blocks:

- 1 In the **Requirements Editor**, load the requirement set that you want to link to.
- 2 Select the requirement to link.
- 3 In the Simulink model, open the MATLAB Function block.
- 4 Select the lines of code that you want to link.
- 5 Right-click the selected code range and select **Requirements > Link to Selection in Requirements Browser**.

Note Requirements linked to MATLAB code lines inside a MATLAB Function block appear in HTML requirements traceability reports, but do not appear the Simulink Report Generator™ Web View. See “Create and Use Web Views of Models” (Simulink Report Generator).

Create Links to Requirements in External Documents

To create links from MATLAB code to requirements in external documents:

- 1 Select a requirement in one of these external documents:
 - Microsoft Word
 - Microsoft Excel
 - IBM Rational DOORS
 - IBM DOORS Next
- 2 In the MATLAB Editor, open the MATLAB code file or plain-text external code file.

Note You cannot create links to MATLAB code in MLX files.


- 3 Select the lines of code that you want to link.
- 4 Right-click the selected code range and select **Requirements**. Depending on the type of your requirements document, select one of these options:
 - **Link to Selection in Word**
 - **Link to Selection in Excel**
 - **Link to Selection in DOORS**
 - **Link to Selected Item(s) in DOORS Next**

For more information about configuring Requirements Toolbox to work with these third-party products, see “Configure Requirements Toolbox for Interaction with Microsoft Office and IBM DOORS” on page 6-2 and “Configure IBM DOORS Next Session” on page 8-4.

Verify Requirements with MATLAB Tests

You can verify requirements with MATLAB code by creating links to class-based and function-based tests, then running the tests.

When you create a link from a requirement to a MATLAB unit test, Requirements Toolbox sets the link type to `Verify`, which enables the software to verify the requirement. For more information, see “Review Requirements Verification Status” on page 4-6.

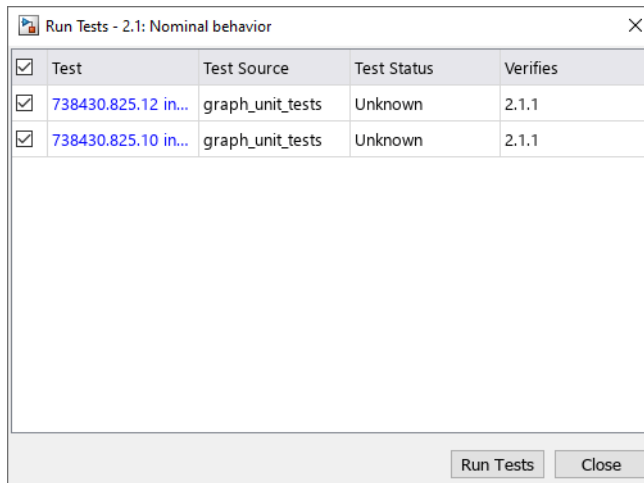
To verify the requirement, run the test at the MATLAB command line or by using the **Requirements Editor**. To view the verification status in the **Requirements Editor**, select  **Columns > Verification Status**.

Note Test results from the **Test Browser** and **MATLAB Test Manager** do not affect the verification status in the **Requirements Editor**.

Run Tests by Using the Requirements Editor

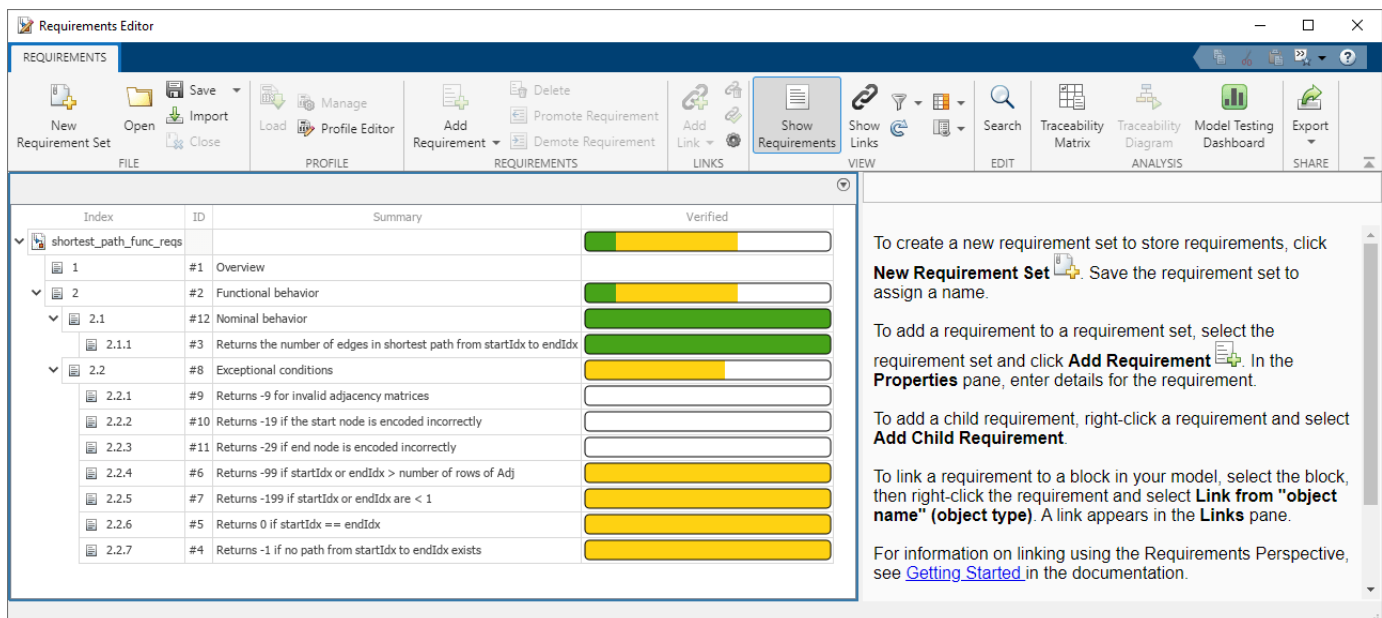
To run MATLAB unit tests for a requirement set, for a parent requirement and all of its descendants, or for a single requirement by using the **Requirements Editor**:

- 1 In the **Requirements Editor**, right-click a requirement set, a parent requirement, or a child requirement that has links to tests and select **Run Tests**.
- 2 In the Run Tests dialog box, confirm that you want to run the linked tests. To omit tests from the run, clear the selection.



3 Click **Run Tests**.

The verification status of the requirements updates after you run the tests.



Run Tests at the MATLAB Command Line

Supposed that you want to verify the requirements in the ShortestPath project by running MATLAB tests programmatically.

To run tests linked to a requirement set programmatically:

1 Open the ShortestPath project.

```
slreqShortestPathProjectStart
```

2 Load the shortest_path_tests_reqs requirement set and open it in the **Requirements Editor**.

```
rs = slreq.open("shortest_path_tests_reqs");
```

- 3 Run the tests that are linked to the requirement set by using `slreq.ReqSet.runTests`.

```
results = runTests(rs);
```

- 4 View the verification status in the **Requirements Editor**. Alternatively, view the verification status at the command line by using `getVerificationStatus`.

```
status = getVerificationStatus(rs)
```

```
status =
    total: 14
    passed: 13
    failed: 0
    unexecuted: 0
    justified: 0
    none: 1
```

View and Edit Links and Linked Line Ranges

You can view linked code ranges in the MATLAB Editor by enabling requirements highlighting. Right-click in the MATLAB Editor and select **Requirements > Enable Requirements Highlighting**.

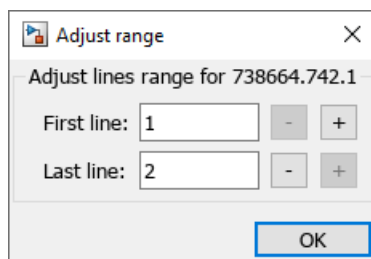
To edit links, use the **Requirements Editor**. For more information, see “View and Edit Links” on page 3-37.

You can also edit the starting and ending lines for a linked line range in the MATLAB Editor or at the MATLAB command line.

Edit Linked Line Ranges in the MATLAB Editor

To edit the line numbers for a linked line range in the MATLAB Editor:

- 1 Right-click a highlighted line range and select **Adjust line range**.
- 2 In the Adjust range dialog box, use the **+** and **-** buttons to change the first and last lines of the line range. Alternatively, enter the line number in the field.



- 3 Click **OK**. The MATLAB Editor updates the highlighting.

Edit Linked Line Ranges Programmatically

To edit the lines for an `slreq.TextRange` object at the MATLAB command line:

- 1 Open the MATLAB code file or plain-text external code file in the MATLAB Editor.


```
open("myAdd.m");
```
- 2 Get the existing `slreq.TextRange` object in the MATLAB code file by passing the file name and the line numbers for the start and end of the line range to the `slreq.getTextRange` function.

```
lr = slreq.getTextRange("myAdd.m",[1 2]);
```

- 3 Modify the line range by using `setLineRange`.

```
setLineRange(lr,1);
```

- 4 The MATLAB Editor updates the code range highlighting. Alternatively, you can confirm the changes by using `show`, `getText`, or `getLineRange`.

Save Links

To save the changes to a link set when you create or edit links to lines of MATLAB code or external code, use one of these approaches:

- In the MATLAB Editor, right-click and select **Requirements > Save Links**.
- In the **Requirements Editor**, click **Show Links**. Select the link set, and click **Save**.
- At the MATLAB command line, use `save`.

Delete Links and Unused Line Ranges

You can delete links to MATLAB code in the MATLAB Editor, the **Requirements Editor**, or at the MATLAB command line.

If you delete links to code ranges in the MATLAB Editor, you can delete the unused line ranges in the MATLAB Editor or at the MATLAB command line.

Delete Links

To delete links in the MATLAB Editor, right-click a highlighted code range and select **Requirements > Delete All Links**. This deletes all incoming and outgoing links to this code range.

To delete links from the **Requirements Editor**, see “Delete Links and Link Sets” on page 3-37.

To delete links at the MATLAB command line, get a handle to an `slreq.Link` object and use `remove`. Alternatively, ensure that the file containing the code range is open in the MATLAB Editor, then get an `slreq.TextRange` object by using `slreq.getTextRange`. Delete the links to the object by using `deleteLinks`.

Delete Unused Line Ranges

To delete unused line ranges in the MATLAB Editor, right-click the line containing the line range and select **Requirements > Delete line range**.

Note If a line range does not have outgoing links, the MATLAB Editor does not highlight the code line. Note the line number before you delete the link so that you can delete the unused line range. Alternatively, you can use the MATLAB command line to get the line ranges in a file and then delete the unused line ranges.

To delete line ranges at the command line, ensure that the file containing the code range is open in the MATLAB Editor, then get an `slreq.TextRange` object by using `slreq.getTextRange`. Delete the code range by using `remove`.

See Also

`slreq.TextRange` | `slreq.createTextRange` | `slreq.getTextRange`

More About

- “Create and Store Links” on page 3-31
- “Load and Resolve Links” on page 3-39
- “Requirements Traceability for Code Generated from MATLAB Code” on page 3-53

Associate Traceability Information with MATLAB Code Lines in Simulink

Traceability management support in the MATLAB Editor is an extension of the Simulink®-based Requirements Management Interface to allow associations between MATLAB® code lines and external artifacts. This capability does not require editing MATLAB files; all traceability data is stored separately. This is similar to "external" storage of RMI links when working with Simulink models, as in "Managing Requirements Without Modifying Simulink Model Files" on page 12-45.

In addition, using the "external storage" mode for managing traceability information, Simulink and Stateflow® users can benefit from finer granularity when associating external documents with contents of MATLAB Function blocks.

The included example model has traceability data associated both with Simulink blocks and individual code lines of MATLAB Function blocks.

Open Example Model

This example demonstrates linking between external documents and MATLAB code lines when modeling stimulated spiking in connected neural cells.

Evaluate the following code to open the `slvndemo_synaptic_transmission` Simulink model in the working directory and set a preference to allow proper communication for files in this example.

```
open('slvndemo_synaptic_transmission.slx');
rmipref('UnsecureHttpRequests',true);
```

There are three Model blocks referencing the same model of a spiking neural cell which can be seen in `slvndemo_neuron.slx`. Evaluate the code to open the model.

```
open('slvndemo_neuron.slx');
```

The neural cell model follows a "Leaky Integrators" equation:

$$\frac{C_m dV}{dt} = I_{total} - \frac{V - V_0}{R_m}$$

C_m, R_m – capacitance and resistance of cell membrane

I_{total} – includes injected stimulation current and all ion channel currents

V_0 – resting cross-membrane potential, typically -70mV

For the purpose of simulation, this is converted to:

$$V \rightarrow V_0 + \int_0^t \frac{1}{C_m} (I_{total} - \frac{V - V_0}{R_m}) dt$$

Two MATLAB Functions between neurons calculate post-synaptic currents. When pre-synaptic depolarization crosses the neurotransmitter release threshold, we increment post-synaptic current by one pulse of given amplitude:

$$I \rightarrow I + I_{\text{amplitude}}$$

The resulting total current decays exponentially according to:

$$\frac{dI}{dt} = -I * \frac{t}{\tau}$$

The next increment is disallowed for a certain time frame after the previous pulse to model the effect of short-term synaptic depression. The model neglects the time delay of axonal transmission.

Simulate Model and View Results

Evaluate the following code to simulate `slvnvdemo_synaptic_transmission` model.

```
sim('slvnvdemo_synaptic_transmission');

### Starting serial model reference simulation build.
### Successfully updated the model reference simulation target for: slvnvdemo_neuron
```

Build Summary

Simulation targets built:

Model	Action	Rebuild Reason
slvnvdemo_neuron	Code generated and compiled.	slvnvdemo_neuron_msf.mexw64 does not exist.

1 of 1 models built (0 models already up to date)
Build duration: 0h 0m 41.144s

Manually check the Scope block for results or evaluate the following code.

```
open_system('slvnvdemo_synaptic_transmission/Scope');
```

The six plots are:

- 1 externally injected electrical current pulse
- 2 injection-stimulated intracellular voltage spiking of the first neuron
- 3 post-synaptic current generated in the second neuron
- 4 synaptically stimulated activity of the second neuron
- 5 post-synaptic current generated in the third neuron
- 6 synaptically stimulated activity of the third neuron

Observe regular spiking of the upstream neuron (plot 2) while stimulation pulse is applied (plot 1). Synaptically induced current in downstream neuron (plot 3) skips some action potentials of the upstream neuron due to short-term neurotransmitter depletion modeled as a temporary turn-off period in the `Synaptic current` function block. Evaluate the code to navigate to the `Synaptic current` block.

```
rmidemo_callback('locate','slvnvdemo_synaptic_transmission/Synaptic current');
```

The downstream neuron is seen to, sometimes, integrate more than one synaptic input to produce a spike (plot 4). The third neuron integrates synaptic inputs from the second neuron (plot 5) and spikes at a later time (plot 6). With the default parameter values, the third neuron may spike 1 or more

times, depending on intentionally introduced random noise in the model, by the `Noise current` block in the `slvndemo_neuron` model. Navigate to the `Noise current` block.

```
rmidemo_callback('locate','slvndemo_neuron/Noise current');
```

The same parameter values are assigned for all three neurons and both synapses. Traceability linking is used to justify parameter values and implementation.

Navigate Between Simulink and Standalone MATLAB Files

The `slvndemo_synaptic_transmission` model runs an external script `synaptic_params.m` to load required parameter values into workspace. If desired, open the script by evaluating the following code: `open('synaptic_params.m')`.

MATLAB code linking allows you to trace from a dependent block in Simulink, not only to a script file but to the specific line that defines a value used in simulation.

Locate the `Stimulation pulse` block in the model manually or evaluate the following code.

```
rmidemo_callback('locate','slvndemo_synaptic_transmission/Stimulation pulse');
```

Right-click the block and select **Requirements > 1. "I_inj = 2e-11; % 20 pA"** to follow the link and view the relevant highlighted region in the MATLAB code file, or evaluate the following:

```
open('synaptic_params.m'); % If desired, open the synaptic parameters script
```

```
rmidemo_callback('view','slvndemo_synaptic_transmission/Stimulation pulse',1); % Follow the Requirements link
```

Notice that in the `synaptic_params.m` script, there is a mismatched parameter value. This is easily detected via Traceability link. Highlight line 12 in the `synaptic_params.m` script which reads `V_syn_release = -2e-2; % -20 mV`. Right click the selection and in the context menu, click **Requirements > 1. slvndemo_synaptic_transmission/Synaptic release threshold (Constant)**. This navigates you back to the Simulink model and highlights the block that is linked to line 12 in `synaptic_params.m`.

In Simulink, click the **Apps** tab and open **Requirements Manager**. Click **Highlight links** in the **Requirements** tab to highlight the links that you just created, or evaluate the following code.

```
rmi('highlightModel','slvndemo_synaptic_transmission');
```

Create Traceability Link for Lines of MATLAB Code

Evaluate the following code to make sure that bidirectional linking is enabled so that you can create two-way traceability links in one step.

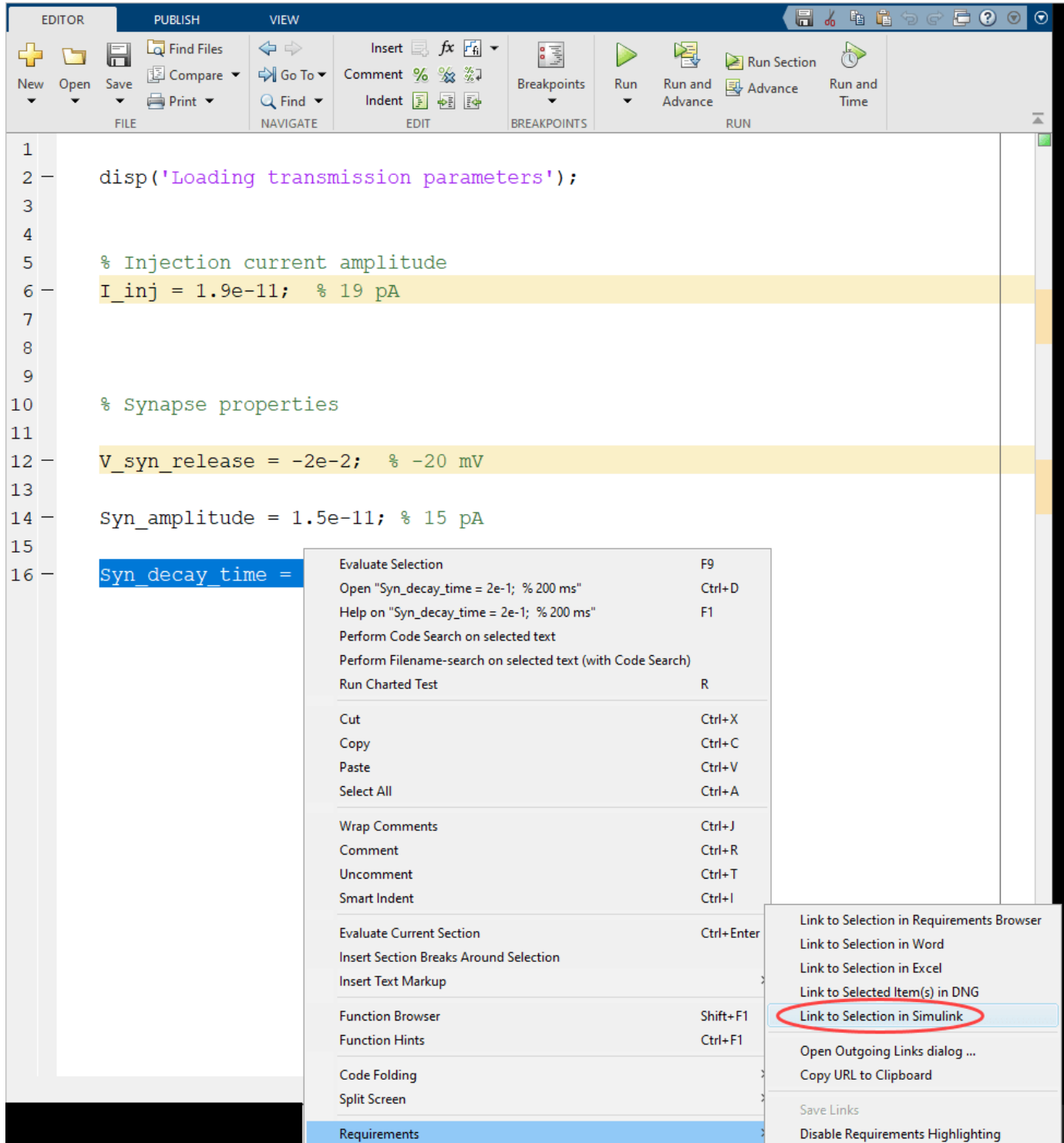
```
rmipref('BiDirectionalLinking',true);
```

The `Synaptic time constant` block in the bottom left corner of the model is not yet linked to its related line in `synaptic_params.m`. In the Simulink model, select the `Synaptic time constant` block. If you can't find this block, evaluate the following code and then select it.

```
rmidemo_callback('locate','slvndemo_synaptic_transmission/Synaptic time constant');
```

Then, in the MATLAB Editor, select the variable name **Syn_decay_time** at the bottom of the `synaptic_params.m` script. Evaluate the following code to open the script, if you haven't already: `open('synaptic_params')`.

Right-click on the selected line and from the context menu, choose **Requirements > Link to Selection in Simulink**. Test the new links by navigating from MATLAB to Simulink and back to MATLAB.



Create Traceability Link for Lines of Code Inside MATLAB Function Blocks

The `slvndemo_synaptic_transmission` model has a MATLAB Function block that you will use trace to parameter value sources to the `Synaptic strength` constant block. In the Simulink model, click on the `Synaptic strength` constant block or evaluate the following code to locate the block. To select it for linking, right-click the block and click **Requirements > Select for Linking with Simulink**.

```
rmidemo_callback('locate','slvndemo_synaptic_transmission/Synaptic strength');
```

Instead of linking the MATLAB function block itself, open the MATLAB code of this block and link specific lines. In the `slvndemo_synaptic_transmission` model, navigate to the `Synaptic current` block and double click it, or evaluate the following code.

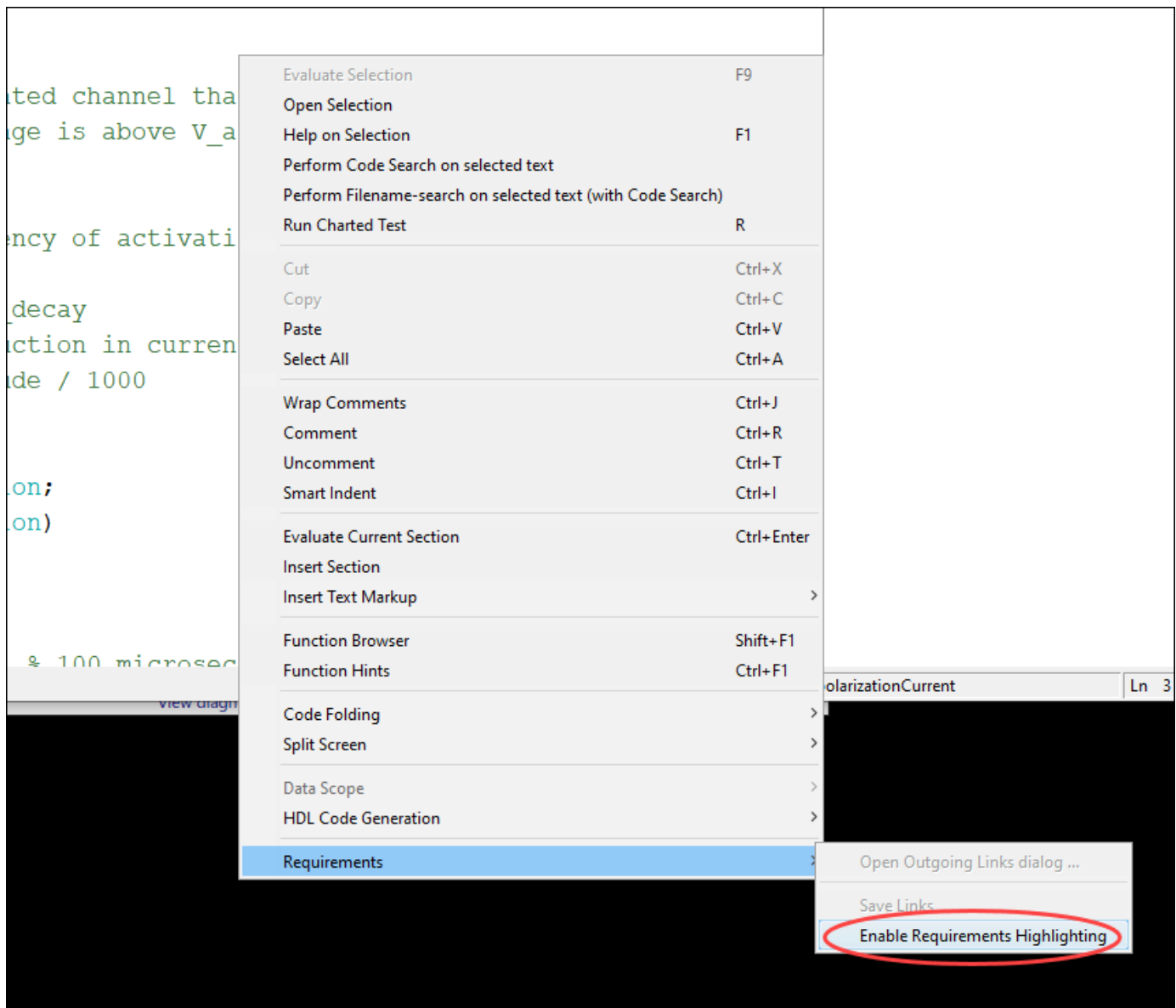
```
rmidemo_callback('emlshow','slvndemo_synaptic_transmission/Synaptic current');
```

In the MATLAB Function Block Editor, find the occurrence of **I_{amplitude}** on line 33 of the MATLAB Function block code. Highlight the line with the cursor to select it and then right-click it. Select **Requirements** from the context menu, then select **Link to Selection in Simulink**.


Create Traceability Link Between Lines in MATLAB Code and Microsoft Word

Until this point we only looked at linking between MATLAB and Simulink. Open the `Ion current calculation` MATLAB Function block that belongs to the `Sodium current calculation` subsystem of the referenced `slvndemo_neuron` model, or evaluate the following code:
`rmidemo_callback('emlshow','slvndemo_neuron/Sodium channel current/Ion current calculation')`.

Right-click in the MATLAB Function Block Editor and select **Requirements > Enable Requirements Highlighting** from the context menu to see which lines of code have requirements links.



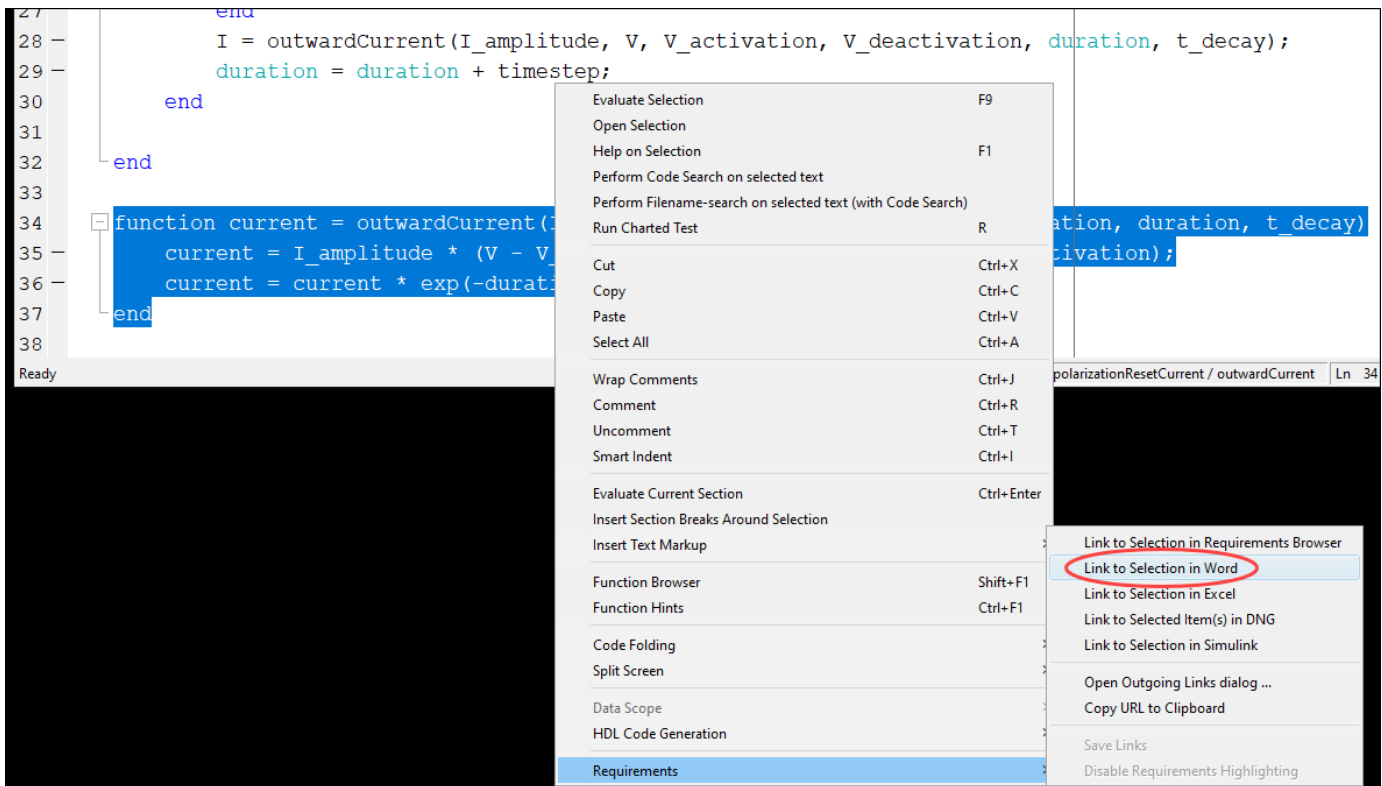
Right-click on a line of code that's been highlighted to indicate it has a requirements link. From the **Requirements** context menu, navigate to a numbered link at the top of the menu. The associated Word document opens to the associated text. You can also open the Word document and visually identify the links by looking for an object similar to the MATLAB icon. You can navigate to the linked code by Ctrl+clicking this linked object.

