

Simulink® Requirements™

User's Guide



MATLAB® & SIMULINK®

R2021b



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

Simulink® Requirements™ User's Guide

© COPYRIGHT 2017–2021 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)

Requirements Definition

Author Requirements in 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
Import Requirements from Third-Party Applications	1-7
Add Requirements to the Path	1-7
Select an Import Mode	1-7
Differences Between Importing and Direct Linking	1-10
Import Requirements from Microsoft Office Documents	1-11
Import Options for Microsoft Word Documents	1-11
Import Options for Microsoft Excel Spreadsheets	1-13
Import Requirements from ReqIF Files	1-16
Choosing an Import Mapping	1-16
Importing Requirements	1-18
Importing Links	1-22
Mapping ReqIF Attributes in Simulink Requirements	1-23
Import Requirements from IBM DOORS Next	1-27
Configure IBM DOORS Next Session	1-27
Import DOORS Next Requirements	1-27
Update Referenced Requirements	1-30
Navigate from Referenced Requirements to Requirements in DOORS Next	1-31
Linking with Referenced Requirements	1-32
Import Requirements from IBM Rational DOORS	1-33
Configure IBM Rational DOORS Session	1-33
Import an Entire Requirements Module	1-33
Import a Subset of Requirements from a Module	1-35
Update the Requirement Set	1-35
Navigate Between Referenced Requirements and Requirements in IBM Rational DOORS	1-36
Export Requirements to ReqIF Files	1-38
Choosing an Export Mapping	1-38
Exporting Requirements	1-40
Exporting Links	1-41

Define Requirements Hierarchy	1-43
Requirement Sets	1-43
Custom Attributes of Requirement Sets	1-43
Create Requirement Set File by Using the Simulink® Requirements™ API	1-45
Customize Requirements with Custom Attributes	1-48
Define Custom Attributes for Requirement Sets	1-48
Set Custom Attribute Values for Requirements	1-49
Edit Custom Attributes	1-50
Custom Attributes for Referenced Requirements	1-50
Import Custom Attributes	1-50
Limitations	1-51
Update Imported Requirements	1-52
Update a Requirement Set	1-52
Update Requirements with Change Tracking Enabled	1-52
Considerations for Microsoft Word Documents	1-53
Import and Update Requirements from a Microsoft Word Document ..	1-54
Export Requirement Sets and Link Sets to Previous Versions of Simulink Requirements	1-56
Export Requirement Sets	1-56
Export Link Sets	1-56
Use Command-line API to Document Simulink Model in Requirements Editor	1-57
Round-Trip Importing and Exporting for ReqIF Files	1-73
Considerations for Importing Requirements	1-73
Edit Imported Content	1-73
Link Requirements to Items in MATLAB and Simulink	1-75
Considerations for Exporting Requirements	1-75
Best Practices and Guidelines for ReqIF Round Trip Workflows	1-77
Managing Requirement Custom IDs	1-77
Guidelines for Updating Referenced Requirements Content	1-77
Guidelines for Editing Referenced Requirements Content	1-77
Guidelines for Adding Details to Imported Requirements	1-77
Guidelines for Exporting Requirements to ReqIF Files	1-77
Manage Custom Attributes for Requirements by Using the Simulink® Requirements™ API	1-79
Create and Edit Attribute Mappings	1-84
Edit the Attribute Mapping for Imported Requirements	1-84
Specify Default ReqIF Requirement Type	1-86
Specify ReqIF Template	1-86
Import Requirements from IBM Rational DOORS by using the API	1-87

Link Blocks and Requirements	2-2
Work with Simulink Annotations	2-4
Track Requirement Links with a Traceability Matrix	2-5
Generate a Traceability Matrix	2-5
Modify the Traceability Matrix View	2-9
Work with Links in the Traceability Matrix	2-14
Export the Traceability Matrix	2-17
Work Programmatically with a Traceability Matrix	2-17
Visualize Links with a Traceability Diagram	2-18
Generate a Traceability Diagram	2-18
Use the Traceability Diagram	2-22
Modify the Traceability Diagram View	2-23
Export the Diagram	2-25
Assess Allocation and Impact	2-27
Assess Requirements Allocation	2-27
Visualize Change Propagation	2-29
Requirement Links	2-32
Linkable Items	2-32
Link Types	2-33
Review Requirement Links	2-36
Resolve Links	2-36
Load Link Information	2-36
Unload Link Information	2-38
Delete a Link Set	2-39
Define Custom Requirement and Link Types	2-40
Create and Register Custom Requirement and Link Types	2-40
Inherited Functionality from the Built-In Type	2-40
Set the Type in the Requirements Editor	2-41
Customize Links with Custom Attributes	2-43
Define Custom Attributes for Link Sets	2-43
Set Custom Attribute Values for Links	2-44
Edit Custom Attributes	2-45
Requirements Consistency Checks	2-46
Check Requirements Consistency in Model Advisor	2-46
Manage Navigation Backlinks in External Requirements Documents ..	2-50
Insert Backlinks in External Requirements Documents	2-50
Use Command-line API to Update or Repair Requirements Links	2-52
Manage Custom Attributes for Links by Using the Simulink® Requirements™ API	2-65
Make Requirements Fully Traceable with a Traceability Matrix	2-70

Modeling System Architecture of Small UAV	2-84
--	-------------

Requirements-Based Verification

3

Review Requirements Implementation Status	3-2
Implement Functional Requirements by Linking to Model Elements	3-2
View the Implementation Status	3-3
Review Requirements Verification Status	3-6
Verify Functional Requirements	3-6
Display Verification Status	3-7
Update Verification Status by Running Tests or Analyses	3-8
Include Verification Status in Report	3-8
Validate Requirements by Analyzing Model Properties	3-9
Justify Requirements	3-16
Linking to a Test Script	3-19
Linking to a Test Script Using the Outgoing Links Editor	3-19
Linking to a Test Script Using the API	3-22
Integrating Results from a MATLAB Unit Test Case	3-25
Include Results from External Sources in Verification Status	3-27
How to Populate Verification Results from External Sources	3-27
Linking to a Result File	3-30
Open Example Files	3-30
Create and Register a Custom Link Type	3-31
Create a Requirement Link	3-32
View the Verification Status	3-34
Integrating Results from a Custom-Authored MATLAB Script as a Test	3-36
Integrating Results from an External Result file	3-40
Integrating results from a custom authored MUnit script as a test	3-44
Fix Requirements-Based Testing Issues	3-48

Change Tracking and Team-Based Workflows

4

Requirements-Based Development in Projects	4-2
Organizing Requirements, Models, and Tests	4-2

Track Changes to Requirement Links	4-3
Enable Change Tracking for Requirement Links	4-3
Review Changes to Requirements	4-3
Resolve Change Issues	4-5
Add Comments to Links	4-6
Manually Check for Using Links Change Tracking	4-7
Compare Requirements Sets	4-8
Compare Two .slreqx Simulink Requirements Sets	4-8
Review Changes in Source-Controlled Files	4-8
Compare Link Sets	4-9
Report Requirements Information	4-10
Report Navigation Links	4-12
Three-way AutoMerge Solution for Requirement Set and Link Set	4-13
Configure Git environment for AutoMerge	4-13
Select and Merge Branches in Git	4-13
Limitations	4-13
Merge Requirement Set and Link Set Files	4-15

Requirements Management Interface Setup

5

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

6

Link to Requirements in Microsoft Word Documents	6-2
Link a Requirement in Word to a Simulink Block	6-2
Link to Requirements in Excel Workbooks	6-7
Navigate from a Model Object to Requirements in an Excel Workbook ...	6-7
Create Requirements Links to the Workbook	6-7
Link Multiple Model Objects to a Microsoft Excel Workbook	6-8
Change Requirements Links	6-8
Navigate to Requirements in Microsoft Office Documents from Simulink	6-10
Enable Linking from Microsoft Office Documents to Simulink Objects ...	6-10
Insert Navigation Objects in Microsoft Office Documents	6-11
Customize Microsoft Office Navigation Objects	6-11
Navigate Between Microsoft Office Requirement and Model	6-12
Managing Requirements for Fault-Tolerant Fuel Control System (Microsoft Office)	6-14

Requirements Traceability with IBM Rational DOORS

7

Configure Simulink Requirements for IBM Rational DOORS Software ..	7-2
Manually Install Additional Files for DOORS Software	7-2
Address DXL Errors	7-3
Link with Requirements in IBM DOORS Next	7-4
Link and Trace Requirements with IBM DOORS Next	7-26
Configure IBM DOORS Next Session	7-26
Linking with Referenced Requirements	7-26
Directly Linking DOORS Next Requirements	7-27
Specifying and Updating the IBM DOORS Next Configuration	7-32
Navigate to Requirements in IBM Rational DOORS Databases from Simulink	7-35
Enable Linking from IBM Rational DOORS Databases to Simulink Objects	7-35
Insert Navigation Objects into IBM Rational DOORS Requirements	7-35
Navigate Between IBM Rational DOORS Requirement and Model Object	7-37
Why Add Navigation Objects to IBM Rational DOORS Requirements? ...	7-38
Customize IBM Rational DOORS Navigation Objects	7-38
Synchronize Simulink Models with IBM Rational DOORS Databases by using Surrogate Modules	7-39
Synchronize a Simulink Model to Create a Surrogate Module	7-39

Create Links Between Surrogate Module and Formal Module in an IBM Rational DOORS Database	7-40
Resynchronize IBM Rational DOORS Surrogate Module to Reflect Model Changes	7-41
Navigate with the Surrogate Module	7-41
Customize IBM Rational DOORS Synchronization	7-43
Synchronization with IBM Rational DOORS Surrogate Modules	7-48
Advantages of Synchronizing Your Model with a Surrogate Module	7-49
Working with IBM Rational DOORS 9 Requirements	7-50
Managing Requirements for Fault-Tolerant Fuel Control System (IBM Rational DOORS)	7-59

Simulink Traceability Between Model Objects

8

Link Model Objects	8-2
Link Objects in the Same Model	8-2
Link Objects in Different Models	8-2
Link Test Cases to Requirements Documents	8-3
Establish Requirements Traceability for Testing	8-3
Link Simulink Data Dictionary Entries to Requirements	8-7
Link Signal Builder Blocks to Requirements and Simulink Model Objects	8-9
Link Signal Builder Blocks to Requirements Documents	8-9
Link Signal Builder Blocks to Model Objects	8-10
Requirements Links for Library Blocks and Reference Blocks	8-13
Introduction to Library Blocks and Reference Blocks	8-13
Library Blocks and Requirements	8-13
Copy Library Blocks with Requirements	8-13
Manage Requirements on Reference Blocks	8-13
Manage Requirements Inside Reference Blocks	8-14
Links from Requirements to Library Blocks	8-15
Navigate to Requirements from Model	8-16
Navigate from Model Object	8-16
Navigate from System Requirements Block	8-16
Link to Requirements Modeled in Simulink	8-18

9

Requirements Traceability for MATLAB Code Lines	9-2
Link MATLAB Code Lines to Requirements in a Requirement Set	9-2
Link MATLAB Code Lines to Requirements Information in External Documents	9-2
Enable or Disable Traceability Links Highlighting for MATLAB Code	9-3
Remove Traceability Links from MATLAB Code Lines	9-4
Traceability for MATLAB Code Lines	9-4
 Associate Traceability Information with MATLAB Code Lines in Simulink	 9-6

URL and Custom Traceability

10

Requirement Links and Link Types	10-2
Requirements Traceability Links	10-2
Supported Model Objects for Requirements Linking	10-2
Links and Link Types	10-2
Link Type Properties	10-3
Outgoing Links Editor	10-6
 Custom Link Types	 10-8
Create a Custom Requirements Link Type	10-8
Implement Custom Link Types	10-13
Why Create a Custom Link Type?	10-14
Custom Link Type Functions	10-14
Custom Link Type Registration	10-15
Custom Link Type Synchronization	10-15
 Implement RMI Extension for Support of Custom Document Type ...	 10-17

Review and Maintain Requirements Links

11

Highlight Model Objects with Requirements	11-2
Highlight Model Objects with Requirements Using Model Editor	11-2
Highlight Model Objects with Requirements Using Model Explorer	11-3
 Navigate to Simulink Objects from External Documents	 11-4
Provide Unique Object Identifiers	11-4
Use the rmiobjnavigate Function	11-4
Determine the Navigation Command	11-4
Use the ActiveX Navigation Control	11-4
Typical Code Sequence for Establishing Navigation Controls	11-5

View Requirements Details for a Selected Block	11-6
Identify Blocks with Links	11-6
Configure Settings	11-6
View Requirements Details	11-6
Create a Requirement Annotation	11-7
Generate Code for Models with Requirements Links	11-8
How Requirements Information Is Included in Generated Code	11-9
Create and Customize Requirements Traceability Reports	11-10
Create Requirements Traceability Report for Model	11-10
Customize Requirements Traceability Report for Model	11-11
Create Requirements Traceability Report for A Project	11-25
Validate Requirements Links	11-26
Validate Requirements Links in a Model	11-26
Validate Requirements Links in a Requirements Document	11-30
Validation of Requirements Links	11-32
Delete Requirements Links from Simulink Objects	11-34
Delete a Single Link from a Simulink Object	11-34
Delete All Links from a Simulink Object	11-34
Delete All Links from Multiple Simulink Objects	11-34
Document Path Storage	11-35
Relative (Partial) Path Example	11-35
Relative (No) Path Example	11-35
Absolute Path Example	11-35
How to Include Linked Requirements Details in Generated Report ...	11-37
Managing Requirements Without Modifying Simulink Model Files ...	11-44

Requirements Management Interface

12

Verification and Validation

13

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 a Model 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 Simulink” on page 1-2
- “Requirement Types” on page 1-6
- “Import Requirements from Third-Party Applications” on page 1-7
- “Import Requirements from Microsoft Office Documents” on page 1-11
- “Import Requirements from ReqIF Files” on page 1-16
- “Import Requirements from IBM DOORS Next” on page 1-27
- “Import Requirements from IBM Rational DOORS” on page 1-33
- “Export Requirements to ReqIF Files” on page 1-38
- “Define Requirements Hierarchy” on page 1-43
- “Create Requirement Set File by Using the Simulink® Requirements™ API” on page 1-45
- “Customize Requirements with Custom Attributes” on page 1-48
- “Update Imported Requirements” on page 1-52
- “ Import and Update Requirements from a Microsoft Word Document” on page 1-54
- “Export Requirement Sets and Link Sets to Previous Versions of Simulink Requirements” on page 1-56
- “Use Command-line API to Document Simulink Model in Requirements Editor” on page 1-57
- “Round-Trip Importing and Exporting for ReqIF Files” on page 1-73
- “Best Practices and Guidelines for ReqIF Round Trip Workflows” on page 1-77
- “Manage Custom Attributes for Requirements by Using the Simulink® Requirements™ API” on page 1-79
- “Create and Edit Attribute Mappings” on page 1-84
- “Import Requirements from IBM Rational DOORS by using the API” on page 1-87

Author Requirements in Simulink

In this section...

“Author and Edit Requirements Content by Using Microsoft Word” on page 1-4

“Customize Requirements Browser View” on page 1-4

“Filter Requirements Content” on page 1-4

In Simulink® Requirements™, 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.

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.

Add these requirements to a model called `crs_controller`.

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

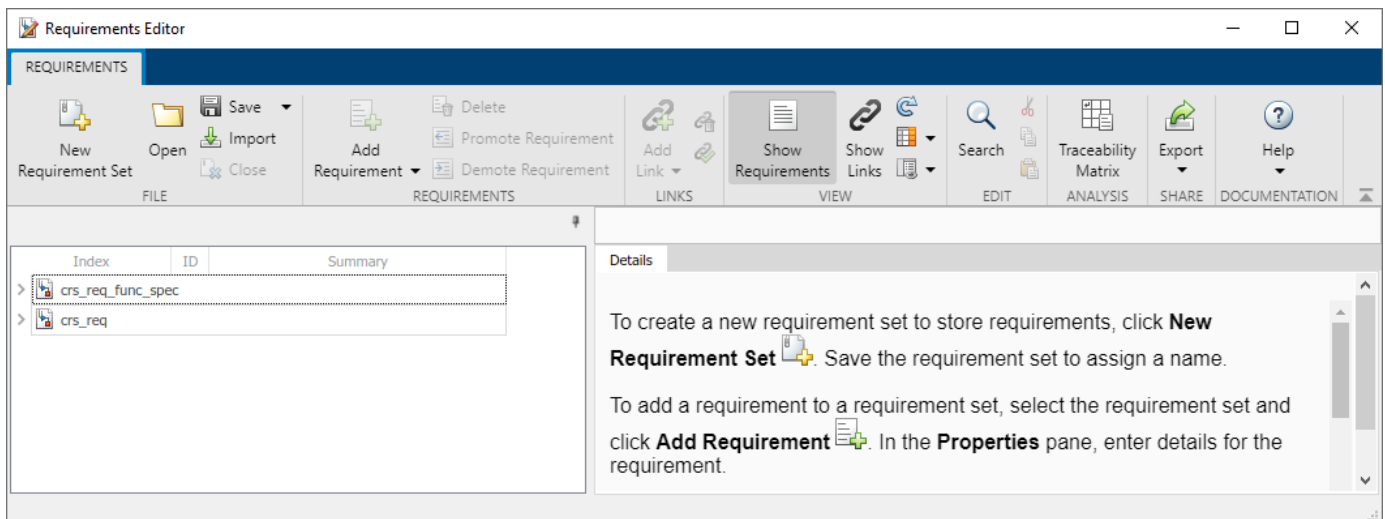
```
slreqCCProjectStart
```

- 2 Open the model. At the command prompt, enter:

```
open_system('models/crs_controller')
```

- 3 Open the Requirements Editor. In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Requirements Editor**.

The Requirements Editor displays the requirements in the Requirements Browser arranged by requirement set. The `crs_controller` model has two requirement sets: `crs_req_func_spec` and `crs_req`.



- 4 Add a requirement set in the Requirements Browser. 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 **Details** 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 Simulink Requirements.

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 the role of the requirement. The requirement type refers to the `slreq.Requirement` object Type property value or the `slreq.Reference` object Type property value, depending on the object type.

When you create or import requirements in Simulink Requirements, you can specify the requirement type in the Requirements Editor in the **Type** list, which is in the **Details** pane, under **Properties**.

Simulink Requirements provides these built-in requirement types:

- **Functional:** Classify requirements that are meant to be implemented or verified in your Model-Based Design workflow. Functional requirements contribute to the Implementation or Verification status of the requirement set that they are in.
- **Container:** Group requirements. Container requirements do not contribute to the Implementation or Verification status of the requirement set that they are in. However, all of the functional requirements under a container requirement contribute to the status.
- **Informational:** Provide supplemental information. Informational requirements and all requirements under them do not contribute to the Implementation or Verification status of the requirement set that they are in.

For more information about implementation and verification status, see “Review Requirements Implementation Status” on page 3-2 and “Review Requirements Verification Status” on page 3-6.

You can also define custom requirement types. Custom requirement types must be a subtype of one of the built-in types. 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 “Define Custom Requirement and Link Types” on page 2-40.

See Also

More About

- “Review Requirements Implementation Status” on page 3-2
- “Review Requirements Verification Status” on page 3-6
- “Define Custom Requirement and Link Types” on page 2-40

Import Requirements from Third-Party Applications

You can author requirements in third-party applications and import them to Simulink Requirements. When you import requirements, you can migrate the requirements and manage them in Simulink Requirements, 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 1-11.
- IBM® Rational® DOORS®. See “Import Requirements from IBM Rational DOORS” on page 1-33.
- IBM DOORS Next. See “Import Requirements from IBM DOORS Next” on page 1-27.
- Applications that use the Requirements Interchange Format (ReqIF™). See “Import Requirements from ReqIF Files” on page 1-16.

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 Simulink Requirements path preference to always use the relative path.

For more information on setting path preferences for requirements documents, see “Document Path Storage” on page 11-35.

Select an Import Mode

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

When you migrate your requirements to Simulink Requirements, 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 Simulink Requirements 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 Simulink Requirements until you update them. Simulink Requirements notifies you when a newer version of the source document is available.

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

Migrate Requirements to Simulink Requirements

If you want to migrate your requirements from the external requirements management application to Simulink Requirements, 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)
- Requirement Identification:** Use bookmarks to identify items and serve as custom IDs (checked)
- Destination(s):** Requirement Set: \examples\CruiseRequirementsExample\crs_req.slreqx; Allow updates from external source (unchecked)

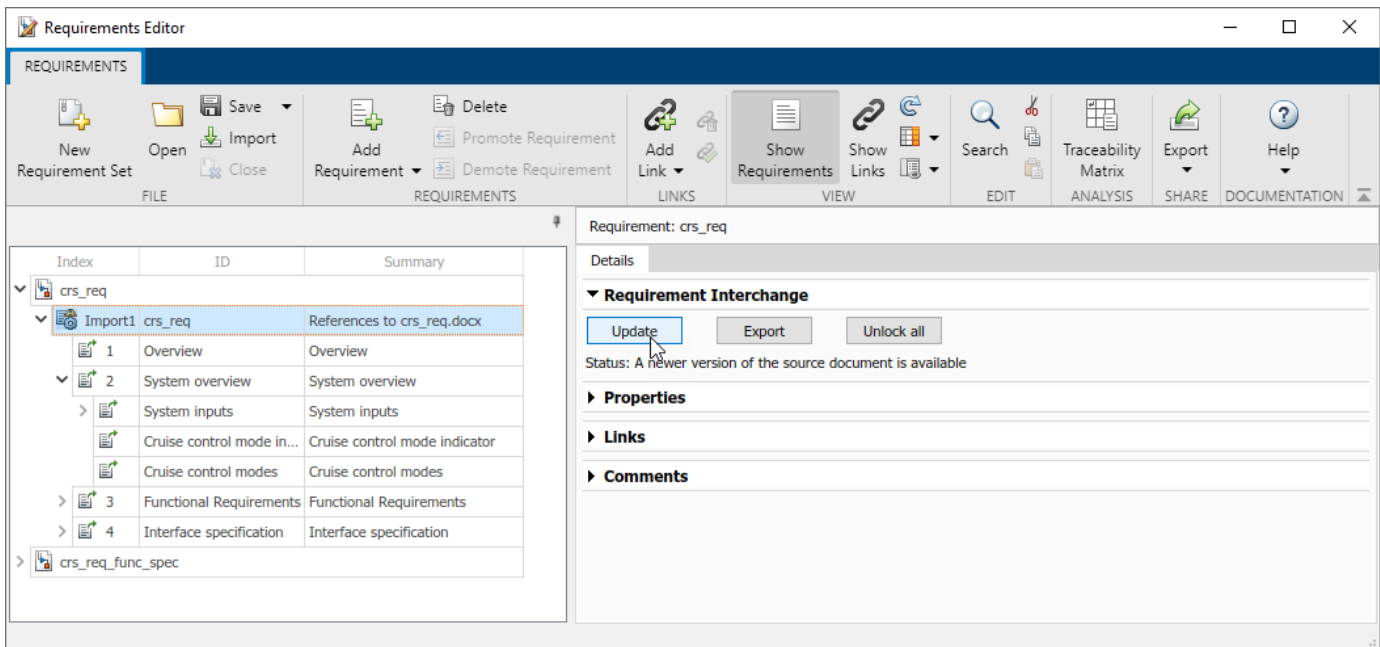
Requirements are imported as `slreq.Requirement` objects and are represented by the requirement icon (📄) in the Requirements Editor and in the Traceability Matrix. Importing requirements as `slreq.Requirement` objects allows you to freely edit, add, delete, and rearrange requirements. Updates you make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

Note You can export your requirements back to third-party applications that support ReqIF files by exporting requirements that are stored in Simulink Requirements 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 Simulink Requirements. In the Requirements Editor, select the top import node, represented by the Import node icon (📄🔗). In the **Details** 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 1-52.



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 **Details** 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 **Details** pane, under **Requirement Interchange**, clicking **Unlock all**. You cannot delete referenced requirements or change their hierarchy within Simulink Requirements, 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 Simulink Requirements. 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 “Customize Requirements with Custom Attributes” on page 1-48. When you register custom attributes within Simulink Requirements 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. 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

Simulink Requirements 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 Simulink Requirements. 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 6-2.) There is no indication when a direct link becomes unresolved unless you run consistency checks.

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

See Also

`slreq.import`

Related Examples

- “Link with Requirements in IBM DOORS Next” on page 7-4
- “Working with IBM Rational DOORS 9 Requirements” on page 7-50

More About

- “Import Requirements from Microsoft Office Documents” on page 1-11
- “Import Requirements from IBM DOORS Next” on page 1-27
- “Import Requirements from ReqIF Files” on page 1-16
- “Update Imported Requirements” on page 1-52
- “Round-Trip Importing and Exporting for ReqIF Files” on page 1-73

Import Requirements from Microsoft Office Documents

You can author requirements in Microsoft Word and Microsoft Excel and import them into Simulink Requirements. 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 1-7.