When the channels open (by detecting the **depolarization** in transmembrane voltage ) , they allow an inward flow of sodium ions, which changes the electrochemical gradient, which in turn produces a further rise in the membrane potential. This then causes more channels to open, producing a greater electric current, and so on. The process proceeds explosively until all of the

Evaluate the following code to open the Word document: `open('NeuralSpikeModeling.docx')`.

To create a similar link to the Ion current calculation Function in Potassium channel current subsystem, open the Word document and use the Find function (Ctrl+F) to find the phrase "**outward current of potassium ions.**" Select this phrase with your cursor in the Word document. Then, open the Ion current calculation MATLAB Function block from above, or evaluate the following code: `rmidemo_callback('emlshow','slvndemo_neuron/Potassium channel current/Ion current calculation')`.

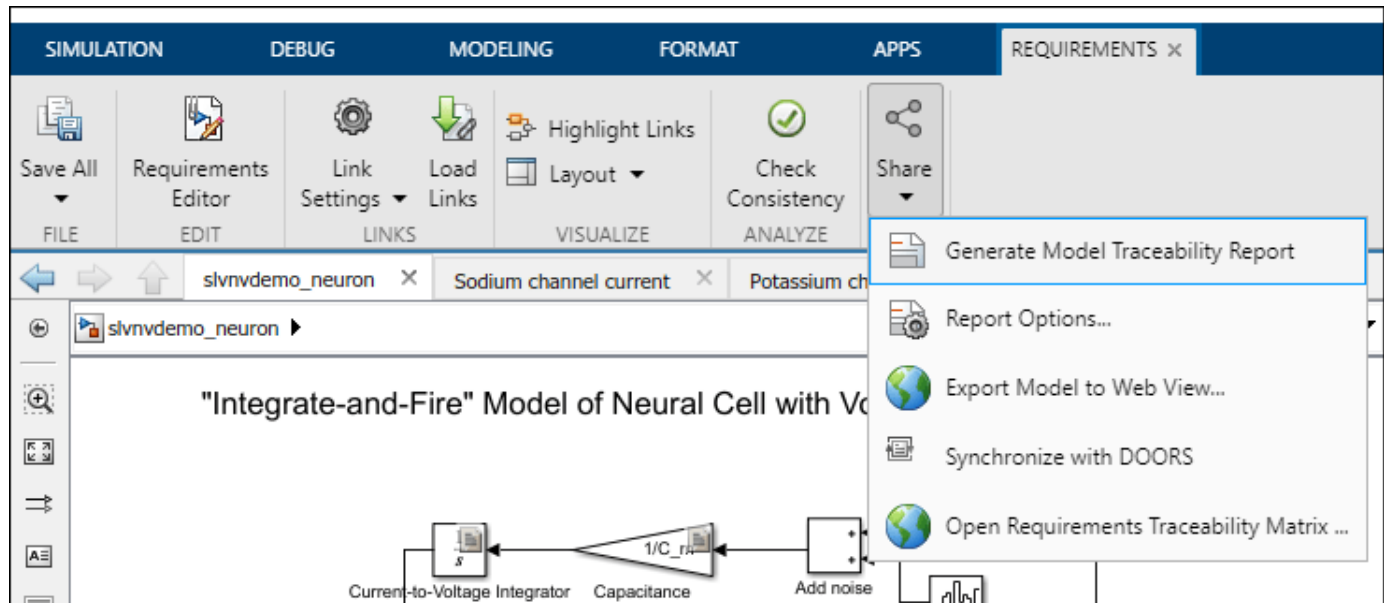
In the MATLAB Function Block Editor, select the implementation for **outwardCurrent** subfunction (lines 34-37). Right-click the selected lines and in the context menu, select **Requirements > Link to Selection in Word**. Minimize the Word document, then navigate from the newly highlighted lines at the bottom of the MATLAB code to the related description in Word. When the correct location is highlighted, use the MATLAB icon to navigate back to code.



Report Traceability Links

Traceability links associated with MATLAB code lines of MATLAB Function blocks are included in the **Requirements Traceability Report** generated for the parent Simulink model. In the **Requirements** tab, click **Share > Generate Model Traceability Report** in the `slvndemo_neuron` Simulink model. Evaluate the following code to open the model.

```
open('slvndemo_neuron.slx');
```



Note that the MATLAB Function block links information is present in the report, including quotations of linked MATLAB Code lines.

URL and Custom Traceability

- “Requirement Links and Link Types” on page 11-2
- “Custom Link Types” on page 11-8
- “Implement RMI Extension for Support of Custom Document Type” on page 11-17

Requirement Links and Link Types

Requirements Traceability Links

When you want to navigate from a Simulink model or from a region of MATLAB code to a location inside a requirements document, you can add requirements traceability links to the model or code.

Requirements traceability 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 Document Types” on page 6-8.)
- A designated location inside the requirements document, such as:
 - Bookmark
 - Anchor
 - ID
 - Page number
 - Line number
 - Cell range
 - Link target
 - Keywords that you define

Supported Model Objects for Requirements Linking

You can associate requirements links between the following types of Simulink model objects:

- Simulink block diagrams and subsystems
- Simulink blocks and annotations
- Simulink data dictionary entries
- Signal Builder signal groups
- Stateflow charts, subcharts, states, transitions, and boxes
- Stateflow functions
- Lines of MATLAB code
- Simulink Test Manager test cases

Links and Link Types

Requirements links are the data structures, managed by Simulink, that identify a specific location within a document. You get and set the links on a block using the `rmi` command.

Links and link types work together to perform navigation and manage requirements. The `doc` and `id` fields of a link uniquely identify the linked item in the external document. The RMI passes both of these values 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 “Outgoing Links Editor” on page 11-6, this string appears on the Document type drop-down list for a Simulink or Stateflow object.
IsFile	A Boolean property that indicates if the linked documents are files within the computer file system. If a document is a file: <ul style="list-style-type: none"> • The software uses the standard method for resolving the path. • In the Outgoing Links Editor, when you click Browse, the file selection dialog box opens.
Extensions	An array of file extensions. Use these file extensions as filter options in the Outgoing Links Editor when you click Browse . 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 Outgoing Links Editor Location drop-down list.
NavigateFcn	The MATLAB callback invoked when you click a link. The function has two input arguments: the document field and the ID field of the link: <pre>feval(LinkType.NavigateFcn, Link.document, Link.id)</pre>
ContentsFcn	The MATLAB callback invoked when you click the Document Index tab in the Outgoing Links Editor. 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 Document field contents. <p>The function returns three outputs:</p> <ul style="list-style-type: none"> • Labels • Depths • Locations
BrowseFcn	The MATLAB callback invoked when you click Browse in the Outgoing Links Editor. You do not need this function when the link type is a file. The function takes no input arguments and returns a single output argument that identifies the selected document.

Property	Description
CreateURLFcn	<p>The MATLAB callback that constructs a path name to the requirement. This function uses the document path or URL to create a specific requirement URL. The requirement URL is based on a location identifier specified in the third input argument. The input arguments are:</p> <ul style="list-style-type: none"> • Full path name to the requirements document • Info about creating a URL to the document (if applicable) • Location of the requirement in the document <p>This function returns a single output argument specified as a character vector. Use this argument when navigating to the requirement from the generated report.</p>
IsValidDocFcn	<p>The MATLAB callback invoked when you run a requirements consistency check. The function takes one input argument—the fully qualified name for the requirements document. It returns true if the document can be located; it returns false if the document cannot be found or the document name is invalid.</p>
IsValidIdFcn	<p>The MATLAB callback invoked when you run a requirements consistency check. This function takes two input arguments:</p> <ul style="list-style-type: none"> • Fully qualified name for the requirements document • Location of the requirement in the document <p>IsValidIdFcn returns true if it finds the requirement and false if it cannot find that requirement in the specified document.</p>
IsValidDescFcn	<p>The MATLAB callback invoked when you run a requirements consistency check. This function has three input arguments:</p> <ul style="list-style-type: none"> • Full path to the requirements document • Location of the requirement in the document • Requirement description label as stored in Simulink <p>IsValidDescFcn returns two outputs:</p> <ul style="list-style-type: none"> • True if the description matches the requirement, false otherwise. • The requirement label in the document, if not matched in Simulink.

Property	Description
DetailsFcn	<p>The MATLAB callback invoked when you generate the requirements report with the Include details from linked documents option. This function returns detailed content associated with the requirement and has three input arguments:</p> <ul style="list-style-type: none"> • Full path to the requirements document • Location of the requirement in the document • Level of details to include in report (Unused) <p>The DetailsFcn returns two outputs:</p> <ul style="list-style-type: none"> • Numeric array that describes the hierarchical relationship among the fragments in the cell array • Cell array of formatted fragments (paragraphs, tables, et al.) from the requirement
SelectionLinkFcn	<p>The MATLAB callback invoked when you use the selection-based linking menu option for this document type. This function has two input arguments:</p> <ul style="list-style-type: none"> • Handle to the model object that will have the requirement link • True if a navigation object is inserted into the requirements document, or false if no navigation object is inserted <p>SelectionLinkFcn returns the requirements link structure for the selected requirement.</p>
GetResultFcn	<p>The MATLAB callback invoked when you link external test cases with the requirements to the custom link type file. It is used in the custom link type file and fetches external results to integrate with verification statuses.</p> <p>This function has one input argument:</p> <ul style="list-style-type: none"> • <code>link</code>: This is a <code>slreq.Link</code> object. The function identifies the source and destination of the link. <p>The function returns a single output argument, <code>result</code> which is specified as a <code>struct</code> with the following fields:</p> <ul style="list-style-type: none"> • <code>status</code> (Required): This is a value from <code>slreq.verification.Status</code> (Pass, Fail, Stale, or Unknown) • <code>timestamp</code> (Optional): Skip this field or mark <code>NaN</code> to avoid stale result detection. • <code>info</code> (Optional): This should be a character, vector or string. The value of <code>info</code> is printed as a diagnostic on the tooltip of the status. • <code>error</code> (Optional): This should be a character, vector or string. The value of <code>error</code> is printed as a diagnostic on the tooltip of the status. If provided, it takes precedence over the <code>info</code> field.

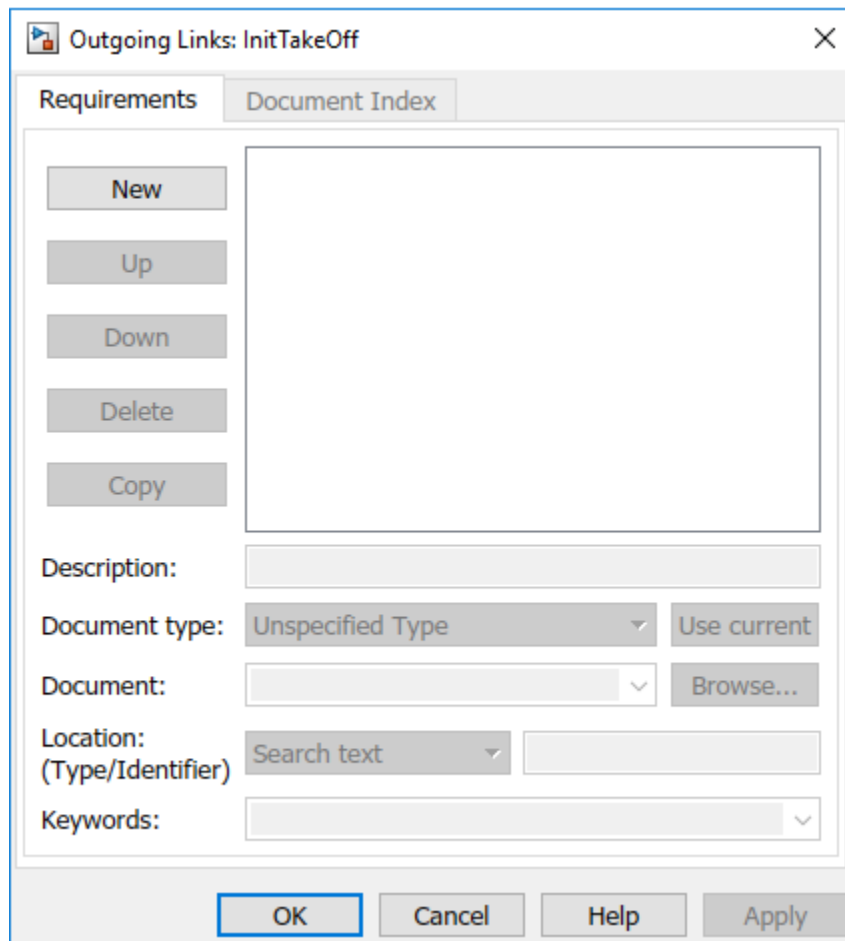
Outgoing Links Editor

Manage Requirements Traceability Links Using the Outgoing Links Editor

You can create, edit, and delete requirements traceability links using the Outgoing Links Editor. To open the Outgoing Links Editor:

- in the Simulink Editor, right-click on a model object that has a requirements traceability link. From the context menu, select **Requirements > Open Outgoing Links dialog**.
- in the MATLAB Editor, right-click inside a region of code that has a requirements traceability link. From the context menu, select **Requirements > Open Outgoing Links dialog**.

The Outgoing Links Editor opens, as shown below.



In the Outgoing Links Editor, you can:

- Create requirements links from one or more Simulink model objects or MATLAB code lines.
- Customize information about requirements links, including specifying user keywords to filter requirements highlighting and reporting.
- Delete existing requirements links.

- Modify the stored order of requirements to control the order of labels in context menus for linked objects.

Requirements Tab

On the **Requirements** tab, you specify detailed information about the link, including:

- Description of the requirement (up to 255 words). If you create a link using the document index, *unless* a description already exists, the name of the index location becomes the description for the link .
- Path name to the requirements document.
- Document type (Microsoft Word, Microsoft Excel, IBM Rational DOORS, MuPAD®, HTML, text file, etc.).
- Location of the requirement (search text, named location, or page or item number).
- User-specified keyword or keyword.

Document Index Tab

The **Document Index** tab is available only if you have specified a file in the **Document** field on the **Requirements** tab that supports indexing. On the **Document Index** tab, the RMI generates a list of locations in the specified requirements document for the following types of requirements documents:

- Microsoft Word
- IBM Rational DOORS
- HTML files
- MuPAD

Note The RMI cannot create document indexes for PDF files.

From the document index, select the desired requirement from the document index and click **OK**. *Unless* a description already exists, the name of the index location becomes the description for the link.

If you make any changes to your requirements document, to load any newly created locations into the document index, you must click **Refresh**. During a MATLAB session, the RMI does not reload the document index unless you click the **Refresh** button.

Custom Link Types

Note Requirements Management Interface (RMI) provides legacy functionality. Starting in R2022a, use `slreq.refreshCustomizations`. For more information, see “Define Custom Requirement and Link Types by Using `sl_customization` Files” on page 3-42.

Create a Custom Requirements Link Type

In this example, you implement a custom link type to a hypothetical document type, a text file with the extension `.abc`. Within this 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 (`"`).

You will create a document index listing 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.

For this example, the file is `rmicustabcinterface.m`, containing the function, `rmicustabcinterface`, that implements the ABC files shipping with your installation.

- 2 To view this function, at the MATLAB prompt, type:

```
edit rmicustabcinterface
```

The file `rmicustabcinterface.m` opens in the MATLAB Editor. The content of the file is:

```
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 1984-2010 The MathWorks, Inc.

% 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 required

% 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

function openFileToLine(fileName, lineNum)
if lineNum > 0
    if matlab.desktop.editor.isEditorAvailable
        matlab.desktop.editor.openAndGoToLine(fileName, lineNum);
    end
else
    edit(fileName);
end

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

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{:}];
items = sort(items);
items = strcat('@',items);

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

depths = [];

```

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

```
rmi register rmicustabcinterface
```

The ABC file type appears on the "Outgoing Links Editor" on page 11-6 drop-down list of document types.

- 4 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 *matlabroot\toolbox\slvnx\rmidemos\demo_req_1.abc*. *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 of "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 of "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 of "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 of "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 of "Glide Slope Coupled"

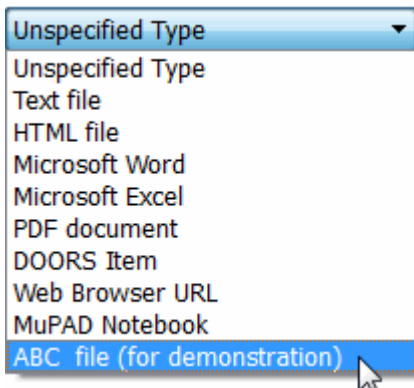
- 5 Open the `aero_dap3dof` model. At the MATLAB command line, enter:
- ```
openExample('simulink_aerospace/DevelopingTheApolloLunarModuleDigitalAutopilotExample')
```

Close the Apollo Lunar Module Descent Animation.

- 6 In the `aero_dap3dof` model, right-click the Reaction Jet Control subsystem and select **Requirements > Open Outgoing Links dialog**.

The Outgoing Links Editor opens.

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



- 8 Set **Document type** to ABC file (for demonstration) and browse to the `matlabroot\toolbox\slvnr\rmidemos\demo_req_1.abc` file. The browser shows only the files with the `.abc` extension.
- 9 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:

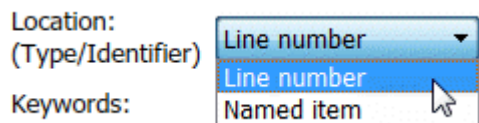
- `>` — Line number in a file
- `@` — Named item, such as a bookmark, function, or HTML anchor

---

**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).



- 10 Select **Line number**. Enter the number 26, which corresponds with the line number for the Altitude Hold requirement in `demo_req_1.abc`.
- 11 In the **Description** field, enter Altitude Hold, to identify the requirement by name.
- 12 Click **Apply**.

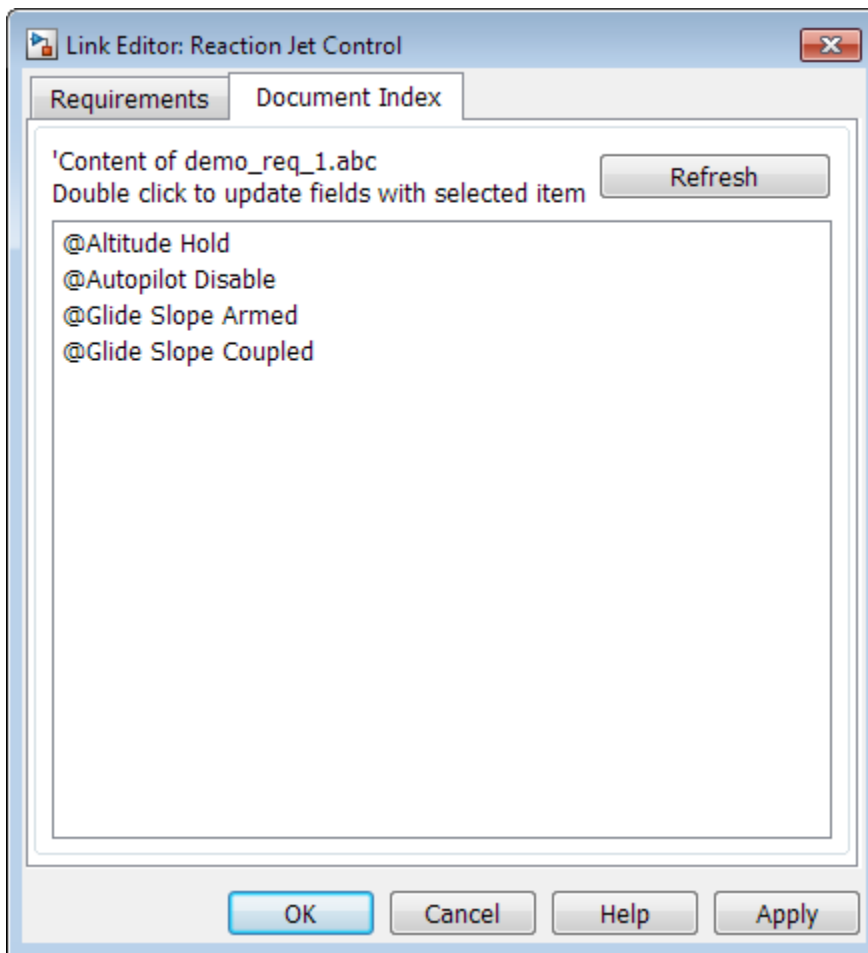
- 13 Verify that the Altitude Hold requirement links to the Reaction Jet Control subsystem. Right-click the subsystem and select **Requirements > 1. "Altitude Hold"**.

### Create 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. 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 this ABC file, the `ContentsFcn` function in `rmicustabcinterface.m` extracts the requirements names and inserts `@` before each name.

For the `demo_req_1.abc` file, in the **Outgoing Links: Reaction Jet Control** dialog box, click the **Document Index** tab. The `ContentsFcn` function generates the document index automatically.



## Implement Custom Link Types

To implement a custom link type:

- 1 Create a MATLAB function file based on the custom link type template, as described in "Custom Link Type Functions" on page 11-14.

- 2 Customize the custom link type file to specify the link type properties and custom callback functions required for the custom link type, as described in “Link Type Properties” on page 11-3.
- 3 Register the custom link type using the `rmi` command 'register' option, as described in “Custom Link Type Registration” on page 11-15.

## 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 from your model to these types of documents.

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

The custom link type API allows you to define MATLAB functions that enable linking between your Simulink model and your custom requirements document type. These functions also enable new link creation and navigation between the model and documents.

For example, navigation involves opening a requirements document and finding the specific requirement record. When you click your custom link in the content menu of a linked object in the model, Simulink uses your custom link type navigation function to open the document and highlight the target requirement based on the implementation provided. The navigation function you implement uses the available API to communicate with your requirements storage application.

Typically, MATLAB runs an operating system shell command or uses ActiveX communication for sending navigation requests to external applications.

Alternatively, if your requirements are stored as custom variants of text or HTML files, you can use the built-in editor or Web browser to open the requirements document.

## Custom Link Type Functions

To create a MATLAB function file, start with the custom link type template, located in:

```
matlabroot\toolbox\slrequirements\linktype_examples\linktype_TEMPLATE.m
```

Your custom link type function:

- Must exist on the MATLAB path with a unique function and file name.
- Cannot require input arguments.
- Must return a single output argument that is an instance of the requirements link type class.

To view similar files for the built-in link types, see the following files in *matlabroot*\toolbox\slrequirements\linktype\_examples\:

```
linktype_rmi_doors.m
linktype_rmi_excel.m
linktype_rmi_html.m
linktype_rmi_text.m
```

## Custom Link Type Registration

Register your custom link type by passing the name of the MATLAB function file to the `rmi` command as follows:

```
rmi register mytargetfilename
```

Once you register a link type, it appears in the “Outgoing Links Editor” on page 11-6 as an entry in the **Document type** drop-down list. A file in your preference folder contains the list of registered link types, so the custom link type is loaded each time you run MATLAB.

When you create links using custom link types, the software saves the registration name and the other link properties specified in the function file. When you attempt to navigate to such 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.

You can remove a link type with the following MATLAB command:

```
rmi unregister mytargetfilename
```

## Custom Link Type Synchronization

After you implement custom link types for RMI that allow you to establish links from Simulink objects to requirements in your requirements management application (RM application), you can implement synchronization of the links between the RM application and Simulink using Requirements Toolbox functions. Links can then be reviewed and managed in your RM application environment, while changes made are propagated to Simulink.

You first create the surrogate objects in the RM application to represent Simulink objects of interest. You then automate the process of establishing traceability links between these surrogate objects and other items stored in the RM application, to match links that exist on the Simulink side. After modifying or creating new associations in the RM application, you can propagate the changes back to Simulink. You use Requirements Toolbox to implement synchronization of links for custom requirements documents. However, this functionality is dependent upon the automation and inter-process communication APIs available in your RM application. You use the following Requirements Toolbox functions to implement synchronization of links between RM applications and Simulink.

To get a complete list of Simulink objects that may be considered for inclusion in the surrogate module:

```
[objHs, parentIdx, isSf, objSIDs] = rmi...
('getObjectsInModel', modelName);
```

This command returns:

- `objHs`, a complete list of numeric handles
- `objSIDs`, a complete list of corresponding session-independent Simulink IDs
- `isSf`, a logical array that indicates which list positions correspond to which Stateflow objects
- `parentIdx`, an array of indices that provides model hierarchy information

When creating surrogate objects in your RM application, you will need to store `objSIDs` values - not `objHs` values - because `objHs` values are not persistent between Simulink sessions.

To get Simulink object Name and Type information that you store on the RM application side:

```
[objName, objType] = rmi('getObjLabel', sLObjectHandle);
```

To query links for a Simulink object, specified by either numeric handle or SID:

```
linkInfo = rmi('getLinks', sLObjectHandle)
linkInfo = rmi('getLinks', sigBuildertHandle, m)
% Signal Builder group "m" use case.
linkInfo = rmi('getLinks', [modelName objSIDs{i}]);
```

`linkInfo` is a MATLAB structure that contains link attributes. See the `rmi` function reference page for more details.

After you retrieve the updated link information from your RM application, populate the fields of `linkData` with the updated values, and propagate the changes to Simulink:

```
rmi('setLinks', sLObjectHandle, linkData)
```

For an example MATLAB script implementing synchronization with a Microsoft Excel Workbook, see the following:

```
edit([matlabroot '/toolbox/slrequirements/...
linktype_examples/slSurrogateInExcel.m'])
```

You can run this MATLAB script on the example model `slvnvdemo_fuelsys_officereq` to generate the Excel workbook surrogate for the model.



# Implement RMI Extension for Support of Custom Document Type

Requirements Management Interface (RMI) provides tools for creating and reviewing links between model-based design elements and requirements documents. RMI provides built-in support for many document types. Additionally, you can implement custom link-type extensions to enable linking to other document types. This example illustrates implementation of RMI extension for linking with Microsoft® PowerPoint® presentations.

## Files to Use with This Example

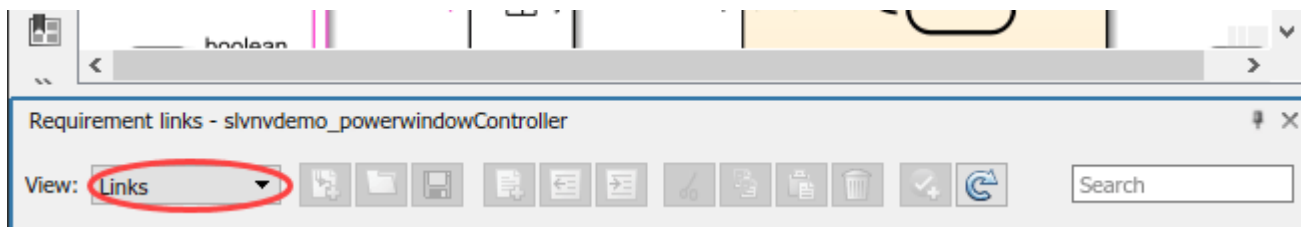
For the purposes of this example tutorial, you will link objects in the `slvnvdemo_powerwindowController.slx` model with slides in the `powerwindowController.pptx` PowerPoint presentation. Open the Simulink® model `slvnvdemo_powerwindowController.slx`.

```
open_system('slvnvdemo_powerwindowController');
```

## Set Up Requirements Manager to Work with Links

- 1 In the **Apps** tab, open **Requirements Manager**.
- 2 In the **Requirements** tab, ensure **Layout > Requirements Browser** is selected.
- 3 In the **Requirements Browser**, in the **View** drop-down menu, select **Links**.

In this example, you will work exclusively in the **Requirements** tab and any references to toolbar buttons are in this tab.



## Store Links Externally

In the `slvnvdemo_powerwindowController` model, configure the settings to store links externally. In the **Requirements** tab, select **Link Settings > Default Link Storage**. This will open the **Requirements Settings** dialog box. Under the heading **Default storage mode for traceability data** select **Store externally (in a separate \*.slmx file)**. Alternatively, evaluate the following code.

```
rmipref('StoreDataExternally', true);
```

## Installed Link Type Definition Files

Depending on the application you use for your custom-type documents, you can implement basic support, including link creation via the **Outgoing Links** dialog box and link navigation via context menu shortcuts, or you may choose to implement a more feature-rich support with selection linking via context menu shortcuts, consistency checking, etc.

In this tutorial you will use a Custom Link Type definition which was created from scratch. To find out more about the Custom Link Type extension API, please refer to the included `linktype_TEMPLATE.m` by evaluating the following:

```
edit([matlabroot, '/toolbox/slrequirements/linktype_examples/linktype_TEMPLATE.m'])
```

You can also review the actual linktype definition files used by the released product. For an example, refer to the minimal Text File link type by evaluating the following:

```
edit([matlabroot, '/toolbox/slrequirements/linktype_examples/linktype_rmi_text.m'])
```

You can also refer to the more advanced Microsoft Excel® Workbook link type:

```
edit([matlabroot, '/toolbox/slrequirements/linktype_examples/linktype_rmi_excel.m'])
```

### Create and Register a Stubbed Link Type File

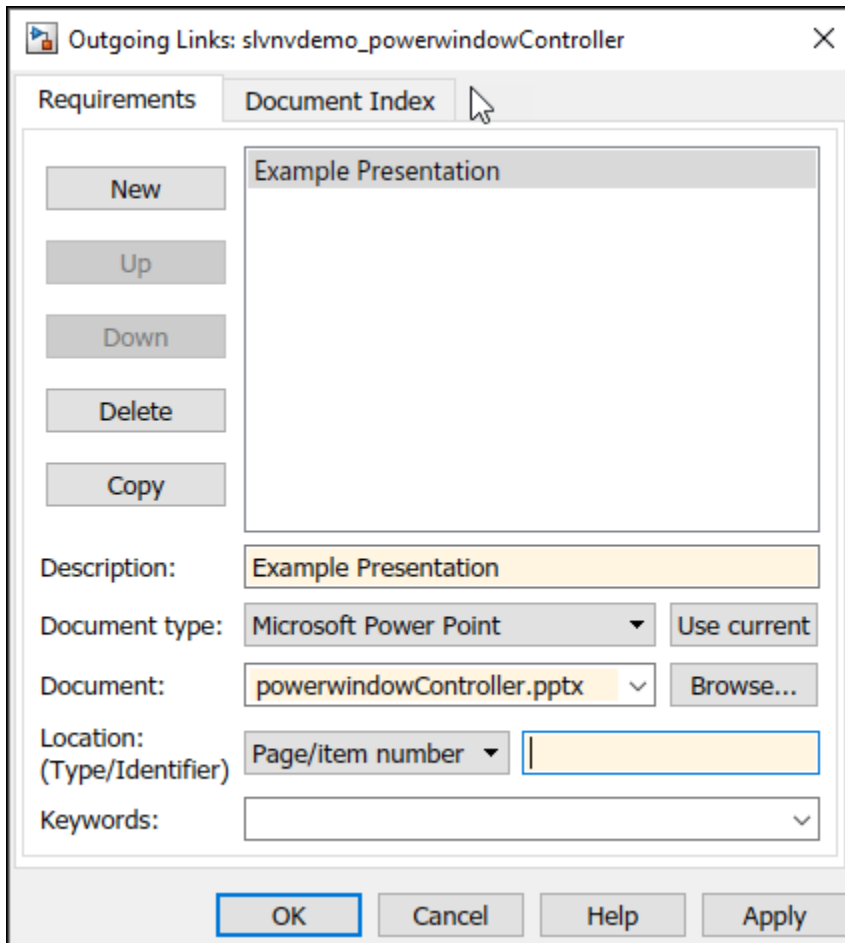
The file `rmidemo_pp_linktype.m` in the current working directory contains link type information for the RMI to work with Microsoft PowerPoint files. Register the link type with the RMI by evaluating the following.

```
rmi('register', 'rmidemo_pp_linktype')
```

This instructs RMI to recognize the filename extensions `.ppt` and `.pptx` as supported files and to use the methods defined here for RMI functionality.

### Create the First Link

- Right-click the background of the `slvndemo_powerwindowController` example model. In the context menu, select **Requirements at This Level > Open Outgoing Links Dialog...** to bring up the Outgoing Links dialog box.
- Click **New** to create a new link.
- Expand the **Document type** drop-down list. Select **Microsoft PowerPoint** at the bottom of the list.
- Use the **Browse** button to locate `powerwindowController.pptx`.
- Enter a **Description** label, like *Example Presentation*.
- Click **OK** to save the new link and close the dialog.



Alternatively, you can evaluate the following code to create the link. This code fills in the link destination information first, then uses the `rmi` function to create links.

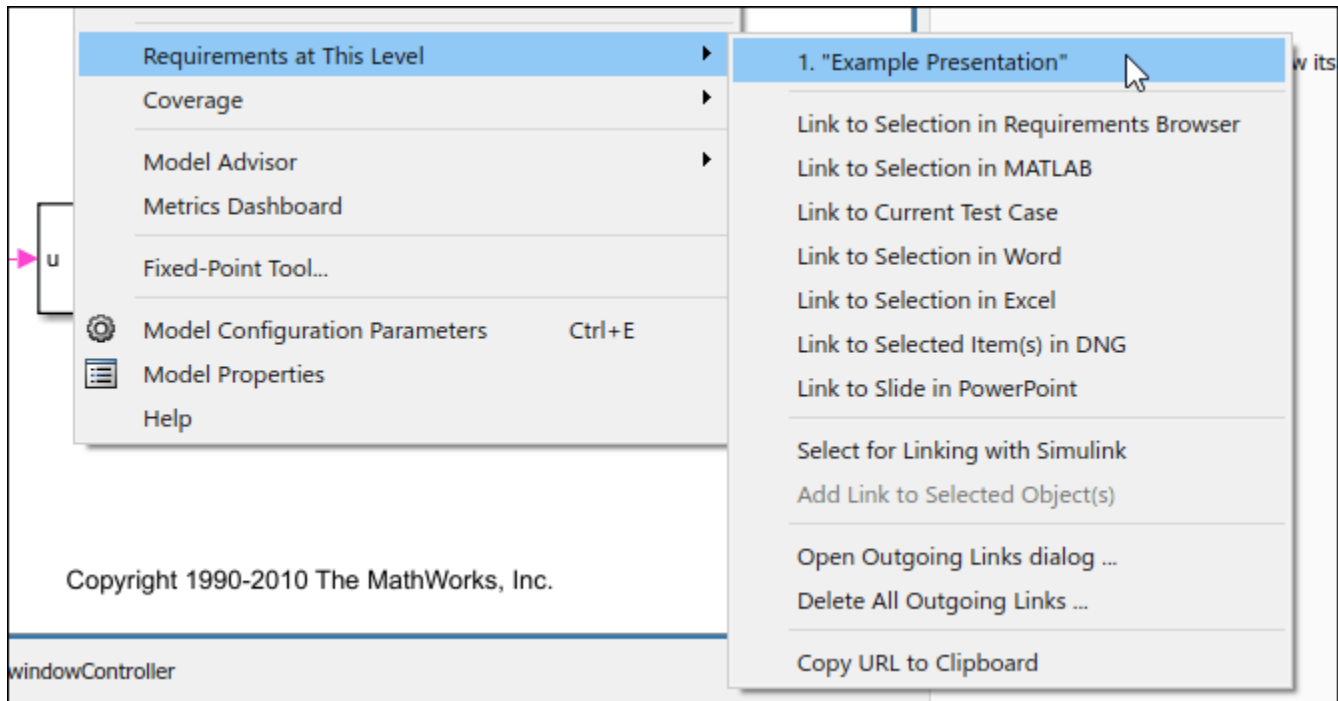
```
firstReq = rmi('createempty');
firstReq.reqsys = 'rmidemo_pp_linktype';
firstReq.doc = 'powerwindowController.pptx';
firstReq.description = 'Example presentation';
rmi('set', 'slnvdemo_powerwindowController', firstReq);
```

### Navigation to the Document

Navigation to the PowerPoint document is provided with functions in the `rmidemo_pp_linktype.m` file. Implementation of this and all other methods requires detailed knowledge of the APIs available in the application that manages the requirements documents. For this Microsoft PowerPoint example you will use COM API. You will use the `actxserver` command in MATLAB® to create a connection with the PowerPoint application. Then, you will use calls like `Application.Presentations.Open(FILENAME)` to manipulate the PowerPoint application via the `rmidemo_pp_linktype` methods. See Microsoft's Developer Reference pages for information on which PowerPoint Objects and Methods are available via COM API.

The `rmidemo_pp_linktype.m` file contains functions to find the correct `.pptx` file.

Return to the Simulink model for `slvndemo_powerwindowController`. Right-click the Simulink diagram background and navigate to **Requirements at This Level** again from the context menu. Notice the new navigation shortcut at the top of the submenu. When you click this new shortcut, MATLAB opens the PowerPoint file.



You can navigate to the link the same way as before, or by evaluating the following:

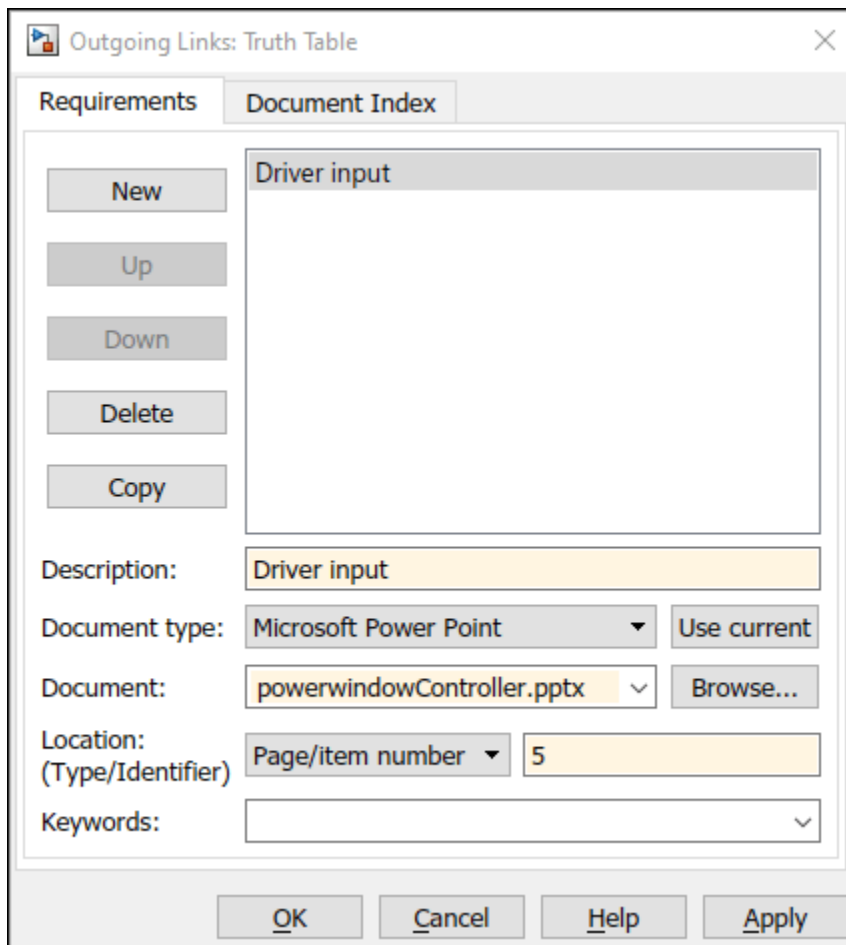
```
rmi('view','slvndemo_powerwindowController', 1)
```

### Implement Navigation to a Given Slide Number

Suppose you want to link the Truth Table block that connects to the driver input of the control subsystem block to the corresponding slide number 5 in the PowerPoint presentation. Navigate to the Truth Table block or evaluate the following code.

```
rmdemo_callback('locate','slvndemo_powerwindowController/Truth Table')
```

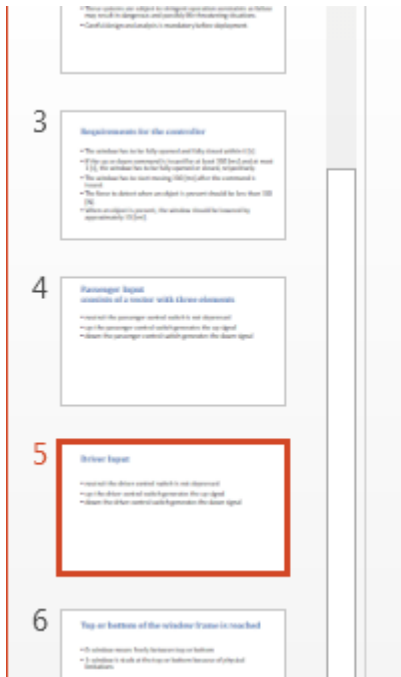
- Right-click the block, select **Requirements > Open Outgoing Links Dialog...** to bring up the Outgoing Links dialog box.
- Click **New** to create a new link.
- Specify the document type and filename as before.
- Enter *Driver input* into the **Description** field.
- Enter *5* into the **Location/Identifier** input field.
- Click **OK** to save the new link.



If you navigate this link from the Simulink diagram, the document will open as before, but it will now scroll down to 5th slide. The helper `goToSlide()` method along with code in the `NavigateFcn()` function open the correct slide.

```
function goToSlide(hDoc, slideNum)
 disp(['Opening slide #' num2str(slideNum)]);
 hDoc.Slides.Item(slideNum).Select;
end
```

Navigate to the link by selecting the Truth Table block, right-clicking and selecting **Requirements > 1. "Driver input"**. The PowerPoint presentation window should scroll down to the 5th slide.



## Driver Input

- neutral: the driver co
- up: the driver control
- down: the driver cont

Alternatively, create the link by evaluating the following code. This code fills in the link destination information first, then uses the `rmi` function to create links.

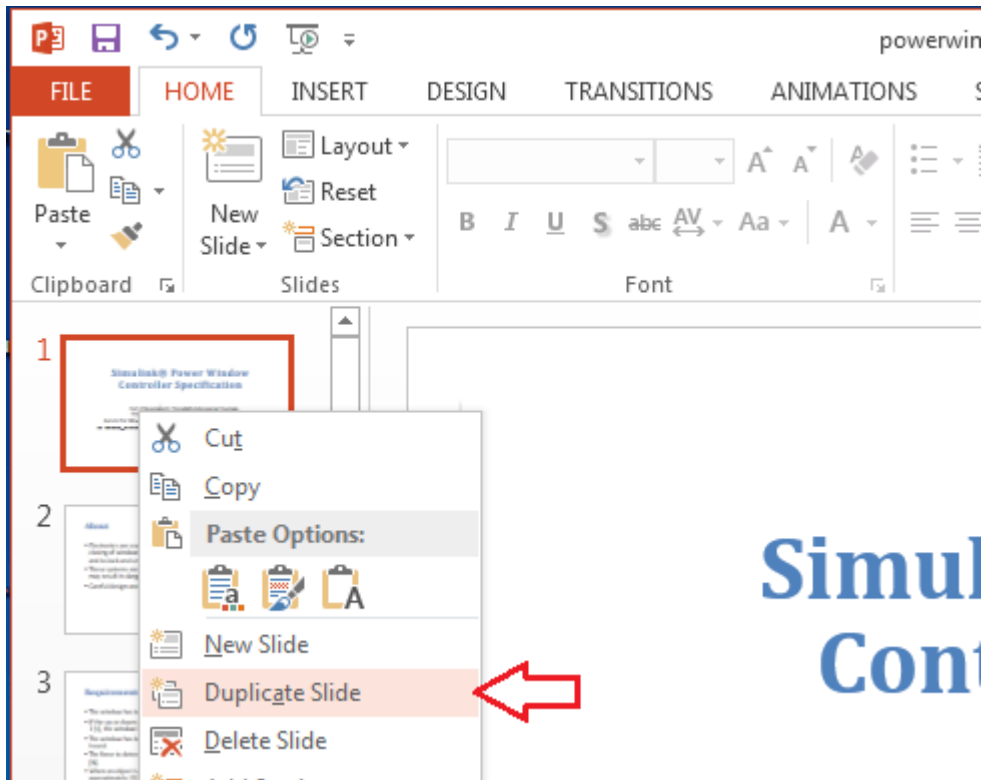
```
secondReq = rmi('createempty');
secondReq.reqsys = 'rmidemo_pp_linktype';
secondReq.doc = 'powerwindowController.pptx';
secondReq.description = 'Driver input';
secondReq.id = '#5';
rmi('set', 'slvndemo_powerwindowController/Truth Table', secondReq);
```

You can navigate to the link the same way as before, or by evaluating the following:

```
rmi('view', 'slvndemo_powerwindowController/Truth Table', 1)
```

### Linking and Navigation to Slide ID

Linking to a stored slide number can be problematic: links may get stale when you modify the document. For example, duplicate the first slide in our presentation:



Now all the other slides shift down by one. Navigation from the Driver Input Truth Table block will bring up the wrong slide. You need to use a different location type, other than Page/Item number.

- Select the same Truth Table block which connects to the driver input of the control subsystem. The following code navigates to the Truth Table block.

```
rmdemo_callback('locate', 'slvndemo_powerwindowController/Truth Table')
```

- Right-click the block, select **Requirements > Open Outgoing Links Dialog...** to bring up the Outgoing Links dialog box.
- Click **New** to create a new link.
- Select **Named Item** from the **Location (Type/Identifier)** drop-down list.
- Enter **260** into the **Location** input field.
- Click **OK** to save the modified link.

"260" is a persistent ID for the Driver Input slide (more on this below).

Now, after this change, navigation from the Driver Input Truth Table block will bring up the correct slide, even after its order number changes.

Unfortunately, one cannot see slide IDs in the PowerPoint application UI. To find out the ID for the 5th slide, you can use the COM API:

```
>> hApp = actxGetRunningServer('powerpoint.application');
>> hDoc = hApp.ActivePresentation;
>> hDoc.Slides.Item(5).SlideID
ans =
 260
```

More user-friendly solutions to this problem are covered in the sections below.

Alternatively, you can create the link using the following code. This code fills in the link destination information first, then uses the `rmi` function to create links.

```
betterLink = rmi('createempty');
betterLink.reqsys = 'rmidemo_pp_linktype';
betterLink.doc = 'powerwindowController.pptx';
betterLink.description = 'Driver input - better link';
betterLink.id = '@260';
rmi('set', 'slvnvdemo_powerwindowController/Truth Table', betterLink);
```

You can navigate to the link destination the same way as before, or evaluate the following:  
`rmi('view', 'slvnvdemo_powerwindowController/Truth Table', 1)`

### Linking Using Document Index Tab

As shown above, you can create persistent links that do not become stale after slides in linked presentation are re-ordered, but you do not have easy access to persistent *SlideID* values in PowerPoint. One possible solution is to select a desired slide in the Document Index tab of the Outgoing Links dialog. The content of the Document Index tab is controlled by the `ContentsFcn()` method in the linktype definition file. you can provide implementation for this method, such that the persistent *SlideID* value is stored by RMI when creating a link, instead of the volatile *SlideNumber* value.

The `ContentsFcn()` methods returns three arrays:

- `labels` to use for Document Index list items and navigation shortcuts
- `depths` to indicate the hierarchical relationship of listed items (unused in this example)
- `locations` to use for stored unique IDs

The `ContentsFcn()` implementation relies on the following PowerPoint API call to populate *location* values:

```
hDoc.Slides.Item(k).SlideID
```

This ensures persistent navigation information even after slide order changes. Note that you use `@` as a prefix for `locations` values, to indicate that the number that follows stores the *Named Item* location value instead of slide (page) number location value.

Use the **Document Index** tab in the **Outgoing Link Editor** to create a link.

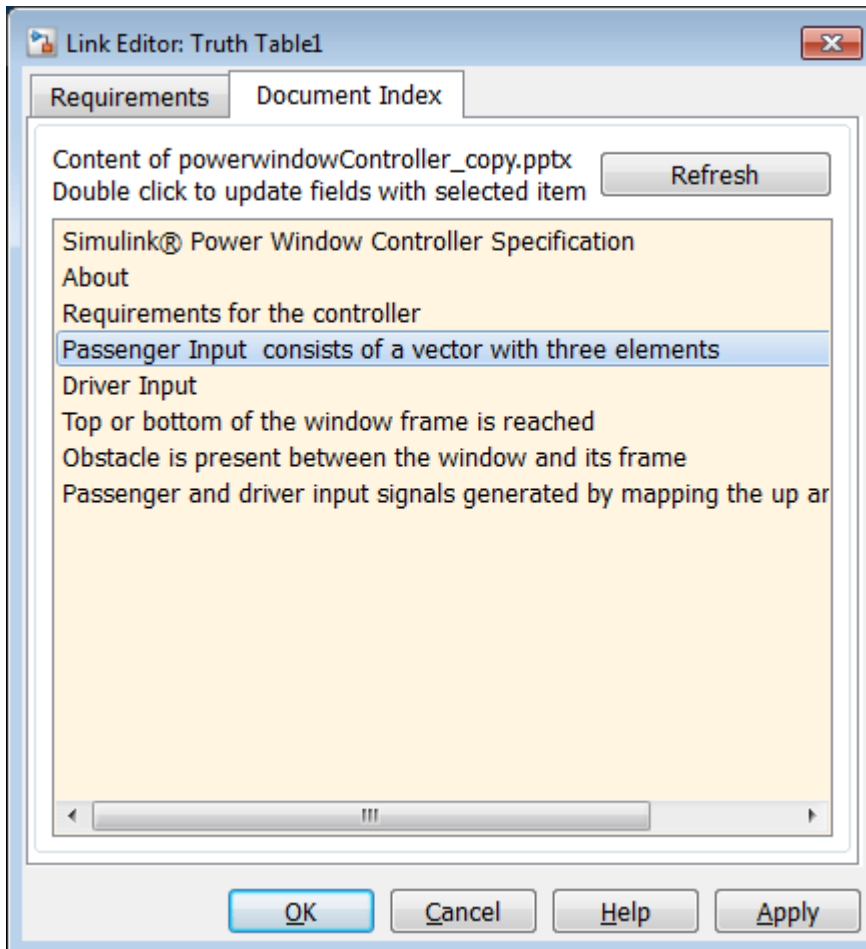
- Navigate to the `Truth Table1` block which connects to the `passenger` input of the `control subsystem` block. The following code navigates to the `Truth Table1` block.

```
rmidemo_callback('locate', 'slvnvdemo_powerwindowController/Truth Table1')
```

- Right-click the block, select **Requirements > Open Outgoing Links Dialog...** to bring up the Outgoing Links dialog box.
- Click **New** to create a new link.
- Specify `Microsoft PowerPoint` as the **Document type**.
- Specify `powerwindowController.pptx` as the **Document** from the **Browse** menu.
- Leave the **Description** input.



- Instead of specifying **Location** manually, switch to the **Document Index** tab, locate the line that corresponds to Passenger Inputs slide, and double-click the line.
- Notice that the remaining input fields are automatically filled with the correct information.
- Click **OK** to save the new link.



Navigate to the link by right-clicking the Truth Table1 block and selecting **Requirements > 1."Passenger Input consists of a vector with three elements in powerwindowController.pptx"**. This link should work correctly even after slides are shifted or re-ordered.

Alternatively you can create the link by evaluating the following code. The link ID is created the same way as in the previous section, where a persistent ID is set. This code fills in the link destination information first, then uses the `rmi` function to create links.

```
indexLink = rmi('createempty');
indexLink.regsys = 'rmidemo_pp_linktype';
indexLink.doc = 'powerwindowController.pptx';
indexLink.description = 'Passenger input - linked via Index tab';
indexLink.id = '@259';
rmi('set', 'slvnvdemo_powerwindowController/Truth Table1', indexLink);
```

Navigate to the link the same way as above, or evaluate the following:

```
rmi('view','slvndemo_powerwindowController/Truth Table1', 1)
```

### **Link to the Current Slide in PowerPoint**

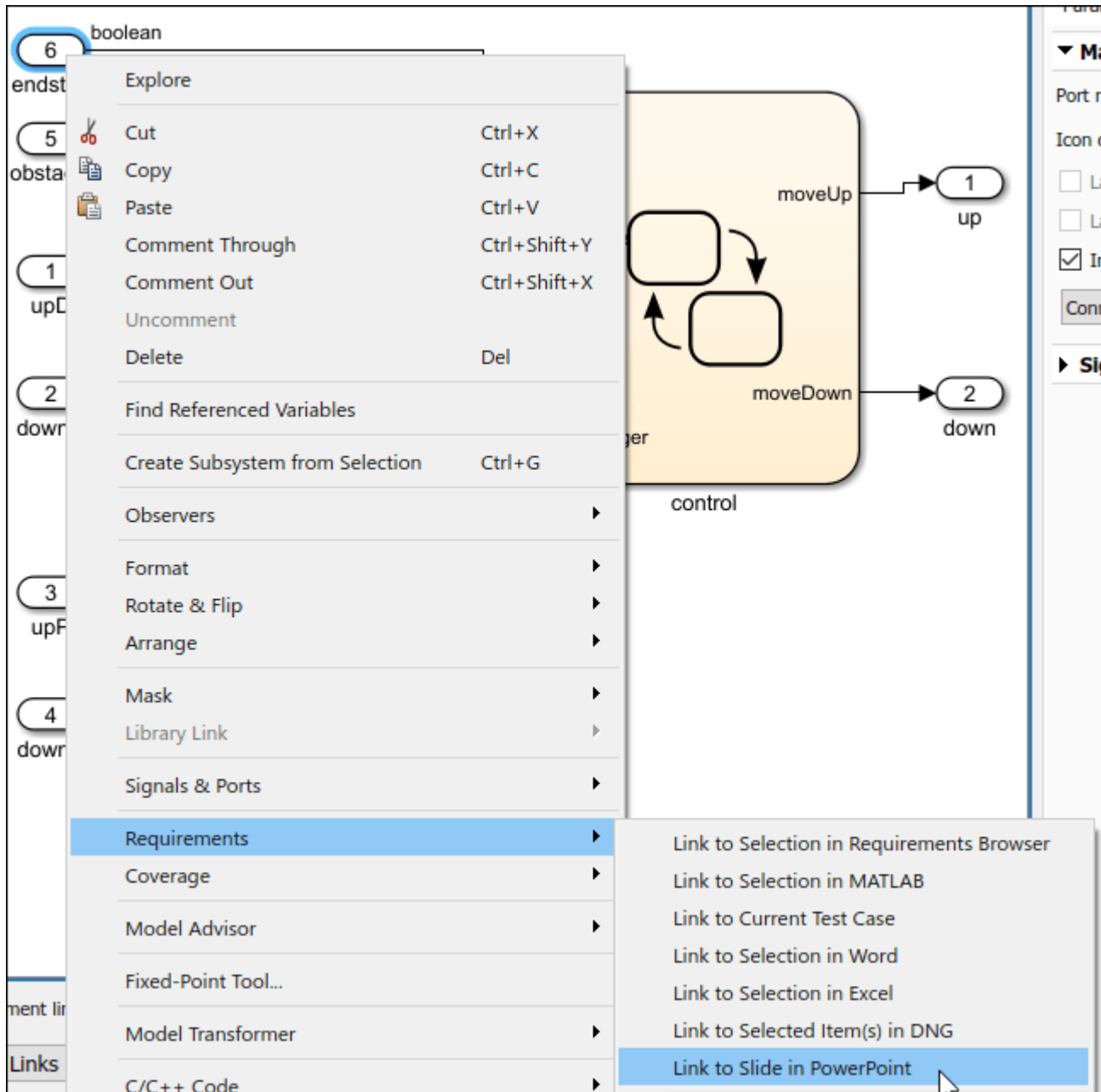
An even better way to support robust persistent links is via Selection Linking Shortcuts. The RMI API allows you to define the `SelectionLinkFcn()` function for linking with the current object in the current document. In the next step of our tutorial, you will automate linking to the current slide in the currently open PowerPoint presentation.

The **Requirements** section of the context menus display a shortcut for linking with the current slide in PowerPoint.

- In your copy of the PowerPoint presentation example, navigate to slide 6 titled Top or bottom of the window frame is reached.
- In the Simulink diagram, right-click the endstop block.

```
rmidemo_callback('locate','slvndemo_powerwindowController/endstop')
```

- Right-click the block and select **Requirements > Link to Slide in PowerPoint** from the context menu.



The RMI will automatically create a link to the *SlideID* that corresponds to the current location in the active presentation. The RMI will try to use the header text of the target slide as a label for the new link.

To navigate to the link, right-click the `endstop` block again and select **Requirements > 1."Top or bottom of the window frame is rea...**". The PowerPoint program should open to the correct slide.

Alternatively, you can create the link using the following code. The link ID is created the same way as in the previous section, where a persistent ID is set. This code fills in the link destination information first, then uses the `rmi` function to create links.

```
selectionLink = rmi('createempty');
selectionLink.reqsys = 'rmdemo_pp_linktype';
selectionLink.doc = 'powerwindowController.pptx';
selectionLink.description = 'Endstop signal - selection link';
selectionLink.id = '@261';
rmi('set', 'slvndemo_powerwindowController/endstop', selectionLink);
```

You can navigate to the link the same way as above, or you can evaluate the following:  
`rmi('view', 'slvndemo_powerwindowController/endstop', 1)`

### Create Bidirectional Links

As a final step of this tutorial you will extend the `SelectionLinkFcn()` function to optionally insert a hyperlink in the current slide, for navigation from PowerPoint to the linked object in Simulink.

Your PowerPoint link type allows automated insertion of Simulink navigation controls into linked slides, when you use **Link to Slide in PowerPoint** shortcut in the context menu for Simulink objects. To activate this feature, in the Simulink model select the **Requirements** tab. Then select **Link Settings > Linking Options**. Alternatively, evaluating the following code will open this dialog box:  
`rmi_settings_dlg`.

Under the **When creating selection-based links** heading, make sure that **Modify destination for bidirectional linking** is checked. Alternatively, the following code will set these settings.

```
origMcState = rmipref('UnsecureHttpRequests', true);
origTwoWayPref = rmipref('BiDirectionalLinking', true);
```

Beginning in R2019a, MATLAB's embedded HTTP service is activated on a secure port 31515, but not on an unsecure port 31415. Because our navigation URLs cannot use the secure port without certificate installation, you should also select the **Enable external connectivity at MATLAB startup** checkbox at the bottom of this tab.

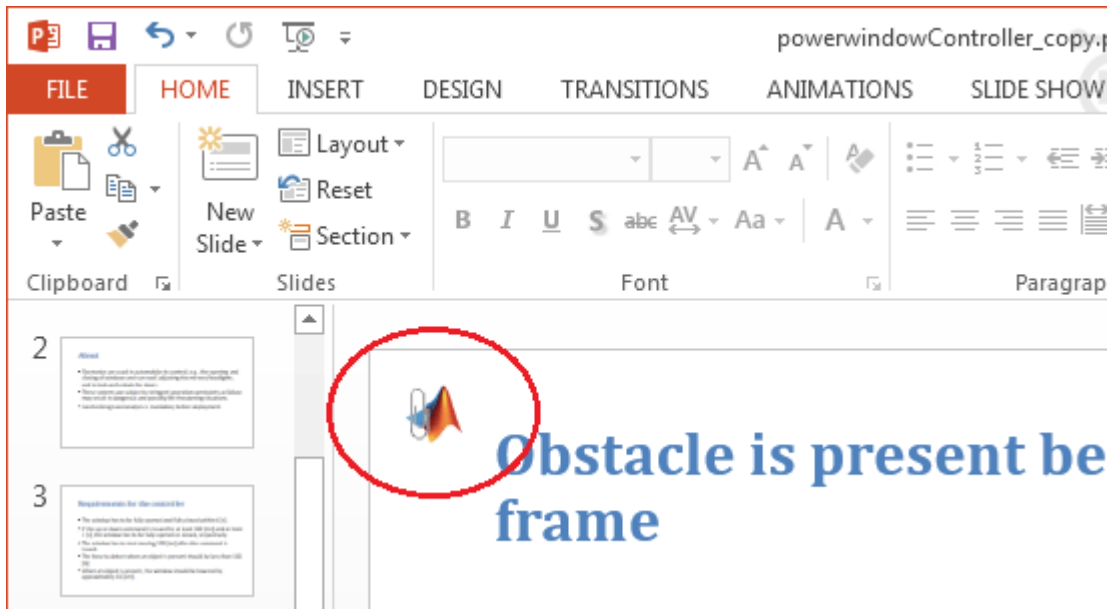
To try this out, repeat the selection linking procedure for the `obstacle` signal input block, to associate it with the corresponding slide in the example presentation.

- Navigate to slide 7 in `powerwindowController.pptx` (make it the active slide).
- Navigate to the `obstacle` block in the Simulink model.

```
rmdemo_callback('locate', 'slvndemo_powerwindowController/obstacle')
```

- Right-click the block and select **Requirements > Link to Slide in PowerPoint** from the context menu.