To import requirements from a Microsoft Office document:

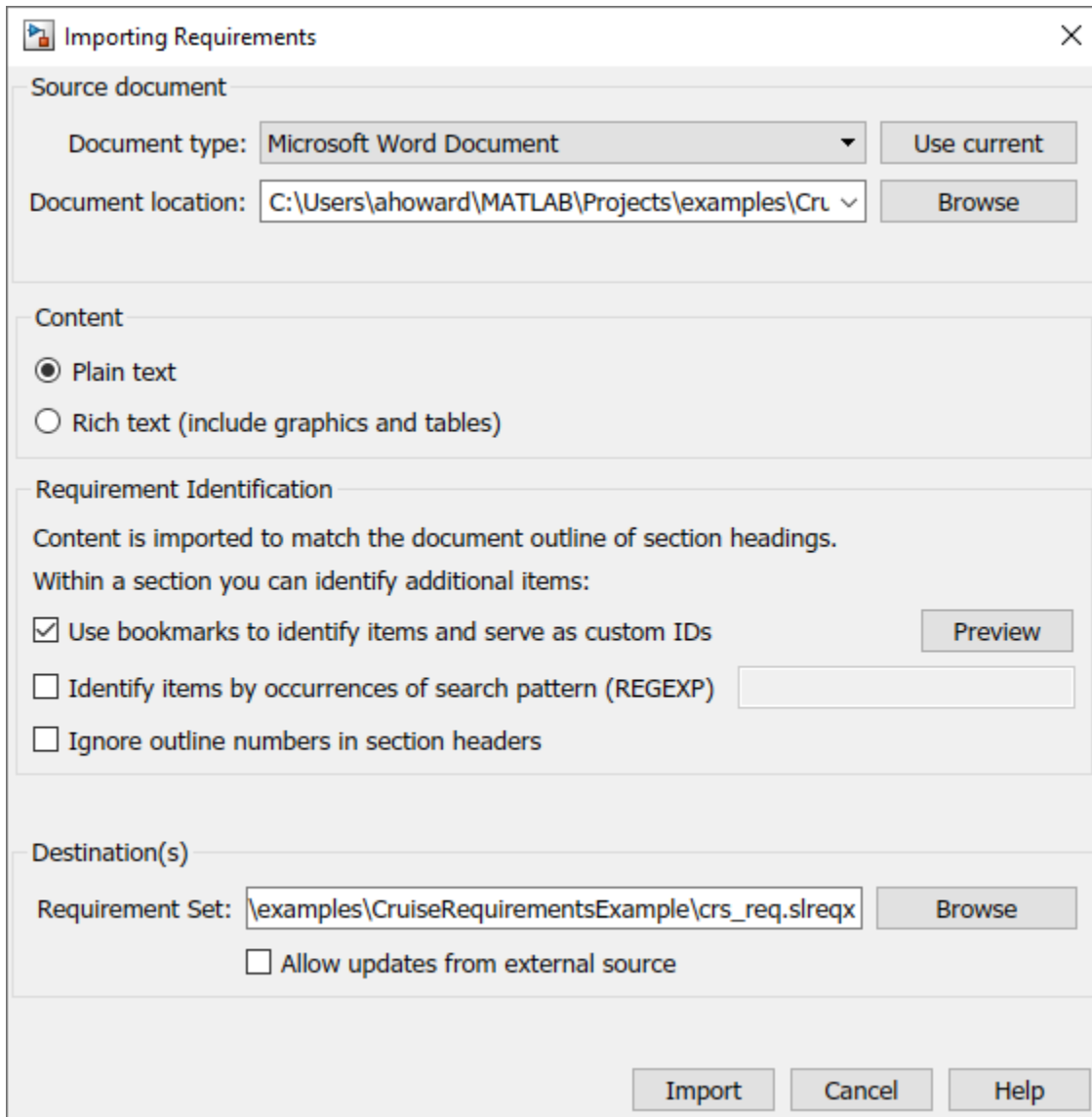
- 1 Open the Requirements Editor. At the MATLAB command line, enter:

```
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 1-11 and “Import Options for Microsoft Excel Spreadsheets” on page 1-13. To read more about importing requirement sets or referenced requirements, see “Select an Import Mode” on page 1-7.
- 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, Simulink Requirements 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 and unnecessary 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:

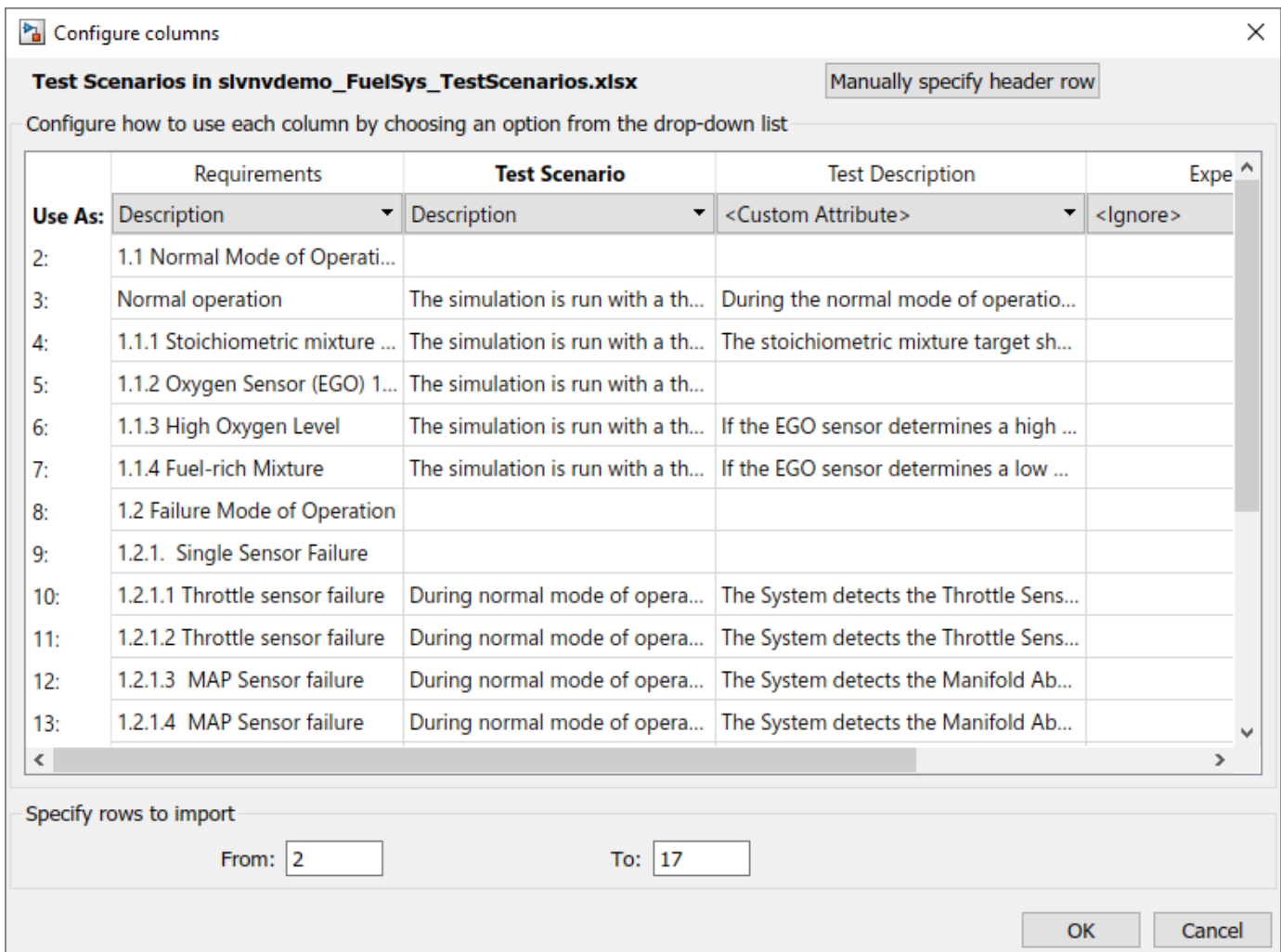
- Source document:**
 - Document type: Microsoft Excel Spreadsheet
 - Document location: C:\Users\ahoward\Downloads\slvndemo_
 - Sheet name: Test Scenarios
 - Use worksheet name as item ID prefix
- Content:**
 - Text only (better performance)
 - Include graphics and layout
- Requirement Identification:**
 - Specify rows and columns (Pattern: ,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 “Customize Requirements with Custom Attributes” on page 1-48.

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.



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 Update Requirements from a Microsoft Word Document” on page 1-54

More About

- “Import Requirements from Third-Party Applications” on page 1-7

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 1-7.

Choosing an Import Mapping

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 Simulink Requirements and their ReqIF counterparts.

Item	Representation in Simulink Requirements	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 1-48 	<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 2-65	<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 Simulink Requirements. 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.

Simulink Requirements 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 Simulink Requirements” on page 1-23.

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, Simulink Requirements 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 “Customize Requirements with Custom Attributes” on page 1-48.

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 Simulink Requirements” on page 1-23.

When you import links using the built-in mapping, Simulink Requirements imports `SpecRelation` objects as links and maps the `SpecRelationType` objects to link types in Simulink Requirements. 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 2-33.

Using the Generic Mapping During Import

When you import a ReqIF file and use the generic mapping, Simulink Requirements 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 “Customize Requirements with Custom Attributes” on page 1-48.

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 Simulink Requirements” on page 1-23.

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 2-33.

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 Simulink Requirements.

- 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.

Importing Requirements

Source document

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

Document location: C:\Users\jdoe\ReqIF\CruiseControlReqs.reqif Browse

Source specifications

Import a single specification: Requirement Spec

Combine all specifications into one Requirement Set

Import each specification into a separate Requirement Set

Source links

Import links

Attribute mapping

Source tool: Generic

Generic ReqIF files

Destination(s)

Requirement Set: ects\CruiseControlDesign\CruiseControlReqs.slreqx Browse

Allow updates from external source

Import Cancel Help

- 5 Under **Attribute mapping**, in the **Source tool** drop-down, select the desired attribute mapping. See “Choosing an Import Mapping” on page 1-16.
- 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**. If you want to migrate your requirements to Simulink Requirements, clear **Allow updates from external source**. For more information about import options, see “Select an Import Mode” on page 1-7.
- 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.

Importing Requirements

Source document

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

Document location: C:\Users\jdoe\ReqIF\CruiseContr 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

Attribute mapping

Source tool: Generic

Generic ReqIF files

Destination(s)

Requirement Set: Design\CruiseControlDesignSpec.slreqx Browse

Allow updates from external source

Import Cancel Help

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 1-22 and “Considerations for ReqIF Files with Multiple Specifications” on page 1-73.

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 Simulink Requirements 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
- 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, Simulink Requirements 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 Simulink Requirements

If you link a requirement in Simulink Requirements 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 1-41.

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 Simulink Requirements default link storage. For more information, see “Requirements Link Storage” on page 5-4. 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 Simulink Requirements


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 1-16.

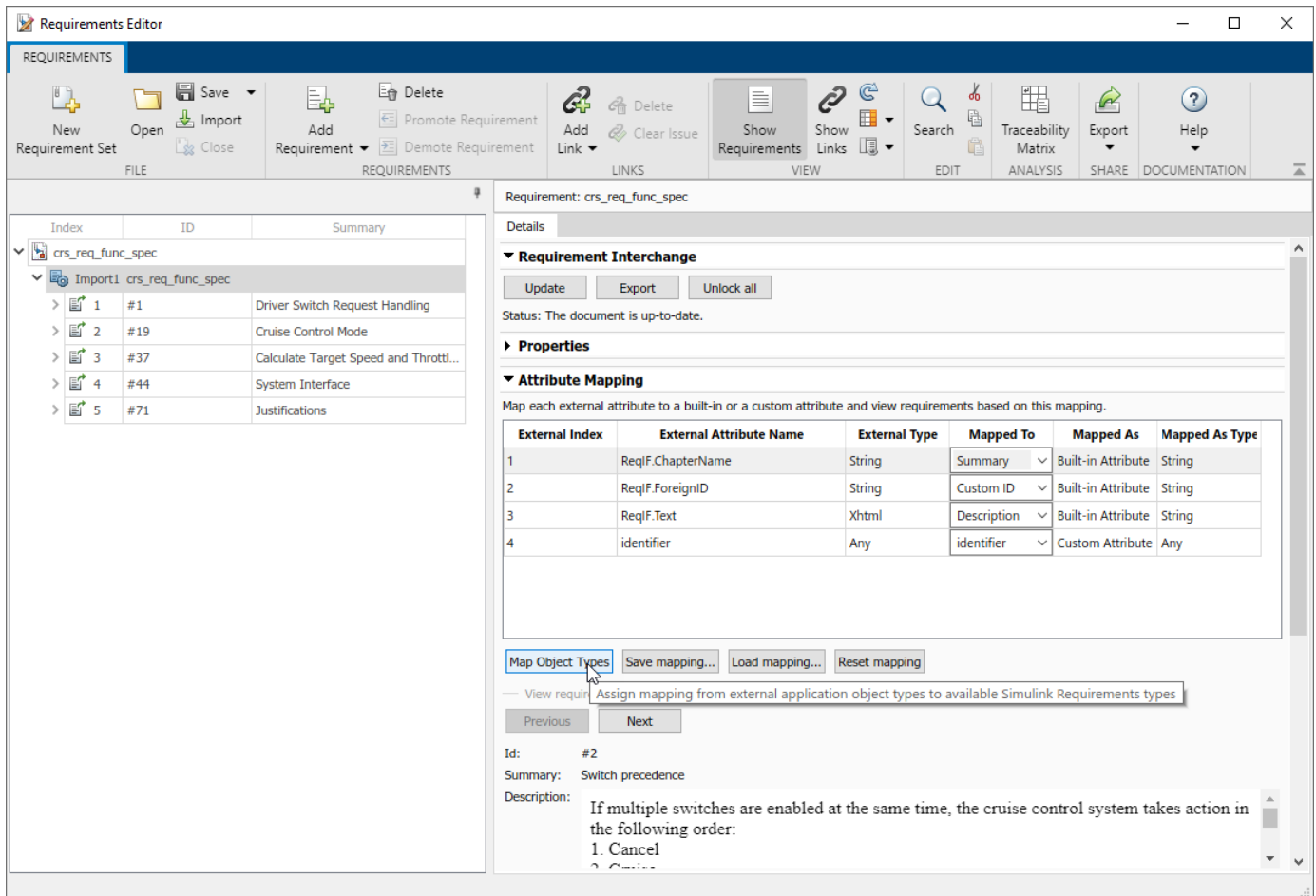
After you import the requirements, you can edit the `SpecObjectType` object attribute mapping.

Select the Import node, which is denoted by , or the top-level requirement, depending on how you imported the requirements. In the **Details** 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 1-84.

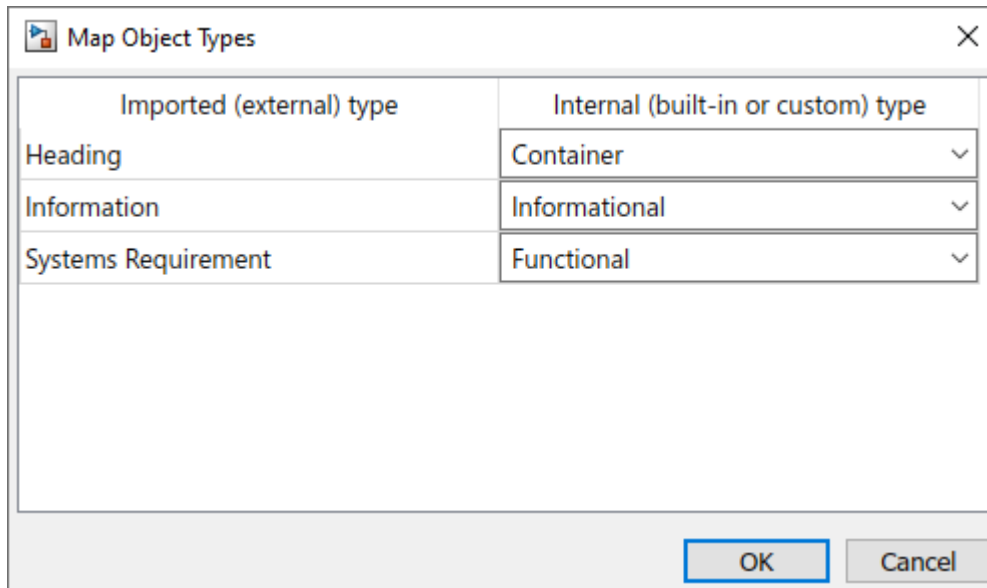
Map SpecObjectTypes to Requirement Types

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

- 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 **Details** pane, under **Attribute Mapping**, click **Map Object Types**.

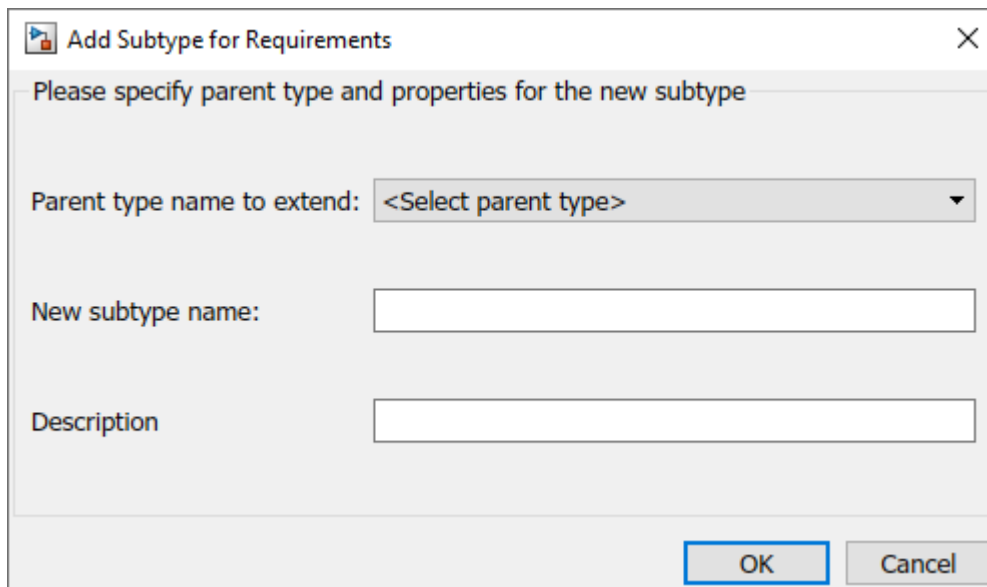


- 3 The Map Object Types dialog appears. **Imported (external) type** lists the SpecObjectType objects and **Internal (built-in or custom) type** lists the available Simulink Requirements requirement types. Map each SpecObjectType object by selecting a requirement type from the list. For more information 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” on page 2-40.



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.

See Also

slreq.import

More About

- “Export Requirements to ReqIF Files” on page 1-38
- “Round-Trip Importing and Exporting for ReqIF Files” on page 1-73
- “Import Requirements from Third-Party Applications” on page 1-7
- “Create and Edit Attribute Mappings” on page 1-84
- “Update Imported Requirements” on page 1-52

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

Configure IBM DOORS Next Session

To interface with IBM DOORS Next, you must configure MATLAB every session. At the MATLAB command prompt, enter:

```
slreq.dngConfigure
```

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. If you do not see a port number, enter the default value of 443. In the Server Login Name and Server Login Password dialog boxes, enter your login credentials. 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>`. For more information about configurations, see “Specifying and Updating the IBM DOORS Next Configuration” on page 7-32.

MATLAB then tests the connection 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 section of `slreq.dngConfigure`.

Import DOORS Next Requirements

You can import requirements by selecting a DOORS Next module or by creating a query. Because Simulink Requirements imports requirements as `slreq.Reference` objects, you must import the requirements into a new requirement set.

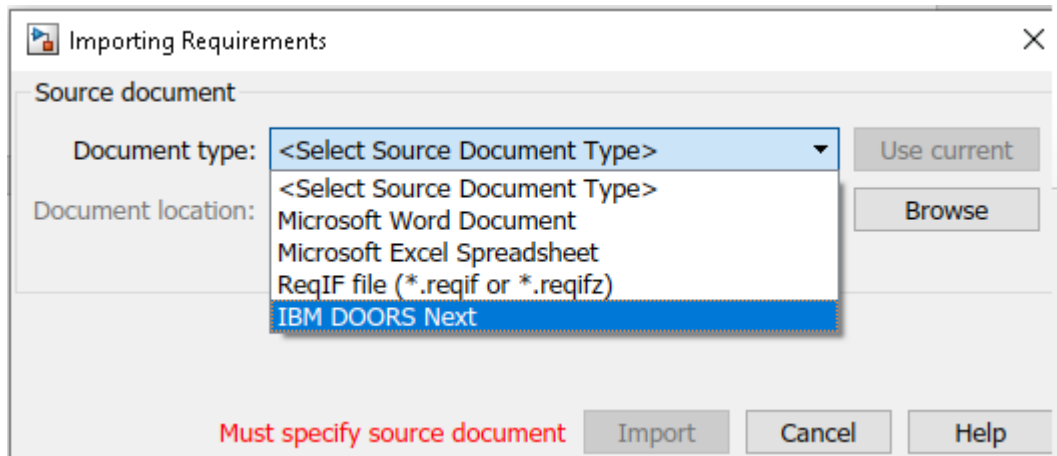
Note When you import requirements from a DOORS Next project that has configuration management enabled, you must select the desired configuration context for your MATLAB session before importing. For more information about enabling configuration management in DOORS Next, see Configuration management in the RM application on the IBM website. For more information about configuring your MATLAB session, see `slreq.dngConfigure`. The imported requirement set is associated with this configuration for updating on page 1-30 and navigating on page 1-31 referenced requirements. For more information about configuration management, see “Specifying and Updating the IBM DOORS Next Configuration” on page 7-32.

Importing a Requirements Module

When you import requirements from a DOORS Next module, the Simulink Requirements 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.



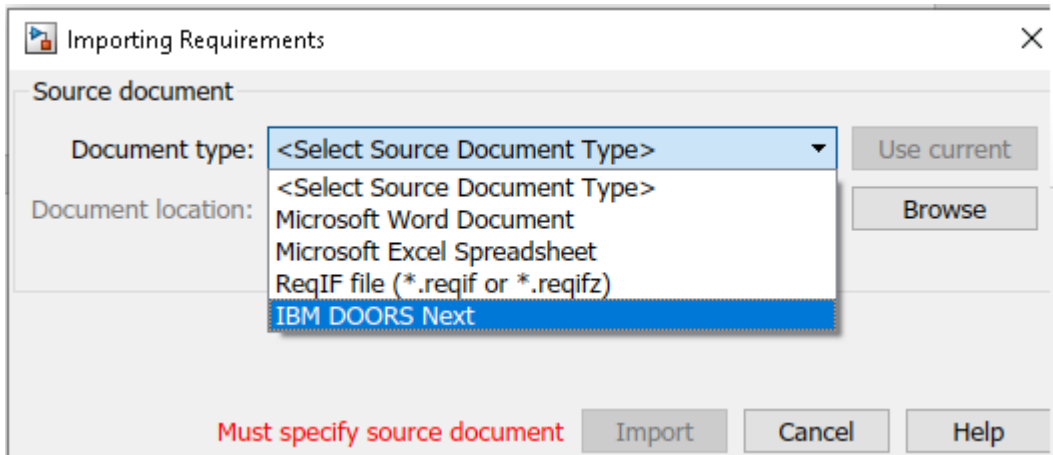
- 3 Set **Document location** to the project that you want to work with.
- 4 Under **Get requirements from**, select **Full module hierarchy (takes time if large module)**. Wait for the **DNG Module** list to populate.

- 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 Simulink Requirements. When the import completes, the Requirements Editor displays the hierarchy of the imported items.

Importing Requirements by Using Queries

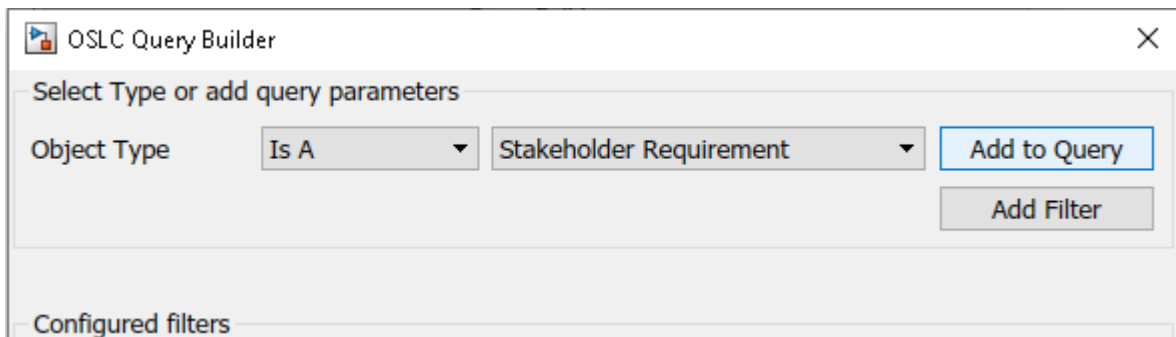
- 1 Open the Requirements Editor:
slreq.editor
- 2 In the Requirements Editor, click **Import**.

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 **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.



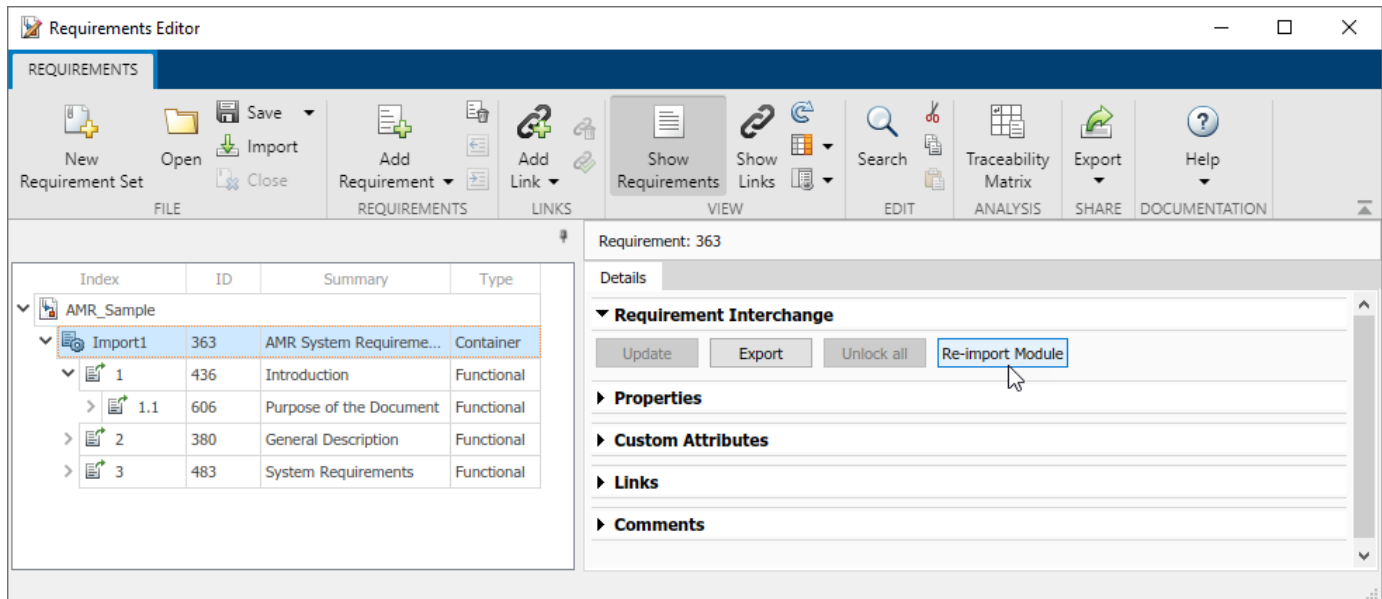
- 6 To create a query with other query parameters, click **Add Filter**.

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 Simulink Requirements.

Update Referenced Requirements

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



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 **Details** 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 **Details** 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 Simulink Requirements, you can link these referenced requirements the same way you link other `slreq.Reference` objects. For more information, see “Requirement Links” on page 2-32.

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 Simulink Requirements. For more information, see “Inserting Backlinks in DOORS Next” on page 7-27.

See Also

`slreq.dngConfigure`

Related Examples

- “Link with Requirements in IBM DOORS Next” on page 7-4

More About

- “Link and Trace Requirements with IBM DOORS Next” on page 7-26
- “Import Requirements from Third-Party Applications” on page 1-7
- “Import Requirements from ReqIF Files” on page 1-16

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. Simulink Requirements 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. At the MATLAB command prompt, enter:

```
rmi setup doors
```

For more information, see “Configure Simulink Requirements for IBM Rational DOORS” on page 5-2.

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 1-87.

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

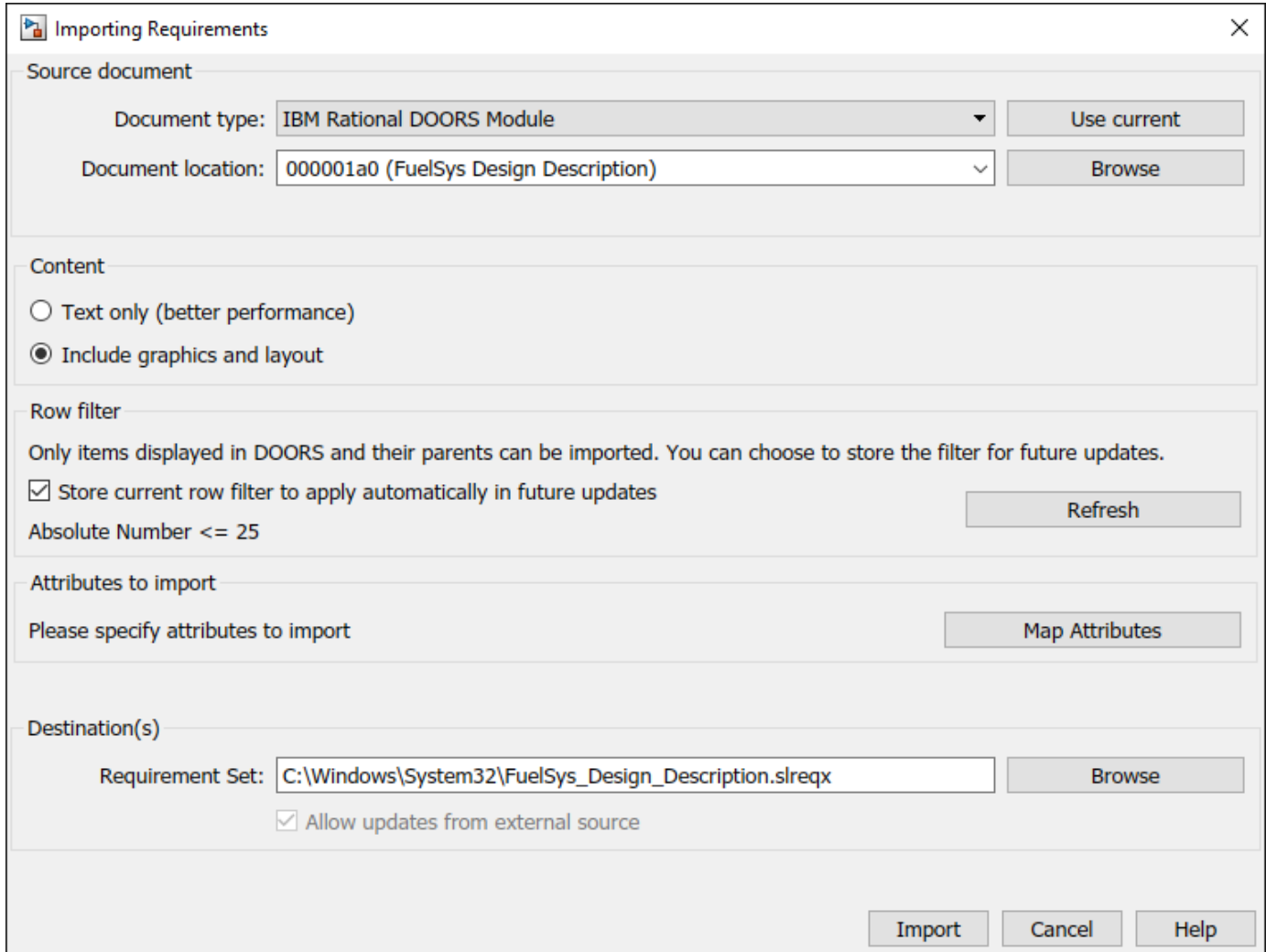
- At the MATLAB command line, enter:

```
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.

- 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**. Simulink Requirements imports only the requirements that match the currently applied filter. For more information, see “Import a Subset of Requirements from a Module” on page 1-35.



- Under **Attributes to import**, click **Map Attributes** to select the attributes to import from your requirements module. Simulink Requirements 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 1-84.

- Under **Destination(s)**, enter the name and file path for the requirement set. Click **Browse** to select a save location.
- Click **Import**. When the import completes, the Requirements Editor displays the requirements hierarchy.

Simulink Requirements 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 1-52.

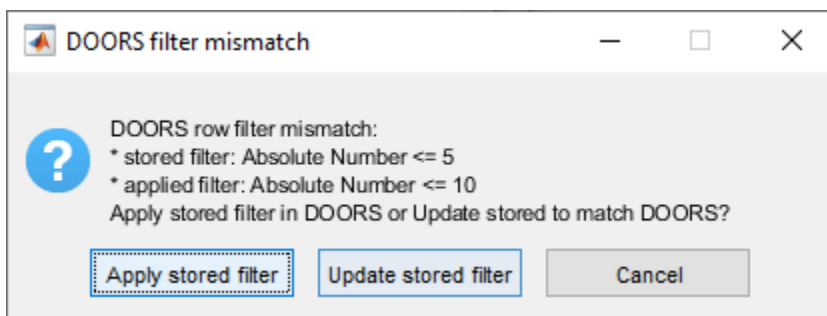
Import a Subset of Requirements from a Module

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 Simulink Requirements. For more information, see “Import an Entire Requirements Module” on page 1-33.

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. Simulink Requirements 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, Simulink Requirements 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 1-52.

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.

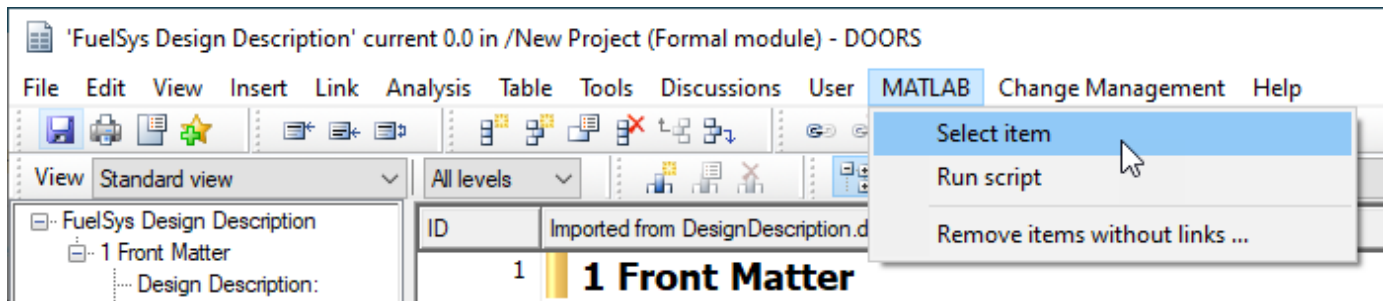
Navigate from MATLAB to IBM Rational DOORS to MATLAB

To navigate from the Requirements Editor to the original requirement in, select the referenced requirement in the Requirements Editor. In the **Details** 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 Simulink Requirements, select the requirement, then click **MATLAB > Select item**. The referenced requirement opens in the Requirements Editor.



See Also

slreq.import

Related Examples

- “Import Requirements from IBM Rational DOORS by using the API” on page 1-87
- “Working with IBM Rational DOORS 9 Requirements” on page 7-50

More About

- “Import Requirements from Third-Party Applications” on page 1-7
- “Import Requirements from IBM DOORS Next” on page 1-27
- “Import Requirements from ReqIF Files” on page 1-16
- “Update Imported Requirements” on page 1-52
- “Create and Edit Attribute Mappings” on page 1-84

- “Manage Navigation Backlinks in External Requirements Documents” on page 2-50

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 Simulink Requirements 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 Simulink Requirements 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.

Simulink Requirements 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:

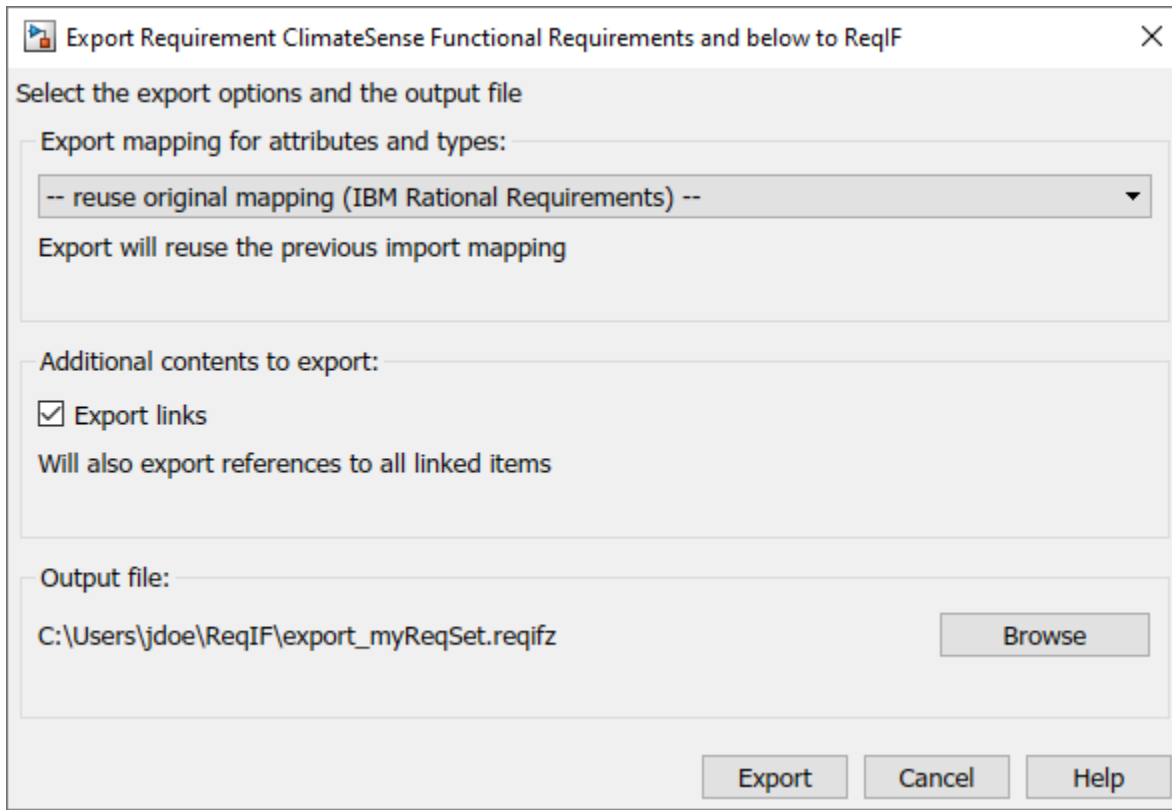
- A generic mapping
- The same mapping for import and export

For more information about ReqIF round-trips, see “Round-Trip Importing and Exporting for ReqIF Files” on page 1-73.

When you export requirements authored in Simulink Requirements, 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 Simulink Requirements. For more information, see “Map SpecObjectTypes to Requirement Types” on page 1-23. 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 all 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 Simulink Requirements requirement type to a custom requirement type. For more information about creating custom requirement types, see “Define Custom Requirement and Link Types” on page 2-40.

When you export links to a ReqIF file by using the generic mapping, Simulink Requirements exports the links as `SpecRelation` objects with the associated `SpecRelationType` object `LongName` attribute set to the same value as the link type in Simulink Requirements. For more information about link types, see “Link Types” on page 2-33.

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 1-16.

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 Simulink Requirements” on page 1-23

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

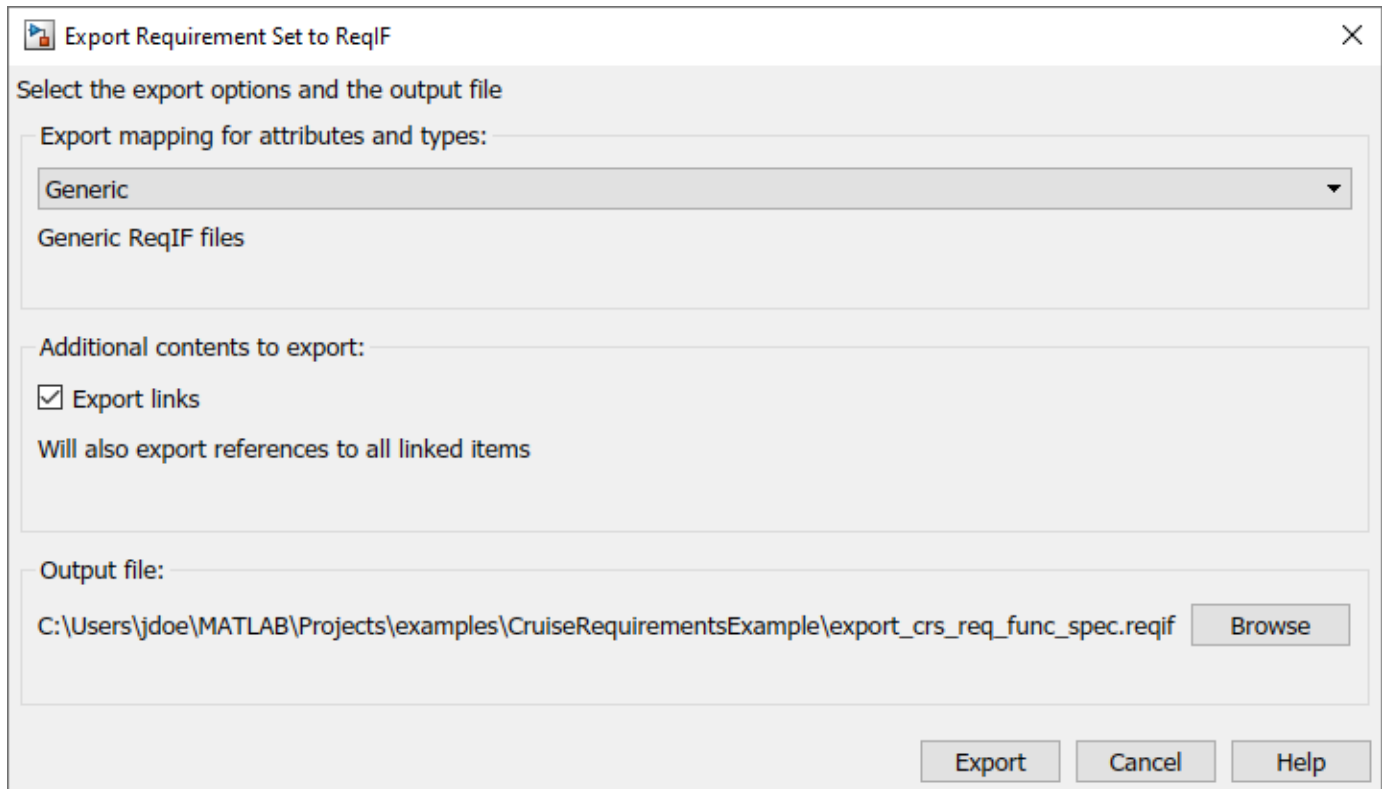
Exporting Requirements

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 Simulink Requirements.

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 1-38.
- 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 1-40.

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 1-38.

If you link a requirement in Simulink Requirements 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	<code>SpecObjectType longName</code> 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 <ul style="list-style-type: none">• Requirement• Referenced requirement	Simulink DDEntry Simulink Requirements object

For all 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 Simulink Requirements 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 Simulink Requirements” on page 1-23.

See Also





More About

- “Import Requirements from ReqIF Files” on page 1-16
- “Round-Trip Importing and Exporting for ReqIF Files” on page 1-73
- “Import Requirements from Third-Party Applications” on page 1-7

Define Requirements Hierarchy

Using Simulink Requirements, 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 Simulink Requirements 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 3-2 and “Review Requirements Verification Status” on page 3-6.

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 “Customize Requirements with Custom Attributes” on page 1-48 for more information about creating custom attributes for requirements.

See Also

More About

- “Customize Requirements with Custom Attributes” on page 1-48
- “Review Requirements Implementation Status” on page 3-2
- “Review Requirements Verification Status” on page 3-6

Create Requirement Set File by Using the Simulink® Requirements™ API

This example shows how to use the Simulink® Requirements™ 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.

```
sl_refresh_customizations;
```

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 with Custom Attributes

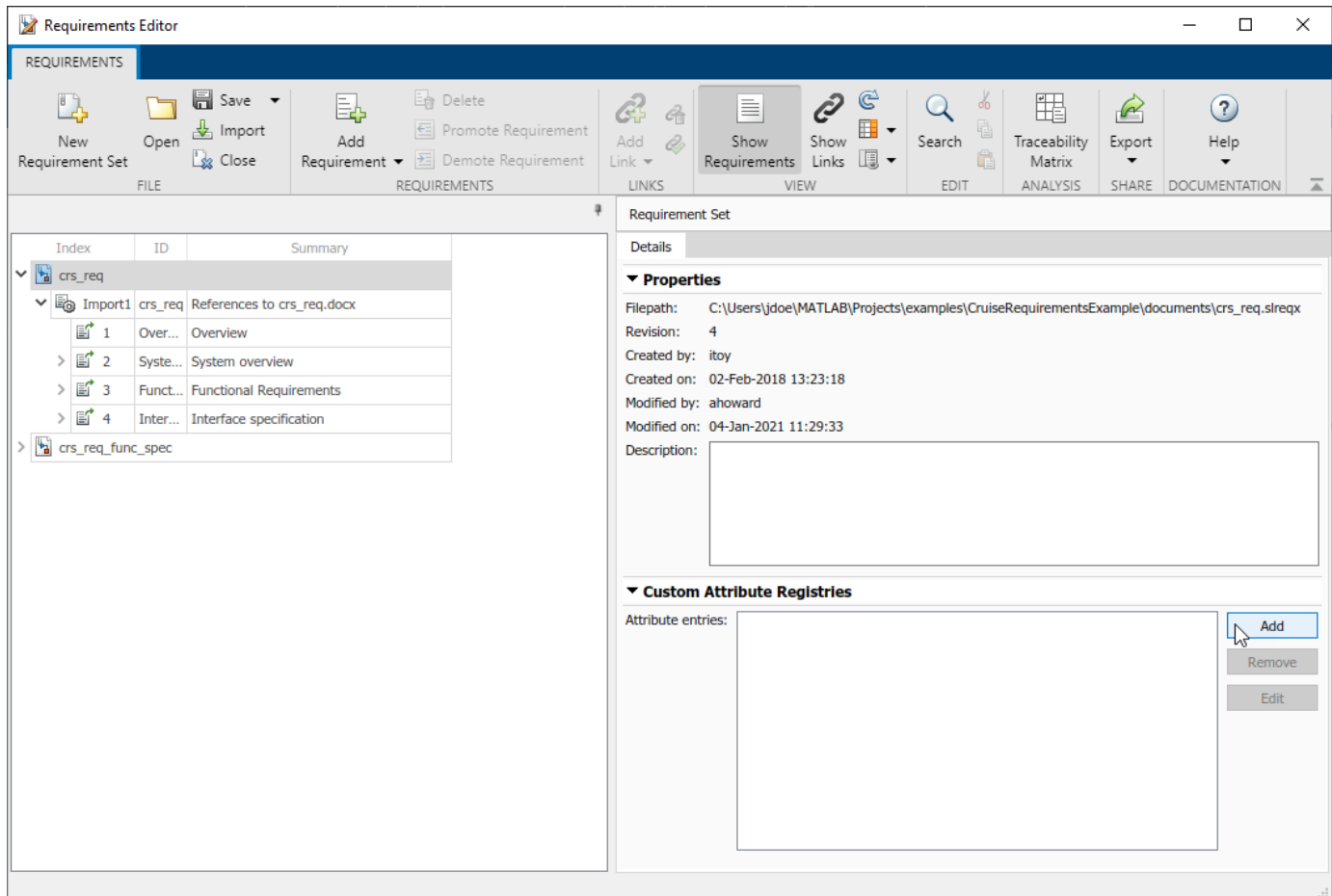
In this section...
"Define Custom Attributes for Requirement Sets" on page 1-48
"Set Custom Attribute Values for Requirements" on page 1-49
"Edit Custom Attributes" on page 1-50
"Custom Attributes for Referenced Requirements" on page 1-50
"Import Custom Attributes" on page 1-50
"Limitations" on page 1-51

When you create a requirement set using Simulink Requirements, 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.

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 **Details** 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.



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 **Details** pane, under **Custom Attributes**, enter the desired value in the field.

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.

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 **Details** pane, under **Custom Attribute Registries**, select the custom attribute you want to edit and click **Edit**.

For custom attributes of any type, you can edit the name and description. For **Combobox** custom attributes, you can also edit the drop-down menu options. You can edit the value of each option in the drop-down menu (excluding **Unset**), or add and remove options. If you edit the value of an option or remove an option, then requirements that had been set to that option will be reset to the default value, **Unset**.

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 Simulink Requirements, if you select **Allow updates from external source**, the requirements are imported as referenced requirements (slreq.Reference objects). For more information, see “Select an Import Mode” on page 1-7.

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 **Details** pane, under **Properties**, click **Unlock**. Alternatively, you can unlock the referenced requirements by selecting the top import node and, in the **Details** 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 Simulink Requirements” on page 1-23.

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 Simulink Requirements. For information about importing requirements from IBM DOORS Next using direct import, see “Link and Trace Requirements with IBM DOORS Next” on page 7-26.

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 1-13.

Limitations

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

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.

See Also


Related Examples

- “Manage Custom Attributes for Requirements by Using the Simulink® Requirements™ API” on page 1-79

More About


- “Customize Links with Custom Attributes” on page 2-43

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 **Details** 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 **Details** 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 Simulink Requirements. See “Customize Requirements with Custom Attributes” on page 1-48 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 Simulink Requirements, 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

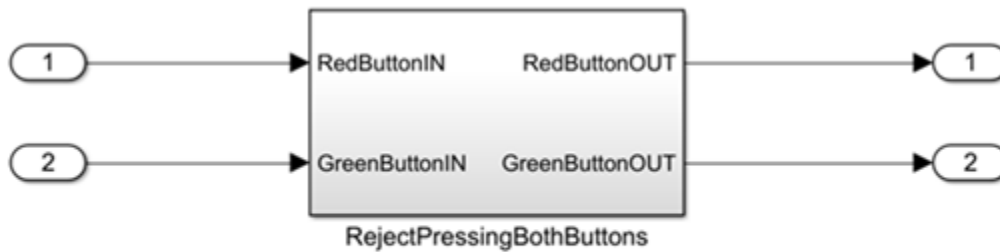
More About

- “Import Requirements from Third-Party Applications” on page 1-7
- “Import Requirements from Microsoft Office Documents” on page 1-11
- “Import Requirements from ReqIF Files” on page 1-16

Import and Update Requirements from a Microsoft Word Document

This example shows you how to import and update requirements from Microsoft® Word requirement documents. The model demonstrates a simple two-button switch that passes through outputs only when one switch is pressed at a time.

This functionality is available only on Microsoft Windows® platforms.



Simulink® Requirements™ rejectDoublePress.slx
Copyright 2018, The MathWorks, Inc.

This example uses a Microsoft Word document, `Reject_Double_Button_Press_Model_Requirements.docx`. This document contains a set of functional requirements for the `rejectDoublePress` model. Open the document from `matlab/examples/slrequirements`. The requirements in the document appear in outline format with custom bookmarks for navigation. To get the best results while importing and updating requirements, set up your Microsoft Word documents with document outlines and custom bookmarks.

Open the model.

```
open_system('rejectDoublePress');
```

Import Requirements

- 1 In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Requirements Editor**.
- 2 In the Requirements Editor, click **Import**.
- 3 Select **Microsoft Word document** from the **Document type** menu. Click **Browse** and select **Reject_Double_Button_Press_Model_Requirements.docx** as the **Document location**.
- 4 Under **Content** select **Plain text**. Under **Requirement Identification**, select **Use bookmarks to identify items and serve as custom IDs** and **Ignore outline numbers in section headers**. Leave **Allow updates from external source** checked. For more information on import options, see “Import Options for Microsoft Word Documents” on page 1-11.

The requirements from the Microsoft Word document are imported into the destination requirement set under a top-level node, `Import1`.

Update Referenced Requirements

Requirements that you import as referenced requirements retain their references to the source requirements document. To change your imported requirements, you can make the changes in the source document and update your requirement set from within Simulink® Requirements™.

- 1 In the `Reject_Double_Button_Press_Model_Requirements.docx` document, add a new requirement: **2.1.5** The Red and Green Button outputs shall be 0 if no buttons are pressed.
- 2 In Microsoft Word, click **Insert > Bookmark**. Create a bookmark called `Red_and_Green_Button_Output_2_1_5` for the new requirement and save the Microsoft Word document.
- 3 In the Requirements Editor, select the top-level node (`Import1`) of the destination requirement set. In the **Details** pane under **Requirement Interchange**, click **Update** to update the referenced requirements.
- 4 Select `Import1` and view the changes in the **Comments** side bar. The **Revision** number and the **UpdatedOn** values are updated for the requirement set.

For more information on updating requirements, see “Update Imported Requirements” on page 1-52.

Unlock Referenced Requirements

You can also edit a referenced requirement by unlocking it. In the Requirements Editor, select the referenced requirement that you want to edit. In the **Details** pane, under **Properties** click **Unlock**. You can unlock all referenced requirements in the requirement set by selecting the import node and in the **Details** pane, under **Requirement Interchange** click **Unlock All**.

If you want to revert changes that you made to a requirement after unlocking it, you can update your referenced requirements. Select the top-level node (`Import1`) of the destination requirement set. In the **Details** pane under **Requirement Interchange**, click **Update** to update the referenced requirements.

Cleanup

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

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

See Also

`slreq.ReqSet`

More About

- “Import Requirements from Microsoft Office Documents” on page 1-11
- “Link to Requirements in Microsoft Word Documents” on page 6-2
- “Import Requirements from Third-Party Applications” on page 1-7
- “Define Requirements Hierarchy” on page 1-43

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

You can export requirement sets and link sets to files that are compatible with previous versions of Simulink Requirements 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.

To open the Requirements Editor, at the MATLAB command prompt, enter:

```
slreq.editor
```

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, Simulink Requirements also exports a link set file that is compatible with the selected 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 1-56.

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 Simulink Version”.

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

See Also

`exportToVersion`

Use Command-line API to Document Simulink Model in Requirements Editor

This example uses Simulink® and Simulink Requirements® APIs to automatically capture and link Simulink model structure, for the purpose of documenting the design in Simulink Requirements Editor. Automation will also help to repair or migrate requirements traceability data after replacing or modifying linked artifacts. The use of the following command-line APIs is demonstrated:

- `slreq.new` for creating a new Requirement Set
- `slreq.ReqSet` for adding entries to a Requirement Set
- `add` for adding child requirements
- `slreq.Requirement` for filling-in the Description field
- `slreq.createLink` for creating link from SRC to DEST
- `slreq.find` for locating Simulink Requirements objects
- `setDestination` for re-connecting the destination end of an existing link
- `setSource` for moving an existing link to the new source object
- `isResolvedSource` for identifying links whose source object cannot be found
- `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.

USE CASE 1: Link with Simulink Model Surrogate in Simulink Requirements

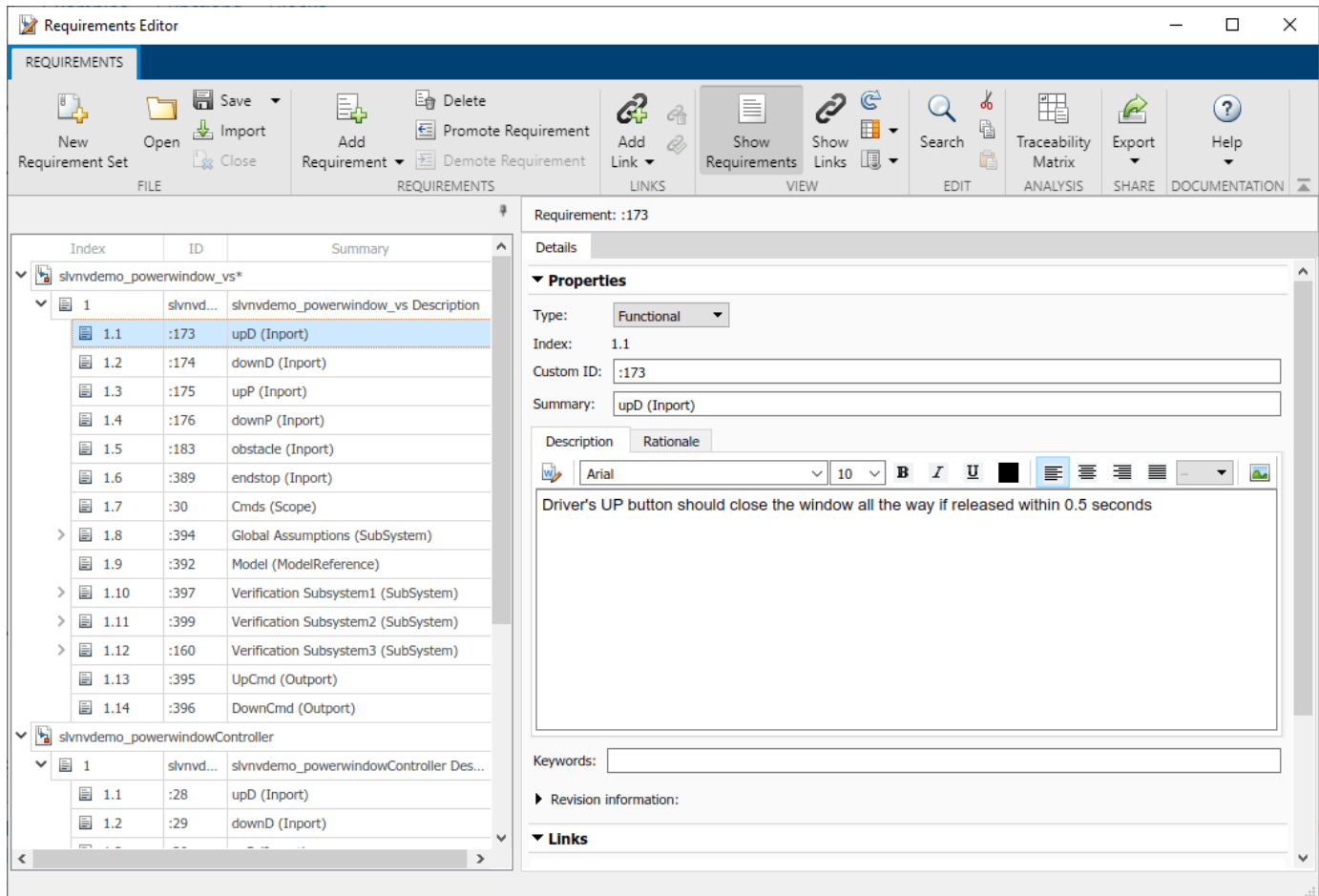
You want to use the Simulink Requirements product to create a detailed description of your Simulink design, and you want to organize your Requirements collection in a hierarchy that matches your Simulink models. You also want an easy way to navigate between the items of this Requirements collection and the corresponding elements in your design.