You should now see a new RMI icon inserted at the top-left corner of the slide.



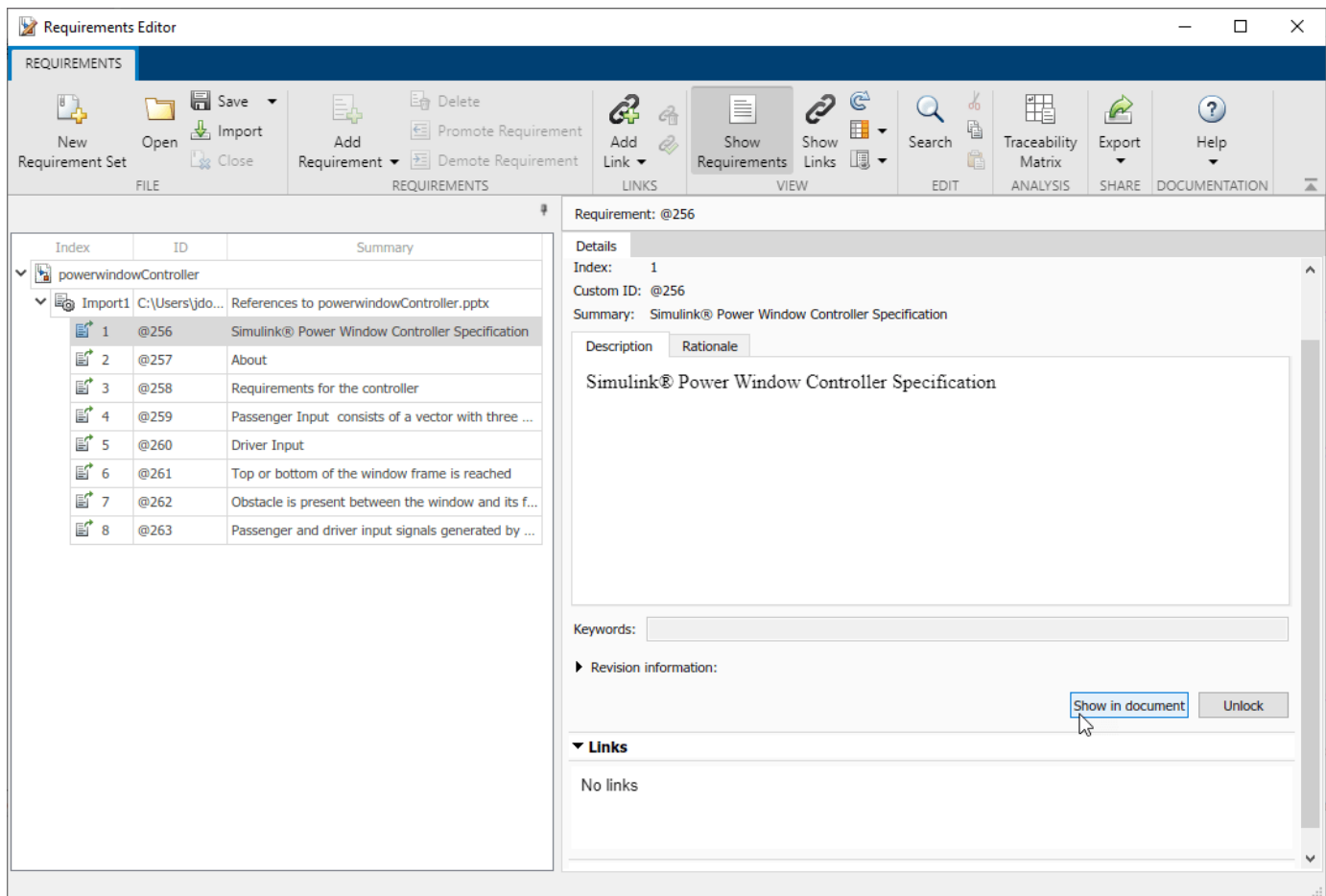
Follow Microsoft PowerPoint's instructions to follow the link, and it should highlight the corresponding block in the `slvnvdemo_powerwindowController` model.

### Importing Items from PowerPoint Document into Requirements Toolbox

Requirements Toolbox™ includes *document import* capability, if your Custom Linktype definition includes all the needed pieces. Using the customization file `rmdemo_pp_linktype.m` and `slreq.import()` API, you can automatically pull in the contents as objects of type `slreq.Reference` or `slreq.Requirement`, and save into `.slreqx` file. Refer to `slreq.import` for further information.

Because our custom document type definition does not provide implementation for `HtmlViewFcn()`, only plain-text import will work.

Make sure the document is open in PowerPoint before you run the `slreq.import()` command. The importer will display the number of imported items, which for our case corresponds to the number of slides. Use `slreq.editor` command to bring-up the **Requirements Editor**. Expand the document node to browse imported items. Click "Show in document" button to navigate from imported *reference* to original item in source document.



Alternatively, follow these steps to import the requirements from the command line.

- Make sure that the `powerwindowController.pptx` document is open before import:

```
rmi('view', 'slvndemo_powerwindowController', 1)
```

- Import the requirements using:

```
slreq.import('rmdemo_pp_linktype', 'AsReference', true, 'RichText', false)
```

- View the requirements in the **Requirements Editor** with `slreq.editor`

### Where to Go from Here

As opposed to linking to a Slide as a whole, you may want to modify the `SelectionLinkFcn()` implementation to link to a specific text or picture in the slide. Refer to Microsoft's Developer Reference pages for information on how to adjust the anchoring and appearance of Simulink navigation controls. For example, instead of inserting an icon with a hyperlink, you may want to attach a hyperlink to the selected text on the current slide.

If you need to link to a *Search text* pattern, irrespective of which slide includes the stored text, you can extend the declaration of supported location types to include the `?` character:

```
linkType.LocDelimiters = '#@?';
```

You should then provide an additional case for `switch(locationStr(1))` in the `NavigateFcn()` method. The corresponding `findText()` helper queries the PowerPoint Presentation object for all `TextFrame` items in all `Slides` and selects the item with the matching text.

The RMI link type template supports other methods, depending on your needs. For example, to have your custom links covered by Requirements Consistency Checking, consider implementing the following methods:

- `IsValidDocFcn()`
- `IsValidIdFcn()`
- `IsValidDescFcn()`

To adjust the way your links are displayed in generated reports, you can use:

- `CreateURLFcn()`
- `UrlLabelFcn()`
- `DocDateFcn()`
- `DetailsFcn()`
- `HtmlViewFcn()`

If your application is not file-based, but uses a proprietary database to store requirements documents, you must mark the link type as "not a file":

```
linkType.IsFile = 0;
```

and provide a specialized implementation for `BrowseFcn()`. This is the function that gets called when you click the **Browse** button in the Outgoing Links dialog.

```
rmi('edit', 'slvndemo_powerwindowController');
```

### Cleanup

Cleanup commands. Unregisters `rmidemo_pp_linktype`, clears open requirement sets without saving changes, and closes open models without saving changes.

```
rmi('unregister', 'rmidemo_pp_linktype');
slreq.clear();
bdclose all;
```





# Review and Maintain Requirements Links

---

- “Highlight Model Objects with Requirements” on page 12-2
- “Navigate to Simulink Objects from External Documents” on page 12-4
- “View Requirements Details for a Selected Block” on page 12-6
- “Generate Code for Models with Requirements Links” on page 12-8
- “Create and Customize Requirements Traceability Reports” on page 12-11
- “Create Requirements Traceability Report for A Project” on page 12-26
- “Validate Requirements Links” on page 12-27
- “Delete Requirements Links from Simulink Objects” on page 12-35
- “Document Path Storage” on page 12-36
- “How to Include Linked Requirements Details in Generated Report” on page 12-38
- “Managing Requirements Without Modifying Simulink Model Files” on page 12-45
- “Compare Requirement Sets Using Comparison Tool” on page 12-50

## Highlight Model Objects with Requirements

To review traceability in your model, you can highlight model objects that have requirements links.

### Highlight Model Objects with Requirements Using Model Editor

If you are working in the Simulink Editor and want to see which model objects in the `slvndemo_fuelsys_officereq` model have requirements, follow these steps:

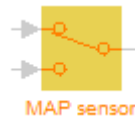
- 1 Open the example model:

```
slvndemo_fuelsys_officereq
```

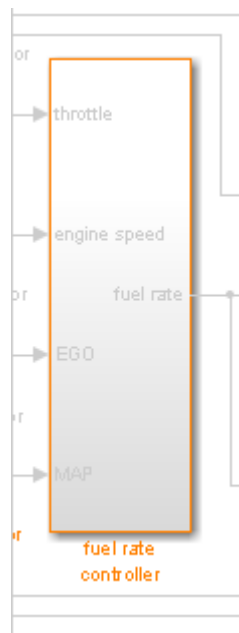
- 2 Select **Coverage Highlighting** from the **Coverage** app.

Two types of highlighting indicate model objects with requirements:

- Yellow highlighting indicates objects that have requirements links for the object itself.



- Orange outline indicates objects, such as subsystems, whose child objects have requirements links.



Objects that do not have requirements are colored gray.




- 3 You remove the highlighting from the model from the **Coverage** app. Alternatively, you can right-click anywhere in the model, and select **Remove Highlighting**.

While a model is highlighted, you can still manage the model and its contents.

## Highlight Model Objects with Requirements Using Model Explorer

If you are working in Model Explorer and want to see which model objects have requirements, follow these steps:

- 1 Open the example model:  
`slvndemo_fuelsys_officereq`
- 2 In the **Modeling** tab, click **Model Explorer**.
- 3 To highlight all model objects with requirements, click the **Highlight items with requirements on model** icon ().

The Simulink Editor window opens, and all objects in the model with requirements are highlighted.

---

**Note** If you are running a 64-bit version of MATLAB, when you navigate to a requirement in a PDF file, the file opens at the beginning of the document, not at the specified location.

---

## Navigate 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.

### Provide Unique Object Identifiers

Whenever you create a requirement link for a Simulink or Stateflow object, the RMI uses 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.

### Use the `rmiobjnavigate` Function

The `rmiobjnavigate` function identifies the Simulink or Stateflow object, highlights that object, and brings the editor window to the front of the screen. When you navigate to a Simulink model from an external application, invoke this function.

The first time you navigate to an item in a particular model, you might experience a slight delay while the software initializes the communication API and the internal data structures. You do not experience a long delay on subsequent navigation.

### Determine the Navigation Command

To create a requirement link for a Simulink or Stateflow object, at the MATLAB prompt, use the following command to find the navigation command, where `obj` is a handle or a uniquely resolved name for the object:

```
[navCmd, objPath] = rmi('navCmd', obj);
```

The return values of the `navCmd` method are:

- `navCmd` — A character vector that navigates to the object when evaluated by the MATLAB software.
- `objPath` — A character vector that identifies the model object.

Send `navCmd` to the MATLAB software for evaluation when navigating from the external application to the object `obj` in the Simulink model. Use `objPath` to visually identify the target object in the requirements document.

### Use 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 displayed in the control tooltip. The `MLEvalCmd` property is the character vector 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 the RMI API as follows:
  - a Create an empty link structure using the following command:
 

```
rmi('createempty')
```
  - b Fill in the link structure fields based on the target location in the requirements document.
  - c Attach the link to the object using the following command:
 

```
rmi('cat')
```
- 4 Determine the MATLAB navigation command that you must embed in the external tool, using the `navCmd` method:
 

```
[navCmd, objPath] = 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 in the property.

When using ActiveX navigation objects provided by the external tool, set the `MLEvalCmd` property to the `navCmd` and set the `tooltipstring` property to `objPath`.

You define the MATLAB code implementation of this procedure as the `SelectionLinkFcn` function in the link type definition file. The following files in `matlabroot\toolbox\slrequirements\linktype_examples` contain examples of how to implement this functionality:

```
linktype_rmi_doors.m
linktype_rmi_excel.m
linktype_rmi_html.m
linktype_rmi_text.m
```

## View Requirements Details for a Selected Block

When a Simulink block has linked requirements, you can view the requirement details in the Simulink canvas with the **Requirements Manager** app.

### Identify Blocks with Links

You can use the **Requirements Manager** app to identify blocks with links. In Simulink, in the **Apps** tab, open the **Requirements Manager**. Blocks that have associated outgoing links have a requirements badge (📄). You can also highlight blocks that have associated outgoing links when, in the **Requirements** tab, you click **Highlight Links**.

### Configure Settings

To configure settings in the **Requirements Manager** so that you can view requirement details in the Simulink canvas:

- 1 In the **Requirements** tab, ensure that **Layout > Requirements Browser** and **Layout > Property Inspector** are selected.
- 2 In the **Requirements** pane, in the **View** drop-down menu, select **Requirements**.

### View Requirements Details

Once you've identified blocks that have associated outgoing links, you can select a block to see if it has a linked requirement. When you select a block, all outgoing link destination summaries are displayed in the **Property Inspector**, in the **Info** tab, under **Links**. You can identify linked requirements by the icon displayed next to the requirement summary: 📄 (slreq.Requirement object), 📄➔ (slreq.Reference object or direct link to requirement in third-party application), or 📄🔓 (unlocked slreq.Reference).

To view the requirement details, click the requirement link in the **Info** tab. If the requirement is stored in Requirements Toolbox, the **Requirements** pane will scroll to the linked requirement and highlight it. If the block only has one linked requirement and you select the block, the **Requirements** pane will automatically scroll to the requirement and highlight it.

Requirements - crs\_controller

View: Requirements

| Index             | ID | Summary                        | Implemented | Verified |
|-------------------|----|--------------------------------|-------------|----------|
| crs_req_func_spec |    |                                |             |          |
| 1                 | #1 | Driver Switch Request Handling |             |          |
| 1.1               | #2 | Switch precedence              |             |          |
| 1.2               | #3 | Avoid repeating commands       |             |          |
| 1.3               | #4 | Long Switch recognition        |             |          |
| 1.4               | #7 | Cancel Switch Detection        |             |          |

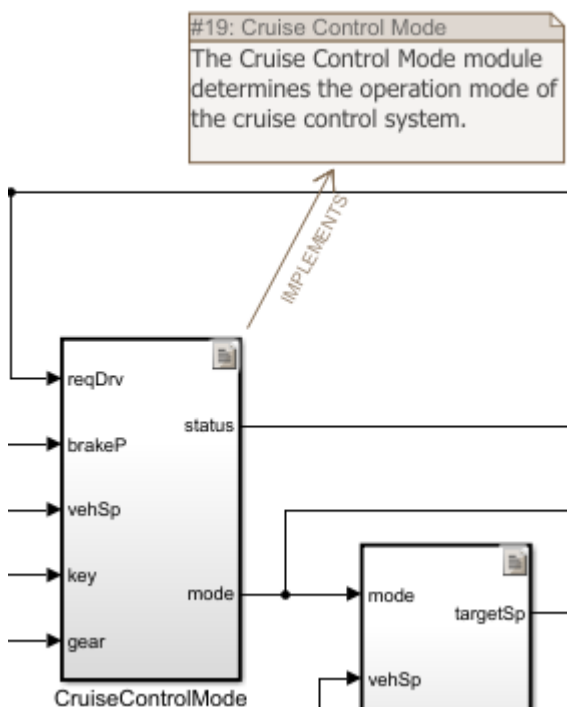
Select the highlighted requirement in the **Requirements** pane. The **Property Inspector** displays the requirement details.

**Note** If the Simulink block is linked to a requirement in a third-party application, you cannot view the requirement details directly in Simulink. When you click the requirement summary in the **Info** tab, the source document will open in the third-party application.

## Create a Requirement Annotation

You can use annotations to quickly view requirements details and call out linked requirements. Click the requirements badge (📄) on a Simulink block to see the outgoing links. To create an annotation:

- 1 Click the requirements badge (📄) on a Simulink block to view the linked requirements.
- 2 Click **Show** next to the requirement link to add an annotation to the Simulink canvas. The annotation displays the requirement ID, requirement summary, and link type. You can show or hide the requirement description by double-clicking the annotation.



- 3 Click the annotation. The **Property Inspector** displays additional requirement details.

You cannot create an annotation for requirements stored in third-party tools.

## See Also

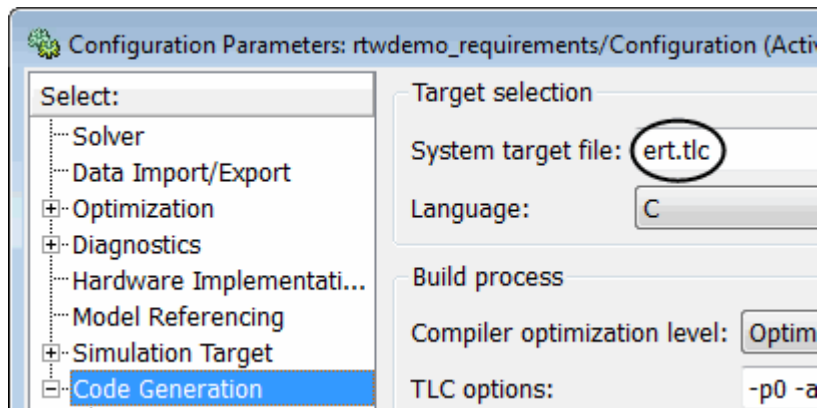
“Navigate to Requirements from Model” on page 9-16

## Generate Code for Models with Requirements Links

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

- 1 Open the `rtwdemo_requirements` example model.
- 2 In the **Modeling** tab, click **Model Settings**.
- 3 In the **Select** tree of the Configuration Parameters dialog box, select the **Code Generation** node.

The currently configured system target must be an ERT target.



- 4 Under **Code Generation**, select **Comments**.
- 5 In the **Custom comments** section on the right, select the **Requirements in block comments** check box.
- 6 Under **Code Generation**, select **Report**.
- 7 On the **Report** pane, select:
  - **Create code generation report**
  - **Open report automatically**
- 8 Press **Ctrl+B** to build the model.
- 9 In the code-generation report, open `rtwdemo_requirements.c`.
- 10 Scroll to the code for the Pulse Generator block, `clock`. The comments for the code associated with that block include a hyperlink to the requirement linked to that block.

```

rtwdemo_requirements.c 119
rtwdemo_requirements.h 120 /* DiscretePulseGenerator: '<Root>/clock'
rtwdemo_requirements_p 121 *
rtwdemo_requirements_t 122 * Block requirements for '<Root>/clock':
123 * 1. Clock period shall be consistent with chirp tolerance
124 */

```

- 11 Click the link `Clock period shall be consistent with chirp tolerance` to open the HTML requirements document to the associated requirement.

---

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

---



## How Requirements Information Is Included in 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. The Embedded Coder software generates code for Embedded Real-Time (ERT) targets.

If the model has any links to requirements, the Embedded Coder software inserts information about the requirements links into the code comments.

For example, if a block has a requirement link, the software generates code for that block. In the code comments for that block, the software inserts:

- Requirement description
- Hyperlink to the requirements document that contains the linked requirement associated with that block

---

### Note

- You must have a license for Embedded Coder to generate code for an embedded real-time application.
  - If you use an external `.req` file to store your requirement links, to avoid stale comments in generated code, before code generation, you must save any change in your requirement links. For information on how to save, see “Save Requirements Links in External Storage” on page 6-4.
- 

Comments for the generated code include requirements descriptions and hyperlinks to the requirements documents in the following locations.

| Model Object with Requirement             | Location of Code Comments with Requirements Links                                                                                                                                                                            |
|-------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Model                                     | In the main header file, <code>&lt;model&gt;.h</code>                                                                                                                                                                        |
| Nonvirtual subsystem                      | At the call site for the subsystem                                                                                                                                                                                           |
| Virtual subsystem                         | 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, <code>&lt;model&gt;.h</code> . |
| Nonsubsystem block                        | In the generated code for the block                                                                                                                                                                                          |
| MATLAB code line in MATLAB Function block | In the generated code for the MATLAB code line(s)                                                                                                                                                                            |

## See Also

### More About

- “Verify Generated Code by Using Code Tracing” (Embedded Coder)
- “Link Blocks and Requirements” on page 3-2

- “Requirements Traceability for Code Generated from MATLAB Code” on page 3-53

# Create and Customize Requirements Traceability Reports

## Create Requirements Traceability Report for Model

To create the default requirements report for a Simulink model:

- 1 Open the example model:  
`slvndemo_fuelsys_officereq`
- 2 Make sure that your current working folder is writable.
- 3 In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, select **Share > Generate Model Traceability 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.

## Requirements Report for slvndemo\_fuelsys\_officereq

.username

17-Jun-2010 10:57:04

---

### Table of Contents

- [1. Model Information for "slvndemo\\_fuelsys\\_officereq"](#)
- [2. Document Summary for "slvndemo\\_fuelsys\\_officereq"](#)
- [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 - MAP Estimate](#)
- [10. Chart - control logic](#)

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
- MATLAB code lines with requirements in MATLAB Function blocks

For detailed information about requirements reports, see “Customize Requirements Traceability Report for Model” on page 12-12.

### **If Your Model Has Library Reference Blocks**

To include requirements links associated with library reference blocks, you must select **Include links in referenced libraries and data dictionaries** under the **Report** tab of the **Requirements Settings**, as described in “Customize Requirements Report” on page 12-20.

### **If Your Model Has Model Reference Blocks**

By default, requirements links within model reference blocks in your model are not included in requirements traceability reports. To generate a report that includes requirements information for referenced models, follow the steps in “Report for Requirements in Model Blocks” on page 12-19.

## **Customize Requirements Traceability Report for Model**

### **Create Default Requirements Report**

If you have a model that contains links to external requirements documents, you can create an HTML report that contains summarized and detailed information about those links. In addition, the report contains links that allow you to navigate to both the model and to the requirements documents.

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

---

**Note** If the model for which you are creating a report contains Model blocks, see “Report for Requirements in Model Blocks” on page 12-19.

---

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

- 1 Open the example 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 Keep the example model open, but close the requirements document.

To generate a default requirements report for the `slvndemo_fuelsys_officereq` model, in the **Requirements** tab, select **Share > Generate Model Traceability 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.

The report is saved with the default name, *model\_name\_requirements.html*. If you generate a subsequent report on the same model, the new report file overwrites any earlier report file.

The report contains the following content:

### 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

17-Jun-2010 10:57:04

---

### Table of Contents

- [1. Model Information for "slvndemo\\_fuelsys\\_officereq"](#)
- [2. Document Summary for "slvndemo\\_fuelsys\\_officereq"](#)
- [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 - MAP Estimate](#)
- [10. Chart - control logic](#)

### List of Tables

The **List of Tables** includes links 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/Sensor correction and Fault Redundancy/MAP Estimate Requirements](#)
- 10.1. [Stateflow objects with 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 for "slvndemo\_fuelsys\_officereq"**

Table 1.1. slvndemo\_fuelsys\_officereq Version Information

|                         |                             |                             |                    |
|-------------------------|-----------------------------|-----------------------------|--------------------|
| <i>ModelVersion</i>     | 1.159                       | <i>ConfigurationManager</i> | none               |
| <i>Created</i>          | Tue Jun 02<br>16:11:43 1998 | <i>Creator</i>              | The MathWorks Inc. |
| <i>LastModifiedDate</i> | Sat Jun 12<br>02:31:44 2010 | <i>LastModifiedBy</i>       |                    |

**Documents Summary**

The **Documents Summary** section lists all the requirements documents to which objects in the slvndemo\_fuelsys\_officereq model link, along with some additional information about each document.

## Chapter 2. Document Summary for "slvnvdemo\_fuelsys\_officereq"

Table 2.1. Requirements documents linked in model

| ID   | Document paths stored in the model                                                | Last modified           | # links |
|------|-----------------------------------------------------------------------------------|-------------------------|---------|
| DOC1 | ERROR: unable to locate<br>slvnvdemo_FuelSys_RequirementsSpecification.docx       | unknown                 | 1       |
| DOC2 | <a href="#">fuelsys_req_docs\slvnvdemo_FuelSys_DesignDescription.docx</a>         | 29-Oct-2009<br>10:56:01 | 8       |
| DOC3 | <a href="#">fuelsys_req_docs\slvnvdemo_FuelSys_RequirementsSpecification.docx</a> | 29-Oct-2009<br>10:56:02 | 6       |
| DOC4 | <a href="#">fuelsys_req_docs\slvnvdemo_FuelSys_TestScenarios.xlsx</a>             | 29-Oct-2009<br>10:56:03 | 2       |

- **ID** — The ID. In this example, **DOC1**, **DOC2**, **DOC3**, and **DOC4** are short names for the requirements documents linked from this model.

Before you generate a report, in the Settings dialog box, on the **Reports** tab, if you select **Use document IDs in requirements tables**, links with these short names are included throughout the report when referring to a requirements document. When you click a short name link in a report, the requirements document associated with that document ID opens.

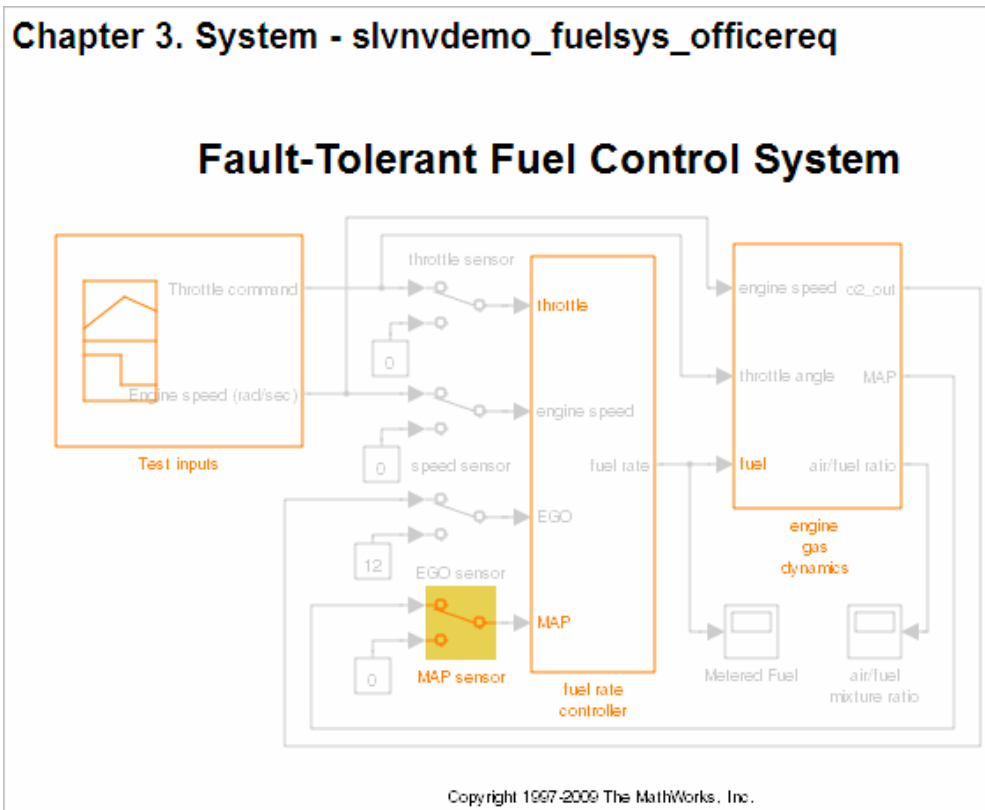
When your requirements documents have long path names that can clutter the report, select the **Use document IDs in requirements tables** option. This option is disabled by default, as you can see in the examples in this section.

- **Document paths stored in the model** — Click this link to open the requirements document in its native application.
- **Last modified** — The date the requirements document was last modified.
- **# links** — The total number of links to a requirements document.

### System

Each **System** section includes:

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



- A list of requirements associated with the model or model object. In this example, click the target document name 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<br>Microsoft Word Document | <a href="#">slvndemo FuelSys DesignDescription.docx</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 the link next to **Target:** 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="#">fuelsys_req_docs\slvndemo_FuelSys_TestScenarios.xlsx</a>, at 'Simulink requirement item 2'</p> |

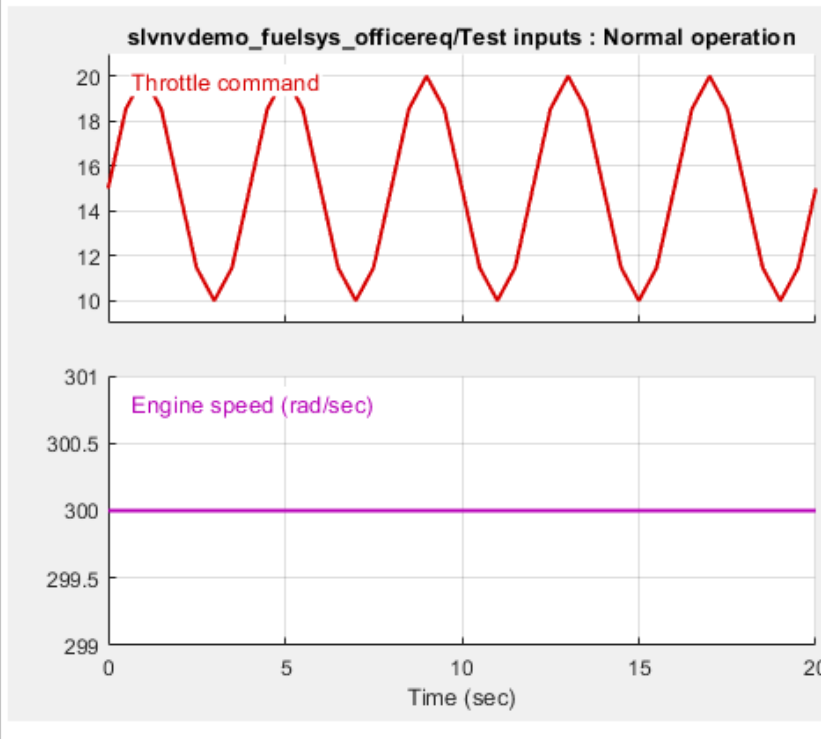
The preceding table does not include these blocks in the top-level model because:

- 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 the link under **Target (document name and location ID)** to open the requirements document associated with this signal group in the Test inputs block.

**Table 4.3. Test inputs : Normal operation signal requirements**

| Link# | Link Description           | Link Target (document name and location ID)                                                            |
|-------|----------------------------|--------------------------------------------------------------------------------------------------------|
| 1.    | "Normal mode of operation" | <a href="#">fuelsys_req_docs/slvndemo_FuelSys_TestScenarios.xlsx, at "Simulink_requirement_item_3"</a> |

[Show in Simulink](#)



**Chart**

Each **Chart** section reports on requirements in Stateflow charts, and includes:

- A graphic of the Stateflow chart that identifies each state.
- A list of elements that have requirements.

To navigate to the requirements document associated with a chart element, click the link next to **Target**.

Table 10.1. Stateflow objects with requirements

| Name                                 | Requirements                                                                                                                                                                                                                       |
|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| warmup                               | Link#1 label: "During a calibratable warm up period the oxygen sensor correction shall be disabled."<br>Target: <a href="#">fuelsys_req_docs\slvndemo_FuelSys_RequirementsSpecification.docx, at 'Simulink requirement item 3'</a> |
| [speed==0 & press < zero_thresh]/... | Link#1 label: "Speed sensor failure detection"<br>Target: <a href="#">fuelsys_req_docs\slvndemo_FuelSys_DesignDescription.docx, at 'Simulink requirement item 6'</a>                                                               |
| Rich_Mixture                         | Link#1 label: "Enriched mixture usage"<br>Target: <a href="#">fuelsys_req_docs\slvndemo_FuelSys_DesignDescription.docx, at 'Simulink requirement item 4'</a>                                                                       |
| Warmup                               | Link#1 label: "During a calibratable warm up period the oxygen sensor correction shall be disabled."<br>Target: <a href="#">fuelsys_req_docs\slvndemo_FuelSys_RequirementsSpecification.docx, at 'Simulink requirement item 3'</a> |

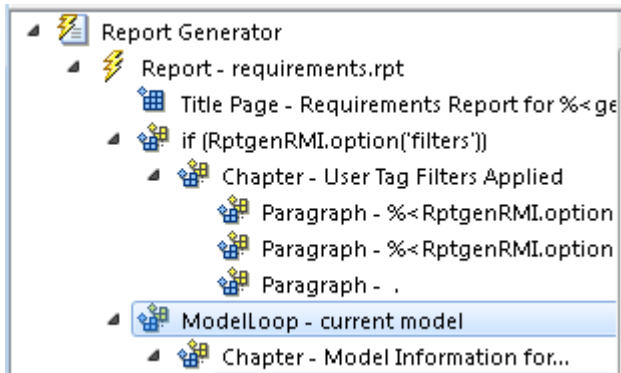
### Report for Requirements in Model Blocks

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.

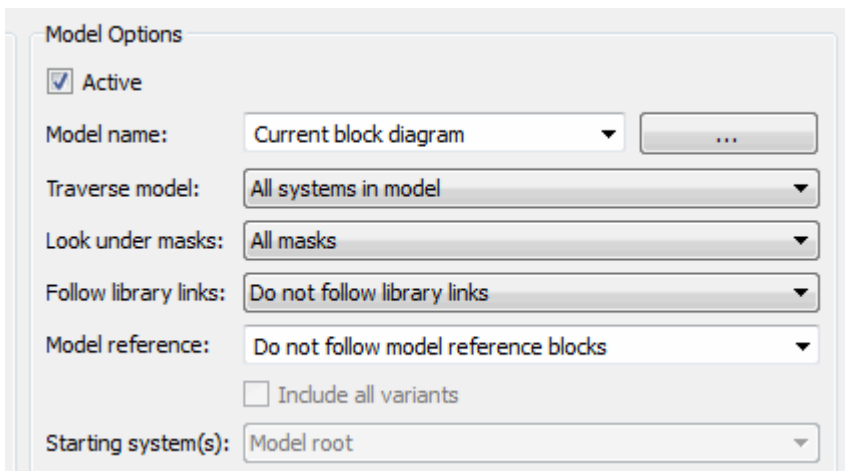
If you have a Simulink Report Generator license, before generating a requirements report, take the following steps:


- 1 Open the model for which you want to create a requirements report. This workflow uses the example model `slvndemo_fuelsys_officereq`.
- 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 .

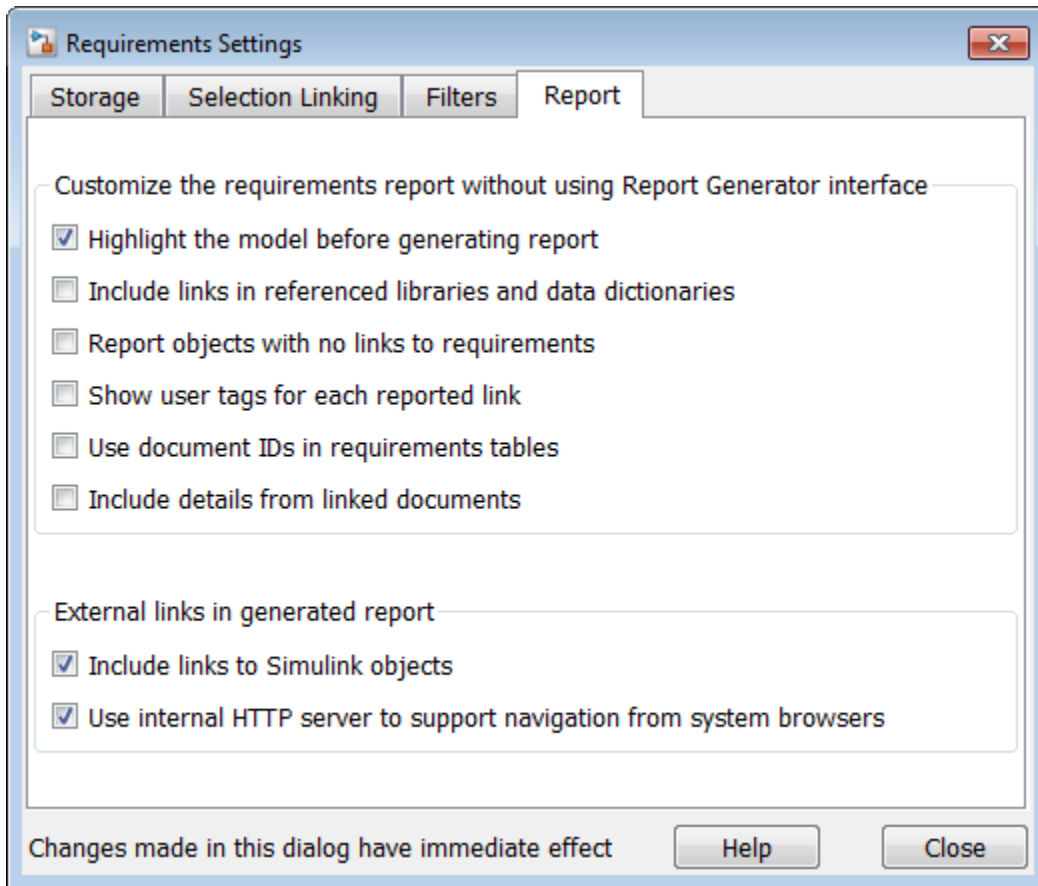
### Customize Requirements Report

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 the Simulink Report Generator software:

- “Customize Requirements Report Using the RMI Settings” on page 12-20
- “Customize Requirements Report Using Simulink Report Generator” on page 12-23

### Customize Requirements Report Using the RMI Settings

There are several options for customizing a requirements report using the Requirements Settings dialog box.



On the **Report** tab, select options that specify the contents that you want in the report.

| Requirements Settings Report Option                                | Description                                                                                                                                                                                              |
|--------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Highlight the model before generating report</b>                | Enables highlighting of Simulink objects with requirements in the report graphics.                                                                                                                       |
| <b>Include links in referenced libraries and data dictionaries</b> | Includes requirements links in referenced libraries in the generated report.                                                                                                                             |
| <b>Report objects with no links to requirements</b>                | Includes lists of model objects that have no requirements.                                                                                                                                               |
| <b>Show user keywords for each reported link</b>                   | Lists the user keywords, if any, for each reported link.                                                                                                                                                 |
| <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. |

| Requirements Settings Report Option                                        | Description                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Include details from linked documents</b>                               | Includes additional content from linked requirements. The following requirements documents are supported: <ul style="list-style-type: none"> <li>• Microsoft Word</li> <li>• Microsoft Excel</li> <li>• IBM Rational DOORS</li> </ul>                                                                                                                                        |
| <b>Include links to Simulink objects</b>                                   | Includes links from the report to objects in Simulink.                                                                                                                                                                                                                                                                                                                       |
| <b>Use internal HTTP server to support navigation from system browsers</b> | Specifies use of internal MATLAB HTTP server for navigation from generated report to documents and model objects. By selecting this setting, this navigation is available from system browsers as long as the MATLAB internal HTTP server is active on your local host. To start the internal HTTP server, at the MATLAB command prompt, type <code>rmi('httpLink')</code> . |

To see how these options affect the content of the report:

- 1 Open the `slvndemo_fuelsys_officereq` model:

```
slvndemo_fuelsys_officereq
```
- 2 In the **Requirements Viewer** tab, click **Link Settings**.
- 3 In the Requirements Settings dialog box, click the **Report** tab.
- 4 For this example, select **Highlight the model before generating report**.

When you select this option, before generating the report, the graphics of the model that are included in the report are highlighted so that you can easily see which objects have requirements.

- 5 To close the Requirements Settings dialog box, click **Close**.
- 6 Generate a requirements report. In the Requirements tab, select S.

The requirements report opens in a browser window so that you can review the content of the report.

- 7 If you do not want to overwrite the current report when you regenerate the requirements report, rename the HTML file, for example, `slvndemo_fuelsys_officereq_requirements_old.html`.

The default report file name is `model_name_requirements.html`.

- 8 In the **Apps** tab, select **Requirements Manager**.
- 9 In the **Requirements** tab, select **Share > Generate Model Traceability Report**.
  - **Show user keywords for each reported link** — The report lists the user keywords (if any) associated with each requirement.
  - **Include details from linked documents** — The report includes additional details for requirements in the following types of requirements documents.

| Requirements Document Format | Includes in the Report                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Microsoft Word               | Full text of the paragraph or subsection of the requirement, including tables.                                                                                                                                                                                                                                                                                                                                             |
| Microsoft Excel              | If the target requirement is a group of cells, the report includes all those cells as a table. If the target requirement is one cell, the report includes that cell and all the cells in that row to the right of the target cell.                                                                                                                                                                                         |
| IBM Rational DOORS           | By default, the report includes: <ul style="list-style-type: none"> <li>• <b>DOORS Object Heading</b></li> <li>• <b>DOORS Object Text</b></li> <li>• All other attributes except <b>Created Thru</b>, attributes with empty string values, and system attributes that are false.</li> </ul> <p>Use the <code>RptgenRMI.doorsAttribs</code> function to include or exclude specific attributes or groups of attributes.</p> |

- 10 Close the Requirements Settings dialog box.
- 11 Generate a new requirements report. In the **Requirements** tab, select **Share > Generate Model Traceability Report**.
- 12 Compare this new report to the report that you renamed in step 7:
  - User keywords associated with requirements links are included.
  - Details from the requirement content are included as specified in step 9.
- 13 When you are done reviewing the report, close the report and the model.

To see an example of including details in the requirements report, enter this command at the MATLAB command prompt:

```
slvndemo_powerwindow_report
```

### Customize Requirements Report Using Simulink Report Generator

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 that 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>                                                                            |
|---------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| <b>Missing Requirements Block Loop (Simulink Report Generator)</b>        | Applies all child components to blocks that have no requirements                                     |
| <b>Missing Requirements System Loop (Simulink Report Generator)</b>       | Applies all child components to systems that have no requirements                                    |
| <b>Requirements Block Loop (Simulink Report Generator)</b>                | Applies all child components to blocks that have requirements                                        |
| <b>Requirements Documents Table (Simulink Report Generator)</b>           | Inserts a table that lists requirements documents                                                    |
| <b>Requirements Signal Loop (Simulink Report Generator)</b>               | Applies all child components to signal groups with requirements                                      |
| <b>Requirements Summary Table (Simulink Report Generator)</b>             | Inserts a property table that lists requirements information for blocks with associated requirements |
| <b>Requirements System Loop (Simulink Report Generator)</b>               | Applies all child components to systems with requirements                                            |
| <b>Subsystem Requirements Table (Simulink Report Generator)</b>           | Inserts a table that lists system and subsystem requirements                                         |
| <b>Data Dictionary Traceability Table (Simulink Report Generator)</b>     | Inserts a table that links data dictionary information to requirements                               |
| <b>MATLAB Code Traceability Table (Simulink Report Generator)</b>         | Inserts a table that links MATLAB code to requirements                                               |
| <b>Simulink Test Suite Traceability Table (Simulink Report Generator)</b> | Inserts a table that links a Simulink test suite to 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 the Simulink Report Generator documentation.

### **Generate Requirements Reports Using Simulink**

When you have a model open in Simulink, the Model Editor provides two options for creating requirements reports:

#### **System Design Description Report**

The System Design Description report describes a system design represented by the current Simulink model.

You can use the System Design Description report to:

- Review a system design without having the model open.



- Generate summary and detailed descriptions of the design.
- Assess compliance with design requirements.
- Archive the system design in a format independent of the modeling environment.
- Build a customized version of the report using the Simulink Report Generator software.

To generate a System Design Description report that includes requirements information:

- 1 Open the model for which you want to create a report.
- 2 In the **Modeling** tab, select **Compare > System Design Description Report**.
- 3 In the Design Description dialog box, select **Requirements traceability**.
- 4 Select any other options that you want for this report.
- 5 Click **Generate**.

As the software is generating the report, the status appears in the MATLAB command window.

The report name is the model name, followed by a numeral, followed by the extension that reflects the document type (.pdf, .html, etc.).

If your model has linked requirements, the report includes a chapter, **Requirements Traceability**, that includes:

- Lists of model objects that have requirements with hyperlinks to display the objects
- Images of each subsystem, highlighting model objects with requirements

### **Design Requirements Report**

In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Share > Generate Model Traceability Report**. This option creates a requirements report, as described in “Create Default Requirements Report” on page 12-12.

To specify options for the report, select **Share > Report Options**. Before generating the report, on the **Report** tab, set the options that you want. For detailed information about these options, see “Customize Requirements Report” on page 12-20.

## **See Also**

### **More About**

- “Report Requirements Information” on page 5-12
- “Create and Store Links” on page 3-31

## Create Requirements Traceability Report for A Project

To create a report for requirements traceability data in a project:

- 1 Open your project.
- 2 At the MATLAB command prompt, enter the following:

```
rmi('projectreport')
```

The MATLAB Web browser opens, showing the traceability report for the project.

This top-level HTML report contains a separate section for Simulink model files, MATLAB code files, and other files included in the project. For each individual file with one or more associated requirements links, a separate HTML report, or sub-report, shows the requirements traceability data for that file. The top-level report contains links to each sub-report.

If you have a MATLAB file with requirements traceability links that is not part of a project, you can create a separate report for the MATLAB file using the `rmi('report', matlabfilepath)` command. For more information, see `rmi`.

# Validate Requirements Links

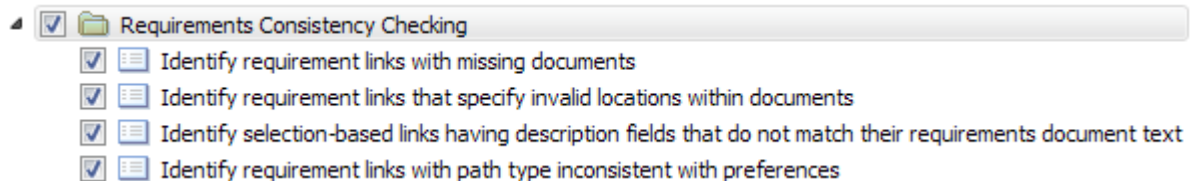
## Validate Requirements Links in a Model

### Check Requirements Links with the Model Advisor

To make sure that every requirements link in your Simulink model has a valid target in a requirements document, run the Model Advisor Requirements consistency checks:

- 1 Open the example model:  
slvndemo\_fuelsys\_officereq
- 2 Open the Model Advisor to run a consistency check. In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Check Consistency**.

In the **Requirements 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.

| Consistency Check                                                                 | Problem Identified                                                                                                                                                                                                                                                                              |
|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Identify requirement links with missing documents</b>                          | The Model Advisor cannot find the requirements document. This might indicate a problem with the path to the requirements document.                                                                                                                                                              |
| <b>Identify requirement links that specify invalid locations within documents</b> | The Model Advisor cannot find the designated location for the requirement. This check is implemented for: <ul style="list-style-type: none"> <li>• Microsoft Word documents</li> <li>• Microsoft Excel documents</li> <li>• IBM Rational DOORS documents</li> <li>• Simulink objects</li> </ul> |

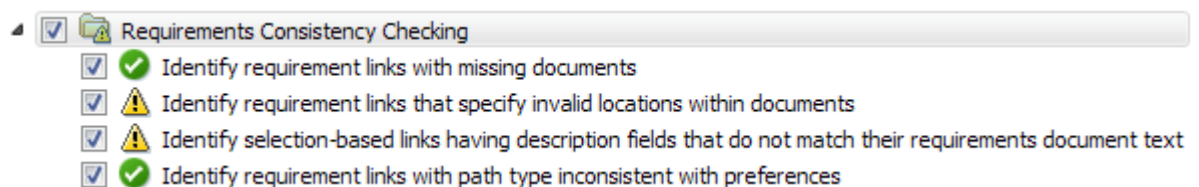
| Consistency Check                                                                                                  | Problem Identified                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Identify selection-based links having description fields that do not match their requirements document text</b> | <p>The <b>Description</b> field for the link does not match the requirements document text. When you create selection-based links, the Requirements Management Interface (RMI) saves the selected text in the link <b>Description</b> field. This check is implemented for:</p> <ul style="list-style-type: none"> <li>• Microsoft Word documents</li> <li>• Microsoft Excel documents</li> <li>• IBM Rational DOORS documents</li> <li>• Simulink objects</li> </ul>                                                                                                                                  |
| <b>Identify requirement links with path type inconsistent with preferences</b>                                     | <p>The path to the requirements document does not match the <b>Document file reference</b> field in the Requirements Settings dialog box <b>Selection Linking</b> tab. This might indicate a problem with the path to the requirements document.</p> <p>On Linux systems, this check is named <b>Identify requirement links with absolute path type</b>. The check reports a warning for each requirements links that uses an absolute path.</p> <hr/> <p><b>Note</b> For information about how the RMI resolves the path to the requirements document, see “Document Path Storage” on page 12-36.</p> |

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. Your DOORS database must include the module that contains the target requirements.
- 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 Report box includes a link to the HTML report.

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

---

**Note** To step through an example that uses the Model Advisor to check requirements links in an IBM Rational DOORS database, run the “Managing Requirements for Fault-Tolerant Fuel Control System (IBM Rational DOORS)” on page 8-39 example in the MATLAB command prompt.

---

### Fix Invalid Requirements Links Detected by the Model Advisor

In “Check Requirements Links with the Model Advisor” on page 12-27, three requirements consistency checks generate warnings in the `slvndemo_fuelsys_officereq` model.

#### Resolve Warning: Identify requirement links that specify invalid locations within documents

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.

**Inconsistencies:**

The following requirements link to invalid locations within their documents. The specified location (e.g., bookmark, line number, anchor) within the requirements document could not be found. To resolve this issue, edit each requirement and specify a valid location within its requirements document.

| <b>Block</b>                                                                                                       | <b>Requirements</b>                          |
|--------------------------------------------------------------------------------------------------------------------|----------------------------------------------|
| <a href="#">slvndemo_fuelsys_officereq/fuel_rate_controller/Sensor_correction_and Fault Redundancy/Terminator1</a> | <a href="#">This section will be deleted</a> |

This check identifies a link that specifies a location that does not exist in the Microsoft Word requirements document, `slvndemo_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 “Outgoing Links Editor” on page 11-6 for this link, under **Requirements**, click the hyperlink.
- 3 To fix the problem from the Outgoing Links dialog, 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 **Requirements Consistency Checking** category of checks.

- Click **Run Selected Checks** again, and verify that the warning no longer occurs.

**Resolve Warning: Identify selection-based links having description fields that do not match their requirements document text**

To fix the warnings about the **Description** field not matching the requirements document text:

- 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.

**Unable to check:**

- Failed to locate item @[Simulink\\_requirement\\_item\\_7](#) in [fuelsys\\_req\\_docs\slvndemo\\_FuelSys\\_DesignDescription.docx](#)

---

**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).

| Block                                                                                                               | Current description                       | External description                                                                                                                                                                                                                        |                        |
|---------------------------------------------------------------------------------------------------------------------|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| <a href="#">slvndemo_fuelsys_officereq/Test inputs</a>                                                              | <a href="#">Normal mode of operation</a>  | The simulation is run with a throttle input that ramps from 10 to 20 degrees over a period of two seconds, then back to 10 degrees over the next two seconds. This cycle repeats continuously while the engine is held at a constant speed. | <a href="#">Update</a> |
| <a href="#">slvndemo_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> |

The first message indicated that the model contains a link to a bookmark named **Simulink\_requirement\_item\_7** in the requirements document that does not exist.

In addition, this check identified the following mismatching text between the requirements blocks and the requirements document:

- The **Description** field in the Test inputs Signal Builder block link is **Normal mode of operation**. The requirement text is **The simulation is run with a throttle input that ramps from 10 to 20 degrees over a period of two seconds, then back to 10 degrees over the next two seconds. This cycle repeats continuously while the engine is held at a constant speed.**
- The **Description** field in the MAP Estimate block link is **Manifold pressure failure**. The requirement text in `slvndemo_FuelSys_DesignDescription.docx` is **Manifold pressure failure mode**.

- 2 Get more information about this link:
  - a To navigate to a block, under **Block**, click the hyperlink.
  - b To open the “Outgoing Links Editor” on page 11-6 for this link, under **Current Description**, click the hyperlink.
- 3 Fix this problem in one of two ways:
  - In the Model Advisor, click **Update**. This action automatically updates the **Description** field for that link so that it matches the requirement.
  - In the Link Editor, manually edit the link from the block so that the **Description** field matches the selected requirements text.
- 4 In the Model Advisor, select the **Requirements Consistency Checking** category of checks.
- 5 Click **Run Selected Checks** again, and verify that the warning no longer occurs.

## Validate Requirements Links in a Requirements Document

### Check Links in a Requirements Document

To check the links in a requirements document:

- 1 At the MATLAB command prompt, enter:

```
rmi('checkdoc', docName)
```

docName is a character vector that represents one of the following:

- Module ID for a DOORS requirements document
- Full path name for a Microsoft Word requirements document
- Full path name for a Microsoft Excel requirements document

The rmi function creates and displays an HTML report that lists all requirements links in the document.

The report highlights invalid links in red. For each invalid link, the report includes brief details about the problem and a hyperlink to the invalid link in the requirements document. The report groups together links that have the same problem.

- 2 Double-click the hyperlink under **Document content** to open the requirements document at the invalid link.

The navigation controls for the invalid link has a different appearance than the navigation controls for the valid links.

- 3 When there are invalid links in your requirements document, you have the following options:

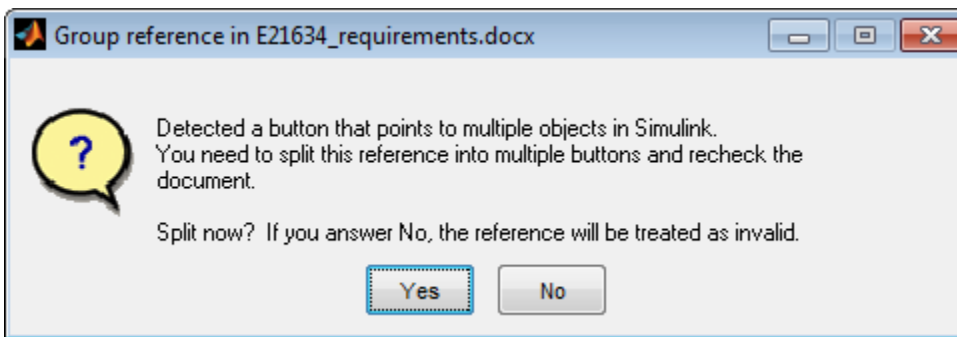
| If you want to...                                                            | Do the following...                                                                      |
|------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| Fix the invalid links                                                        | Follow the instructions in “Fix Invalid Links in a Requirements Document” on page 12-32. |
| Keep the changes to the navigation controls without fixing the invalid links | Save the requirements document.                                                          |

| If you want to...        | Do the following...                                |
|--------------------------|----------------------------------------------------|
| Ignore the invalid links | Close the requirements document without saving it. |

### When Multiple Objects Have Links to the Same Requirement

When you link multiple objects to the same requirement, only one navigation object is inserted into the requirements document. When you double-click that navigation object, all of the linked model objects are highlighted.

If you check the requirements document using the 'checkdoc' option of the rmi function and the check detects a navigation object that points to multiple objects, the check stops and displays the following dialog box.



You have two options:

- If you click **Yes**, or you close this dialog box, the RMI creates additional navigation objects, one for each model object that links to that requirement. The document check continues, but the RMI does not recheck that navigation; the report only shows one link for that requirement. To rerun the check so that all requirements are checked, at the top of the report, click **Refresh**.
- If you click **No**, the document check continues, and the report identifies that navigation object as a broken link.

### Fix Invalid Links in a Requirements Document

Using the report that the rmi function creates, you may be able to fix the invalid links in your requirements document.

In the following example, rmi cannot locate the model specified in two links.

**References with Unresolved Models - 1 unique problem in 2 links**

| Document content                           | Target model     |
|--------------------------------------------|------------------|
| <a href="#">Transmission Requirements</a>  | sf_car_doors.mdl |
| <a href="#">Engine Torque Requirements</a> |                  |

To fix invalid links:



- 1 In the report, under **Document content**, click the hyperlink associated with the invalid requirement link.

The requirements document opens with the requirement text highlighted.

- 2 In the requirements document, depending on the document format, take these steps:
  - In DOORS:
    - a Select the navigation control for an invalid link.
    - b Select **MATLAB > Select item**.
  - In Microsoft Word, double-click the navigation control.

A dialog box opens that allows you to fix, reset, or ignore all the invalid links with a given problem.

- 3 Click one of the following options.

| To...                                                                                                                                    | Click...         |
|------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| Navigate to and select a new target model or new target objects for these broken links.                                                  | <b>Fix all</b>   |
| Reset the navigation controls for these invalid links to their original state, the state before you checked the requirements document.   | <b>Reset all</b> |
| Make no changes to the requirements document. Any modifications rmi made to the navigation controls remain in the requirements document. | <b>Cancel</b>    |

- 4 Save the requirements document to preserve the changes made by the rmi function.

## Validation of Requirements Links

Requirements links in a model can become outdated when requirements change over time. Similarly, links in requirements documents may become invalid when your Simulink model changes, for example, when the model, or objects in the model, are renamed, moved, or deleted. The Requirements Toolbox software provides tools that allow you to detect and resolve these problems in the model or in the requirements document.

- “When to Check Links in a Requirements Document” on page 12-33
- “How the rmi Function Checks a Requirements Document” on page 12-34

### When to Check Links in a Requirements Document

When you enable **Modify destination for bidirectional linking** and create a link between a requirement and a Simulink model object, the RMI software inserts a navigation control into your requirements document. These links may become invalid if your model changes.

To check these links, the 'checkDoc' option of the rmi function reviews a requirements document to verify that all the navigation controls represent valid links to model objects. The checkDoc command can check the following types of requirements documents:

- Microsoft Word
- Microsoft Excel

- IBM Rational DOORS

The `rmi` function only checks requirements documents that contain navigation controls; to check links in your Simulink model, see “Validate Requirements Links in a Model” on page 12-27.



---

**Note** For more information about inserting navigation controls in requirements documents, see:

- “Insert Navigation Objects in Microsoft Office Documents” on page 7-12
  - “Insert Navigation Objects into IBM Rational DOORS Requirements” on page 8-15
- 

### How the `rmi` Function Checks a Requirements Document

`rmi` performs the following actions:

- Locates all links to Simulink objects in the specified requirements document.
- Checks each link to verify that the target object is present in a Simulink model. If the target object is present, `rmi` checks that the link label matches the target object.
- Modifies the navigation controls in the requirements document to identify any detected problems. This allows you to see invalid links at a glance:
  - Valid link: 
  - Invalid link: 

## Delete Requirements Links from Simulink Objects

### Delete 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 > Open Outgoing Links dialog**.
- 2 In the top-most pane of the Link Editor, select the link that you want to delete.
- 3 Click **Delete**.
- 4 Click **Apply** or **OK** to complete the deletion.

### Delete 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 Outgoing 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.

### Delete All Links from Multiple Simulink Objects

To delete all requirements links from a group of Simulink model objects in the same model diagram or Stateflow chart:

- 1 Select the model objects whose requirements links you want to delete.
- 2 Right-click one of the objects and select **Requirements > Delete All Outgoing Links**.
- 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 child objects inside subsystems, Model blocks, or Stateflow charts.

## Document Path Storage

When you create a requirements link, the RMI stores the location of the requirements document with the link. If you use selection-based linking or browse to select a requirements document, the RMI stores the document location as specified by the **Document file reference** option on the Requirements Settings dialog box, **Selection Linking** tab. The available settings are:

- Absolute path
- Path relative to current folder
- Path relative to model folder
- Filename only (on MATLAB path)

You can also manually enter an absolute or relative path for the document location. A relative path can be a partial path or no path at all, but you must specify the file name of the requirements document. If you use a relative path, the document is not constrained to a single location in the file system. With a relative path, the RMI resolves the exact location of the requirements document in this order:

- 1 The software attempts to resolve the path relative to the current MATLAB folder.
- 2 When there is no path specification and the document is not in the current folder, the software uses the MATLAB search path to locate the file.
- 3 If the RMI cannot locate the document relative to the current folder or the MATLAB search path, the RMI resolves the path relative to the model file folder.

The following examples illustrate the procedure for locating a requirements document.

### 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<br>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<br><MATLAB path dir>\pid.html<br>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              |

## How to Include Linked Requirements Details in Generated Report

The requirements report is a feature in RMI that scans the Simulink® model for links to external requirements documents and generates a report. When documents are available for reading during requirements report generation, you have an option to insert referenced document fragments into the generated content to produce a more detailed report.

### Open Example Model and Load RMI Links Data.

This example uses the Power Window Controller model and relies on an externally stored set of links. See “Managing Requirements Without Modifying Simulink Model Files” on page 12-45 example for a detailed demonstration of external storage feature in RMI.

Run the following commands to enable externally stored links, associate the example model with externally stored RMI data, and open the Simulink model.

```
rmipref('StoreDataExternally', true);
rmiobj.map('slvndemo_powerwindowController', 'slvndemo_powerwindowOffice.slmx');
```

Mapping ...\\slrequirements-ex02666684\\slvndemo\_powerwindowController.slx to ...\\slrequirements-

```
open_system('slvndemo_powerwindowController');
```

### Navigate Links to See Target Content in Documents

Highlight links in the model to locate objects with links and navigate to documents. In the **Apps** tab open the **Requirements Manager**. In the **Requirements** tab, click **Highlight Links**. Alternatively, evaluate the following code.

```
rmi('highlightModel', 'slvndemo_powerwindowController');
```

The included links demonstrate several possible styles of linking with Microsoft® Office documents:

- The control subsystem links to a major subheader in the Word document. Evaluate the code to navigate to the control subsystem.

```
rmiobj.callback('locate', 'slvndemo_powerwindowController/control');
```

- Both truth tables link to a subheader and a table. Evaluate the code to navigate to the Truth Table and Truth Table1 blocks.

```
rmiobj.callback('locate', {'slvndemo_powerwindowController/Truth Table', ...
 'slvndemo_powerwindowController/Truth Table1'});
```

- Driver-side Mux1 links to a subheader including some bullet points. Evaluate the code to navigate to the Mux1 block.

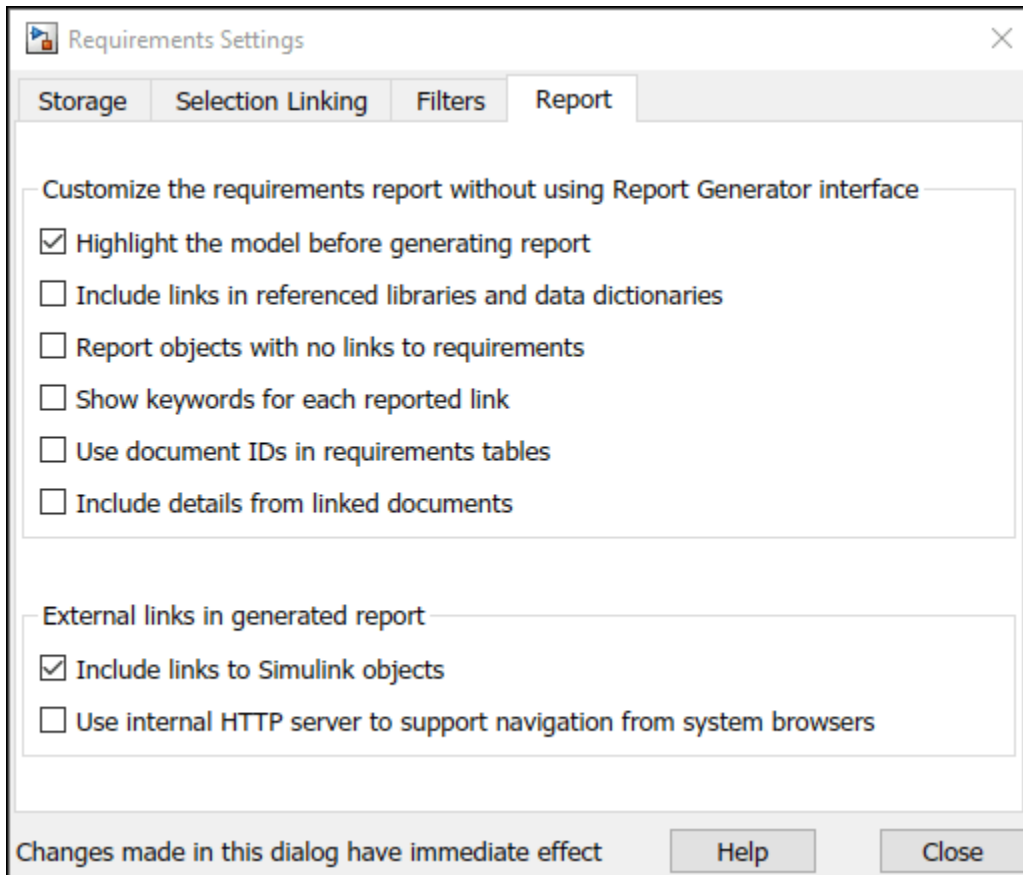
```
rmiobj.callback('locate', 'slvndemo_powerwindowController/Mux1');
```

- Passenger-side Mux4 links to just a subheader. Evaluate the code to navigate to the Mux4 block.

```
rmiobj.callback('locate', 'slvndemo_powerwindowController/Mux4');
```

### Requirements Report Without Document Fragments

- In the Simulink model, in the **Requirements** tab, click **Share > Report Options**.
- Uncheck **Include details from linked documents** in the **Report** tab of Requirements Settings dialog.



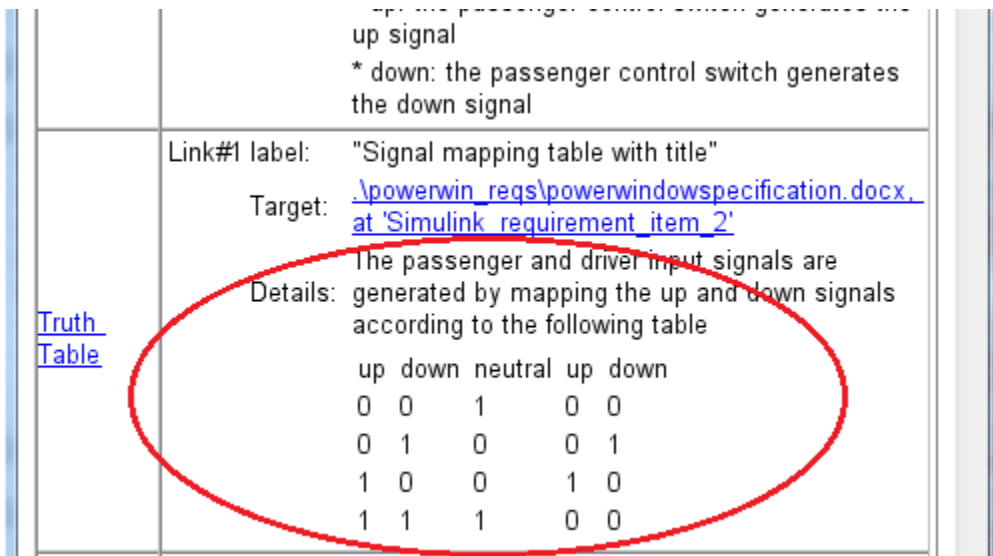
- In the Simulink model in the **Requirements** tab, click **Share > Generate Model Traceability Report**.
- Note that the tables in the report include only short labels of links. These are the same string labels you see in the object context menus and in the **Link Editor** dialog box.

**Table 3.1. Objects in slvndemo\_powerwindowController that have Requirement Links**

| Linked Object                | Requirements Data                                                                                                        |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <a href="#">control</a>      | 1. "High Level Discrete Event Control Specification - header" <a href="#">PowerWindowSpecificationWord.sreqx, at "6"</a> |
| <a href="#">Mux1</a>         | 1. "driver input - includel bullet items" <a href="#">PowerWindowSpecificationWord.sreqx, at "8"</a>                     |
| <a href="#">Mux4</a>         | 1. "passenger input - sub-header" <a href="#">PowerWindowSpecificationWord.sreqx, at "7"</a>                             |
| <a href="#">Truth Table</a>  | 1. "Signal mapping table with title" <a href="#">PowerWindowSpecificationWord.sreqx, at "11"</a>                         |
| <a href="#">Truth Table1</a> | 1. "Signal mapping table with title" <a href="#">PowerWindowSpecificationWord.sreqx, at "11"</a>                         |

**Requirements Report with Documents Content Inserted**

- In the Simulink model in the **Requirements** tab, click **Share > Report Options**.
- This time, check **Include details from linked documents**.
- Re-generate the report by clicking **Share > Generate Model Traceability Report**.
- When the target range in a Microsoft Word document includes a table, the table is now included in generated report.



- When the target location in Microsoft Word is a subheader, child content is included in generated report. Use this feature with caution: linking to a major subheader in the document could mean large amounts of text are copied from the document into the report.



again the next sample time.

|      |                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Mux1 | Link#1 label: "driver input - includel bullet items"<br>Target: <a href="#">_powerwin_reqs\powerwindowsspecification.docx_ at 'Simulink_requirement_item_3'</a><br>Details: driver input<br>* neutral: the driver control switch is not depressed<br>* up: the driver control switch generates the up signal<br>* down: the driver control switch generates the down signal                                              |
| Mux4 | Link#1 label: "passenger input - sub-header"<br>Target: <a href="#">_powerwin_reqs\powerwindowsspecification.docx_ at 'Simulink_requirement_item_4'</a><br>Details: passenger input consists of a vector with three elements<br>* neutral: the passenger control switch is not depressed<br>* up: the passenger control switch generates the up signal<br>* down: the passenger control switch generates the down signal |
|      | Link#1 label: "Signal mapping table with title"                                                                                                                                                                                                                                                                                                                                                                          |

- When the target location is a range of cells in Microsoft Excel, the target worksheet fragment is inserted in the report.

emergencyDown state and is part of the next analysis phase.

|            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [obstacle] | Link#1 label: "ENDSTOP - one cell"<br>Target: <a href="#">_powerwin_reqs\powerwindowsspecification.xlsx_ at 'Simulink_requirement_item_2'</a><br>Details: ENDSTOP   CONTROL   DISCRETE   BOOLEAN   TRUE,'FALSE'                                                                                                                                                                                                                                                                                                                                            |
| [endstop]  | Link#1 label: "AD1.3 - subtable"<br>Target: <a href="#">_powerwin_reqs\powerwindowsspecification.xlsx_ at 'Simulink_requirement_item_1'</a><br>Details: 4x6 range in<br>C:\Work\Aslrtw\matlab\toolbox\slvnv\rmidemos\powerwin_reqs\powerwindowsspecification.xlsx<br>AD1.3            DETECT_OBSTACLE_ENDSTOP<br>ENDSTOP_MIN    DATA                            CONSTANT REAL VALUE: 0.0 [m]<br>ENDSTOP_MAX    DATA                            CONSTANT REAL VALUE: 0.4 [m]<br>OBSTACLE_MAX   DATA                            CONSTANT REAL VALUE: 0.3 [m] |
|            | Link#1 label: "AD1.3 - subtable"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |



- When the target location is a single cell in Microsoft Excel worksheet, the content of cells to the right of the target cell is also inserted into the report.

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| emergencyDown | <p>Link#1 label: "emergencyDown state"</p> <p>Target: <a href="#">\powerwin_reqs\powerwindowsspecification.docx, at 'Simulink_requirement_item_5'</a></p> <p>Details: Links to the following paragraph under 'Driver-side Precedence':<br/>         * On the next sample time, the state machine moves to its emergencyDown state to lower the window a few inches. How far exactly depends on how long the state machine is in the emergencyDown state and is part of the next analysis phase.</p> |
| [obstacle]    | <p>Link#1 label: "ENDSTOP - one cell"</p> <p>Target: <a href="#">\powerwin_reqs\powerwindowsspecification.xlsx, at 'Simulink_requirement_item_2'</a></p> <p>Details: ENDSTOP   CONTROL   DISCRETE   BOOLEAN   TRUE', 'FALSE'</p>                                                                                                                                                                                                                                                                    |
|               | <p>Link#1 label: "AD1.3 - subtable"</p> <p>Target: <a href="#">\powerwin_reqs\powerwindowsspecification.xlsx, at 'Simulink_requirement_item_1'</a></p> <p>Details: 4x6 range in<br/>         C:\Work\A\slrtw\matlab\toolbox\slvm\midemos\powerwin_reqs\powerwindowsspecification.xlsx</p>                                                                                                                                                                                                           |

**Include IBM Rational DOORS Attributes in RMI Report**

For users linking with IBM® Rational® DOORS®, RMI provides more control over which object attributes to include in the requirements tables.

**Table 3.1. slvndemo\_powerwindowController Requirements**

| Link# | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Target (do location ID)                    |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| 1     | <p>Label: 5.3 Scope Description<br/>                     Details: Scope Description<br/>                     * TPS2000 Digital Storage Oscilloscope Series<br/>                     * Powerful productivity from bench to field.<br/>                      * With up to 200 MHz bandwidth and 2 GS/s sample rate, the TPS2000 Oscilloscope Series allows you to safely make floating or differential measurements in a variety of challenging environments. With up to 4-isolated channels and a portable, battery-powered design, the TPS2000 Series allows you to quickly and accurately tackle the tough challenges you face in the field of electronics and power systems - all designed with your safety in mind.</p> <p>Created By: astarovo<br/>                     Created On: 21 January 2011<br/>                     Created Thru: Manual Input<br/>                     Last Modified By: astarovo<br/>                     Last Modified On: 16 March 2011</p> | <p><a href="#">DOORS m Project/Scr</a></p> |
|       | <p>Label: 5.1 Pulse Generator connects to Scope<br/>                     Details: Pulse Generator connects to Scope<br/>                     * 50MHz sine wave frequency<br/>                     * 25MHz pulse frequency<br/>                     * Arbitrary waveform generator with 256k-point, 14-bit resolution<br/>                      * Built-in function generator capability includes: sine, square, triangle, noise, DC, etc.<br/>                     * Precision pulses and square waves with fast (5ns) rise/fall times<br/>                     * Built-in 10MHz external time base for multiple unit synchronization<br/>                     * Built-in AM, FM, PM, FSK, PWM modulation<br/>                     * Frequency sweep and burst capability</p>                                                                                                                                                                                              |                                            |

- The default configuration will include DOORS Object Heading, DOORS Object Text and all other attributes except: "Created Thru", all attributes with empty string values, and system attributes that are false.
- The list of attribute names to include in generated report is stored as part of RMI settings under user *prefdir*.
- Use the `RptgenRMI.doorsAttribs` to include/exclude certain attributes and/or groups of attributes.

```
current_settings = RptgenRMI.doorsAttribs('show')
```

```
current_settings = 6x1 cell
 {'Object Text' }
 {'$AllAttributes$' }
 {'$NonEmpty$' }
 {'-Created Thru' }
```

```
{'+Last Modified By'}
{'+Last Modified On'}
```

**help** `RptgenRMI.doorsAttribs`

`RptgenRMI.doorsAttribs` - IBM Rational DOORS attributes in requirements report  
This MATLAB function returns the DOORS attribute report settings.

## Syntax

```
settings = RptgenRMI.doorsAttribs('show')
tf = RptgenRMI.doorsAttribs('default')
tf = RptgenRMI.doorsAttribs(Name,Value)
```

## Name-Value Arguments

```
type - Types of attributes to include or omit in report
 'all' | 'user' | 'none'
add - Attributes to add to report
 character array
remove - Attributes to remove from report
 character array
nonempty - Include or omit empty attributes
 'on' | 'off'
```

## Output Arguments

```
settings - Current DOORS attribute report settings
 cell array
tf - Changed settings success status
 1 | 0
```

## Examples

```
Show the DOORS Attribute Report Settings
Restore Default DOORS Attributes Report Settings
Include or Omit DOORS Attributes from the Requirements Report by Specifying Type
Explicitly Include or Omit DOORS Attributes from the Requirements Report
Include or Omit Empty User-Defined DOORS Attributes from the Requirements Report
```

See also `rmi`

Introduced in Requirements Toolbox in R2011b  
Documentation for `RptgenRMI.doorsAttribs`  
`doc RptgenRMI.doorsAttribs`

## Managing Requirements Without Modifying Simulink Model Files

You can store link data for Simulink® models by storing link data in the Simulink model `.slx` file or storing links in an external `.slmx` file.

Use external link storage to manage changes to the model file separately from changes to the requirements links. Additionally, using external link storage allows you to manage multiple sets of requirements links for the same model, by loading different `.slmx` files.

This example shows how to work with externally stored RMI links. Click **Open Example** to create a working folder of the example files. Run the following commands:

```
rmimap.map('slvnvdemo_powerwindowController', 'clear');
```

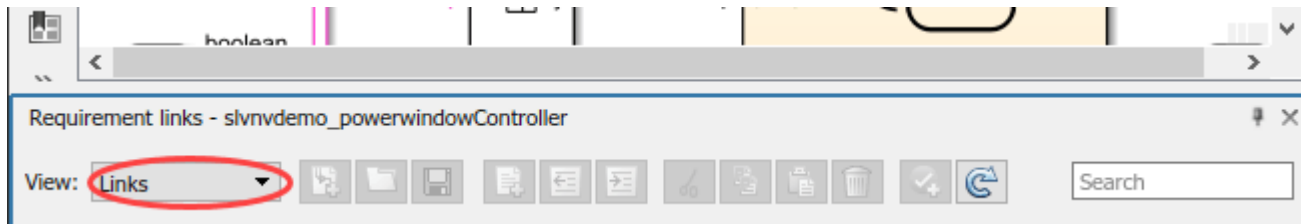
```
Nothing to clear for ...\slrequirements-ex04732634\slvnvdemo_powerwindowController.slx
```

```
open_system('slvnvdemo_powerwindowController');
rmipref('UnsecureHttpRequests', true);
```

### Set Up Requirements Manager to Work with Links

- 1 In the **Apps** tab, open **Requirements Manager**.
- 2 In the **Requirements** tab, ensure **Layout > Requirements Browser** is selected.
- 3 In the **Requirements Browser**, in the **View** drop-down menu, select **Links**.

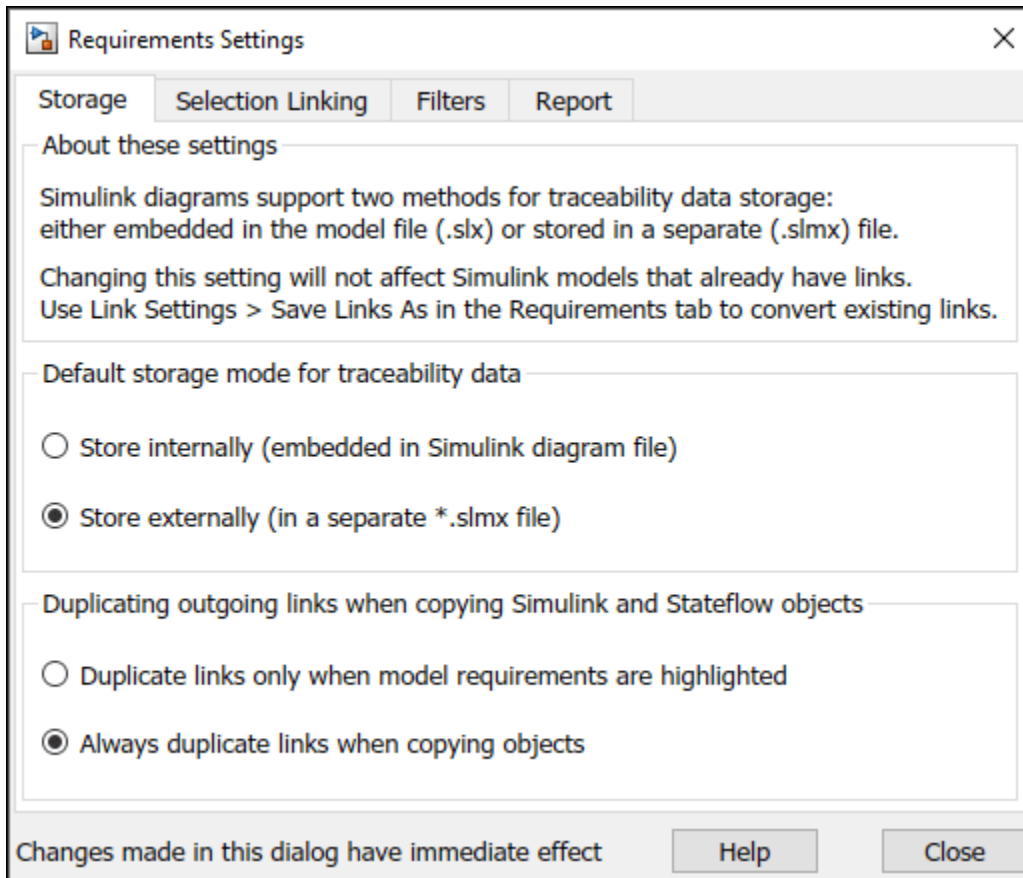
In this example, you will work exclusively in the **Requirements** tab and any references to toolbar buttons are in this tab.



### Configure RMI to Store Links Externally

- 1 In the **Requirements** tab, select **Link Settings > Default Link Storage**. This opens the **Requirements Settings** dialog box.
- 2 Select **Store externally (in a separate \*.slmx file)**.

```
rmipref('StoreDataExternally', true);
```



The default file name for saving requirements links data is *ModelName.slmx*. The links file must be in the same folder as the model for the links to resolve.

### Creating and Managing RMI Links

Create a link from model to document.

- 1 Open the `PowerWindowSpecification.docx` file in the current directory.
- 2 Select the subheader *passenger input consists of a vector with three elements* under the *High Level Discrete Event Control Specification* section.
- 3 Find the block `Mux4`.

```
rmidemo_callback('locate', 'slvndemo_powerwindowController/Mux4');
```

Right click `Mux4` and select **Requirements > Link to Selection in Word**.

You can also enter the following to create the link:

```
testReqLink = rmi('createEmpty');
testReqLink.description = 'testReqLink';
testReqLink.doc = 'PowerWindowSpecification.docx';
testReqLink.id = '?passenger input consists of a vector with three elements';
```

Create the link:

```
rmi('set', 'slvndemo_powerwindowController/Mux4', testReqLink)
```

If the model is still highlighted, the Mux4 block highlights to indicate associated requirements data. New link information is stored separately from the model and saves when the model is saved.

### Saving Requirements Links Data to External Files

When saving requirements data externally, you can save changes to requirements by:

- Clicking **Save** or **Save As** the Simulink model, even if the model does not have unsaved changes.
- Closing the model. You will be prompted to save links changes.
- Clicking **Link Settings > Save Links As**.

Click **Link Settings > Save Links As** and save them with the name `slvnvdemo_powerwindowController.slmx`.

Close the model.

```
close_system('slvnvdemo_powerwindowController',1)
```

Save the links file with the default `ModelName.slmx` name in the model folder, or choose a different file name and/or location.

### Loading Requirements Links from External Files

When you open a model, the RMI will try to load requirements links data from the recently used location for this model. You may also select **Load Links** to choose a different `.slmx` or `.req` file. In this way you can use several sets of links with the same model. For example, you can use links to design change descriptions that are different from links to original design specifications.

Reopen the model and select **Load Links** to open a file browser and point to `slvnvdemo_powerwindowRequirements.slmx` in the working directory, or evaluate the following code.

```
open_system('slvnvdemo_powerwindowController');
otherReqFile = 'slvnvdemo_powerwindowRequirements.slmx';
rmi.map('slvnvdemo_powerwindowController', otherReqFile);
```

Mapping `...\slrequirements-ex04732634\slvnvdemo_powerwindowController.slx` to `...\slrequirements-`

Click **Highlight Links** in the toolstrip to confirm that an alternative set of links is now associated with the model, or evaluate the following code.

```
rmi('highlightModel','slvnvdemo_powerwindowController');
```

You can navigate and modify these links in the same way you would work with embedded (in-model) links.

### Moving RMI Links from Internal to External Storage

A model with existing embedded requirements links can be converted to external storage. Link data will no longer be stored in `.slx` file, but in a new `.slmx` file. Try this out with the following steps.

Open another model that has internally stored RMI data by evaluating the following code.

```
open_system('slvnvdemo_fuel_sys_officereq');
```

Select **Link Settings > Save Links As Link Set File** to open a file browser. Choose a file name for the new external `.slmx` file and click **OK**. The model is resaved with no embedded links, and a new `.slmx` file is created. You can also evaluate the following code.

```
rmidata.saveAs('slvndemo_fuelsys_officereq','slvndemo_fuelsys_officereq.slmx');
```

The requirements links now depend on the external file.

Click **Highlight Links** to confirm that the link data is available, or evaluate the following code.

```
rmi('highlightModel','slvndemo_fuelsys_officereq');
```

Close the model manually and delete the external `.slmx` file, or evaluate the following code.

```
close_system('slvndemo_fuelsys_officereq',1);
```

Use the following code to delete the file:

```
rmidemo_callback('remove','slvndemo_fuelsys_officereq.slmx')
```

Manually reopen the model or evaluate the following code.

```
open_system('slvndemo_fuelsys_officereq');
```

Click **Highlight Links** or evaluate the following to highlight the links:

```
rmi('highlightModel','slvndemo_fuelsys_officereq')
```

Nothing is highlighted because the data is no longer available. Recreate the `slvndemo_fuelsys_officereq.slmx` file and map it to the `slvndemo_fuelsys_officereq` Simulink model by evaluating the following code.

```
rmimap.map('slvndemo_fuelsys_officereq','backup_reqs.slmx');
```

Mapping `...\slrequirements-ex04732634\slvndemo_fuelsys_officereq.slx` to `...\slrequirements-ex04732634\slvndemo_fuelsys_officereq.slmx`

```
rmidata.saveAs('slvndemo_fuelsys_officereq','slvndemo_fuelsys_officereq.slmx');
```

Points to keep in mind before you move internally stored links to an external file:

- You will need to carry an extra `.slmx` file along with the model file.
- Non-default file name and location associations are stored in user preferences. If you move or rename the `.slmx` file outside MATLAB®, you will have to manually point RMI to the new location when the model is reopened.
- When one user has configured a non-default location or name for the `.slmx` file associated with the model, other RMI users will need to manually select **Load links** when they open the model. The specified location will persist in each user's preferences and does not need to change unless files are moved or renamed again.

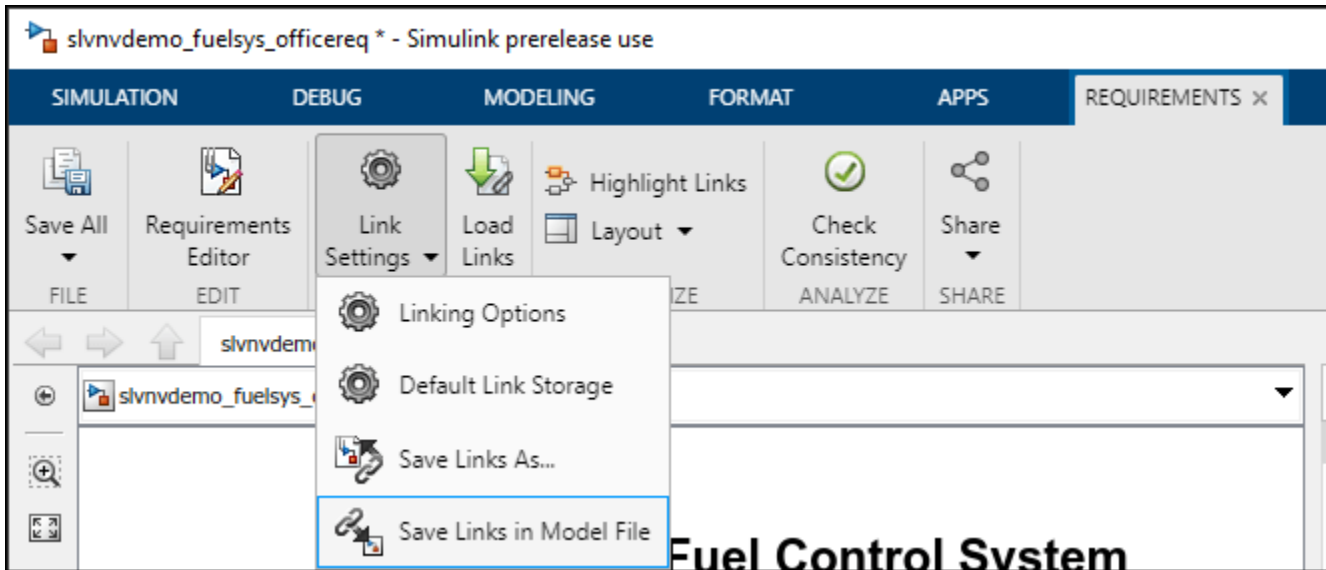
### Moving RMI Links from External to Internal Storage

To *embed* RMI data with the Simulink model, so that all information is in one place and you do not need to track extra files, select **Link Settings > Save Links in Model File**. The external `.slmx` file still exists, but it is not read when you reopen the model that now has *embedded* RMI data. You can try this out with the `slvndemo_fuelsys_officereq.slx` model from the previous section.

Alternatively, evaluate the following code.



```
rmipref('StoreDataExternally',false);
save_system('slvndemo_fuelsys_officereq');
```



Points to keep in mind before you *embed* RMI data with the model file:

- Every change to RMI links will modify the model file.
- External `.slmx` files are disregarded when `.slx` file contains traceability links data.

### Cleanup

Clear the open requirement sets and link sets. Close all open models.

```
slreq.clear;
bdclose 'all';
```

## Compare Requirement Sets Using Comparison Tool

This examples shows you how to use the **Comparison** tool to analyze the comparison results for different types of properties by publishing a comparison report.

### Compare Requirement Sets

The example uses two requirement sets. The `crs_req_func_spec_01` and `crs_req_func_spec_02` files contain the original and changed requirements sets, respectively.

To compare the two requirements sets, in the current folder pane of MATLAB® example, select the two files. Right-click either file and select **Compare Selected Files/Folders**. Alternatively, use the `visdiff` function to compare the two sets by typing this command in the MATLAB command prompt.