For the purpose of this demonstration, consider the `slvndemo_powerwindow_vs.slx` specification model designed for verifying the functional properties of `slvndemo_powerwindowController.slx`.

```
open_system('slvndemo_powerwindow_vs');
open_system('slvndemo_powerwindowController');
```

We use the legacy VNV/RMI product API, `rmi('getObjectsInModel', MODEL)`, to get a hierarchical list of objects in `MODEL`, then use Simulink Requirements `slreq.*` APIs on page 1-0 to automatically generate the surrogate (representation) for each of our Simulink models.

We can then provide related design requirements information in the Description or Rational fields of auto-generated proxy items.



Below is the script that builds one Requirement Set with two model surrogates. The bottom three commands provide an example of how to programmatically fill-in the Description field for a proxy item, but most probably you will do this interactively in the Editor.

```
models = {'slvndemo_powerwindow_vs', 'slvndemo_powerwindowController'};
workDir = tempname;
disp(['Using ' workDir ' to store generated files.']);
```

Using C:\Users\ahoward\AppData\Local\Temp\tp048cdc5c_b22d_44f6_bb6c_85d54e962505 to store generated files.

```
mkdir(workDir);
addpath(workDir);
for modelIdx = 1:length(models)
    modelName = models{modelIdx};
    reqSetFile = fullfile(workDir, [modelName '.slreq']);
    slProxySet = slreq.new(reqSetFile); % create separate ReqSet file with matching name
    open_system(modelName); % will create a proxy item for each object in this Simulink model
    modelNode = slProxySet.add('Id', modelName, 'Summary', [modelName ' Description']);
    [objHs, parentIdx, isSf, SIDs] = rmi('getobjectsInModel', modelName);
    for objIdx = 1:length(objHs)
        if parentIdx(objIdx) < 0 % top-level item is the model itself
            indexedReqs(objIdx) = modelNode; %#ok<SAGROW>
        else
            parentReq = indexedReqs(parentIdx(objIdx));
```

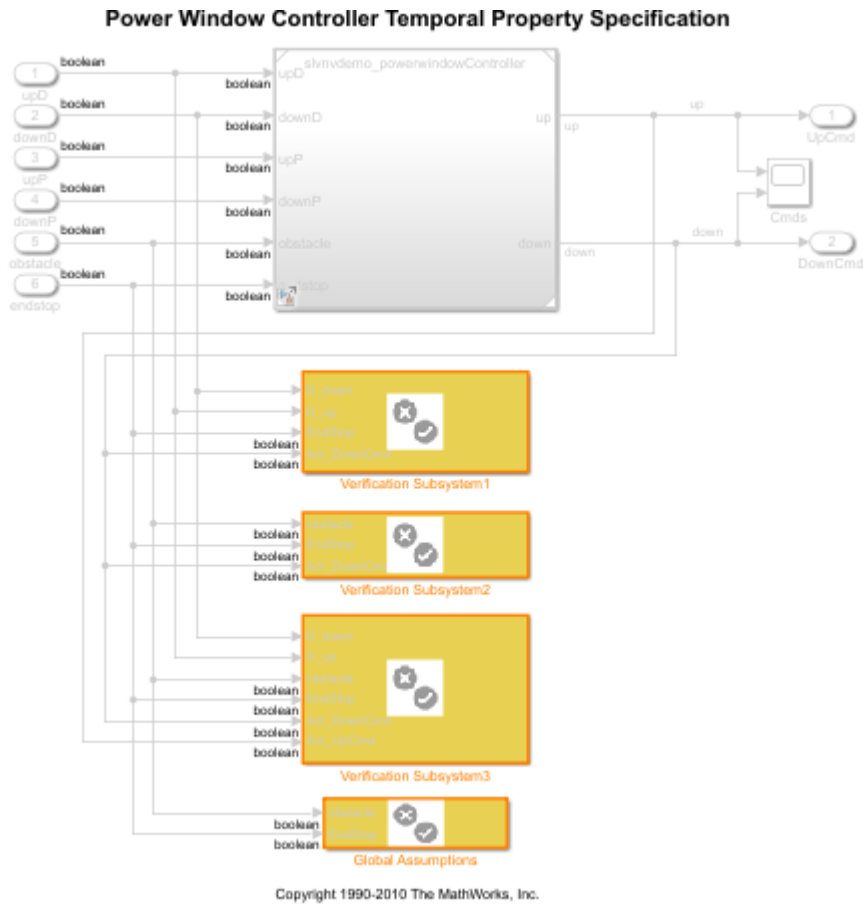
```

if isSf(objIdx)
    sfObj = Simulink.ID.getHandle([modelName SIDs{objIdx}]);
    if isa(sfObj, 'Stateflow.State')
        name = sf('get', objHs(objIdx), '.name');
    elseif isa(sfObj, 'Stateflow.Transition')
        name = sf('get', objHs(objIdx), '.labelString');
    else
        warning('SF object of type %s skipped.', class(sfObj));
        continue;
    end
    type = strrep(class(sfObj), 'Stateflow.', '');
else
    name = get_param(objHs(objIdx), 'Name');
    type = get_param(objHs(objIdx), 'BlockType');
end
indexedReqs(objIdx) = parentReq.add(...
    'Id', SIDs{objIdx}, 'Summary', [name ' (' type ')']); %#ok<SAGROW>
end
end
slProxySet.save(); % save the autogenerated Requirement Set
end
slreq.editor(); % open editor to view the constructed Requirement Set
slProxySet = slreq.find('type', 'ReqSet', 'Name', 'slvndemo_powerwindow_vs');
roItem = slProxySet.find('type', 'Requirement', 'Summary', 'upD (Inport)'); % will...
% provide Description text for this item
roItem.Description = ['Driver's UP button should close the window all the way if...' ...
    ' released within 0.5 seconds'];

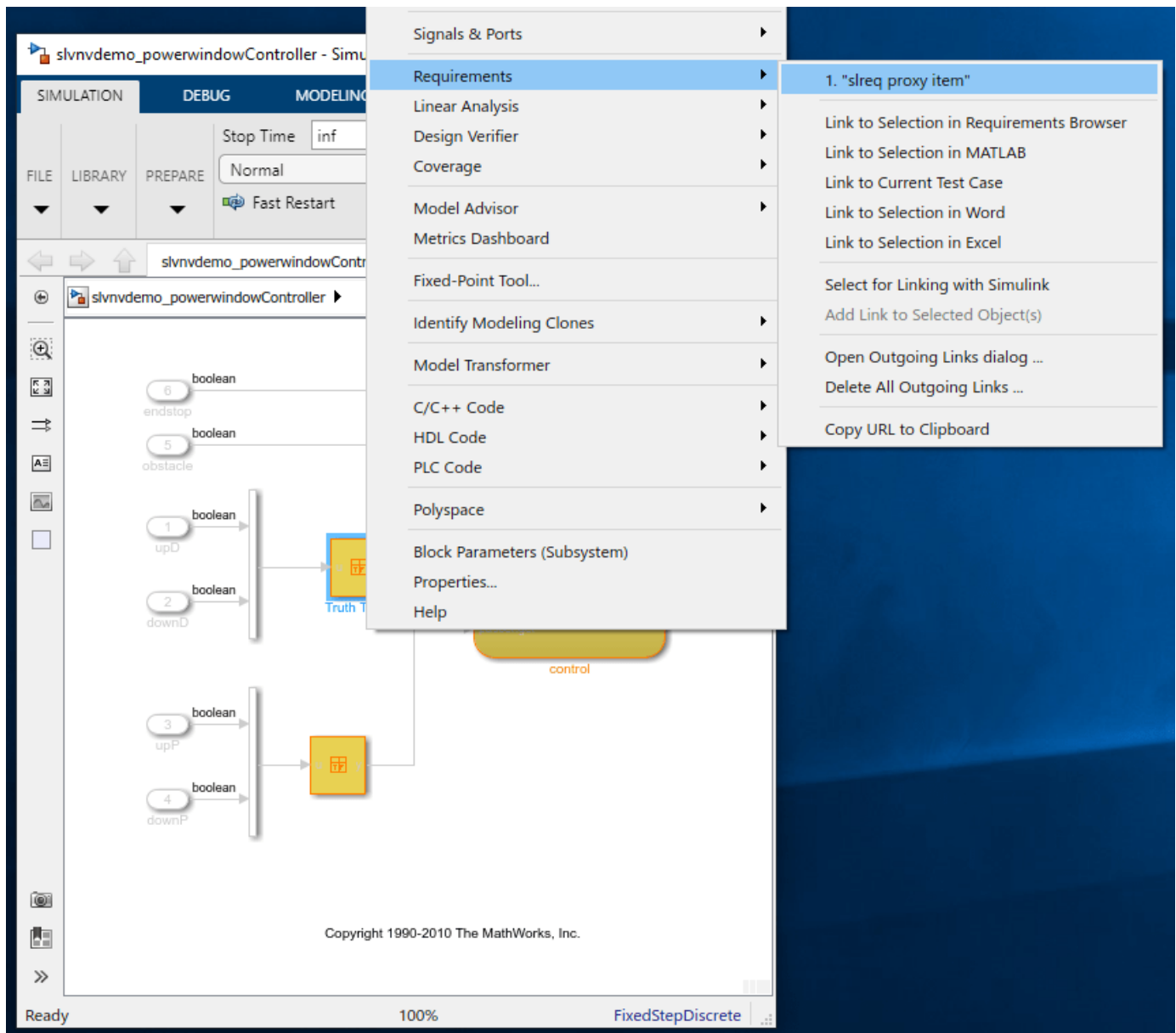
```

Create Traceability Between Model Objects and Proxy Items

Now we can browse the structure of each model in the Requirement Editor, and we can edit the Description fields to provide additional details about each design element. What's missing is an easy way to navigate between the objects in Simulink diagrams and the proxy/surrogate items in Simulink Requirements. The script below demonstrates the use of `slreq.createLink` API to automatically add navigation links. We can choose any desired level of granularity. For the purpose of this example, we will limit linking to SubSystem blocks.

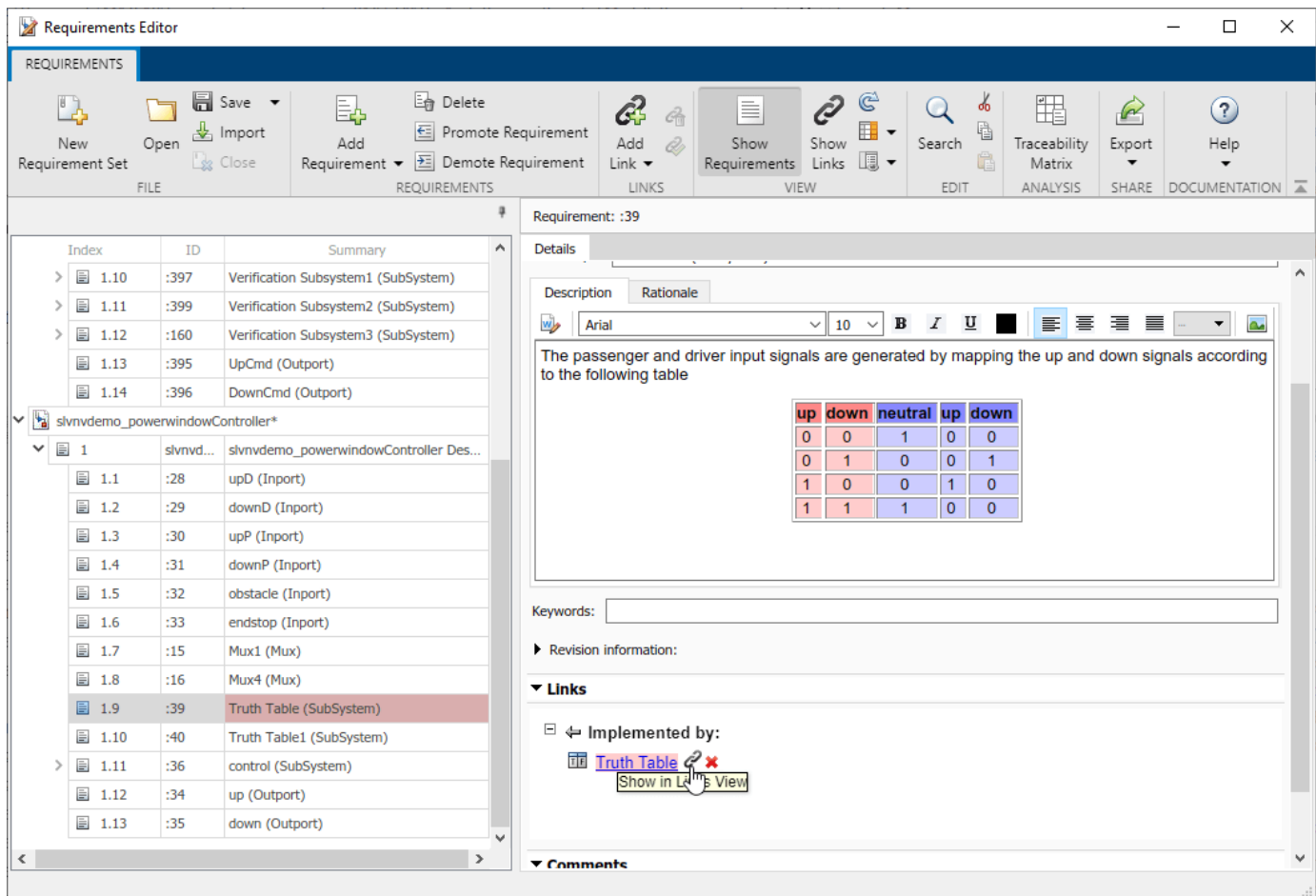


We can also enable highlighting to visualize which Simulink objects received navigation links.



Navigate the link from Simulink block to review the corresponding proxy item in Simulink Requirements Editor. Note the hyperlink to the associated block in the **Details** pane under **Links** at the bottom-right.

1 Requirements Definition



```

for modelIdx = 1:length(models)
    modelName = models{modelIdx};
    counter = 0;
    slProxySet = slreq.find('type', 'ReqSet', 'Name', modelName);
    proxyItems = slProxySet.find('type', 'Requirement');
    for reqIdx = 1:numel(proxyItems)
        roItem = proxyItems(reqIdx);
        if contains(roItem.Summary, '(SubSystem)' % || contains(roItem.Summary, '(State)')
            sid = [modelName roItem.Id];
            disp(['    linking ' sid ' ..']);
            srcObj = Simulink.ID.getHandle(sid);
            link = slreq.createLink(srcObj, roItem);
            link.Description = 'slreq proxy item';
            counter = counter + 1;
        end
    end
    disp(['Created ' num2str(counter) ' links for ' modelName]);
    rmi('highlightModel', modelName);
end

linking slvndemo_powerwindow_vs:394 ..
linking slvndemo_powerwindow_vs:394:224 ..
linking slvndemo_powerwindow_vs:394:272 ..
linking slvndemo_powerwindow_vs:394:271 ..

```

```
linking slvndemo_powerwindow_vs:394:360 ..
linking slvndemo_powerwindow_vs:397 ..
linking slvndemo_powerwindow_vs:397:107 ..
linking slvndemo_powerwindow_vs:397:300 ..
linking slvndemo_powerwindow_vs:397:108 ..
linking slvndemo_powerwindow_vs:397:285 ..
linking slvndemo_powerwindow_vs:397:307 ..
linking slvndemo_powerwindow_vs:399 ..
linking slvndemo_powerwindow_vs:399:650 ..
linking slvndemo_powerwindow_vs:399:214 ..
linking slvndemo_powerwindow_vs:399:218 ..
linking slvndemo_powerwindow_vs:399:273 ..
linking slvndemo_powerwindow_vs:160 ..
linking slvndemo_powerwindow_vs:160:643 ..
linking slvndemo_powerwindow_vs:160:646 ..
linking slvndemo_powerwindow_vs:160:590 ..
linking slvndemo_powerwindow_vs:160:591 ..
linking slvndemo_powerwindow_vs:160:648 ..
linking slvndemo_powerwindow_vs:160:592 ..
```

Created 23 links for slvndemo_powerwindow_vs

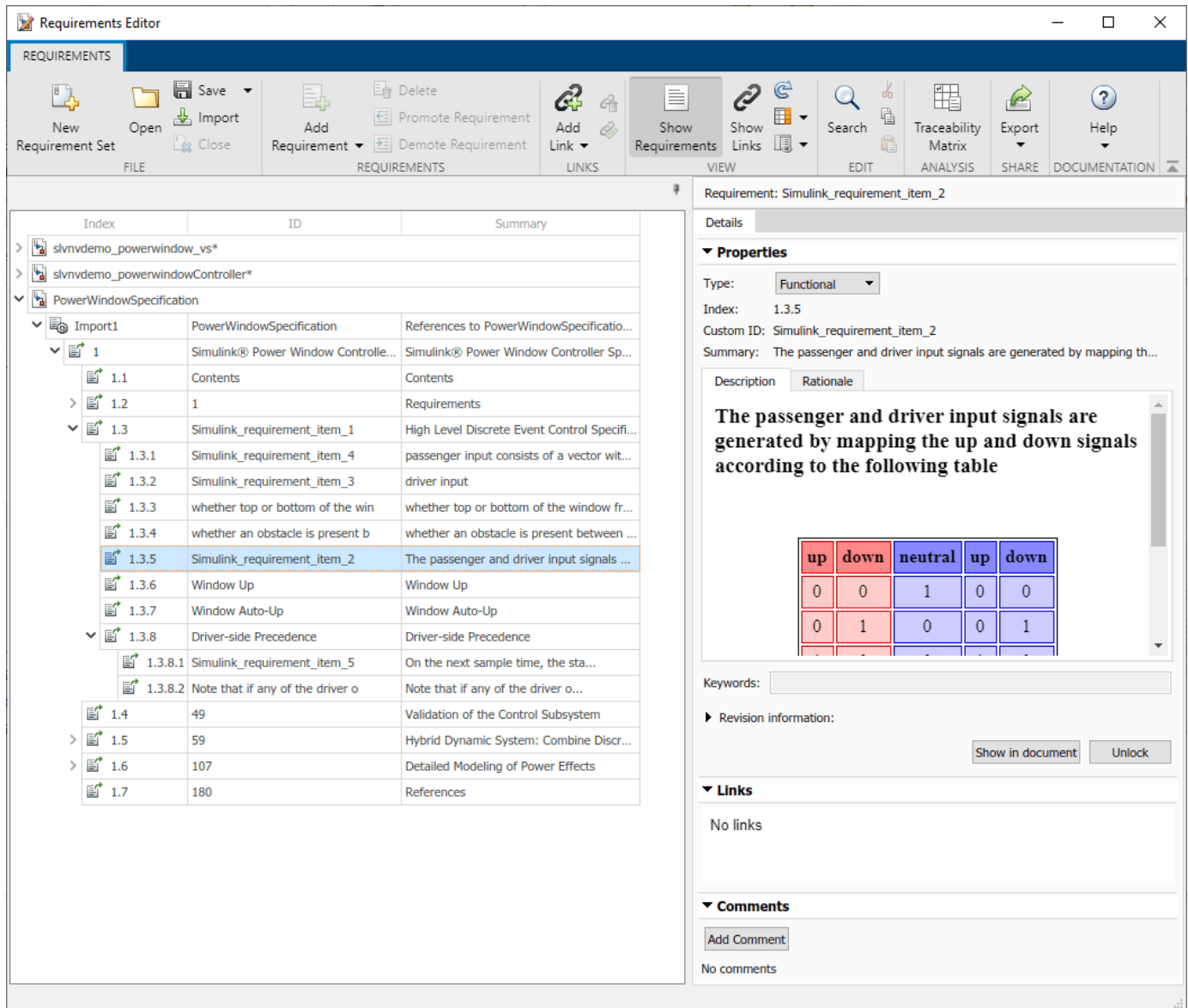
```
linking slvndemo_powerwindowController:39 ..
linking slvndemo_powerwindowController:40 ..
linking slvndemo_powerwindowController:36 ..
```

Created 3 links for slvndemo_powerwindowController

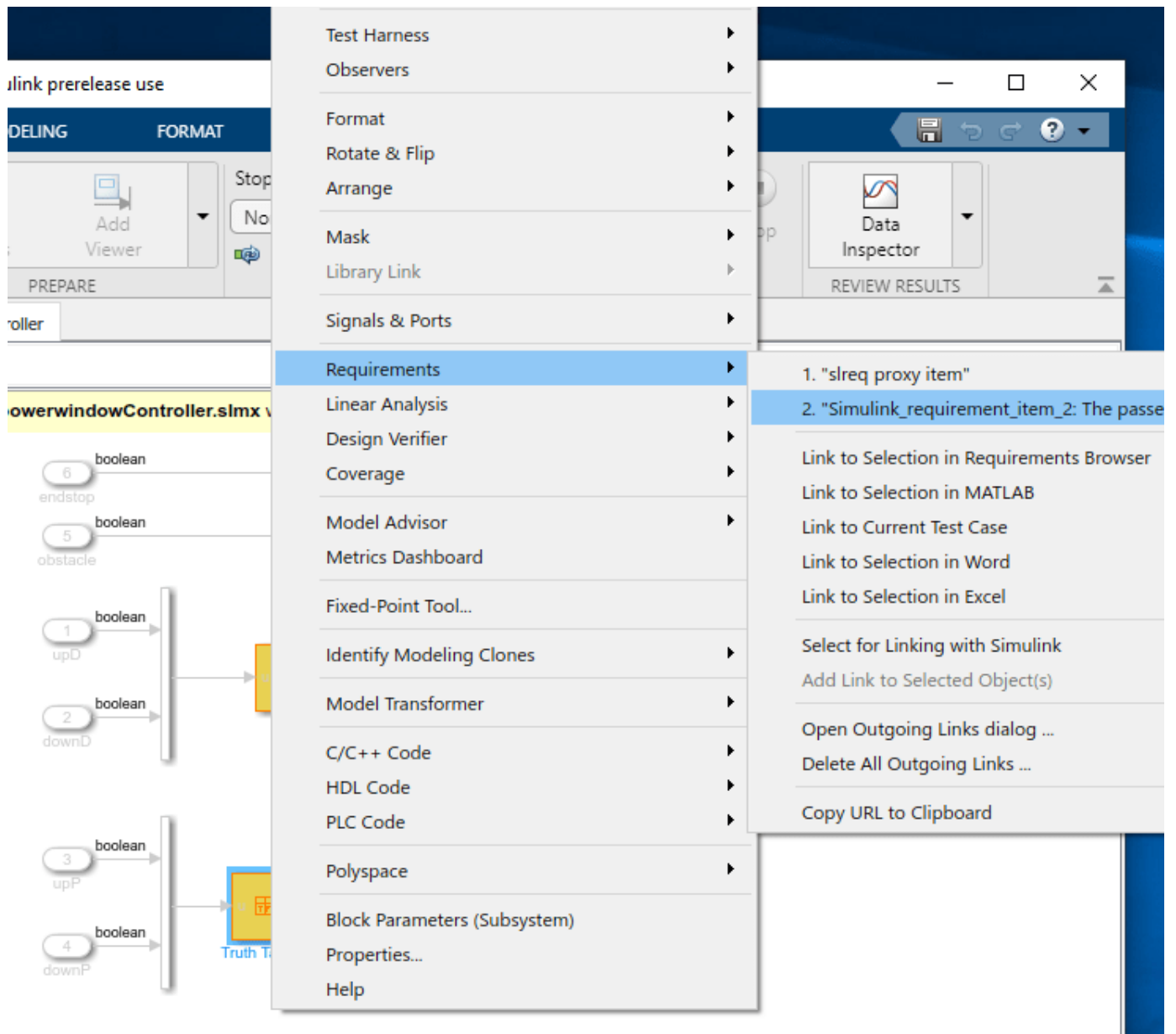
USE CASE 2: Reuse Existing Links After Replacing Linked Destination Artifact

In the course of design project development, there may be need to migrate to a new set of Requirements. If current Requirements have links, and when there is a known rule to associate original linked requirements with the corresponding entries in the new Requirement Set, you may want to automatically migrate the links where possible, to avoid redoing all the linking manually. Migrating existing links is preferred over re-creating new links, because you keep the existing metadata such as keywords, rationale statements, comment history.

To quickly put together an example situation where you may need to migrate links, we will start with the Requirements imported from a Microsoft Word document, then create some links.



We then create some links, either interactively (by drag-and-drop in Requirements Perspective mode) or using the API, to allow navigation between Simulink objects and imported requirements.



Navigation from Simulink object is done either via the context menu or with one click when in Requirements Perspective mode.

up	down	neutral	up	down
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0

```

examplesFolder = fullfile(matlabroot, 'toolbox', 'slrequirements', 'slrequirementsdemos');
docsFolder = fullfile(examplesFolder, 'powerwin_reqs');
addpath(docsFolder); % just in case
externalDocName = 'PowerWindowSpecification';
externalDoc = fullfile(docsFolder, [externalDocName '.docx']);
outputFile = fullfile(workDir, 'ReadOnlyImport.slreqx');
[~,~,roReqSet] = slreq.import(externalDoc, 'ReqSet', outputFile); % without extra...
% arguments the default import mode is "read-only references"
roReqSet.save();
roItem = roReqSet.find('CustomId', 'Simulink_requirement_item_2');
designModel = 'slvnvdemo_powerwindowController';
link1 = slreq.createLink([designModel '/Truth Table'], roItem); % link to read-only item...
% in imported set
link2 = slreq.createLink([designModel '/Truth Table1'], roItem); % link 2nd block to the...
% same read-only item
slreq.show(link1.source()); % highlight te source of 1st link
slreq.show(link1.destination()); % navigate to the target of 1st link
rmi('view', [designModel '/Truth Table1'], 2); % navigate 2nd link from Truth Table1...
% using legacy RMI('view') API

```

Updating Links for Navigation to Alternative Imported Requirement Set

Note that we imported our Word document "as read-only references", which is the default for `slreq.import` command when run without optional arguments. This import mode allows to later update the imported items when the newer version of the source document becomes available. Now, supposed that we changed our mind: we want our imported items to be fully editable in Simulink Requirements Editor, including additions, deletions, and structural moves. Although you can Unlock and edit properties of items imported "as references", you cannot reorder the imported items or add new ones, and if you delete an item, it will re-appear when performing the next update from the modified external document. When unrestricted editing capability is needed, we use a different import mode: "as editable requirements", by providing an additional `AsReference` argument for the `slreq.import` command, and specifying `false` as the value.