```
visdiff('crs_req_func_spec_01.slreqx', 'crs_req_func_spec_02.slreqx');
```

### Understand the Results

By default, the tool highlights the changed requirements.. The tool differentiates between different types of changes using colors. To change the colors, on the **Environment** tab on the MATLAB toolstrip, click **Preferences**, and then click Comparison.

The view in the comparison tool consists of two main parts:

- The top pane displays the requirement sets in a hierarchical manner.
- The bottom pane contains a name-value table with the names and values of specific properties or key parameters of the selected requirement.

Use the arrow button in the table to move the table to its corresponding hierarchy pane.

COMPARISON

Previous Next Refresh Swap Sides Find Linked Scrolling Always Highlight Highlight Now Filter Publish

NAVIGATE HIGHLIGHT FILTER PUBLISH

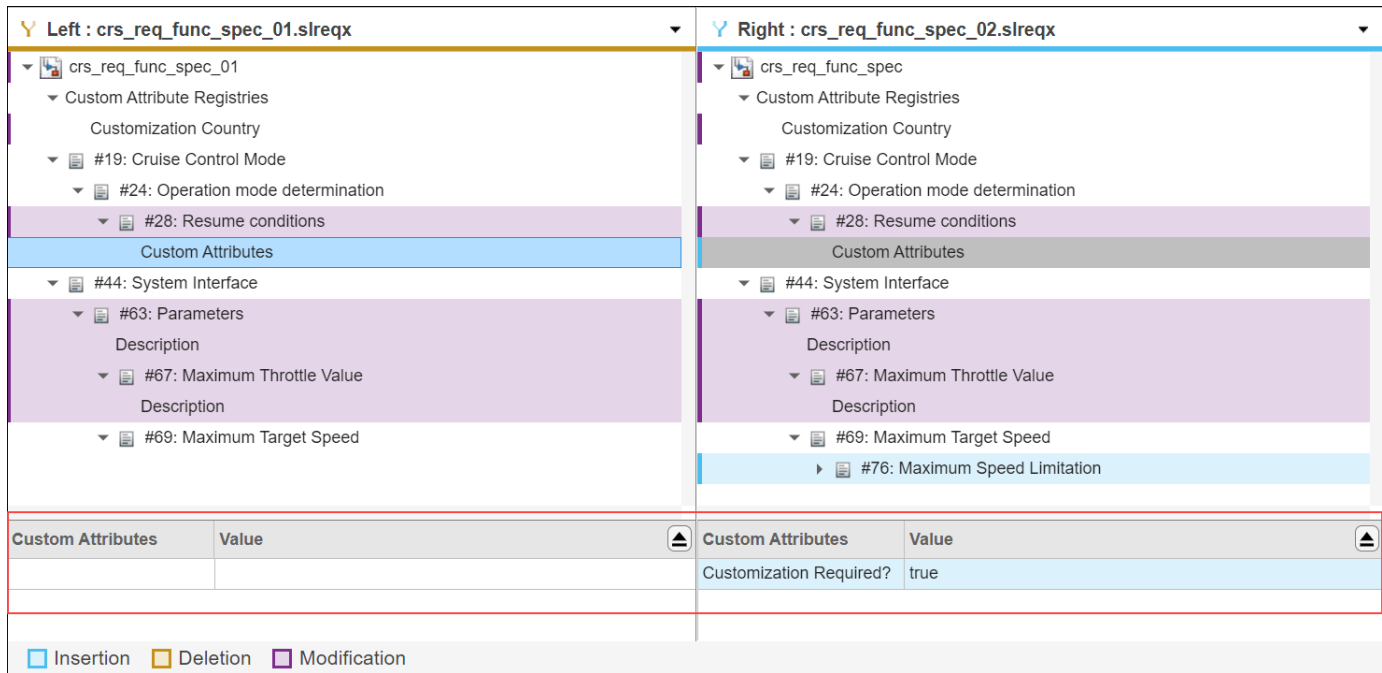
Left : crs\_req\_func\_spec\_01.sreqx Right : crs\_req\_func\_spec\_02.sreqx

| Name          | Value                | Name          | Value                |
|---------------|----------------------|---------------|----------------------|
| lastNumericID | 75                   | lastNumericID | 76                   |
| modifiedOn    | 31-May-2022 13:39:09 | modifiedOn    | 02-Jun-2022 13:54:08 |
| name          | crs_req_func_spec_01 | name          | crs_req_func_spec_02 |
| revision      | 70                   | revision      | 72                   |

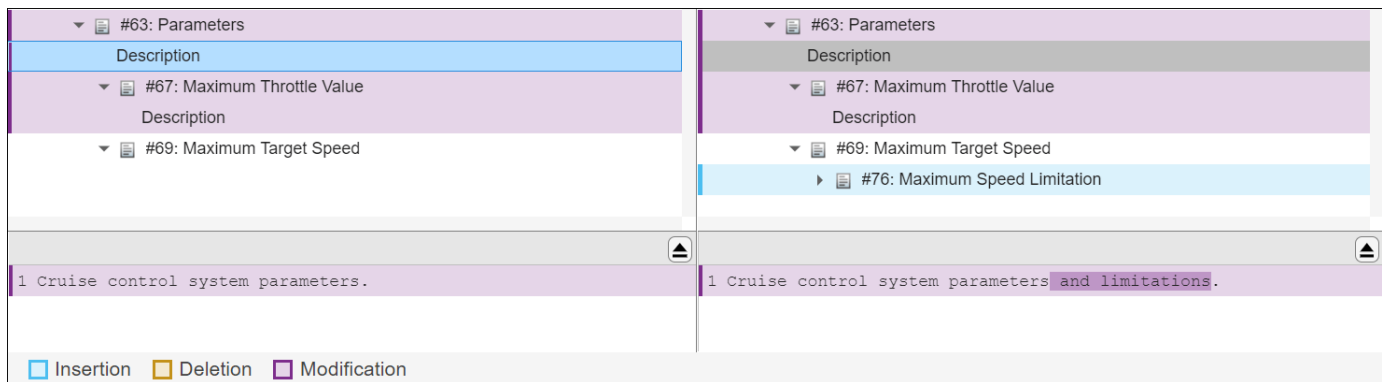
- When you click a modified requirement on either side, the name-value columns show modification details such as the username of the person who modified the requirement, the date on which they modified it, and revision of the file.

| Name       | Value               | Name       | Value               |
|------------|---------------------|------------|---------------------|
| modifiedBy | as...               | modifiedBy | sl...               |
| modifiedOn | 2017-08-03T20:59:14 | modifiedOn | 2022-05-30T04:38:23 |
| revision   | 58                  | revision   | 71                  |

- When you click a custom attribute in a requirement set, you can view the changes to the attribute. For example, the requirement set crs\_func\_spec\_02 is modified as true value for Customization Required? **Custom Attributes.**



- When you click **Description** parameter in the requirements tree, you can view the original and changed description of the selected requirement.



### Stepping Through Differences

Use the **Next** and **Previous** navigation buttons on the **Navigate** tab of the toolbar to navigate between groups of changes in the requirement sets. If the item you select on the left has a match, the tool selects it on the right. Use the **Swap Sides** button to swap the view. **Linked Scrolling** is selected by default. You can scroll the requirement sets on the left pane and right pane independently if you deselect this option.

### Highlight Results

To control highlighting in models, on the **Highlight** tab, select or clear **Always Highlight**. If you select **Always Highlight**, the tool shows the corresponding requirements in Requirements Editor each time you select an entry in the comparison tree.

You can click the **Highlight Now** button to highlight a selected requirement at any time in the Requirements Editor. For example, if you select requirement number #67 (Maximum Throttle

Value) from the `crs_req_func_spec_01.s1reqx` file, and click **Highlight Now**, the requirement opens in the Requirements Editor.

| Index | ID                   | Summary                                   | Verified |
|-------|----------------------|-------------------------------------------|----------|
| >     | crs_req_func_spec    |                                           |          |
| ✓     | crs_req_func_spec_02 |                                           |          |
| >     | 1                    | Driver Switch Request Handling            |          |
| >     | 2                    | Cruise Control Mode                       |          |
| >     | 3                    | Calculate Target Speed and Throttle Value |          |
| ✓     | 4                    | System Interface                          |          |
| >     | 4.1                  | Inputs                                    |          |
| >     | 4.2                  | Outputs                                   |          |
| ✓     | 4.3                  | Parameters                                |          |
|       | 4.3.1                | Maximum Throttle Value                    |          |
|       | 4.3.2                | Minimum Throttle Value                    |          |
| >     | 4.3.3                | Maximum Target Speed                      |          |
|       | 4.3.4                | Minimum Target Speed                      |          |
|       | 4.3.5                | Proportional term                         |          |
|       | 4.3.6                | Integral term                             |          |
|       | 4.3.7                | Threshold value for brake pressure        |          |
| >     | 5                    | Justifications                            |          |

Requirement: #67

**Properties**

Type: Functional

Index: 4.3.1

Custom ID: #67

Summary: Maximum Throttle Value

Description: MS Shell Dlg 2

Rationale: Maximum value for the throttle should be defined as a parameter.

Keywords:

Revision information:

Custom Attributes:

Links: No links

## Filter Results

The **Show Only Changed** option on the **Filter** tab is always selected by default. Clear this option to view changed and unchanged requirement sets.

| Name            | Value                              |
|-----------------|------------------------------------|
| MATLABVersion   | 9.14.0.1960994 (R2023a) Prerelease |
| createdBy       | itoy                               |
| createdOn       | 27-Feb-2017 20:50:39               |
| defaultTypeName | Functional                         |
| lastNumericID   | 75                                 |

| Name            | Value                              |
|-----------------|------------------------------------|
| MATLABVersion   | 9.14.0.1960994 (R2023a) Prerelease |
| createdBy       | itoy                               |
| createdOn       | 27-Feb-2017 20:50:39               |
| defaultTypeName | Functional                         |
| lastNumericID   | 76                                 |

### Publish Results

You can publish the comparison results in a comparison report. The report shows only the changes and not the full contents of the files. You can publish the report in an HTML, Word, or PDF format. To publish the report, on the **Comparison** tab, click **Publish** and select the format.

Alternatively, you can publish the report using the `visdiff` command.

**Note:** You can also compare link sets file in the **Comparison** tool. The comparison tool does not compare requirement or link attributes that you define or add via stereotypes and profiles.

### See Also

“Compare Requirement Sets” on page 5-10

“Compare Link Sets” on page 5-11

### Related Examples

“Three-Way AutoMerge Solution for Requirement Set and Link Set” on page 5-15

# Verification and Validation

---

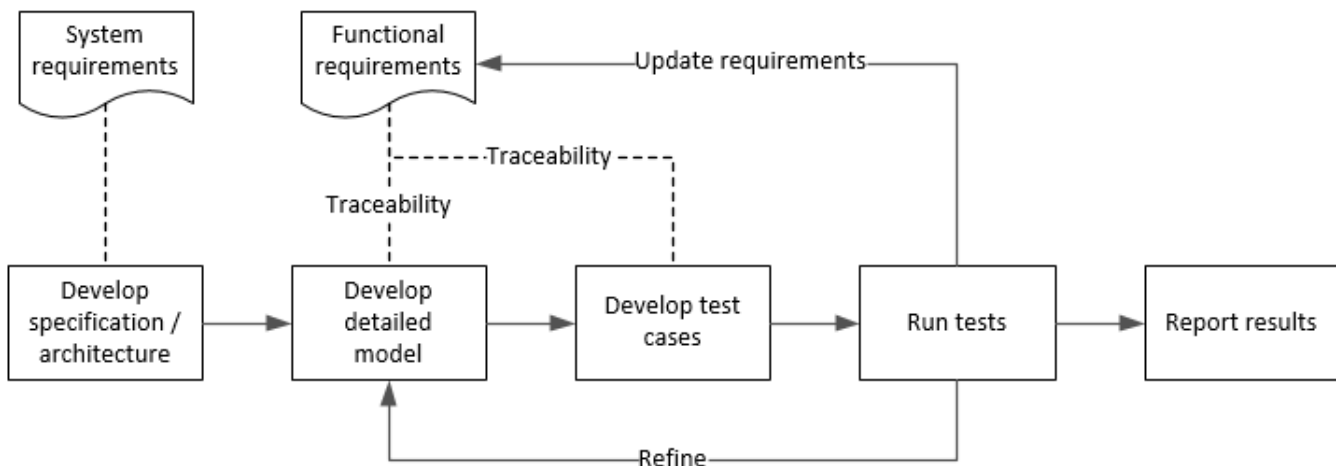
- “Test Model Against Requirements and Report Results” on page 13-2
- “Analyze Models for Standards Compliance and Design Errors” on page 13-7
- “Perform Functional Testing and Analyze Test Coverage” on page 13-9
- “Analyze Code and Test Software-in-the-Loop” on page 13-12

## Test Model Against Requirements and Report Results

### Requirements - Test Traceability Overview

Traceability between requirements and test cases helps you interpret test results and see the extent to which your requirements are verified. You can link a requirement to elements that help verify it, such as test cases in the Test Manager, `verify` statements in a Test Sequence block, or Model Verification blocks in a model. When you run tests, a pass/fail summary appears in your requirements set.

This example demonstrates a common requirements-based testing workflow for a cruise control model. You start with a requirements set, a model, and a test case. You add traceability between the tests and the safety requirements. You run the test, summarize the verification status, and report the results.




In this example, you conduct a simple test of two requirements in the set:

- That the cruise control system transitions to disengaged from engaged when a braking event has occurred
- That the cruise control system transitions to disengaged from engaged when the current vehicle speed is outside the range of 20 mph to 90 mph.

### Display the Requirements

- 1 Open the example project.

```
openExample("shared_vnv/CruiseControlVerificationProjectExample");
pr = openProject("SimulinkVerificationCruise");
```

- 2 In the `models` folder, open the `simulinkCruiseAddReqExample` model.
- 3 Display the requirements. Click the  icon in the lower-right corner of the model canvas, and select **Requirements**. The requirements appear below the model canvas.
- 4 Display the verification and implementation status. Right-click a requirement and select **Verification Status** and **Implementation Status**.



The screenshot displays the Simulink environment with a model named 'simulinkCruiseAddReqExample'. The model includes several input blocks: 'CruiseOnOff' (boolean), 'Brake' (boolean), 'Speed' (single), 'CoastSetSw' (boolean), and 'AccelResSw' (boolean). These inputs feed into a central 'Compute target speed' block. The output of this block is 'tspeed', which is also fed back into the 'CoastSetSw' block. The 'Property Inspector' on the right shows details for Requirement A 1.2, including its type (Functional), index (1.2), and summary ('Set Speed / Decelerate Button'). The 'Requirements' window at the bottom shows a table with columns for Index, ID, Summary, Verified, and Implemented.

| Index                 | ID                      | Summary                      | Verified | Implemented |
|-----------------------|-------------------------|------------------------------|----------|-------------|
| simulinkCruiseChar... |                         |                              |          |             |
| 1                     | Architecture            | Architecture                 |          |             |
| 1.1                   | A 1.1                   | Enable Disable Switch        |          |             |
| 1.2                   | A 1.2                   | Set Speed / Decelerate Bu... |          |             |
| 1.3                   | A 1.3                   | Resume Speed / Accelerat...  |          |             |
| 1.4                   | A 1.4                   | Engaged Output               |          |             |
| 1.5                   | A 1.5                   | Target Speed Output          |          |             |
| 1.6                   | A 1.6                   | Vehicle Speed Input          |          |             |
| 1.7                   | A 1.7                   | Vehicle Brake Input          |          |             |
| 2                     | Functional Requirements | Functional Requirements      |          |             |
| 3                     | Safety Requirements     | Safety Requirements          |          |             |

- 5 In the Project window, open the Simulink Test file `s1ReqTests.mldatx` from the `tests` folder. The test file opens in the Test Manager.

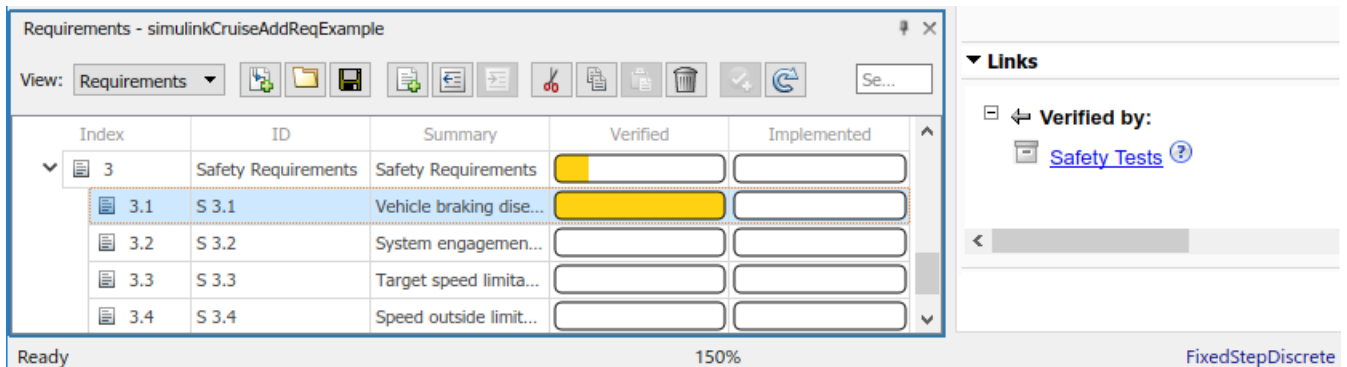
## Link Requirements to Tests

Link the requirements to the test case.

- 1 In the Project window, open the Simulink Test file `s1ReqTests.mldatx` from the `tests` folder. The test file opens in the Test Manager. Explore the test suite and select `Safety Tests`.

Return to the model. Right-click on requirement `S 3.1` and select **Link from Selected Test Case**.

A link to the `Safety Tests` test case is added to **Verified by**. The yellow bars in the **Verified** column indicate that the requirements are not verified.



- 2 Also add a link for item S 3.4.

## Run the Test

The test case uses a test harness `SafetyTest_Harness1`. In the test harness, a test sequence sets the input conditions and checks the model behavior:

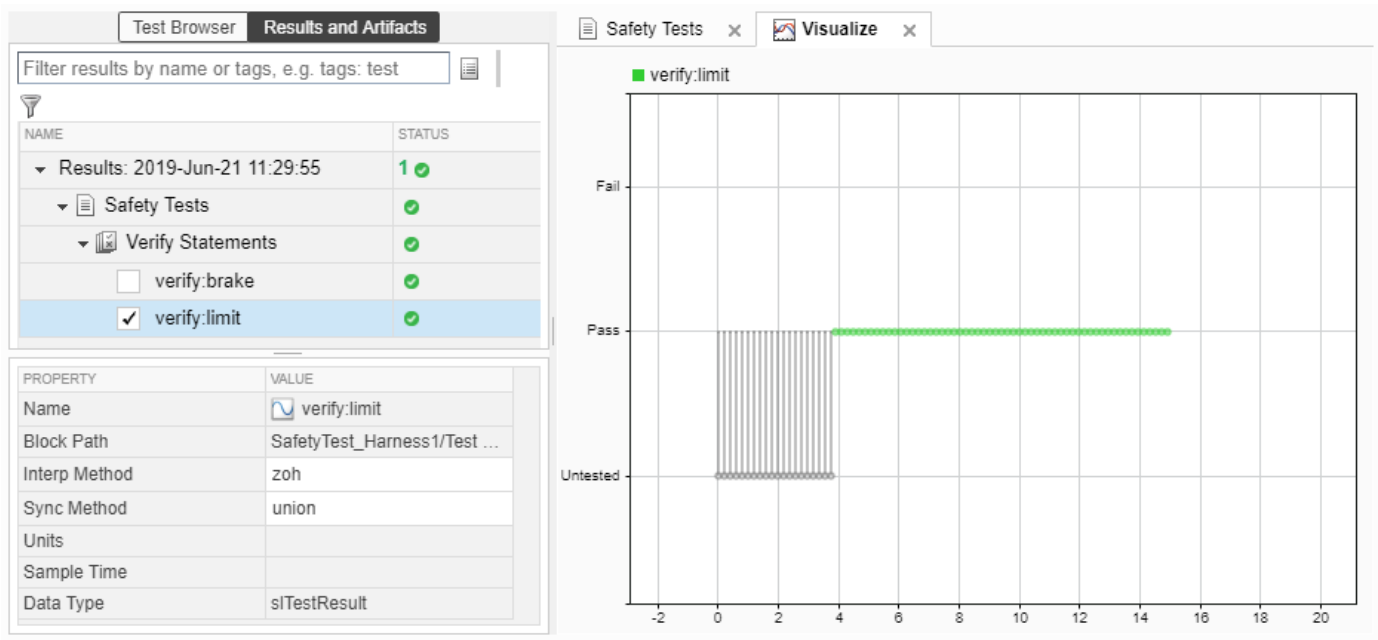
- The `BrakeTest` sequence engages the cruise control, then applies the brake. It includes the `verify` statement

```
verify(engaged == false,...
 'verify:brake',...
 'system must disengage when brake applied')
```

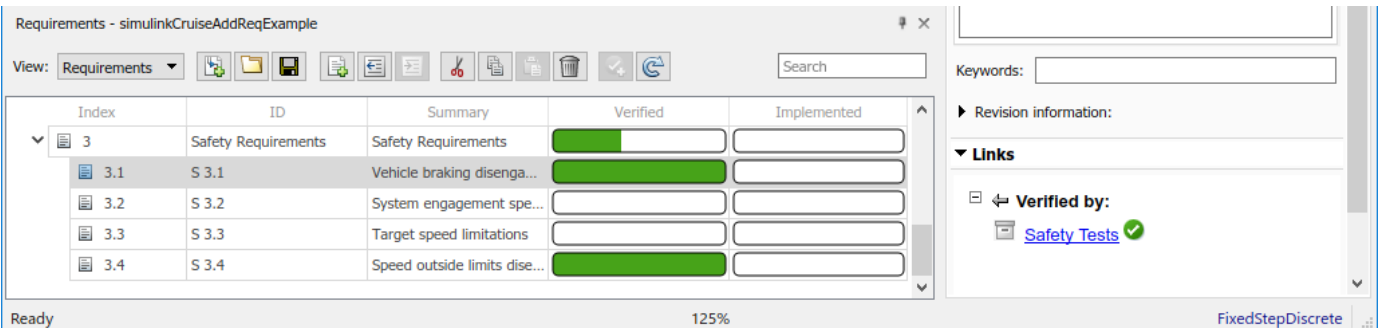
- The `LimitTest` sequence engages the cruise control, then ramps up the vehicle speed until it exceeds the upper limit. It includes the `verify` statement.

```
verify(engaged == false,...
 'verify:limit',...
 'system must disengage when limit exceeded')
```

- 1 Return to the Test Manager. To run the test case, click **Run**.
- 2 When the test finishes, review the results. The Test Manager shows that both assessments pass and the plot provides the detailed results of each `verify` statement.



- 3 Return to the model and refresh the Requirements. The green bar in the **Verified** column indicates that the requirement has been successfully verified.



## Report the Results

- 1 Create a report using a custom Microsoft Word template.
  - a From the Test Manager results, right-click the test case name. Select **Create Report**.
  - b In the Create Test Result Report dialog box, set the options:
    - Title — SafetyTest
    - Results for — All Tests
    - File Format — DOCX
    - For the other options, keep the default selections.
  - c Enter a file name and select a location for the report.
  - d For the **Template File**, select the ReportTemplate.dotx file in the **documents** project folder.
  - e Click **Create**.

- 2 Review the report.
  - a The **Test Case Requirements** section specifies the associated requirements
  - b The **Verify Result** section contains details of the two assessments in the test, and links to the simulation output.

### See Also

#### Related Examples

- “Link to Requirements” (Simulink Test)
- “Validate Requirements Links in a Model” on page 12-27
- “Customize Requirements Traceability Report for Model” on page 12-12

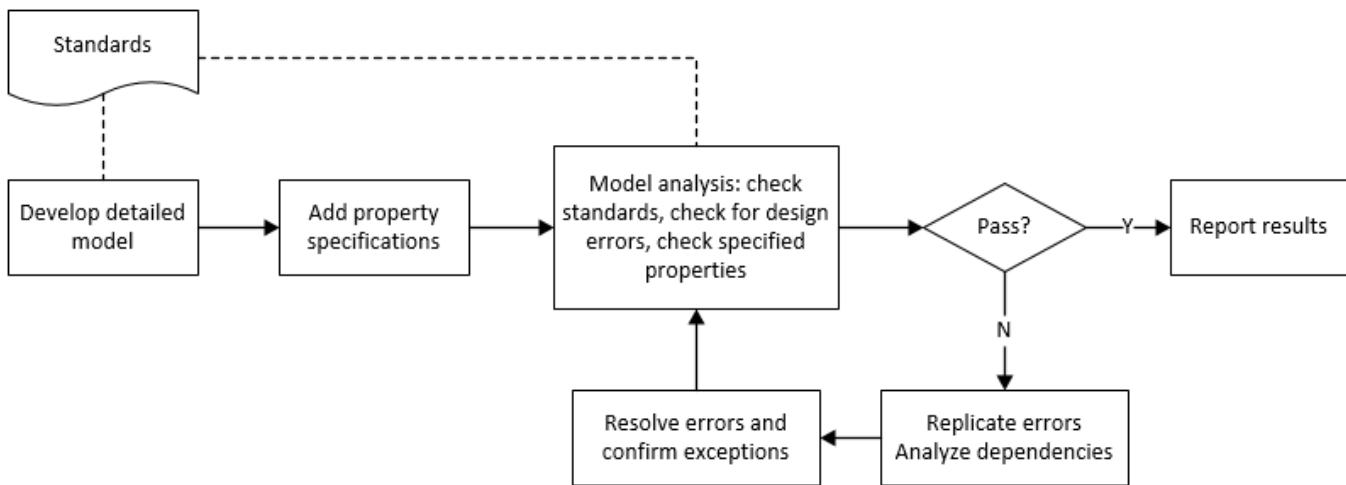
#### External Websites

- Requirements-Based Testing Workflow

# Analyze Models for Standards Compliance and Design Errors

## Standards and Analysis Overview

During model development, check and analyze your model to increase confidence in its quality. Check your model against standards such as MAB style guidelines and high-integrity system design guidelines such as DO-178 and ISO 26262. Analyze your model for errors, dead logic, and conditions that violate required properties. Using the analysis results, update your model and document exceptions. Report the results using customizable templates.



## Check Model for Style Guideline Violations and Design Errors

This example shows how to use the Model Advisor to check a cruise control model for MathWorks® Advisory Board (MAB) style guideline violations and design errors. Select checks and run the analysis on the model. Iteratively debug issues using the Model Advisor and rerun checks to verify that it is in compliance. After passing your selected checks, report results.

### Check Model for MAB Style Guideline Violations

Check that your model complies with MAB guidelines by using the Model Advisor.

- 1 Open the example project. On the command line, enter
 

```
openExample("shared_vnv/CruiseControlVerificationProjectExample");
pr = openProject("SimulinkVerificationCruise");
```
- 2 Open the `simulinkCruiseErrorAndStandardsExample` model.
 

```
open_system simulinkCruiseErrorAndStandardsExample
```
- 3 In the **Modeling** tab, select **Model Advisor**.
- 4 Click **OK** to select `simulinkCruiseErrorAndStandardsExample` from the System Hierarchy.
- 5 Check your model for MAB style guideline violations using Simulink Check.
  - a In the left pane, in the **By Product > Simulink Check > Modeling Standards > MAB Checks** folder, select:

- **Check Indexing Mode**
  - **Check model diagnostic parameters**
- b** Right-click on the **MAB Checks** node and select **Run Checks**.
  - c** To review the configuration parameter settings that violate MAB style guidelines, run the **Check model diagnostic parameters** check.
  - d** The analysis results appear in the right pane on the **Report** tab. Report displays the violation details and the recommended action.
  - e** Click the parameter hyperlinks, which opens the Configuration Parameters dialog box, and update the model diagnostic parameters. Save the model.
  - f** To verify that your model passes, rerun the check. Repeat steps from c to e, if necessary, to reach compliance.
  - g** To generate a results report of the Simulink Check checks, select the **MAB Checks** node, and then, from the toolbar, click **Report**.

### Check Model for Design Errors

While in the Model Advisor, you can also check your model for hidden design errors using Simulink Design Verifier.

- 1** In the left pane, in the **By Products > Simulink Design Verifier** folder, select **Design Error Detection**.
- 2** If not already checked, click the box beside **Design Error Detection**. All checks in the folder are selected.
- 3** From the tool strip, click **Run Checks**.
- 4** After the Model Advisor analysis, from the tool strip, click **Report**. This generates a HTML report of the check analysis.
- 5** In the generated report, click a **Simulink Design Verifier Results Summary** hyperlink. The dialog box provides tools to help you diagnose errors and warnings in your model.
  - a** Review the analysis results on the model. Click the **Compute target speed** subsystem. The Simulink Design Verifier Results Inspector window provides derived ranges that can help you understand the source of an error by identifying the possible signal values.
  - b** Review the harness model or create one if it does not already exist.
  - c** View tests and export test cases.
  - d** Review the analysis report. To see a detailed analysis report, click **HTML** or **PDF**.

### See Also

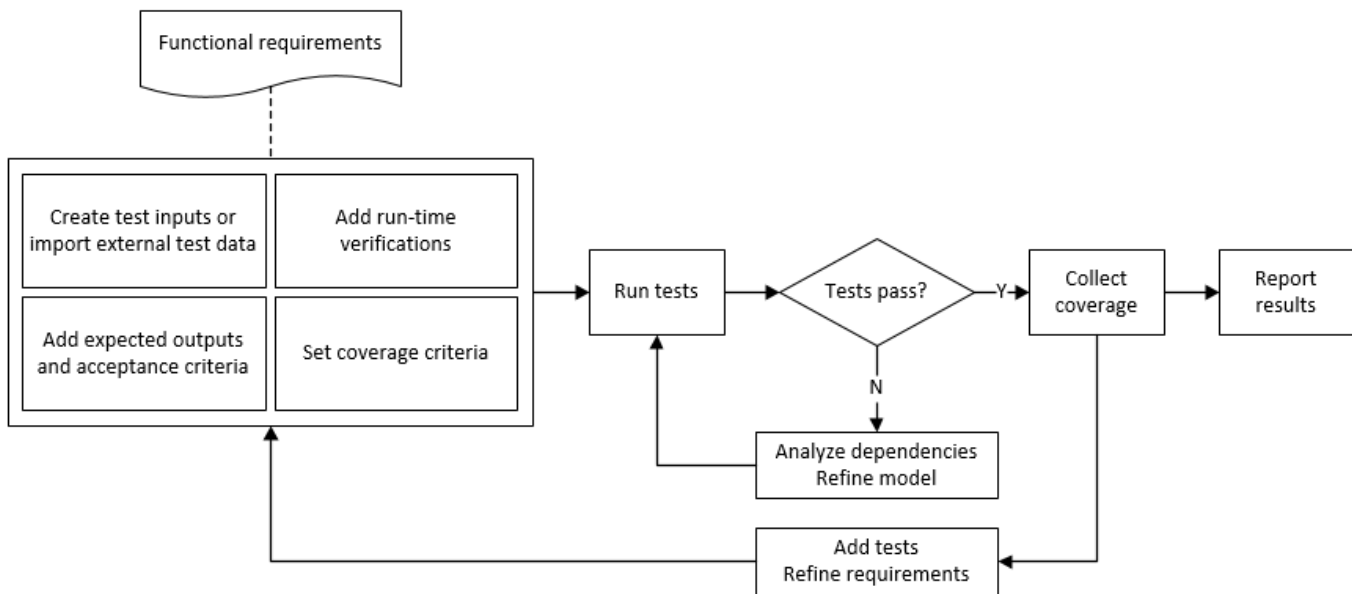
#### Related Examples

- “Check Model Compliance by Using the Model Advisor” (Simulink Check)
- “Collect Model Metrics Using the Model Advisor” (Simulink Check)
- “Analyze Models for Design Errors” (Simulink Design Verifier)
- “Prove Properties in a Model” (Simulink Design Verifier)

## Perform Functional Testing and Analyze Test Coverage

Functional testing begins with building test cases based on requirements. These tests can cover key aspects of your design and verify that individual model components meet requirements. Test cases include inputs, expected outputs, and acceptance criteria.

By collecting individual test cases within test suites, you can run functional tests systematically. To check for regression, add baseline criteria to the test cases and test the model iteratively. Coverage measurement reflects the extent to which these tests have fully exercised the model. Coverage measurement also helps you to add tests and requirements to meet coverage targets.



### Incrementally Increase Test Coverage Using Test Case Generation

This example shows how to perform requirements-based tests for a cruise control model. The tests link to a requirements document. You:

- 1 Run the tests.
- 2 Determine test coverage by using Simulink Coverage.
- 3 Increase coverage with additional tests generated by Simulink Design Verifier.
- 4 Report the results.

#### Open the Test Harness and Model

- 1 Open the project:

```
openExample("shared_vnv/CruiseControlVerificationProjectExample");
pr = openProject("SimulinkVerificationCruise");
```

- 2 Open the model and the test harness. At the command line, enter:

```
open_system simulinkCruiseAddReqExample
sltest.harness.open("simulinkCruiseAddReqExample", "SafetyTest_Harness1")
```

- 3 Load the test suite from “Test Model Against Requirements and Report Results” (Simulink Test) and open the Simulink Test Manager.