This generates a new Requirement Set, to be managed exclusively in Simulink Requirements. There is no connection with the original external document, and you are free to add/move/delete entries as

needed. Now, you do not need to Unlock imported items to modify the Description or other properties:

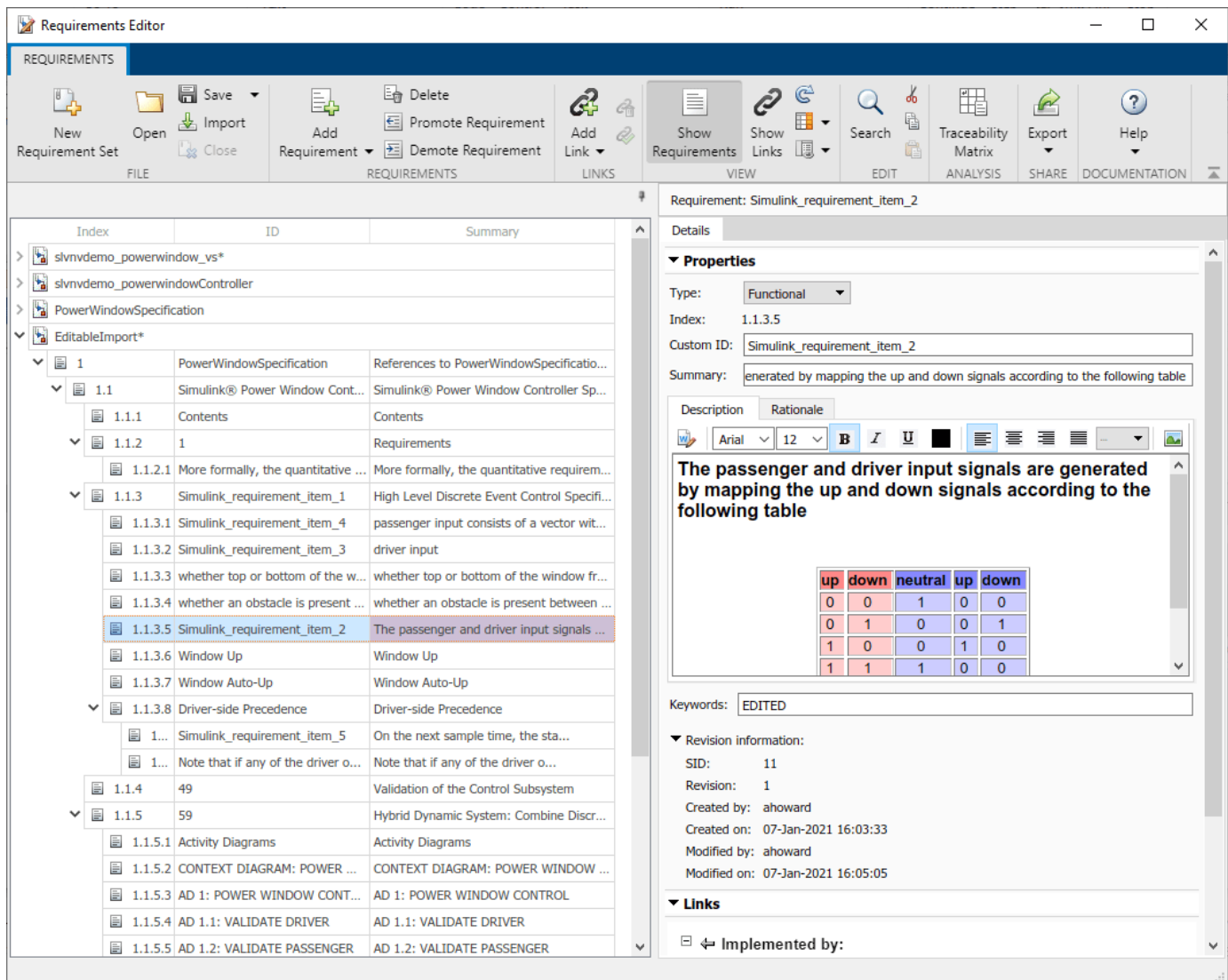
The screenshot shows the Requirements Editor application window. The main window is divided into several panes:

- Top Ribbon:** Contains tabs for REQUIREMENTS, LINKS, VIEW, EDIT, ANALYSIS, SHARE, and DOCUMENTATION. The REQUIREMENTS tab is active, showing options like New, Open, Save, Add, Delete, Promote Requirement, Demote Requirement, Add Link, Show Requirements, Show Links, Search, Traceability Matrix, Export, and Help.
- Left Pane (Index):** A tree view showing the project structure. The 'EditableImport*' folder is expanded, showing a hierarchy of requirements. The requirement 'Simulink_requirement_item_2' is selected and highlighted in blue.
- Right Pane (Details):** Shows the details for the selected requirement 'Simulink_requirement_item_2'. It includes fields for Index (1.1.3.5), Custom ID (Simulink_requirement_item_2), and Summary (mapping the up and down signals according to the following table). Below these is a rich text editor for the Description, which contains the text: "The passenger and driver input signals are generated by mapping the up and down signals according to the following table". A table is embedded in the description:

up	down	neutral	up	down
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

However, there is problem: our previously created links connect from Simulink to the original read-only References, not to the more recently imported editable Requirements. The solution is to create and run a script that redirects the existing links to corresponding items in the newly imported (editable) set. We use `setDestination` API to perform the required updates.

After we loop over all links in the LinkSet, and adjust the affected links to connect with corresponding editable items, when we navigate from the model block, the correct item in editable set opens, and incoming links from both blocks are shown.



Below is the example script that accomplishes this task.

```

outputFile = fullfile(workDir, 'EditableImport.slreqx');
% re-import as Editable Requirements
[~,~,mwReqSet] = slreq.import(externalDoc, 'ReqSet', outputFile, 'AsReference', false);
mwReqSet.save();
linkSet = slreq.find('type', 'LinkSet', 'Name', designModel); % LinkSet for our design model
links = linkSet.getLinks(); % all outgoing links in this LinkSet
updateCount = 0;
for linkIdx = 1:numel(links)
    link = links(linkIdx);
    if strcmp(link.destination.reqSet, [roReqSet.Name '.slreqx']) % if this link points to...
        % an item in read-only ReqSet
        sid = link.destination.sid; % internal identifier of linked read-only item
        roItem = mwReqSet.find('SID', sid); % located the linked read-only item
        id = roItem.Id; % document-side identifier of imported read-only item
        mwItem = mwReqSet.find('Id', id); % located a matching item in Editable...
        % Requirement Set
    end
end

```

```

        link.setDestination(mwItem);
        updateCount = updateCount + 1;
    end
end
disp(['Updated ' num2str(updateCount) ' links from ' designModel]);

Updated 2 links from slvndemo_powerwindowController

slreq.show(link.destination());           % check updated destination of the last...
% link we modified
rmi('view', [designModel '/Truth Table1'], 2); % navigate again (legacy API), editable...
% item selected in RE

```

USE CASE 3: Reuse Existing Outgoing Links After Replacing Source Objects

Consider the situation when you have a SubSystem with lots of links to Requirements, and this subsystem needed to be redesigned or entirely replaced. The new implementation is mostly similar, and you would like to preserve the existing links where possible (where blocks with the same name exists in the same level of the model structure hierarchy). This will allow to limit manual linking steps to only the blocks that did not exist in the original implementation. You use `setSource` API to re-attach the existing links at new source objects after replacing the SubSystem. Note that you cannot simply keep using the old links, because links rely on unique persistent session-independent identifiers (SIDs) to associate the link source object (the Simulink object that "owns" the link), and your replacement SubSystem has new SIDs for each object.

To demonstrate the use of `setSource` API with our example model, we will simply replace two Truth Table blocks that we linked in the previous section with same exact new blocks. Once this is done, the links become `unresolved`, because new Truth Table copies have new SIDs.

In the Requirements Editor, click **Show Links** and notice the orange triangle indicators for all the broken links. There is a total of 4, because each of the replaced blocks had 2 links: one link to the surrogate item in `slvndemo_powerwindowController.slreqx` and another link to an imported requirement in `ReadOnlyImport.slreqx`.

```

slreq.open('slvndemo_powerwindowController.slreqx')

ans =
    ReqSet with properties:

        Description: ''
            Name: 'slvndemo_powerwindowController'
            Filename: 'C:\Users\ahoward\AppData\Local\Temp\tp048cdc5c_b22d_44f6_bb6c_85d54e90'
            Revision: 1
            Dirty: 0
        CustomAttributeNames: {}
            CreatedBy: 'ahoward'
            CreatedOn: 14-Jan-2021 15:22:27
            ModifiedBy: 'ahoward'
            ModifiedOn: 14-Jan-2021 15:22:27

```

Label	Source	Type	Destination
slvndemo_powerwindow_vs.slmx*	Changed source: 0/23		Changed destination: ...
UpdatedModel.slmx*	Changed source: 0/9		Changed destination: ...
sireq proxy item	UpdatedModel:39	Implements	:39 Truth Table (Sub...
sireq proxy item	UpdatedModel:40	Implements	:40 Truth Table1 (Su...
sireq proxy item	control	Implements	:36 control (SubSyste...
link #4	UpdatedModel:39	Implements	Simulink_requirement...
link #5	UpdatedModel:40	Implements	Simulink_requirement...
sireq proxy item	Truth Table	Implements	:39 Truth Table (Sub...
link #4	Truth Table	Implements	Simulink_requirement...
sireq proxy item	New Truth Table1	Implements	:40 Truth Table1 (Su...
link #5	New Truth Table1	Implements	Simulink_requirement...

Link: UpdatedModel:40

Properties

Source: UpdatedModel:40

Type: Implements

Destination: Simulink_requirement_item_2 The passenger and driver input signals are generat

```

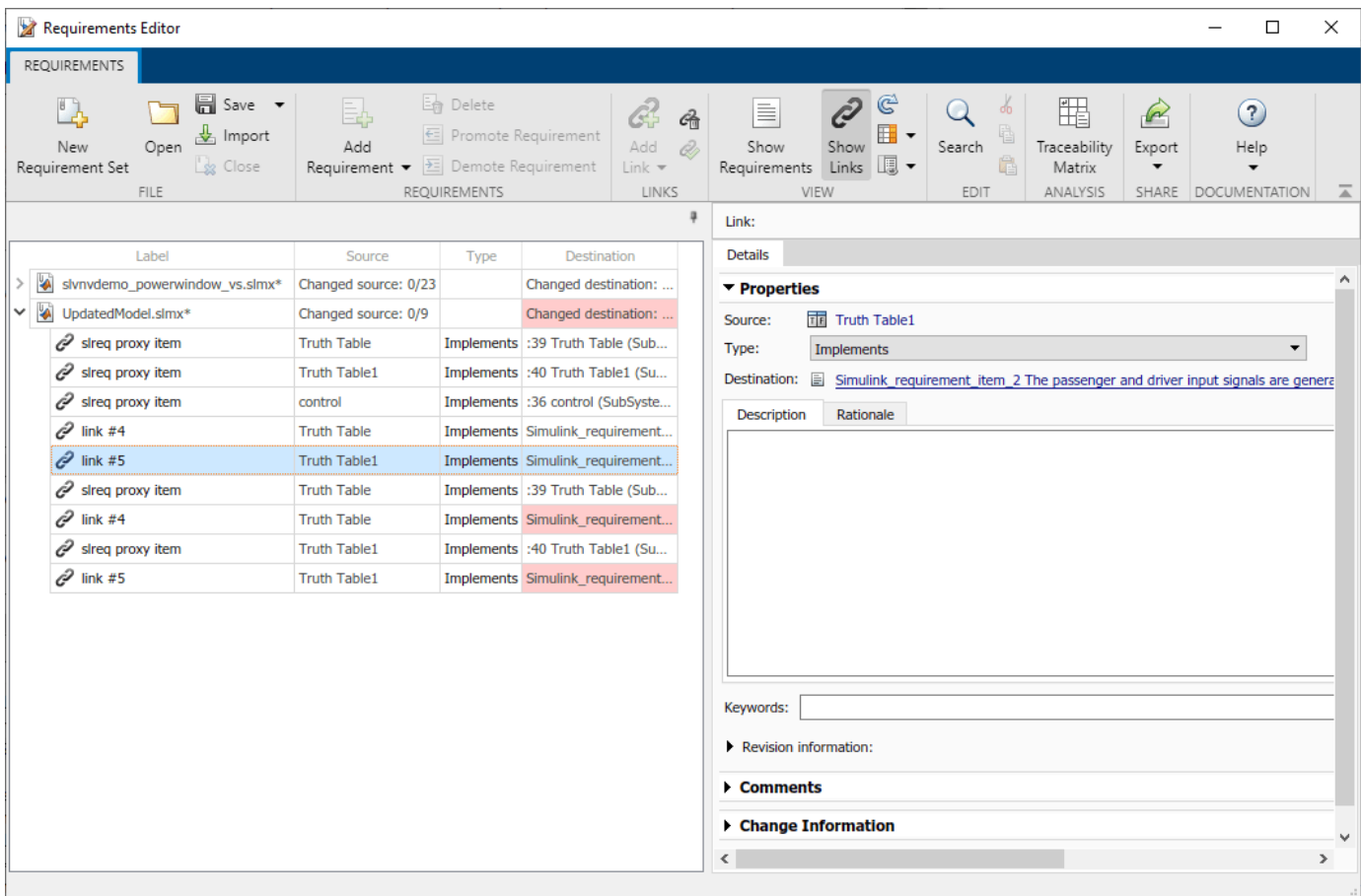
originalModel = 'slvndemo_powerwindowController';
updatedModel = 'UpdatedModel';
save_system(originalModel, fullfile(workDir, [updatedModel '.slx'])); % this also creates...
% .slmx file in workDir
delete_line(updatedModel, 'Mux1/1', 'Truth Table/1'); % disconnect original block
delete_line(updatedModel, 'Truth Table/1', 'control/3'); % disconnect original block
add_block([updatedModel '/Truth Table'], [updatedModel '/New Truth Table']); % create...
% replacement block
delete_block([updatedModel '/Truth Table']); % delete original block
add_line(updatedModel, 'Mux1/1', 'New Truth Table/1'); % reconnect new block
add_line(updatedModel, 'New Truth Table/1', 'control/3'); % reconnect new block
set_param([updatedModel '/New Truth Table'], 'Name', 'Truth Table'); % restore original name
delete_line(updatedModel, 'Mux4/1', 'Truth Table1/1'); % disconnect original block
delete_line(updatedModel, 'Truth Table1/1', 'control/4'); % disconnect original block
add_block([updatedModel '/Truth Table1'], [updatedModel '/New Truth Table1']); % create...
% replacement block
delete_block([updatedModel '/Truth Table1']); % delete original block
add_line(updatedModel, 'Mux4/1', 'New Truth Table1/1'); % reconnect new block
add_line(updatedModel, 'New Truth Table1/1', 'control/4'); % reconnect new block
set_param([updatedModel '/New Truth Table1'], 'Name', 'Truth Table1'); % restore original name

```

Update Source Ends to Repair Broken Links

Now we need to iterate over all the links in the new model, identify the ones with unresolved source using `isResolvedSource` API, then use `setSource` command to fix each broken link. Because we cannot rely on the old SIDs to find the needed new sources of the link, we open the original model to discover the original block's path and name, then locate the corresponding replacement block in the updated model.

See the example script below. When you run this script, it reports 4 links fixed. Check the Links View in Simulink Requirements Editor and confirm that all the links are now resolved, there are no orange icon indicators anywhere.



```
open_system(originalModel);
updatedLinkSet = slreq.find('type', 'LinkSet', 'Name', updatedModel);
links = updatedLinkSet.getLinks();
fixCount = 0;
for linkIdx = 1:numel(links)
    link = links(linkIdx);
    if ~link.isResolvedSource()
        missingSID = link.source.id;
        origBlockHandle = Simulink.ID.getHandle([originalModel missingSID]);
        origBlockPath = getfullname(origBlockHandle);
        [~,blockPath] = strtok(origBlockPath, '/');
        updatedBlockPath = [updatedModel blockPath];
        updatedModelSID = Simulink.ID.getSID(updatedBlockPath);
```

```
        updatedBlockHandle = Simulink.ID.getHandle(updatedModelSID);
        link.setSource(updatedBlockHandle);
        fixCount = fixCount + 1;
    end
end
updatedLinkSet.save();
disp(['Fixed ' num2str(fixCount) ' links in ' updatedModel '.slmx']);
```

```
Fixed 4 links in UpdatedModel.slmx
```

Cleanup

To cleanup after performing the above steps, we close all models and remove all files that were created by scripts in this example. `slreq.clear` command will remove all Requirements and Links data from MATLAB session memory, so as to avoid conflicting with what you do next.

```
slreq.clear();
bdclose('all');
rmpath(workDir);
rmpath(docsFolder);
rmdir(workDir, 's');
% clear stored import location for our document to avoid prompt on rerun
slreq.import.docToReqSetMap(externalDoc, 'clear');
```


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 Simulink Requirements, 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 1-16.

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 1-16.

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 1-75.

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 1-20.

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 1-20 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 1-7.

Edit the Attribute Mapping

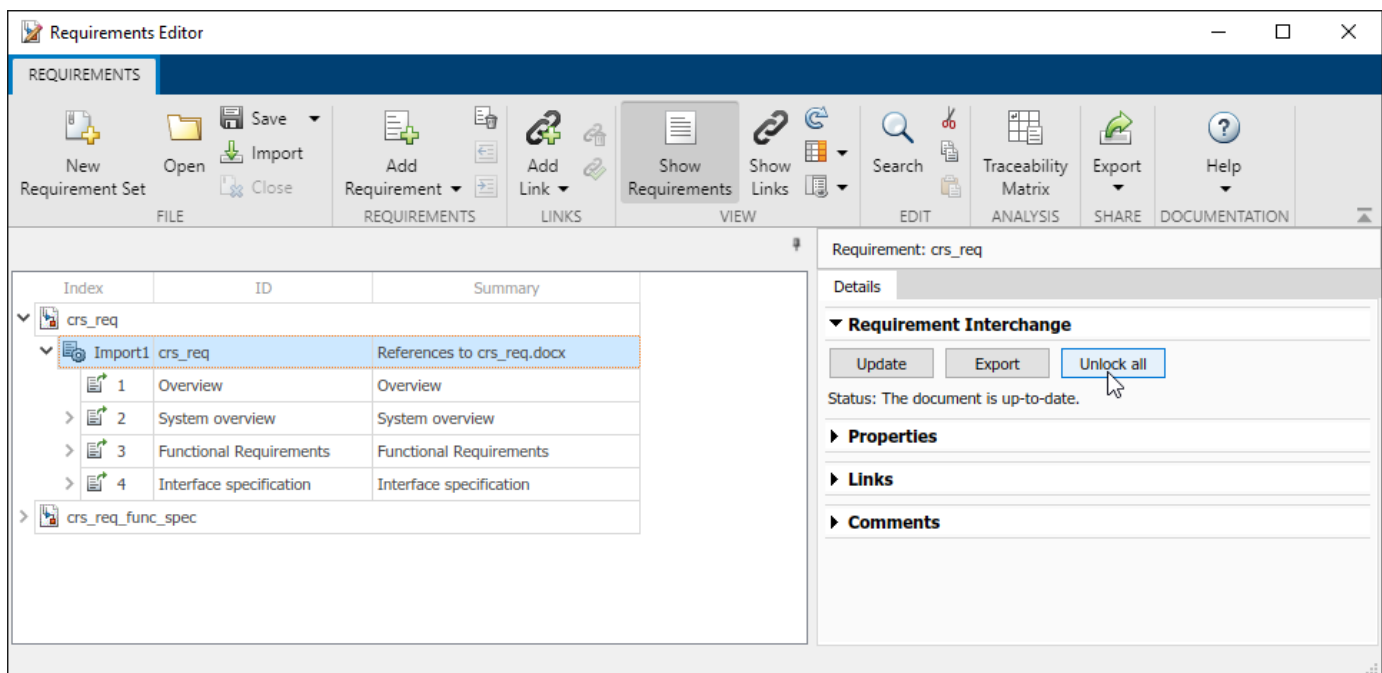
When you import requirements from a ReqIF file, the import process maps SpecObjectType object attributes to built-in requirement properties or requirement custom attributes. For more information about SpecObjectType object attributes, see “Choosing an Import Mapping” on page 1-16.

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 Simulink Requirements” on page 1-23.

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 “Customize Requirements with Custom Attributes” on page 1-48.

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 **Details** pane, under **Requirement Interchange**, click **Unlock all**.



To unlock individual requirements, navigate to the requirement and, in the **Details** 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 **Details** pane under **Custom Attribute Registries**. For more information about managing custom attributes for requirements, see “Customize Requirements with Custom Attributes” on page 1-48. 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 **Details** 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 1-9.

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, Simulink Requirements 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 1-41.

When you re-import the ReqIF file generated by the Simulink Requirements, 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 Simulink Requirements” on page 1-23.

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 1-38.

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 1-38.

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 1-39.

For more information about how the export mapping affects the exported content, see “Choosing an Export Mapping” on page 1-38.

See Also

slreq.import

More About

- “Import Requirements from ReqIF Files” on page 1-16
- “Export Requirements to ReqIF Files” on page 1-38

- “Create and Edit Attribute Mappings” on page 1-84
- “Import Requirements from Third-Party Applications” on page 1-7
- “Update Imported Requirements” on page 1-52
- “Best Practices and Guidelines for ReqIF Round Trip Workflows” on page 1-77

Best Practices and Guidelines for ReqIF Round Trip Workflows

Managing Requirement Custom IDs

- When you import requirements as referenced requirements, Simulink Requirements 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 Simulink Requirements 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 1-7
- “Round-Trip Importing and Exporting for ReqIF Files” on page 1-73
- “Update Imported Requirements” on page 1-52

Manage Custom Attributes for Requirements by Using the Simulink® Requirements™ API

This example shows how to use the Simulink® Requirements™ 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
    Columns 1 through 3
        {'MyCheckboxAttr...'}    {'MyComboboxAttr...'}    {'MyDateTimeAttr...'}
    Column 4
        {'MyEditAttribute'}
```

```
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'
    SID: 3
    FileRevision: 46
    ModifiedOn: 01-Sep-2021 19:06:48
    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
    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
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  
    {'MyCheckboxAttri...'}    {'MyComboboxAttri...'}    {'MyDateTimeAttri...'}  
    
```

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  
Columns 1 through 3  
    {'MyCheckboxAttr...'}    {'MyComboboxAttr...'}    {'MyDateTimeAttr...'}  
Column 4  
    {'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  
    {'MyCheckboxAttri...'}    {'MyComboboxAttri...'}    {'MyDateTimeAttri...'}  

```

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 Simulink® Requirements™ API” on page 2-65

More About

- “Customize Requirements with Custom Attributes” on page 1-48

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 Simulink Requirements requirement properties and custom attributes. For more information, see “Customize Requirements with Custom Attributes” on page 1-48. You can also save this mapping for reuse.

Edit the Attribute Mapping for Imported Requirements

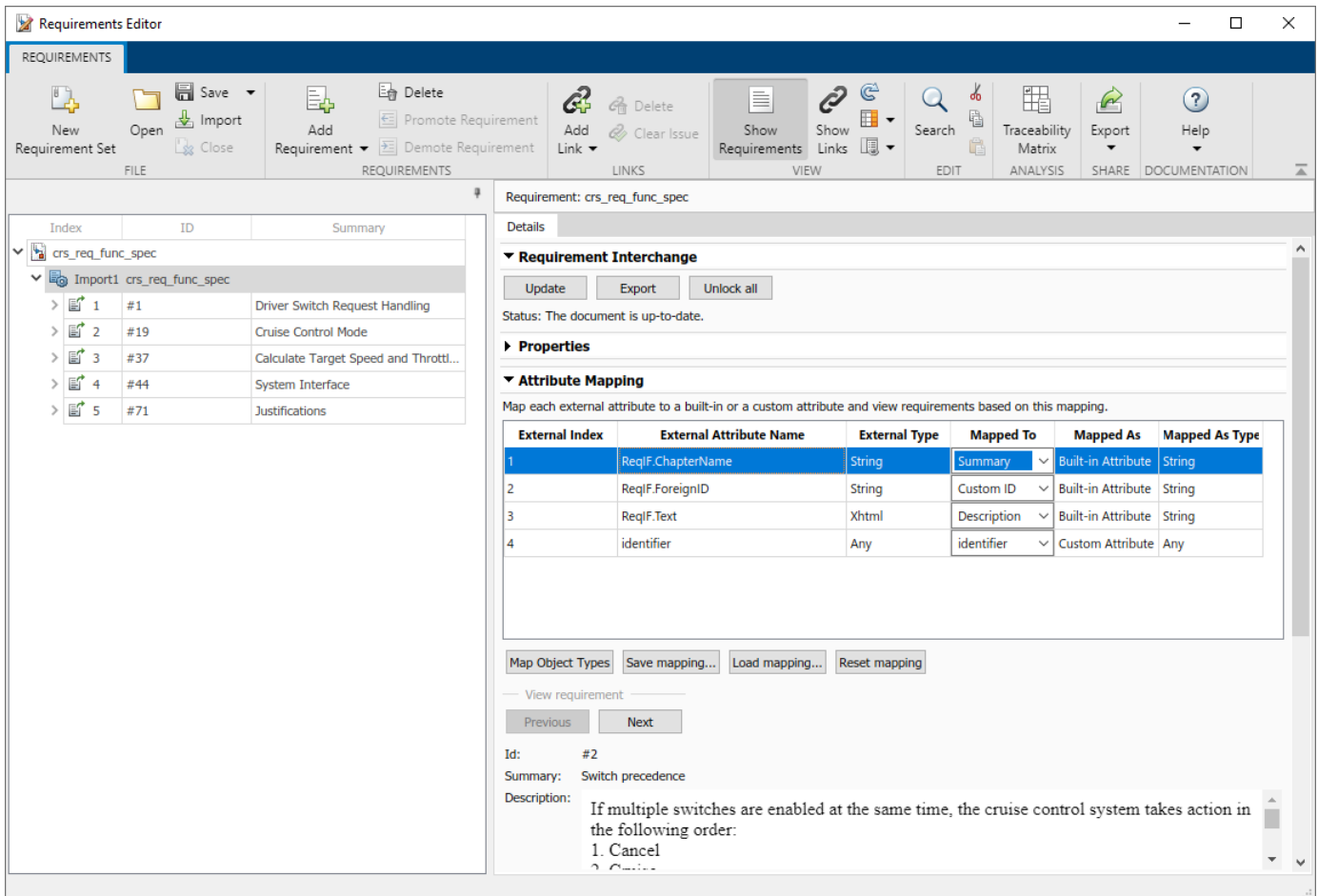
You can import requirements from ReqIF files in the Requirements Editor. For more information, see “Import Requirements from ReqIF Files” on page 1-16.

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 1-16.

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 1-16.
- 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 1-7.

You can see the attribute mappings in the **Details** 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 Simulink Requirements 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, Simulink Requirements 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

More About

- “Import Requirements from ReqIF Files” on page 1-16
- “Export Requirements to ReqIF Files” on page 1-38
- “Round-Trip Importing and Exporting for ReqIF Files” on page 1-73
- “Best Practices and Guidelines for ReqIF Round Trip Workflows” on page 1-77

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 Simulink® Requirements™ API.

Configure IBM Rational DOORS

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

```
rmi setup doors
```

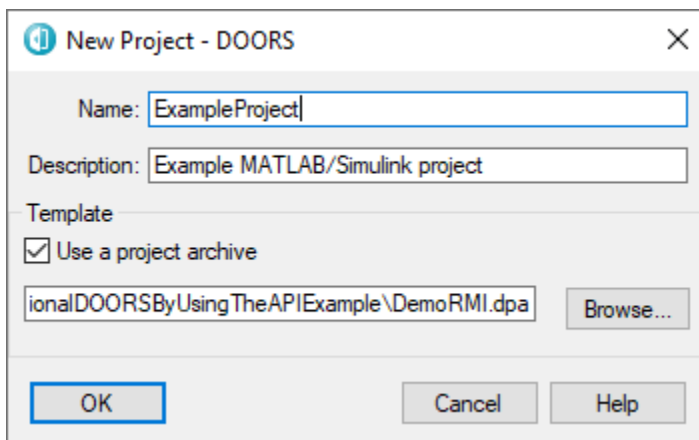
For more information, see “Configure IBM Rational DOORS Session” on page 1-33.

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.
```

```
refCount1 = 59
```

```
reqSetFilePath1 =
```

```
'C:\Users\jdoe\MATLAB\Examples\ImportRequirementsFromIBMRationalDOORSByUsingTheAPIExample\fuelSysReqSpec'
```

```
myReqSet1 =
```

```
ReqSet with properties:
```

```
    Description: ''  
        Name: 'fuelSysReqSpec'  
    Filename: 'C:\Users\jdoe\MATLAB\Examples\ImportRequirementsFromIBMRationalDOORSByUsingTheAPIExample\fuelSysReqSpec'  
    Revision: 1  
        Dirty: 1  
    CustomAttributeNames: {}  
        CreatedBy: 'ahoward'  
        CreatedOn: NaT  
        ModifiedBy: 'ahoward'  
        ModifiedOn: 12-Jul-2021 08:52:09
```

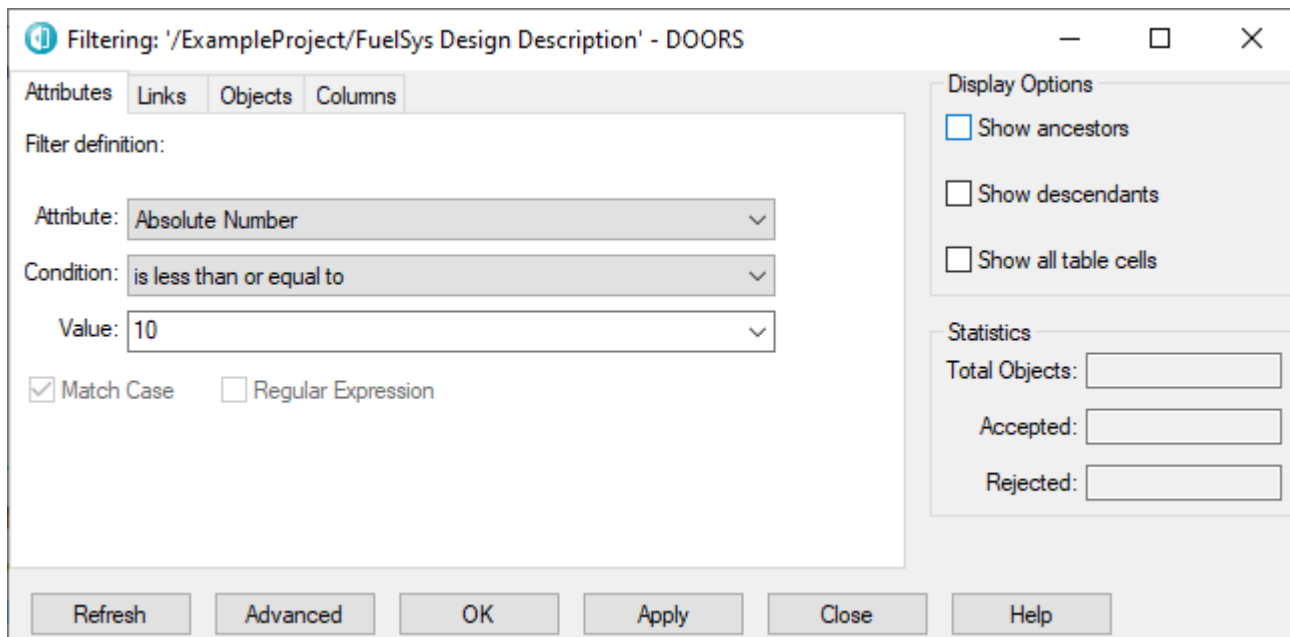
Import a Subset of Requirements from a Module

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.

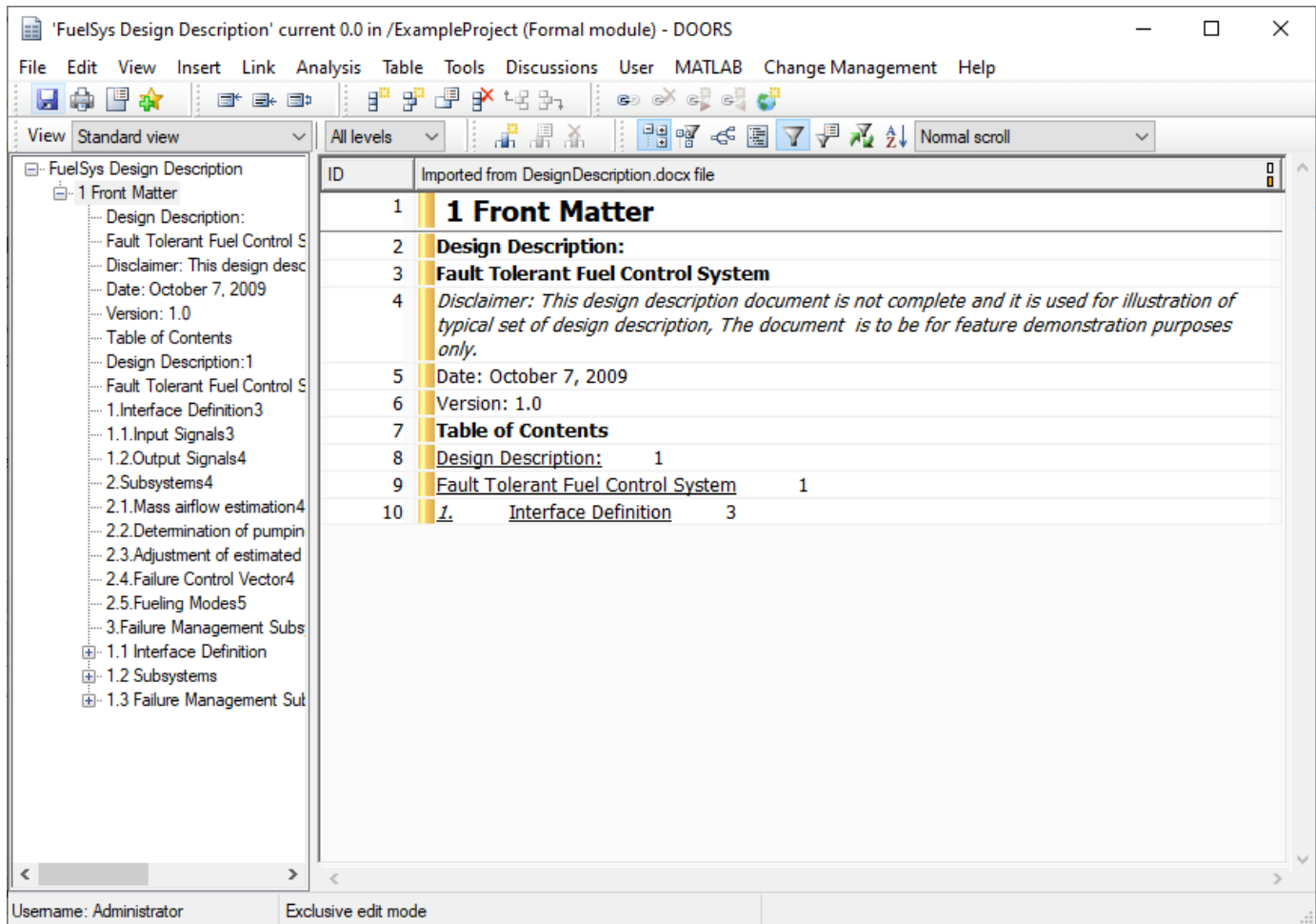
Filter the Requirements Module

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 Simulink Requirements, the process imports only the requirements that match the filter. When you import requirements by using the API, Simulink Requirements 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 Design Description of type linktype_rmi_doors ..  
done.
```

```
refCount2 = 10
```

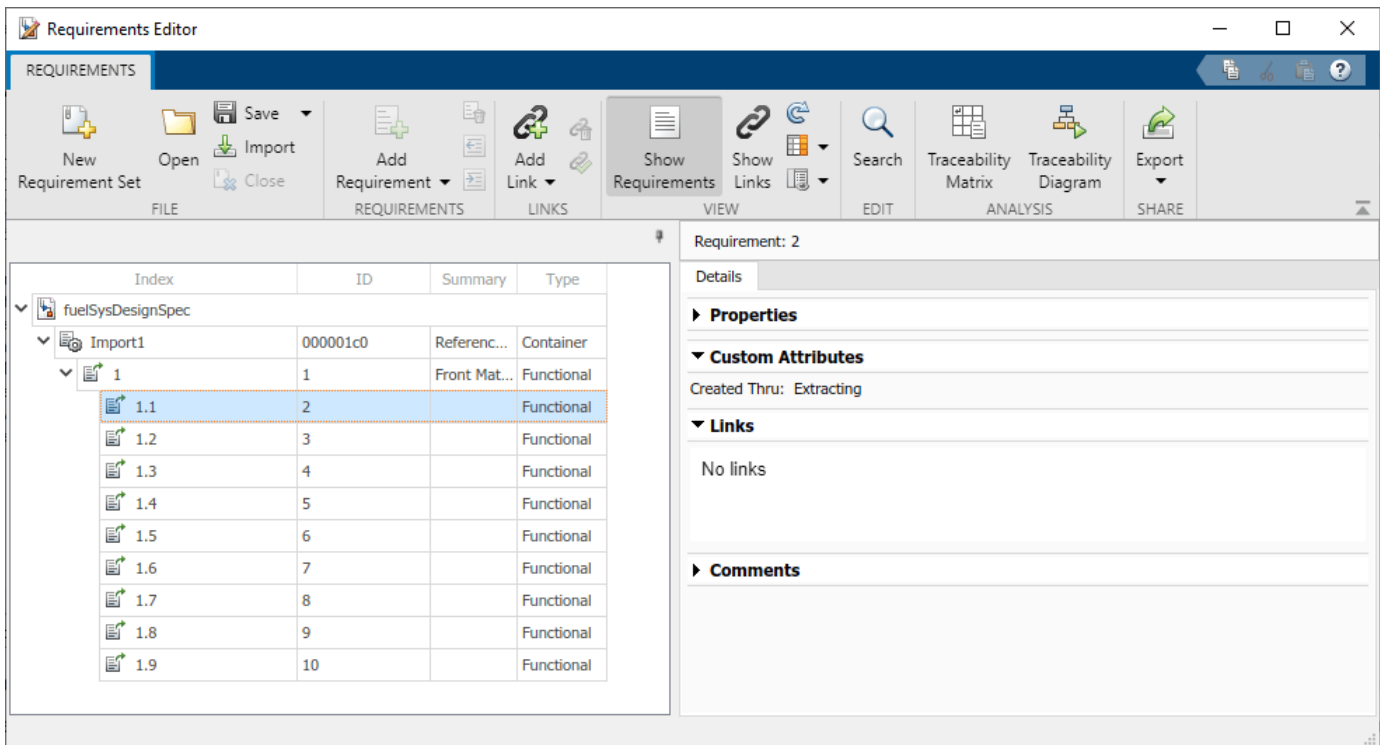
```

reqSetFilePath2 =
'C:\Users\jdoe\MATLAB\Examples\ImportRequirementsFromIBMRationalDOORSByUsingTheAPIExample\fuelSys

myReqSet2 =
  ReqSet with properties:

      Description: ''
      Name: 'fuelSysDesignSpec'
      Filename: 'C:\Users\jdoe\MATLAB\Examples\ImportRequirementsFromIBMRationalDOORSBy
      Revision: 1
      Dirty: 1
      CustomAttributeNames: {'Created Thru'}
      CreatedBy: 'ahoward'
      CreatedOn: NaT
      ModifiedBy: 'ahoward'
      ModifiedOn: 12-Jul-2021 08:52:45
    
```

Simulink Requirements 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 “Customize Requirements with Custom Attributes” on page 1-48.

Update the Filtered Requirement Set

After you import the requirement set, you can update it. For more information, see “Update Imported Requirements” on page 1-52.

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.

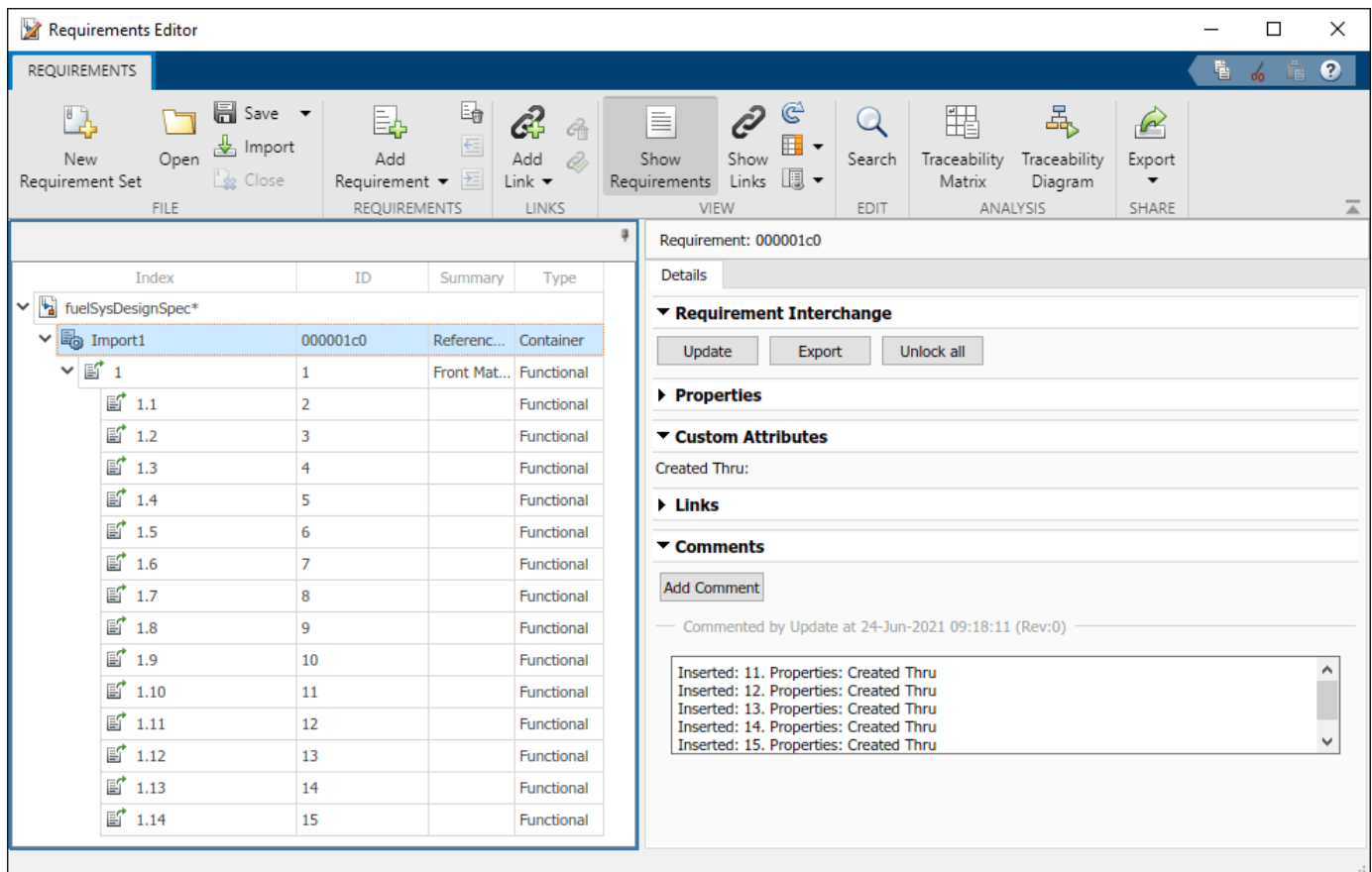
Find the Import node from the requirement set myReqSet2. Update the requirement set.

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

```
Importing from FuelSys Design Description of type linktype_rmi_doors ..
done.
```

```
status =
'Update completed. Refer to Comments on Import1.'
```

Simulink Requirements 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 Design Description of type linktype_rmi_doors ..
done.
```

```
status =
'Update completed. Refer to Comments on Import1.'
```

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

The screenshot shows the Requirements Editor window. The left pane displays a tree view of requirements under 'fuelSysDesignSpec*'. The 'Import1' container contains 5 requirements with IDs 1 through 5, all of type 'Functional'. The right pane shows the details for requirement '000001c0', including custom attributes, links, and comments. The comments section shows a list of properties that were reordered, deleted, or inserted.

Index	ID	Summary	Type
fuelSysDesignSpec*			
Import1	000001c0	Referenc...	Container
1	1	Front Mat...	Functional
1.1	2		Functional
1.2	3		Functional
1.3	4		Functional
1.4	5		Functional

Cleanup

Clear the open requirement sets.