```
pf = fullfile(pr.RootFolder,"tests","slReqTests.mldatx");
tf = sltest.testmanager.TestFile(pf);
sltest.testmanager.view
```

- 4 Open the Test Sequence block. The sequence verifies system disengagement when either:
  - The brake pedal is pressed.
  - Speed exceeds a limit.

### Measure Model Coverage

- 1 In the Simulink Test Manager, select the `slReqTests` test file.
- 2 To enable coverage collection, in the right page under **Coverage Settings**:
  - Select **Record coverage for referenced models**.
  - Specify a coverage filter by using **Coverage filter filename**.
  - Select **Decision**, **Condition**, and **MCDC**.
- 3 Click **Run** on the Test Manager toolstrip.
- 4 After the test completes, select **Results**. The test achieves 50% decision coverage, 41% condition coverage, and 25% MCDC coverage.

| ANALYZED MODEL              | REPORT CO... | DECISION | CONDITION | MCDC |
|-----------------------------|--------------|----------|-----------|------|
| simulinkCruiseAddReqExample | 31           | 50%      | 41%       | 25%  |

### Generate Tests to Increase Model Coverage

- 1 Use Simulink Design Verifier to generate additional tests to increase model coverage. In **Results and Artifacts**, select the `slReqTests` test file and open the **Aggregated Coverage Results** section located in the right pane.
- 2 Right-click the test results and select **Add Tests for Missing Coverage**.
- 3 Under **Harness**, choose Create a new harness.
- 4 Click **OK** to add tests to the test suite using Simulink Design Verifier. The model being tested must either be on the MATLAB path or in the working folder.
- 5 On the Test Manager toolstrip, click **Run** to execute the updated test suite. The test results include coverage for the combined test case inputs, achieving increased model coverage.

Alternatively, you can create and use tests to increase coverage programmatically by using `sltest.testmanager.addTestsForMissingCoverage` and `sltest.testmanager.TestOptions`.



## See Also

### Related Examples

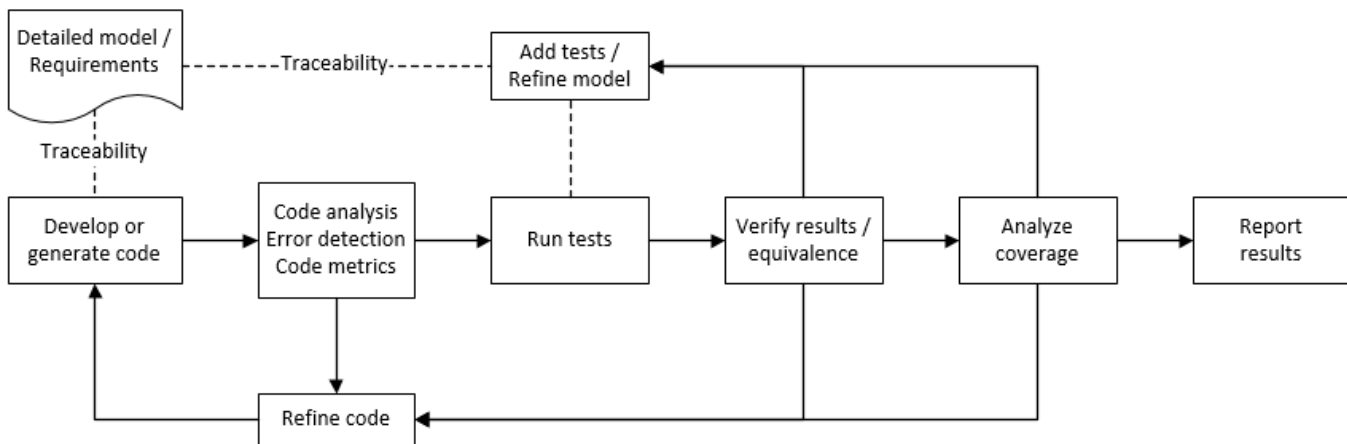
- “Link to Requirements” (Simulink Test)
- “Assess Model Simulation Using verify Statements” (Simulink Test)
- “Compare Model Output to Baseline Data” (Simulink Test)
- “Generate Test Cases for Model Decision Coverage” (Simulink Design Verifier)
- “Increase Test Coverage for a Model” (Simulink Test)

## Analyze Code and Test Software-in-the-Loop

### Code Analysis and Testing Software-in-the-Loop Overview

You can analyze code to detect errors, check standards compliance, and evaluate key metrics such as length and cyclomatic complexity. For handwritten code, you typically check for run-time errors with static code analysis and run test cases that evaluate the code against requirements and evaluate code coverage. Based on the results, you refine the code and add tests.

In this example, you generate code and demonstrate that the code execution produces equivalent results to the model by using the same test cases and baseline results. Then you compare the code coverage to the model coverage. Based on test results, add tests and modify the model to regenerate code.



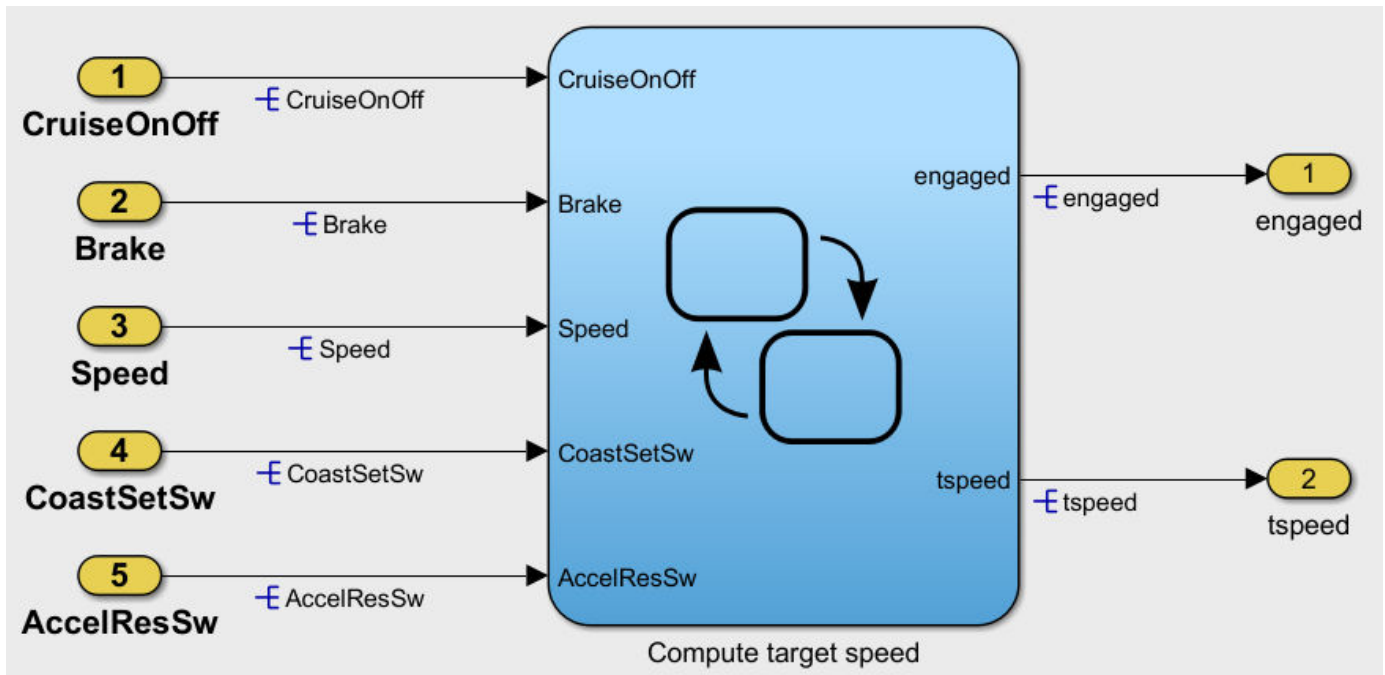
### Analyze Code for Defects, Metrics, and MISRA C:2012

This workflow describes how to check if your model produces MISRA™ C:2012 compliant code and how to check your generated code for code metrics and defects. To produce more MISRA compliant code from your model, you use the code generation and Model Advisor. To check whether the code is MISRA compliant, you use the Polyspace MISRA C:2012 checker and report generation capabilities. For this example, you use the model `simulinkCruiseErrorAndStandardsExample`. To open the model:

- 1 Open the project.

```
openExample("shared_vnv/CruiseControlVerificationProjectExample");
pr = openProject("SimulinkVerificationCruise");
```

- 2 From the project, open the model `simulinkCruiseErrorAndStandardsExample`.

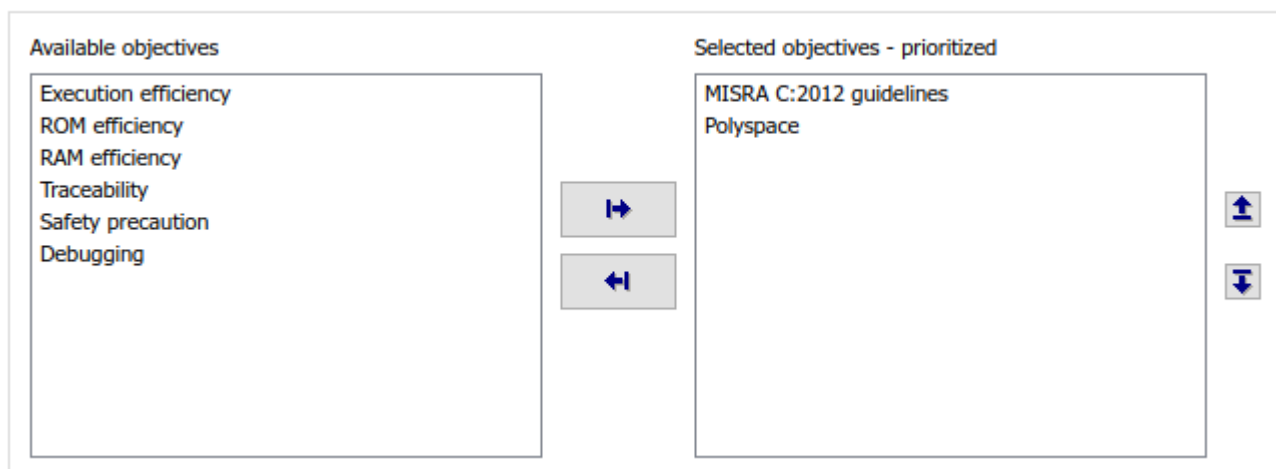


### Run Code Generator Checks

Check your model by using the Code Generation Advisor. Configure code generation parameters to generate code more compliant with MISRA C and more compatible with Polyspace.

- 1 Right-click Compute target speed and select **C/C++ Code > Code Generation Advisor**.
- 2 Select the Code Generation Advisor folder. In the right pane, move Polyspace to **Selected objectives - prioritized**. The MISRA C:2012 guidelines objective is already selected.

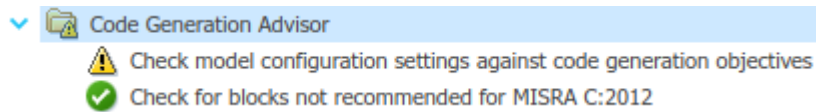
Code Generation Objectives (System target file: ert.tlc)



- 3 Click **Run Selected Checks**.

The Code Generation Advisor checks whether the model includes blocks or configuration settings that are not recommended for MISRA C:2012 compliance and Polyspace code analysis. For this

model, the check for incompatible blocks passes, but some configuration settings are incompatible with MISRA compliance and Polyspace checking.



- 4 Click the check that did not pass. Accept the parameter changes by selecting **Modify Parameters**.
- 5 Rerun the check by selecting **Run This Check**.

### Run Model Advisor Checks

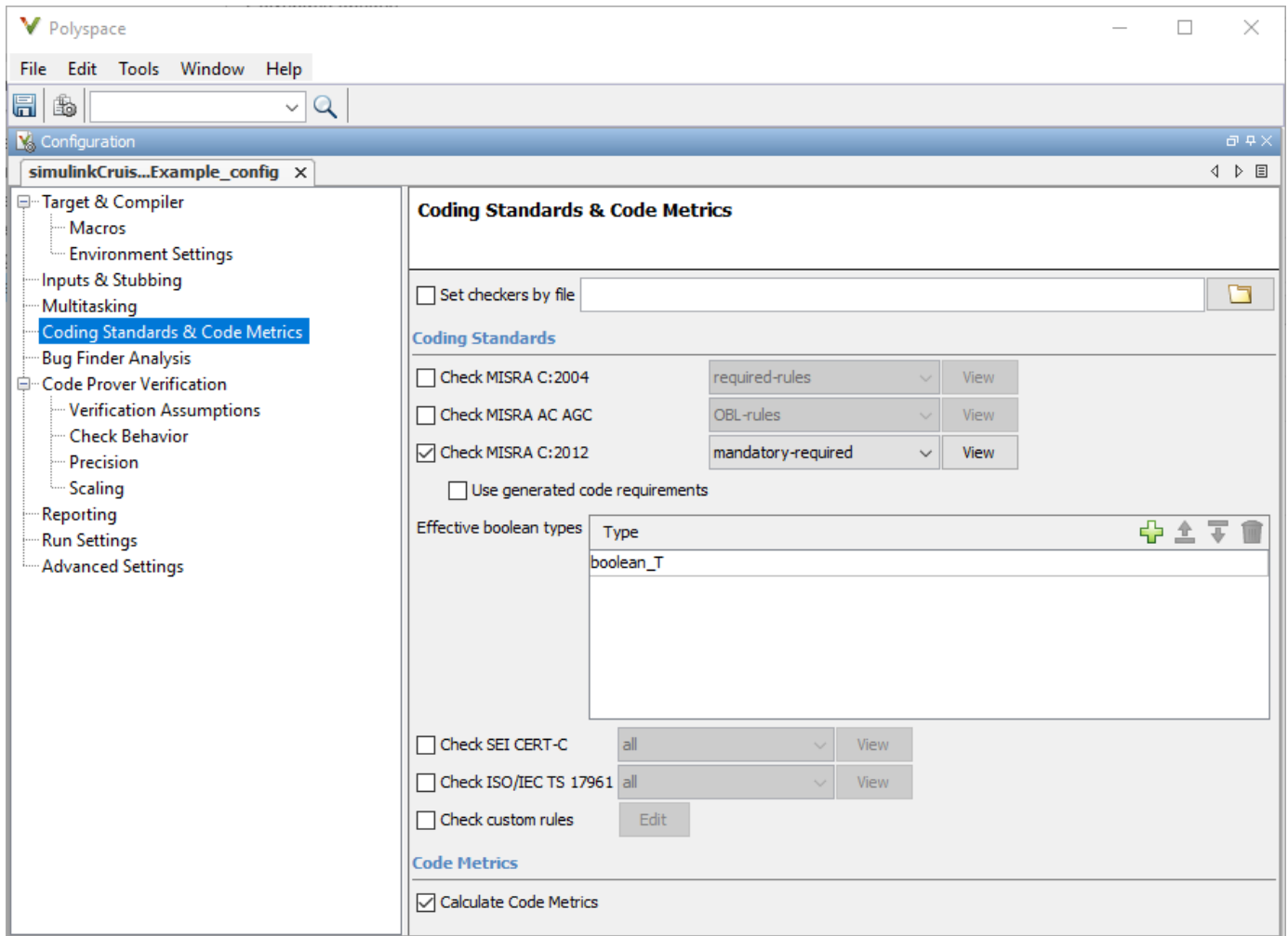
Before you generate code from your model, use the Model Advisor to check your model for MISRA C and Polyspace compliance. This example shows you how to use the Model Advisor to check your model before generating code.

- 1 At the bottom of the Code Generation Advisor window, select **Model Advisor**.
- 2 Under the **By Task** folder, select the **Modeling Standards for MISRA C:2012** advisor checks.
- 3 Click **Run Checks** and review the results.
- 4 If any of the tasks fail, make the suggested modifications and rerun the checks until the MISRA modeling guidelines pass.

### Generate and Analyze Code

After you have done the model compliance checking, you can generate the code. With Polyspace, you can check your code for compliance with MISRA C:2012 and generate reports to demonstrate compliance with MISRA C:2012.

- 1 In the Simulink editor, right-click Compute target speed and select **C/C++ Code > Build This Subsystem**.
- 2 Use the default settings for the tunable parameters and select **Build**.
- 3 After the code is generated, in the Simulink Editor, right-click Compute target speed and select **Polyspace > Options**.
- 4 Click **Configure** to choose more advanced Polyspace analysis options in the Polyspace configuration window.



- 5 On the left pane, click **Coding Standards & Code Metrics**, then select **Calculate Code Metrics** to enable code metric calculations for your generated code.
- 6 Save and close the Polyspace configuration window.
- 7 From your model, right-click Compute target speed and select **Polyspace > Verify > Code Generated For Selected Subsystem**.

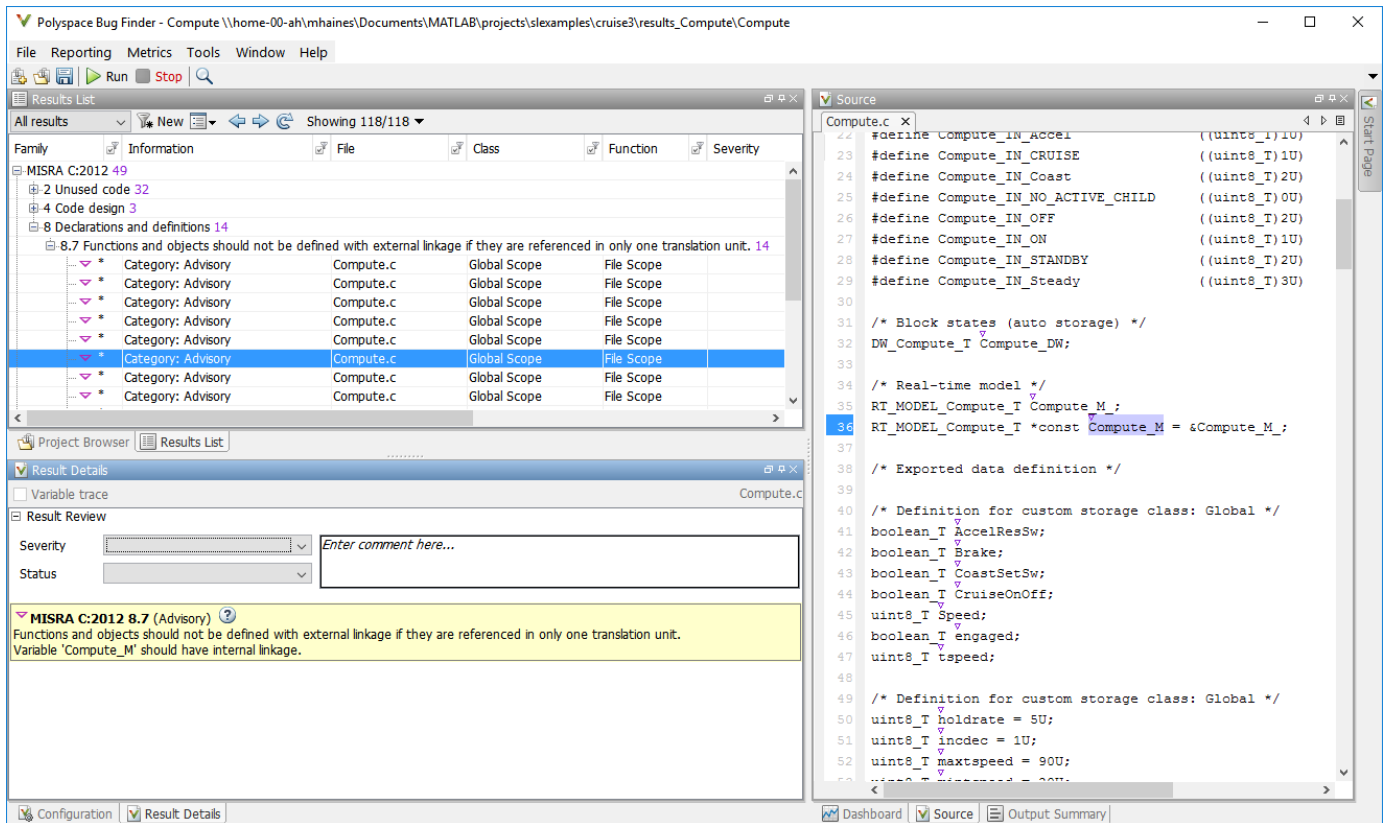
Polyspace Bug Finder analyzes the generated code for a subset of MISRA checks. You can see the progress of the analysis in the MATLAB Command Window. After the analysis finishes, the Polyspace environment opens.

## Review Results

The Polyspace environment shows you the results of the static code analysis.

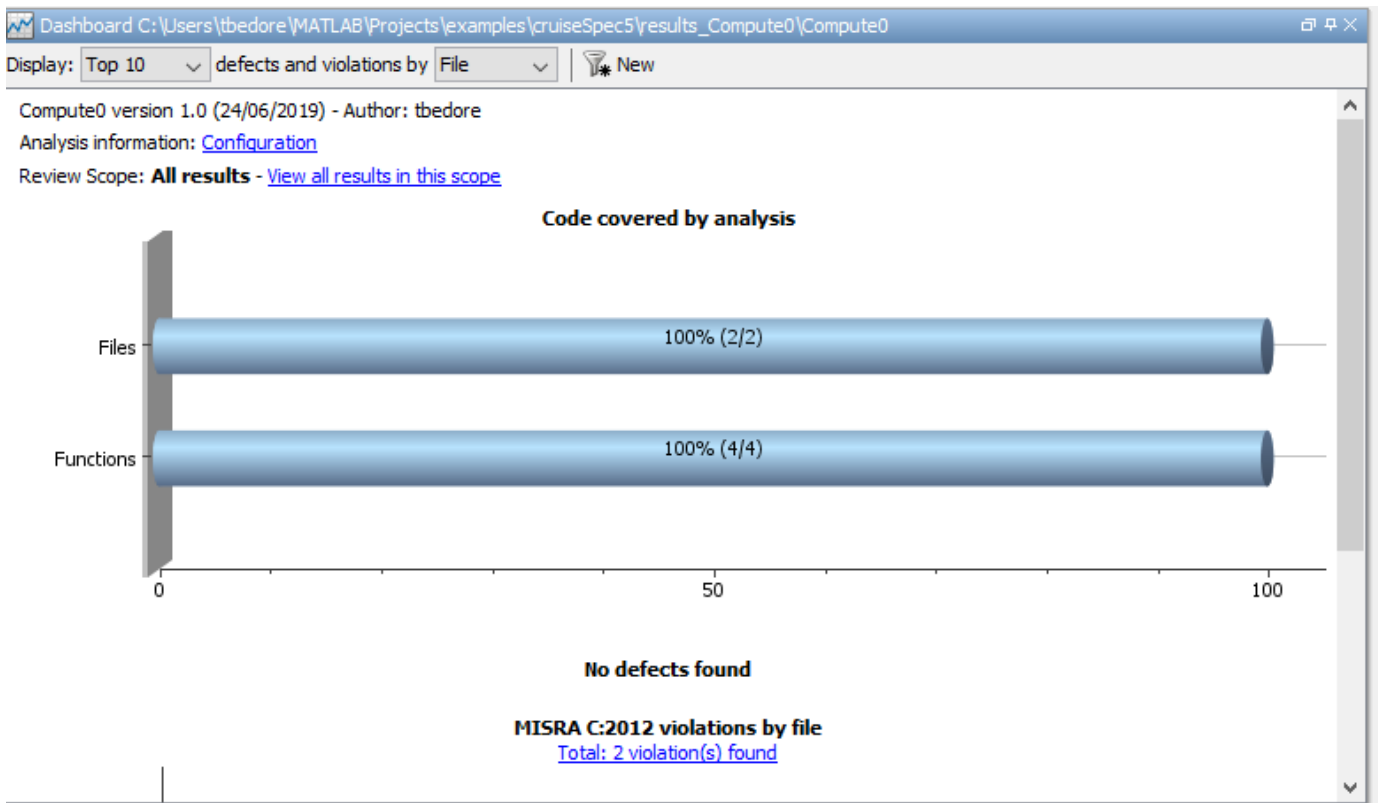
- 1 Expand the tree for rule 8.7 and click through the different results.

Rule 8.7 states that functions and objects should not be global if the function or object is local. As you click through the 8.7 violations, you can see that these results refer to variables that other components also use, such as `CruiseOnOff`. You can annotate your code or your model to justify every result. Because this model is a unit in a larger program, you can also change the configuration of the analysis to check only a subset of MISRA rules.



- 2 In your model, right-click Compute target speed and select **Polyspace > Options**.
- 3 Set the **Settings from** option to Project configuration to choose a subset of MISRA rules in the Polyspace configuration.
- 4 Click **Configure**.
- 5 In the Polyspace window, on the left pane, click **Coding Standards & Code Metrics**. Then select **Check MISRA C:2012** and, from the drop-down list, select **single-unit-rules**. Now Polyspace checks only the MISRA C:2012 rules that are applicable to a single unit.
- 6 Save and close the Polyspace configuration window.
- 7 Rerun the analysis with the new configuration.

The rules Polyspace showed previously were found because the model was analyzed by itself. When you limited the rules Polyspace checked to the single-unit subset, Polyspace found only two violations.



When you integrate this model with its parent model, you can add the rest of the MISRA C:2012 rules.

### Generate Report

To demonstrate compliance with MISRA C:2012 and report on your generated code metrics, you must export your results. If you want to generate a report every time you run an analysis, see [Generate report \(Polyspace Bug Finder\)](#).

- 1 If they are not open already, open your results in the Polyspace environment.
- 2 From the toolbar, select **Reporting > Run Report**.
- 3 Select **BugFinderSummary** as your report type.
- 4 Click **Run Report**.

The report is saved in the same folder as your results.

- 5 To open the report, select **Reporting > Open Report**.

### Test Code Against Model Using Software-in-the-Loop Testing

You previously showed that the model functionality meets its requirements by running test cases based on those requirements. Now run the same test cases on the generated code to show that the code produces equivalent results and fulfills the requirements. Then compare the code coverage to the model coverage to see the extent to which the tests exercised the generated code.

- 1 In MATLAB, in the project window, open the `tests` folder, then open `SILTests.mldatx`. The file opens in the Test Manager.

- 2 Review the test case. On the **Test Browser** pane, navigate to SIL Equivalence Test Case. This equivalence test case runs two simulations for the `simulinkCruiseErrorAndStandardsExample` model using a test harness.
  - Simulation 1 is a model simulation in normal mode.
  - Simulation 2 is a software-in-the-loop (SIL) simulation. For the SIL simulation, the test case runs the code generated from the model instead of running the model.

The equivalence test logs one output signal and compares the results from the simulations. The test case also collects coverage measurements for both simulations.

- 3 Run the equivalence test. Select the test case and click **Run**.
- 4 Review the results in the Test Manager. In the **Results and Artifacts** pane, select **SIL Equivalence Test Case** to see the test results. The test case passed and the results show that the code produced the same results as the model for this test case.

| ANALYZED MODEL                                      | REPORT | SIM MODE   | COM... | DECISION | CONDITION | MCDC | EXECUTION | FUNCTION | FUNCTION... |
|-----------------------------------------------------|--------|------------|--------|----------|-----------|------|-----------|----------|-------------|
| <code>dlv_su32.c</code>                             |        | SIL        | 2      | 0%       | --        | --   | 0%        | 0%       | --          |
| <code>simulinkCruiseErrorAndStandardsExample</code> |        | ModelRe... | 22     | 54%      | 44%       | 17%  | 70%       | 100%     | 33%         |
| <code>simulinkCruiseErrorAndStandardsExample</code> |        | Normal     | 31     | 50%      | 41%       | 25%  | --        | --       | --          |

- 5 Expand the **Coverage Results** section of the results. The coverage measurements show the extent to which the test case exercised the model and the code. When you run multiple test cases, you can view aggregated coverage measurements in the results for the whole run. Use the coverage results to add tests and meet coverage requirements, as shown in “Perform Functional Testing and Analyze Test Coverage” (Simulink Check).

You can also test the generated code on your target hardware by running a processor-in-the-loop (PIL) simulation. By adding a PIL simulation to your test cases, you can compare the test results and coverage results from your model to the results from the generated code as it runs on the target hardware. For more information, see “Code Verification Through Software-in-the-Loop and Processor-in-the-Loop Execution” (Embedded Coder).



## See Also

### Related Examples

- “Run Polyspace Analysis on Code Generated with Embedded Coder” (Polyspace Bug Finder)
- “Test Two Simulations for Equivalence” (Simulink Test)
- “Export Test Results” (Simulink Test)