```
slreq.clear;
```

See Also

`updateFromDocument` | `slreq.import`

More About

- “Import Requirements from IBM Rational DOORS” on page 1-33
- “Working with IBM Rational DOORS 9 Requirements” on page 7-50
- “Import Requirements from Third-Party Applications” on page 1-7

Requirements Traceability and Consistency

- “Link Blocks and Requirements” on page 2-2
- “Track Requirement Links with a Traceability Matrix” on page 2-5
- “Visualize Links with a Traceability Diagram” on page 2-18
- “Assess Allocation and Impact” on page 2-27
- “Requirement Links” on page 2-32
- “Define Custom Requirement and Link Types” on page 2-40
- “Customize Links with Custom Attributes” on page 2-43
- “Requirements Consistency Checks” on page 2-46
- “Manage Navigation Backlinks in External Requirements Documents” on page 2-50
- “Use Command-line API to Update or Repair Requirements Links” on page 2-52
- “Manage Custom Attributes for Links by Using the Simulink® Requirements™ API” on page 2-65
- “Make Requirements Fully Traceable with a Traceability Matrix” on page 2-70
- “Modeling System Architecture of Small UAV” on page 2-84

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.

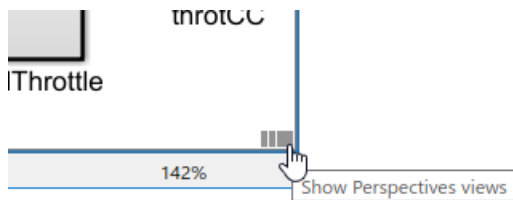
In this tutorial, link requirements to a model by using the model requirements perspective. Visual elements highlight links between requirements and blocks.

- 1 Open the example project by entering

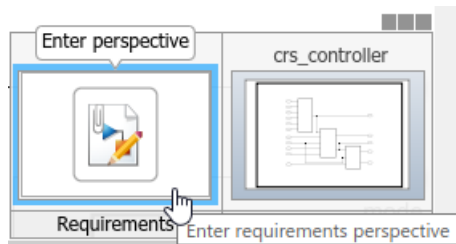
```
slreqCCProjectStart
```

Open `crs_controller` from the `models` folder.

- 2 In the model canvas, click the perspectives control in the lower-right corner.



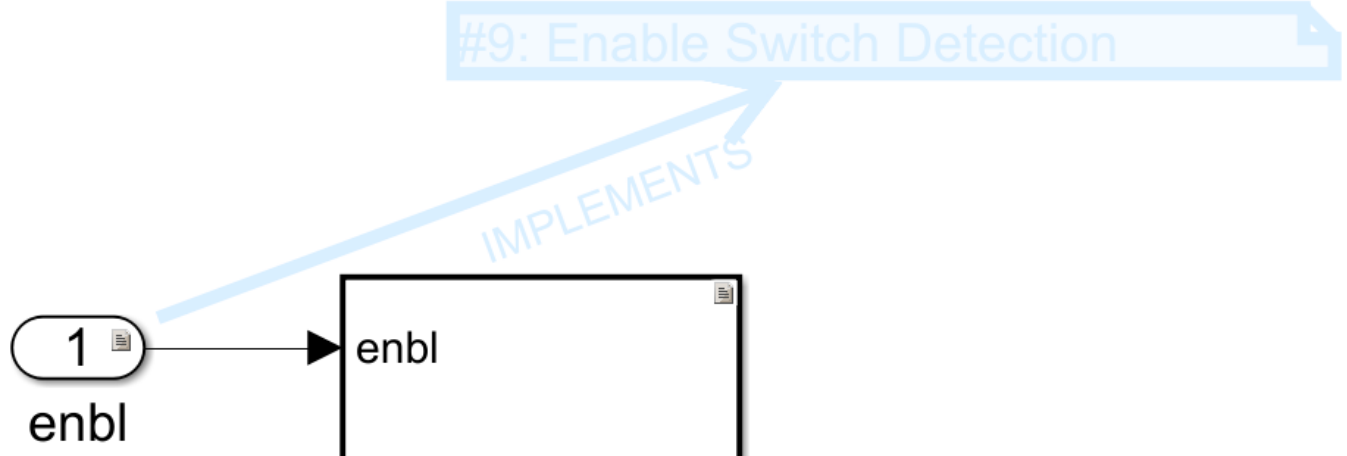
- 3 Open the requirements perspective by clicking the **Requirements** icon.



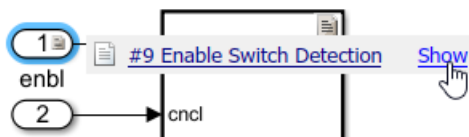
The Requirements Browser appears at the bottom of the model canvas. When you select a requirement, the Property Inspector displays the requirement's properties.

- 4 Link a requirement to a model element:

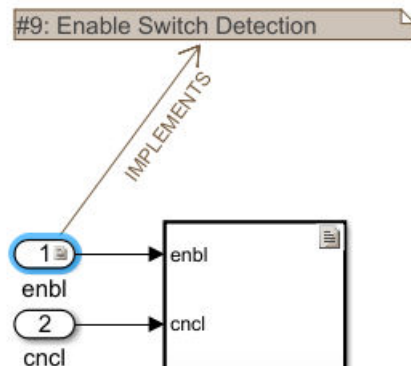
- 1 In the Requirements Browser, search for `Enable Switch Detection`.
- 2 Link to the `enb1 Inport` block by clicking and dragging the requirement to the block. An annotation template appears.
- 3 Place the requirement annotation by clicking on the canvas. Create a link without an annotation by clicking outside the canvas.



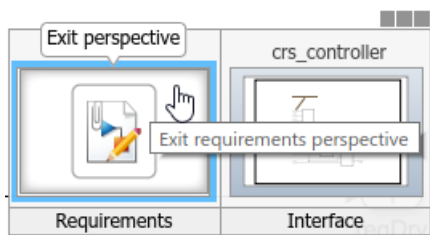
- 5 The block displays a link badge. To display information about the requirement, click the badge and select **Show**.



Clicking **Show** displays the requirement ID, requirement summary, and link type. For information on link types, see “Requirement Links” on page 2-32.



- 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.
- 6 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**.

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:

- Simulink Requirements requirement sets
- Simulink models
- System Composer models
- Simulink Test test files
- Simulink data dictionaries
- MATLAB M-files

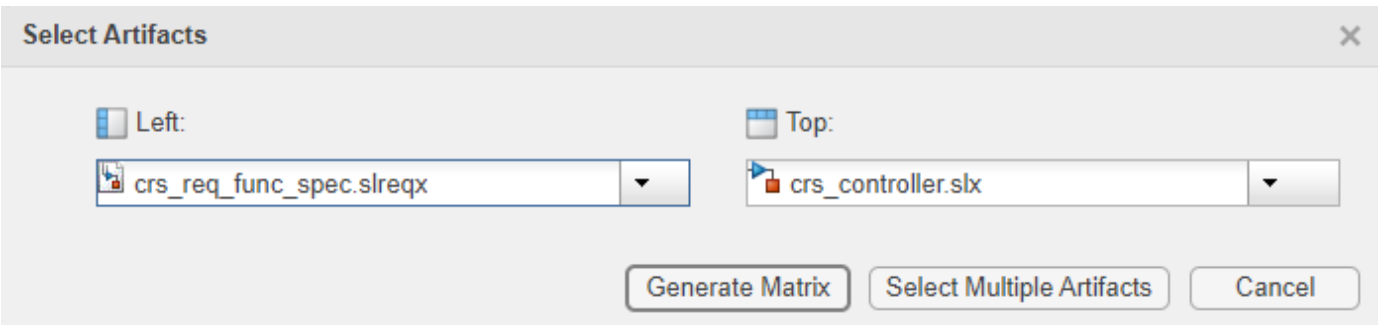
To open the Traceability Matrix window, use one of these approaches:

- 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:

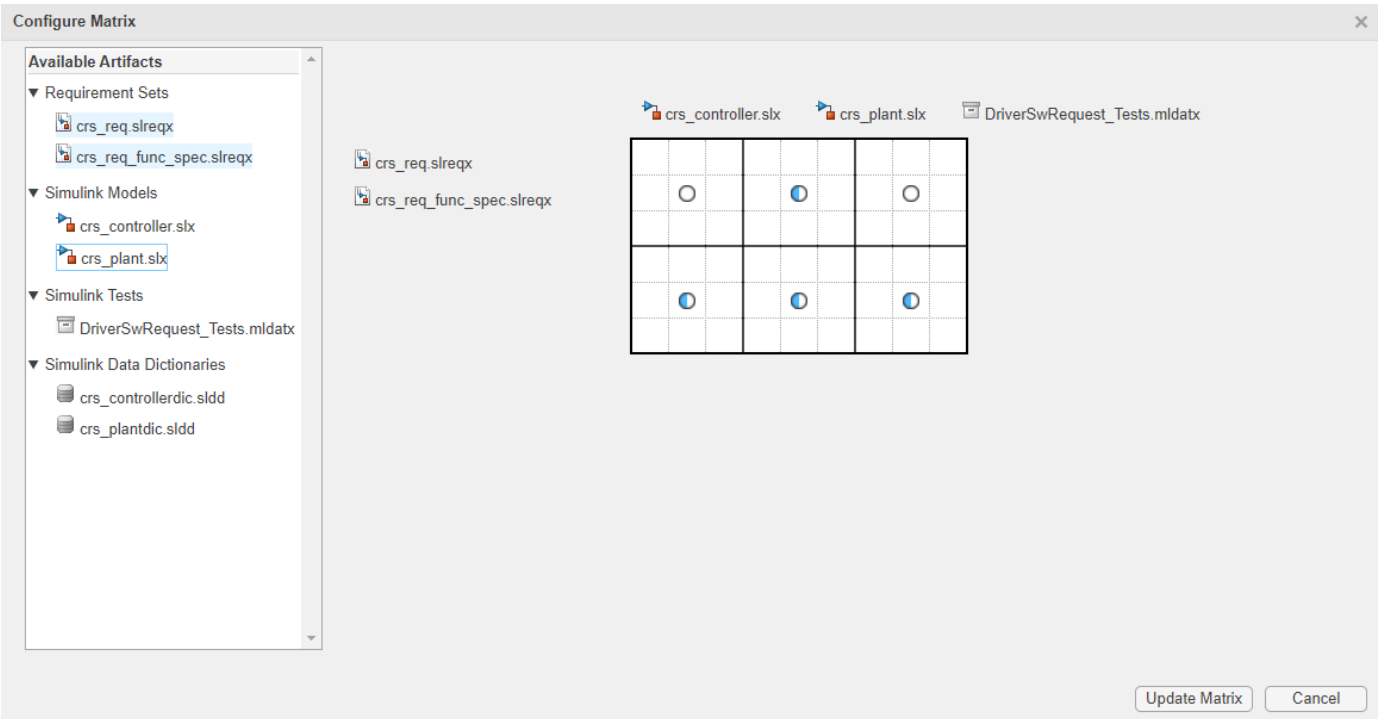
```
slreq.generateTraceabilityMatrix
```

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.

The screenshot displays the Traceability Matrix application window. The interface includes a top navigation bar with a 'HOME' tab and a help icon. Below this 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 area is divided into a 'Filter Panel' on the left and a 'Traceability Matrix' on the right. The 'Filter Panel' has sections for 'Top' (Type: Leaf Block, Subsystem; Link: Missing Links, Missing Expected Links) and 'Left' (Type: Container, Functional, Justification; Link: Missing Links; Change Tracking: With Change Issues; Cell: Type: Implements; Change Tracking: With Change Issues). The 'Traceability Matrix' shows a grid with 'crs_controller' as the column header and 'crs_req_func_spec' as the row header. The grid contains several cells with icons representing links, such as 'Driver Switch Request Handling', 'Cruise Control Mode', 'Disable Cruise Control system', 'Operation mode determination', 'Calculate Target Speed and Th', 'Disabled case', 'Enabled case', 'Activated case', 'Resume mode', 'System Interface', and 'Inputs'. A vertical scrollbar is visible on the right side of the matrix.

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 shows the Traceability Matrix application window. The title bar reads "Traceability Matrix". The interface is divided into several sections:

- HOME (Toolbar):** Contains icons for "Add", "Adjust Artifacts", "Update", "Scope", "Expand All", "Collapse All", "Highlight Missing Links", "Create Link", "Remove Links", "Clear Change Issue", and "Export".
- Filter Panel (Left):**
 - Top:** "Link" section with "Missing Expected Links" and "Missing Links".
 - Type:** "Models" (Leaf Block, Stateflow Object, Subsystem) and "Simulink Tests" (Test Case, Test File, Test Suite).
 - Left:** "Change Tracking" (With Change Issues), "Link" (Missing Links), "Type" (Container, Functional, Justification), and "Cell" (Type, Related to, Change Tracking (With Change Issues)).
- Main Content Area:**
 - Tab: "Simulink Requirements vs Simulink Model".
 - Selected artifacts: "crs_plant, crs_controller, DriverSwRequest Tests" and "crs_req, crs_req_func_spec".
 - Grid: A traceability matrix with rows for requirements and columns for model components. A solid blue vertical line is between "crs_controller" and "CruiseControlMode". A solid blue horizontal line is between "crs_req" and "crs_req_func_spec".

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.

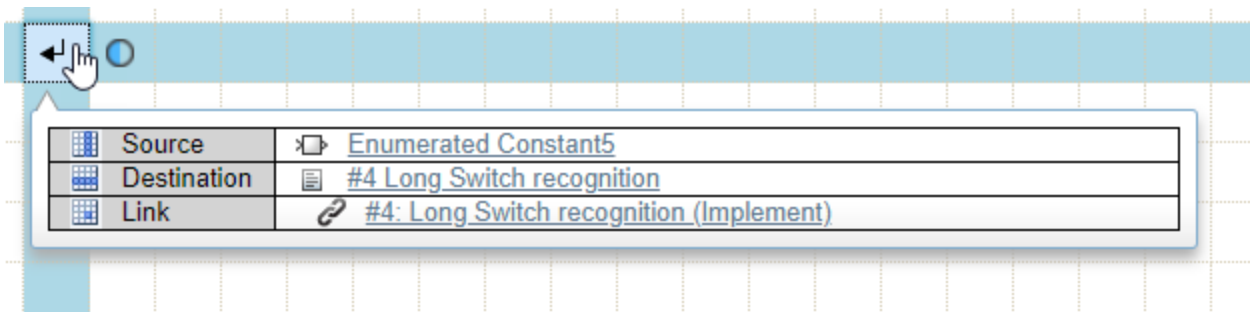
In order to be able to add an artifact to the traceability matrix, the artifact must either:

- 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.

Left	#8 Set Switch Detection
Top	DriverSwRequest
Link	None (Create)
Note	Link exists between some items under this row and column. Expand row or column to see the links.

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.

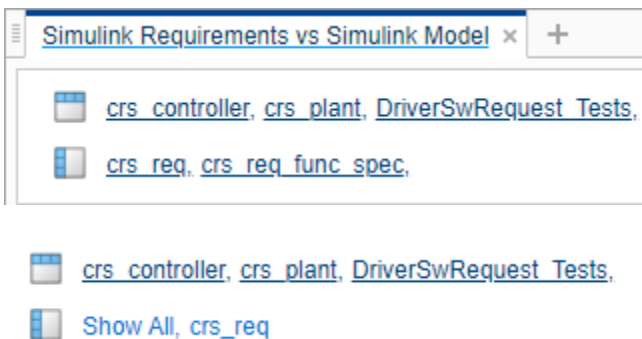
Focus the Display

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

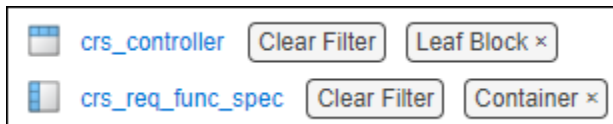
▼ **Change Tracking**

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.

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.

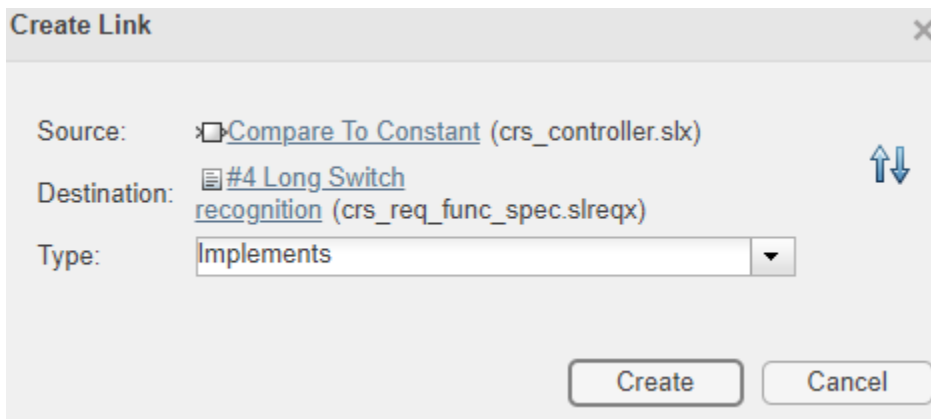
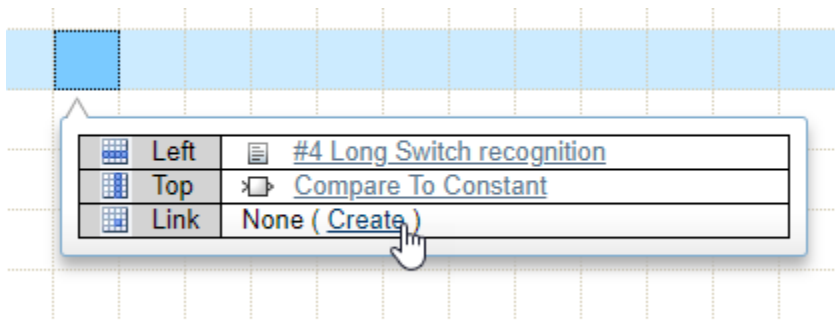
	crs_controller	CruiseControlMode	disableCaseDetection	IsKeyPositionOn	In1	Constant	Relational Operator	Out1	IsShiftDrive	In1
crs_req_func_spec										
#1 Driver Switch Request Hand										
#2 Switch precedence										
#3 Avoid repeating command										
#4 Long Switch recognition										
#7 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 2-10.

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.

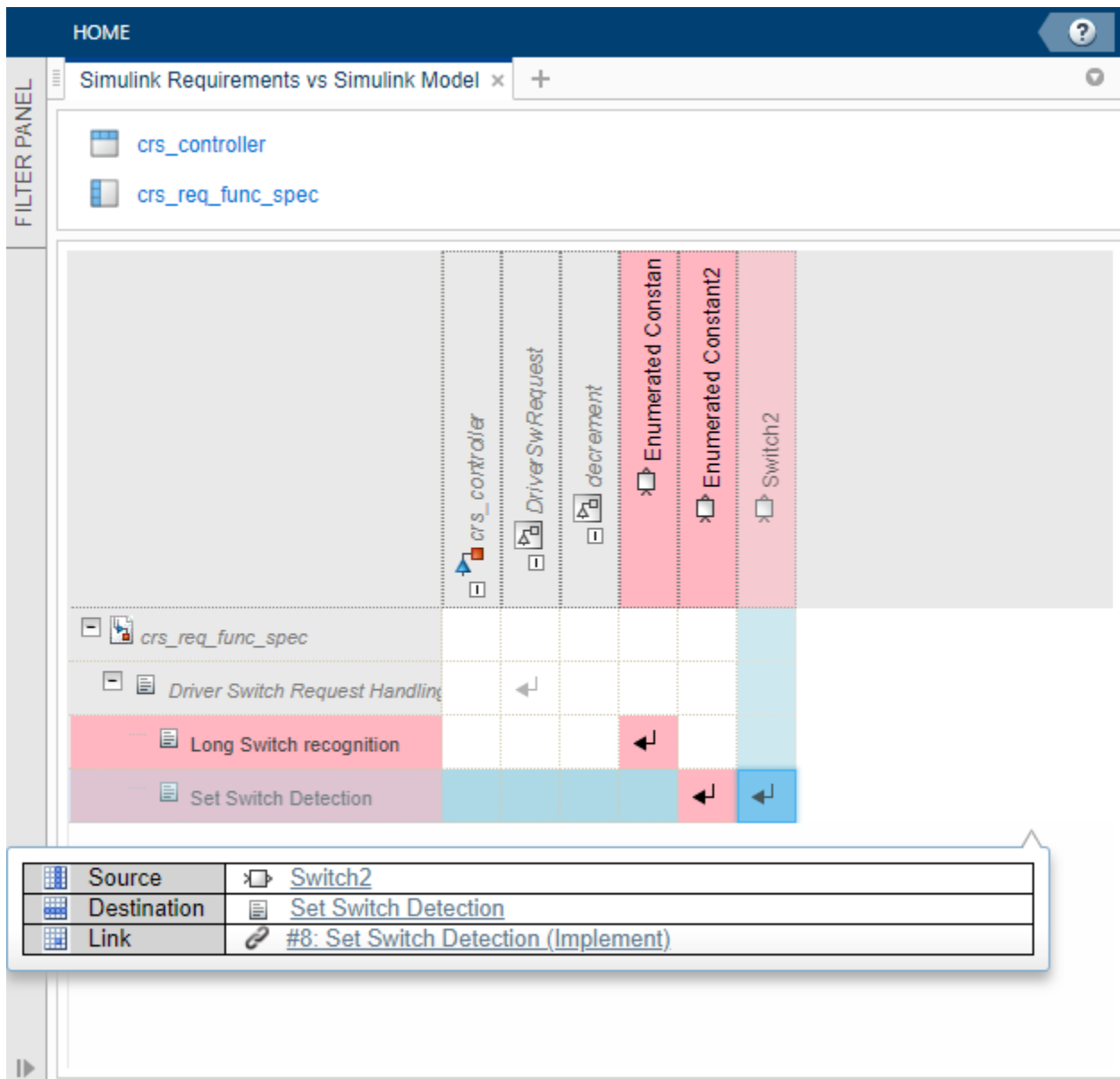
Remove a Link

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 4-3.

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.



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 2-70
- “Programmatically Generate a Traceability Matrix”

More About

- “Link Blocks and Requirements” on page 2-2
- “Define Custom Requirement and Link Types” on page 2-40
- “Requirement Links” on page 2-32
- “Track Changes to Requirement Links” on page 4-3

Visualize Links with a Traceability Diagram

You can visualize the traceability structure of requirements and other Model-Based Design items by using the Traceability Diagram. 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 2-32.

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 2-33. For more information, see “Impact Direction” on page 2-21.

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 2-27.

Generate a Traceability Diagram

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.

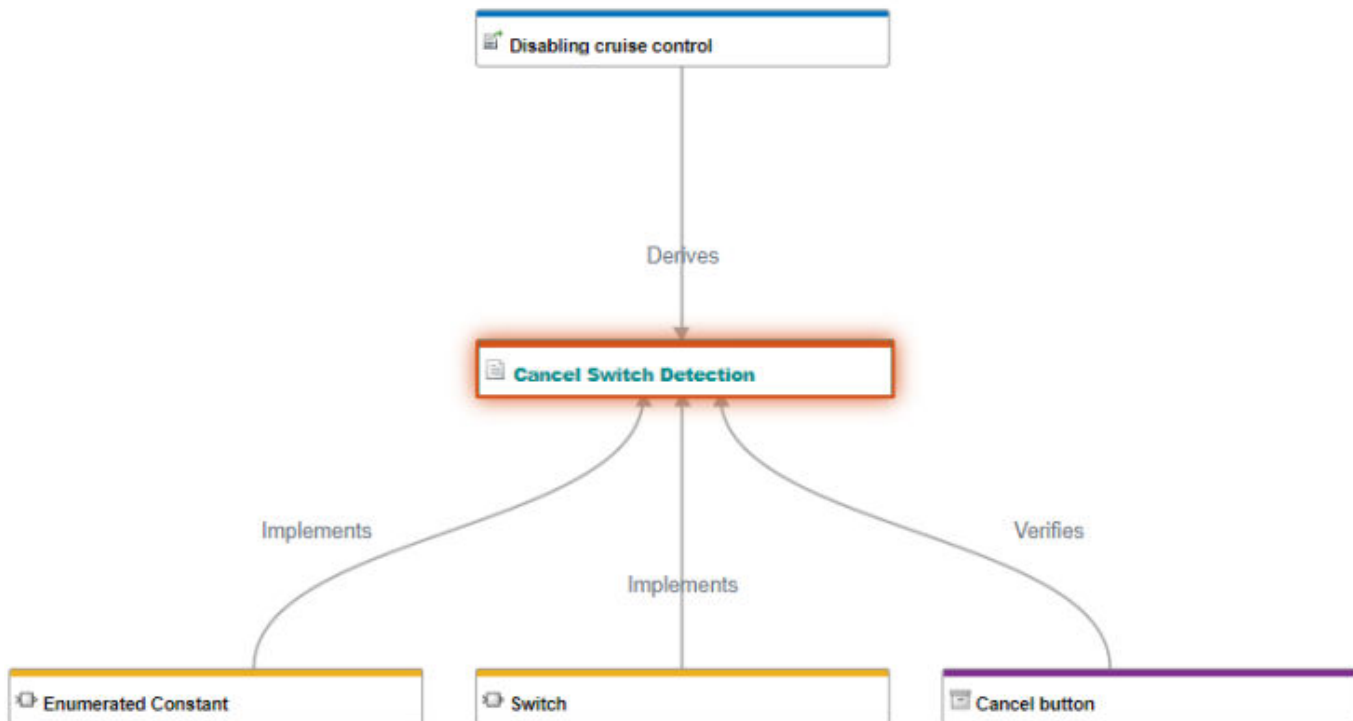
To create a traceability diagram:

- 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`.

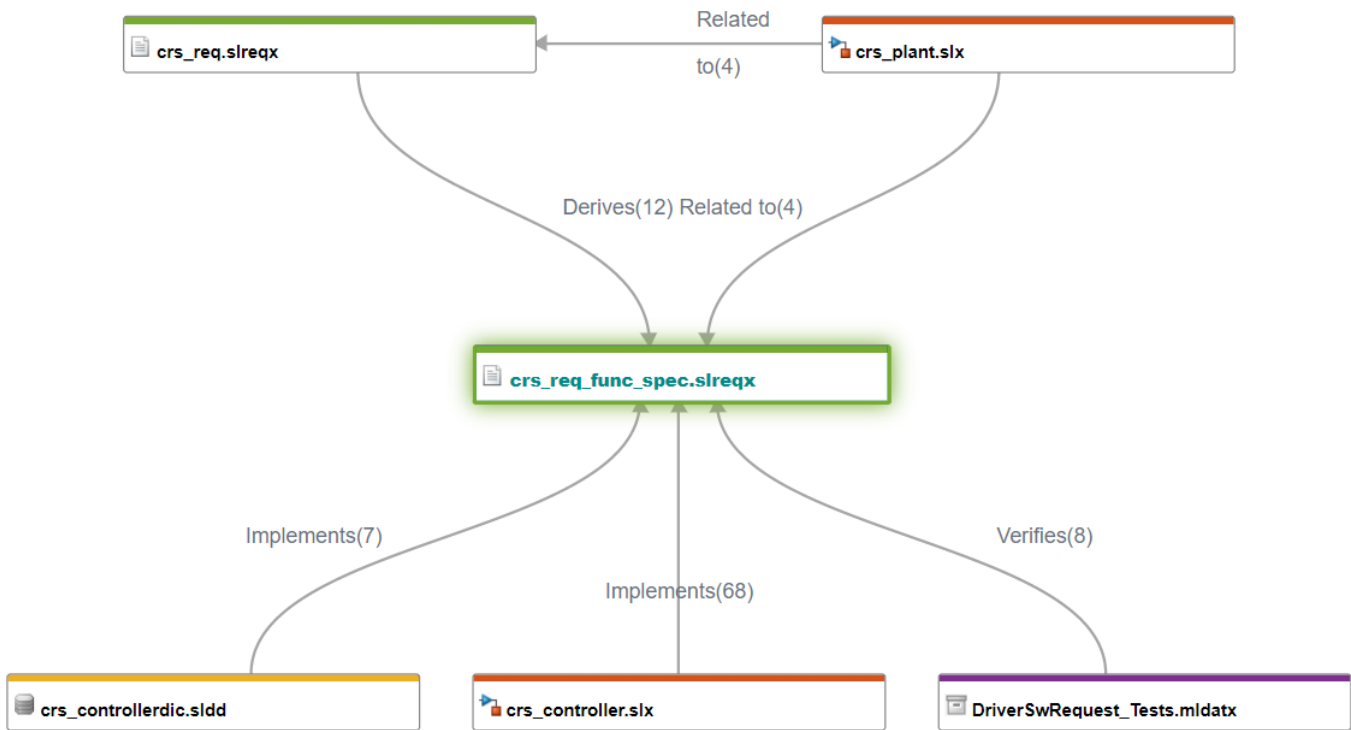
You can create a new diagram from a node in an existing diagram by right-clicking the node and selecting **View Traceability Diagram**.

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.

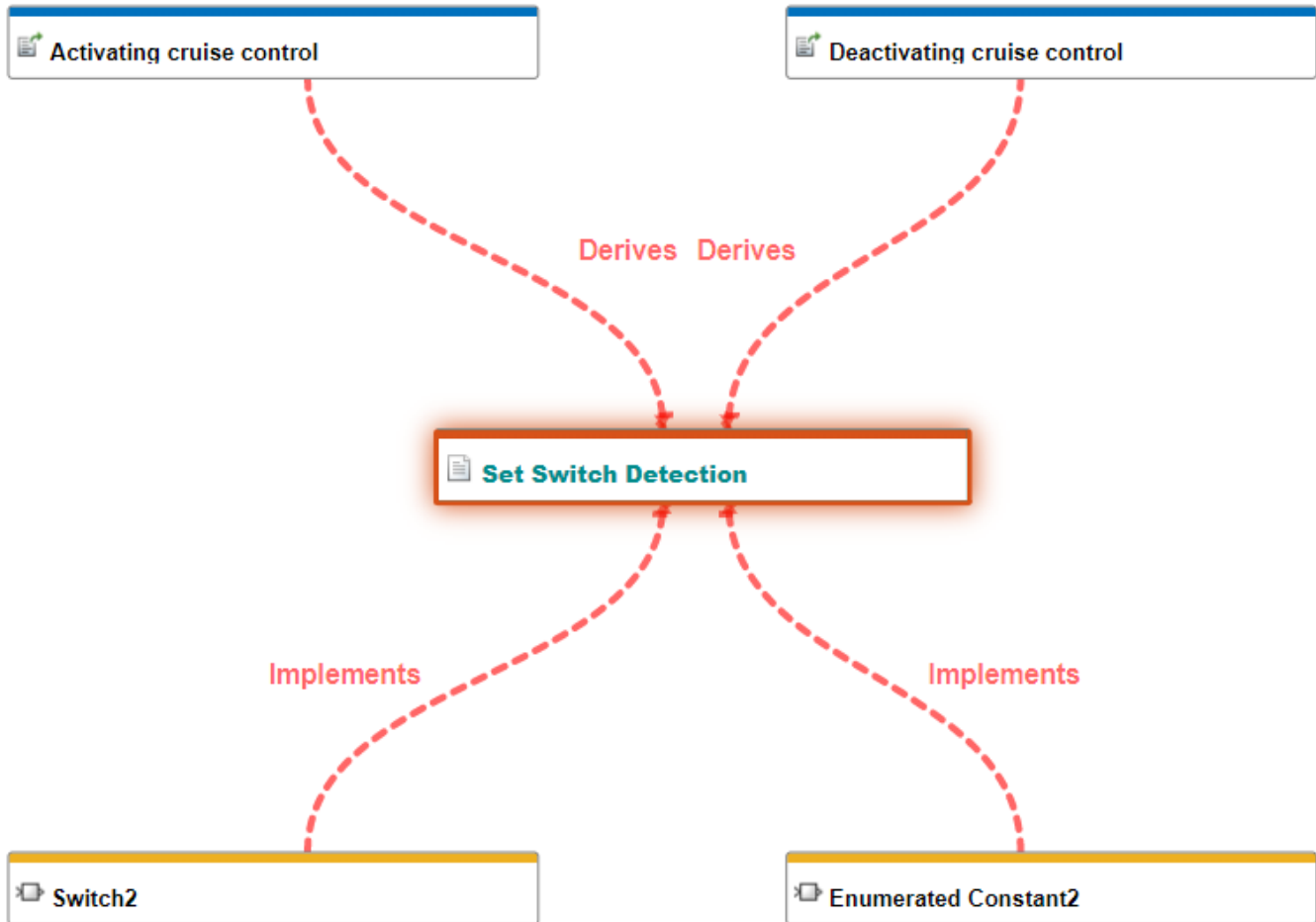


The node border color indicates the artifact that the node belongs to, or the artifact domain, such as Simulink Requirements files, or Simulink models and libraries. The **Legend** pane displays the artifacts, the artifact colors, and the domain that each artifact belongs to.

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 “Resolve Links” on page 2-36.

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 2-33 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 4-3.



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 2-21 and “Link Types” on page 2-33.

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 2-33. 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

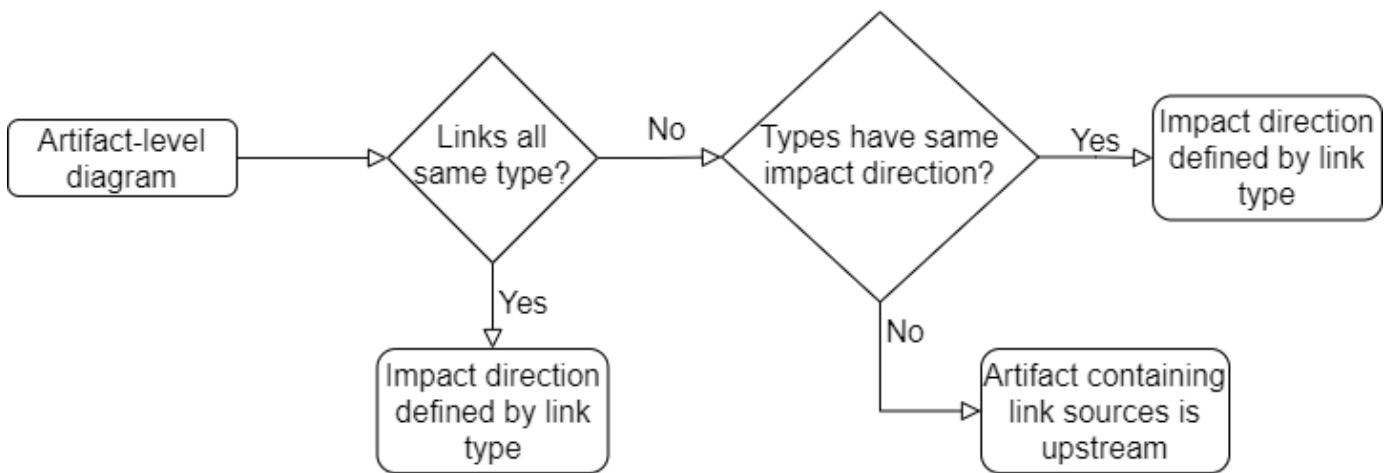
Link Type	Upstream	Relationship	Downstream	Impact direction
Verify	Destination	Verified by	Source	Opposite of link direction
Derive	Source	Derives	Destination	Same as link direction
Refine	Destination	Refined by	Source	Opposite of link direction
Confirm	Source	Confirmed by	Destination	Same as link direction

For example, 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 2-27.

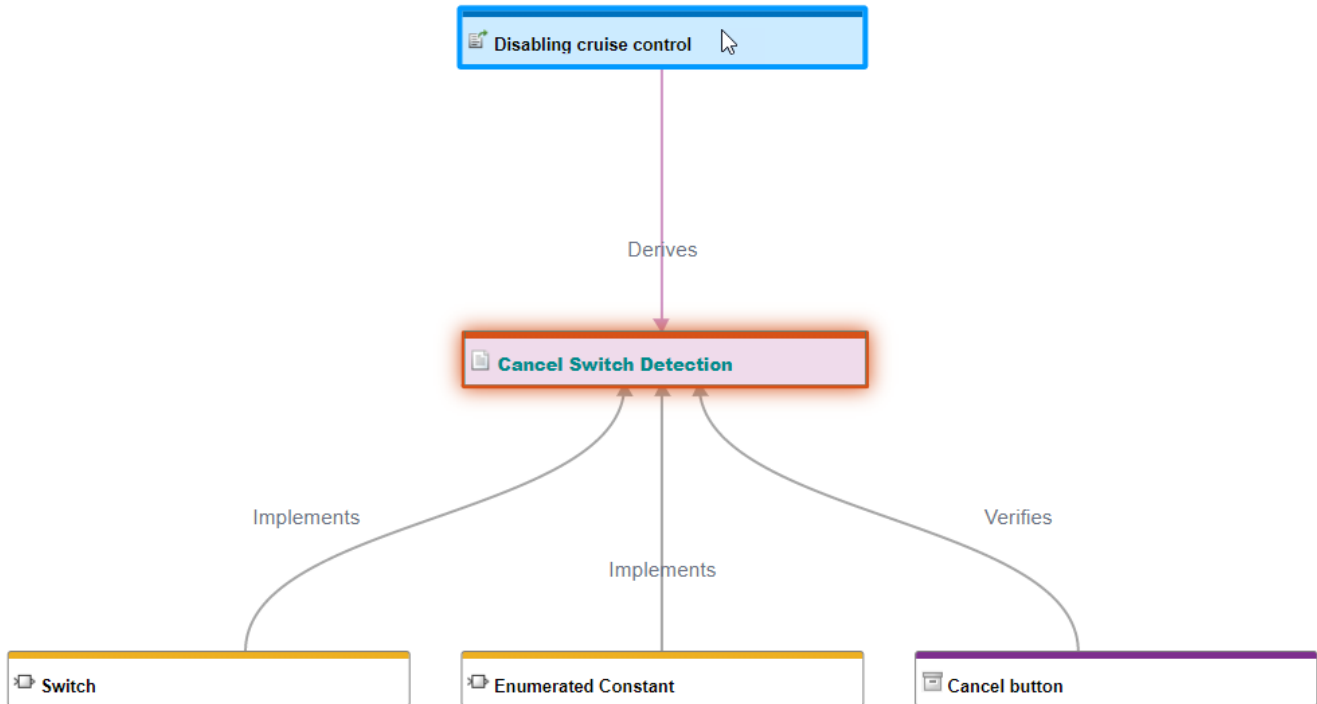
In an artifact-level diagram, if all links between items in two artifacts have the same type, then the link type defines the impact direction. If the links between items in two artifacts have different types, but all types define the impact direction the same way, they use the impact direction defined 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.



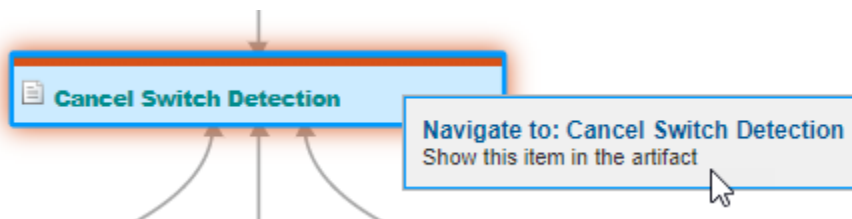
Use the Traceability Diagram

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.



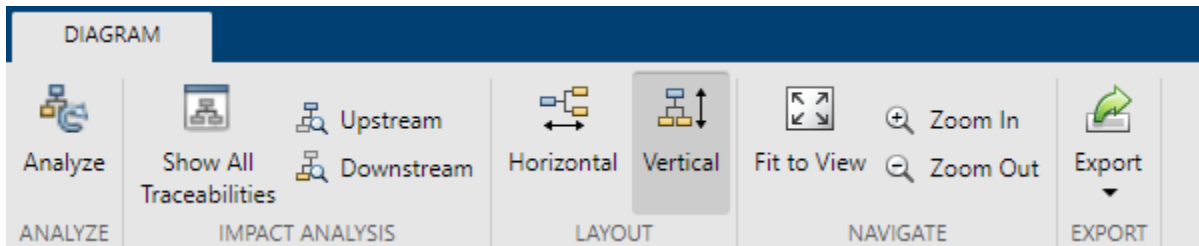
Refresh the Diagram

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 2-20.

Click **Analyze** to refresh the diagram.

Modify the Traceability Diagram View

You can modify the view in the Traceability Diagram by using the toolstrip or the **Legend** and **Overview** panes.

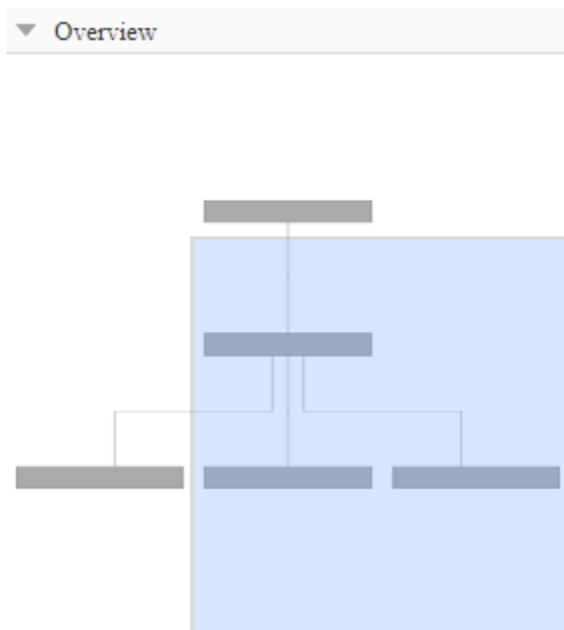


Layout and Navigation

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 all 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 2-27
- “Track Requirement Links with a Traceability Matrix” on page 2-5
- “Track Changes to Requirement Links” on page 4-3
- “Perform an Impact Analysis”
- “Trace Artifacts to Units for Model Testing Analysis” (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 2-32. For more information, see “Visualize Links with a Traceability Diagram” on page 2-18.

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.

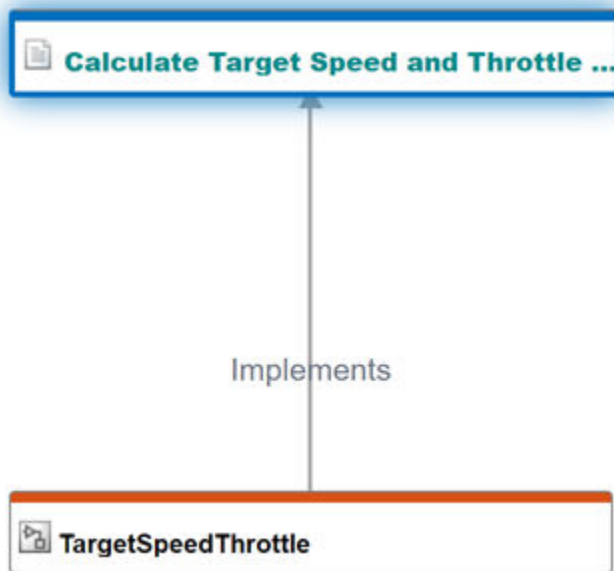
For example, the “Requirements Definition for a Cruise Control Model” describes a cruise control system design. The `crs_req_func_spec` requirement set contains requirements that ensure that the system meets the functional specifications. The `Calculate Target Speed and Throttle Value` requirement is 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 3-16.

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.



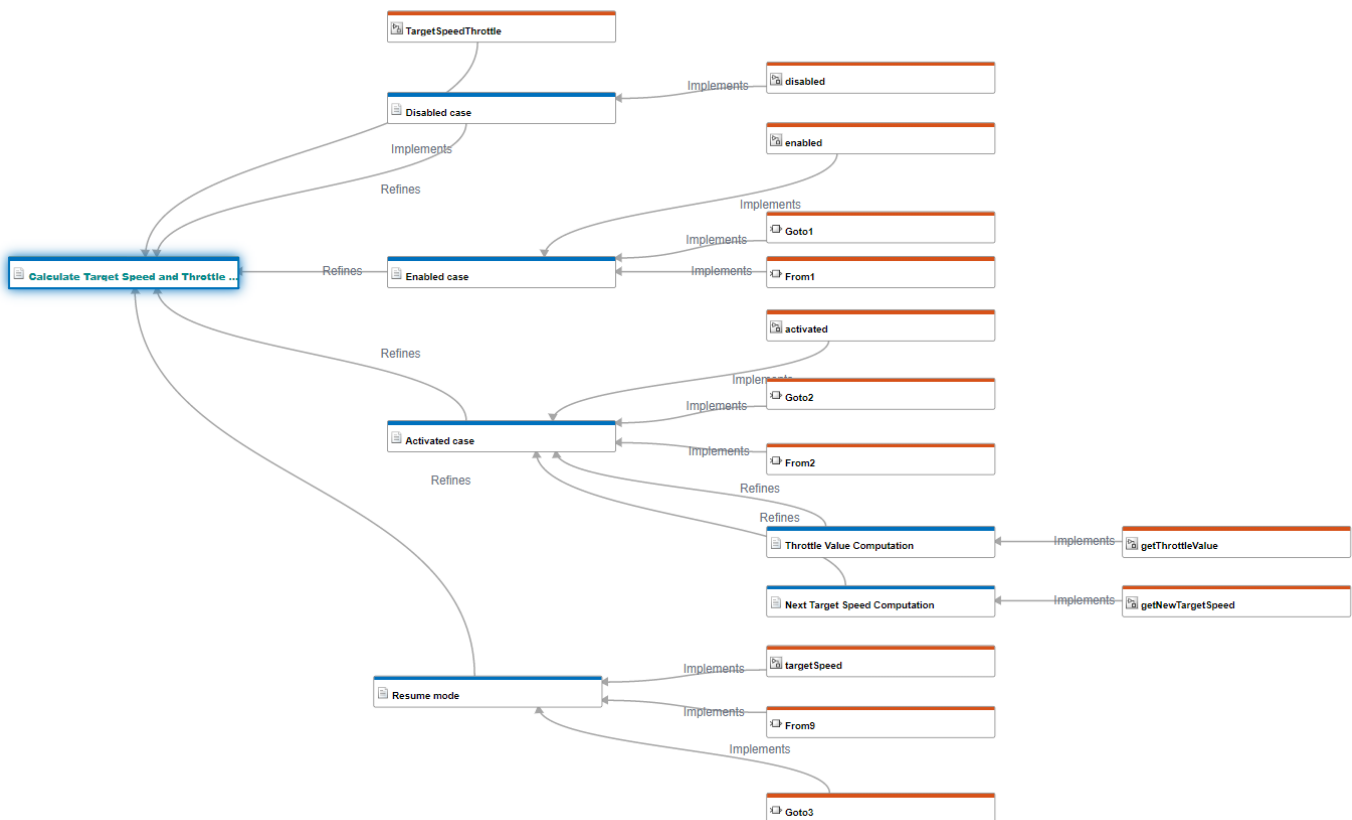
You can use the Requirements Editor, Requirements Perspective, or Traceability Matrix to create links. For more information, see:

- “Link Blocks and Requirements” on page 2-2
- “Link to Test Cases from Requirements”
- “Track Requirement Links with a Traceability Matrix” on page 2-5

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 a Traceability Diagram” on page 2-18.

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

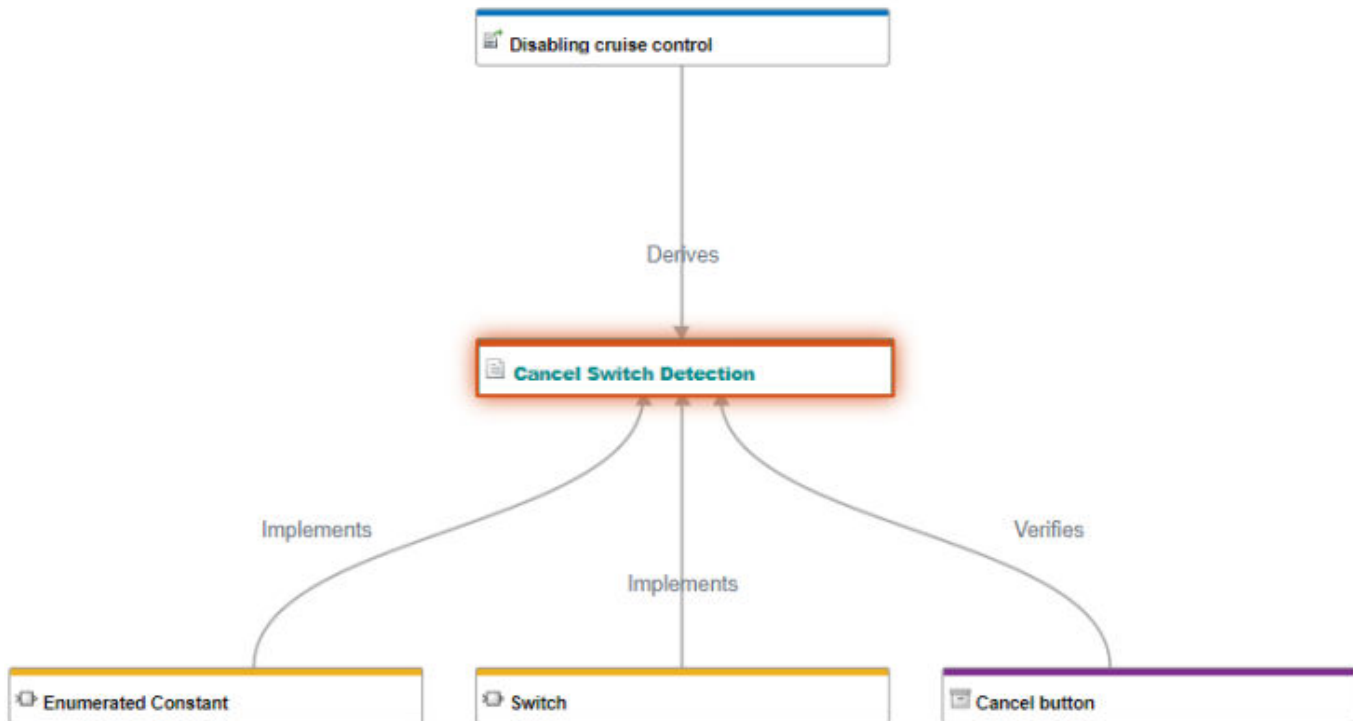
Simulink Requirements allows you to track changes to requirements. If a linked requirement changes, the associated link has a change issue. For more information, see “Track Changes to Requirement Links” on page 4-3.

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.

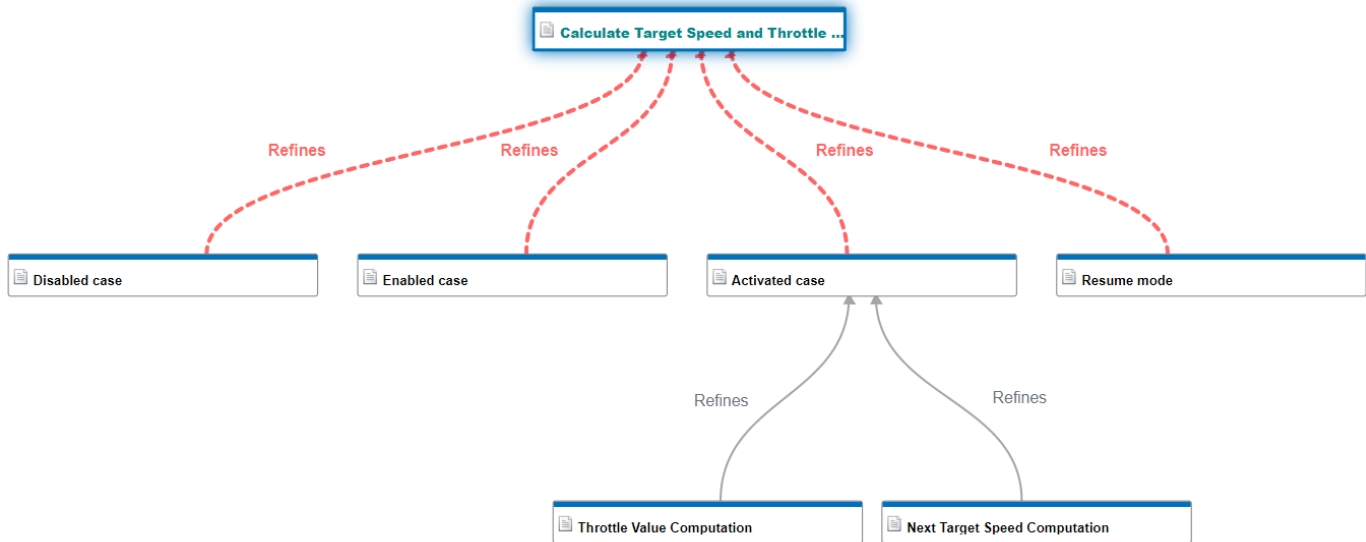
For example, this diagram shows an indirect link between the `Disabling cruise control` requirement and the `Enumerated Constant` Simulink block:



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 2-23.

See Also

`slreq.generateTraceabilityDiagram`

More About

- “Visualize Links with a Traceability Diagram” on page 2-18
- “Review Requirements Implementation Status” on page 3-2
- “Review Requirements Verification Status” on page 3-6
- “Track Changes to Requirement Links” on page 4-3

Requirement Links

Track how your requirements relate to your model design by using Simulink Requirements to create links between your requirements and various Simulink model elements, including blocks, Stateflow objects, Simulink Test test objects, Simulink data dictionary entries, MATLAB code lines, and other requirements.

Each link has a corresponding `slreq.Link` object. The link points from a source item to a destination item.

You can create links to blocks and Stateflow objects from the Simulink Editor by dragging requirements from the **Requirements Browser** in the Requirements Perspective View. You can create links to Simulink Test test objects from the Test Manager or from the Requirements Editor. For more information on linking Simulink model elements to requirements, see “Link Blocks and Requirements” and “Link to Test Cases from Requirements”.

Requirement links identify the requirement by the Session Independent Identifier (SID) instead of the Custom ID, such that the link remains functional if the Custom ID is modified.

Linkable Items

You can create links between these requirements items, model entities, test artifacts, and code:

- Simulink Requirements 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
- Simulink Test objects:
 - Test files
 - Test suites
 - Test cases

- Iterations
- Assessments
- Lines of MATLAB code in:
 - MATLAB .m files.
 - MATLAB Function blocks. For more information, see “Integrate Basic Algorithms Using MATLAB Function Block”.
 - MATLAB-based Simulink tests. For more information, see “Test Models Using MATLAB-Based Simulink Tests” (Simulink Test).

You can set external artifacts like URLs as link destinations by creating MATLAB structures. There are two approaches available:

- Create a link destination structure, then create a link between the requirement and the destination.

```
myLinkDest = struct('domain', 'linktype_rmi_url', 'artifact', ...
  'https://www.mathworks.com', 'id', '')
```

```
myLinkDest =
```

```
struct with fields:
```

```
    domain: 'linktype_rmi_url'
    artifact: 'www.mathworks.com'
        id: ''
slreq.createLink(myReq, myLinkDest);
```

- Create a requirement links data structure using `rmi('createempty')`. See `rmi`.

Link Types

Each link has a type that describes the relationship between the source and destination items. The link type refers to the `slreq.Link` object's `Type` property value.

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, Simulink Requirements sets the link type and designates the items as 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, Simulink Requirements assumes that the model element implements the requirement. It 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 Simulink Requirements sets the link type to `Relate`.

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 **Details** pane, under **Properties**, select the desired link type from the **Type** list.

Simulink Requirements 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 3-2.</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 3-6.</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 3-27.</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. Specify the source and the destination artifacts carefully because these links affect the implementation status and verification status. For more information, see “Review Requirements Implementation Status” on page 3-2 and “Review Requirements Verification Status” on page 3-6.

The link type also affects the impact direction in the Traceability Diagram window. For more information, see “Visualize Links with a Traceability Diagram” on page 2-18.

Custom Link Types

In addition to the built-in types, you can define custom link types. Custom link types must be a subtype of one of the built-in types. 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 “Define Custom Requirement and Link Types” on page 2-40.

Review Requirement Links

You can review links in the Requirements Editor or the Requirements Browser. To view links in the Requirements Editor, click **Show Links**. To view links in the Requirements Browser, select **Links** from the **View** drop-down menu.

When working in the Simulink Editor, you can review requirement links for individual requirements. In the Requirements Browser, select **Requirements** from the **View** drop-down menu and select a requirement. The links are displayed in the Property Inspector, under **Links**.

By default, all the outgoing links from a source artifact are stored in a Link Set file (.slmx). See “Requirements Link Storage” on page 5-4 for more information on requirements links storage.


When you delete a link, all related data is deleted including associated comments and custom attributes.

Resolve Links

For a link to be resolved, you must be able to navigate to the source item and to the destination item. If the source, destination, or both are not available, the link is unresolved. The source or destination items can be unavailable because:

- The design 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 because the link sources are not available.
- The design artifact loaded, but the specified ID does not exist. For example, if you delete a linked requirement, the link becomes unresolved because the stored ID no longer corresponds to a valid item.

If a link is unresolved because the specified ID does not exist, it is a broken link.

To see the unresolved links, in the Requirements Editor, click **Show Links**. Unresolved links are denoted by .

If a link is unresolved because the source or destination is not loaded, you can resolve the link by loading the file that contains the unloaded source or destination. If a link is broken, you can use the `setSource` and `setDestination` methods to repair the link.

Load Link Information

For artifacts such as requirement sets, Simulink models, data dictionaries, test files, and MATLAB files, all link information related to the artifacts that are on the MATLAB or Project path are automatically loaded when that artifact is loaded.

The link information loading can be summarized by the rules as follows:

- Rule 1: Loading an artifact such as requirement sets, Simulink models, data dictionaries, test files, and MATLAB files, that are on the MATLAB or Project path loads all incoming and outgoing link sets for that artifact. Each artifact can have one outgoing link set and one or more link sets containing link information from other artifacts.

- Rule 2: If the loaded artifact has outgoing links to a requirement set, then the requirement set is also loaded along with the link information. This loaded requirement set is also eligible to follow Rule 1 to further load link information.

The application of these rules can be illustrated using the `slreqCCProjectStart` project in three scenarios. Follow these steps:

- 1 Close all the Simulink models and requirement sets before opening the project.
- 2 Load the `slreqCCProjectStart` in the MATLAB:

```
slreqCCProjectStart
```

- 3 *Scenario 1:*

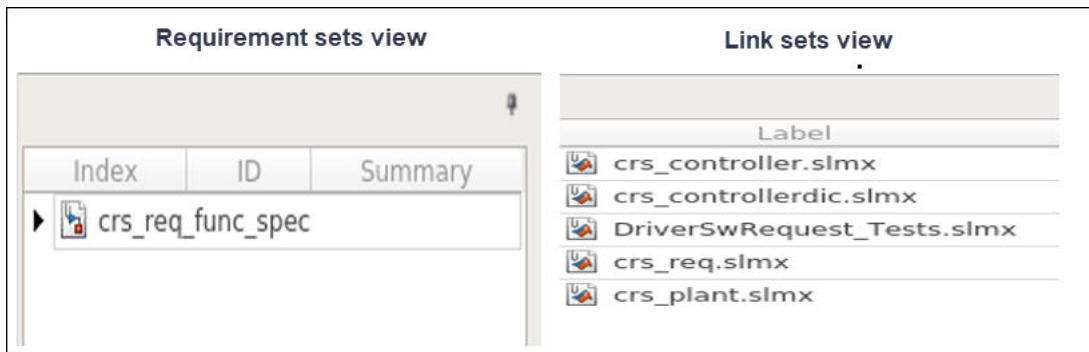
- a Open the model `crs_controller.slx`:

```
open_system('crs_controller.slx');
```

- b Open Requirements Editor:

```
slreq.editor
```

- c The Requirements Editor shows following information:



Link information from outgoing link set `crs_controller.slmx` is loaded according to Rule 1.

Requirement set `crs_req_func_spec.slmx` and link sets `crs_req.slmx`, `crs_controllerdic.slmx`, `DriverSwRequest_Tests.slmx`, and `crs_plant.slmx` are loaded according to Rule 2.

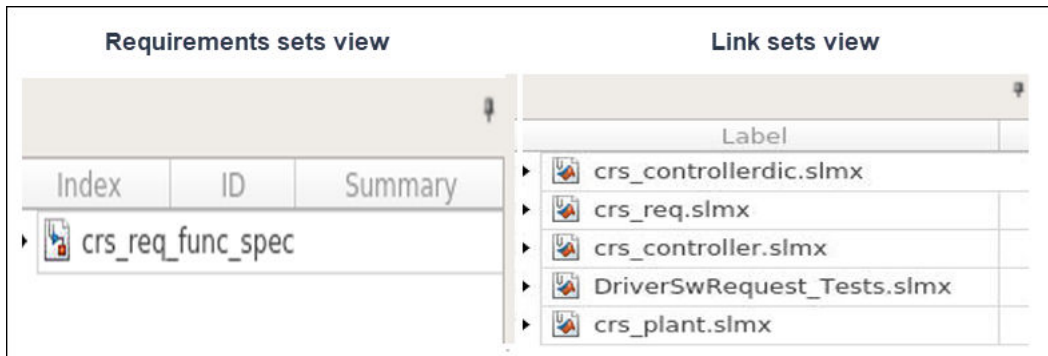
- d Close the model and the Requirements Editor.

- 4 *Scenario 2:*

- a Open the requirement set `crs_req_func_spec.slmx`:

```
slreq.open('crs_req_func_spec.slmx');
```

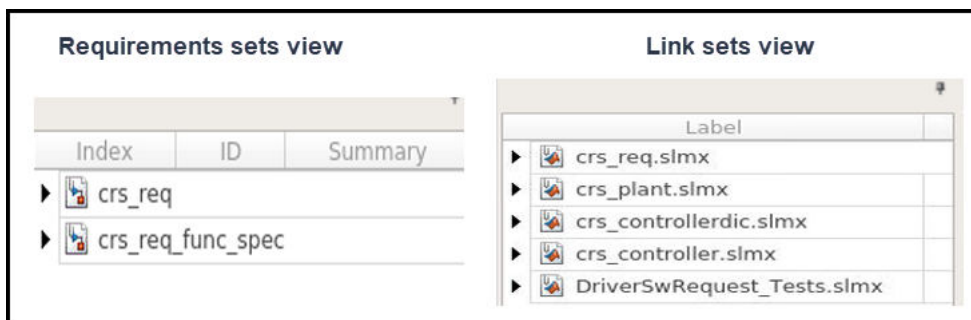
- b The Requirements Editor opens and shows following information:



Link sets View

All link sets which contain incoming link information for loaded requirement set are loaded according to Rule 1.

- c Close the Requirements Editor.
- 5 Scenario 3:
 - a Open the model `crs_plant.slx`:
`open_system('crs_plant.slx');`
 - b Open Requirements Editor:
`slreq.editor`
 - c The Requirements Editor shows following information:



Link information from outgoing link set `crs_plant.slmx` is loaded according to Rule 1.

Requirement sets `crs_req_func_spec.slreqx` and `crs_req.slreqx` and all link sets which contain incoming link information for these requirements are loaded according to Rule 2.

- d Close the model and the Requirements Editor.

Link information is not automatically loaded if you save your links with the model as an embedded link set. You can also load link information by using the `slreq.refreshLinkDependencies` command.

Unload Link Information

Link information is automatically unloaded when you unload all the related artifacts from memory.

Delete a Link Set

Link sets are stored in `.slmx` files. Deleting the `.slmx` file while the links are loaded in memory may lead to unexpected behavior.

Note If you want to delete a link set file associated with a Simulink model, ensure that the links are stored externally. To read more about how to store links externally from a Simulink model, see “Requirements Link Storage” on page 5-4.

To delete a link set:

- 1 Locate the `.slmx` file. By default, when you create a link, it is stored in a link set with the same name as the artifact that the source item belongs to. The `.slmx` file is stored in the same directory as the source artifact.
- 2 It is best to close all loaded artifacts before deleting a link set. This includes requirement sets, Simulink Test files, MATLAB code, Simulink data dictionaries, and Simulink, Stateflow or System Composer models. Manually close all of these artifacts.
- 3 At the MATLAB command line, clear loaded links by entering:

```
slreq.clear
```
- 4 Delete the `.slmx` file.

After deleting the link set file, you can re-open artifacts as needed.

See Also

`setSource` | `setDestination` | `slreq.refreshLinkDependencies`

More About

- “Track Requirement Links with a Traceability Matrix” on page 2-5
- “Customize Links with Custom Attributes” on page 2-43
- “Define Custom Requirement and Link Types” on page 2-40

Define Custom Requirement and Link Types

All requirement and link objects in Simulink Requirements 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 2-33 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.

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 to ensure that the test item contributes to the verification status. For more information, see “Link Types” on page 2-33.

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 2-33.

```
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:

```
sl_refresh_customizations
```

For more information, see “Register Customizations with Simulink”.

Inherited Functionality from the Built-In Type

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.

Custom Requirement Type Contribution to Implementation and Verification Status

Only the functional built-in requirement type contributes to the implementation and verification status. When you create a requirement type that is a subtype of **Functional**, it also contributes to

the implementation and verification status. Custom requirement types that are subtypes of the other built-in types do not contribute to those statuses. For more information, see “Review Requirements Implementation Status” on page 3-2 and “Review Requirements Verification Status” on page 3-6.

In the example code above, the **Heading** custom requirement type does not contribute to the implementation or verification statuses because it is a subtype of the built-in **Container** requirement type.

Custom Link Type Contribution to Implementation and Verification Status

To implement a functional requirement, you must link the requirement to a Model-Based Design item by using a link with the link type **Implement**. For more information, see “Review Requirements Implementation Status” on page 3-2.

Similarly, to verify a functional requirement, you must link the requirement to a supported test item with the link type **Verify** or, if it is linked to an external test result, the link type **Confirm**. For more information, see “Review Requirements Verification Status” on page 3-6 and “Include Results from External Sources in Verification Status” on page 3-27.

When you create a custom link type that is a subtype of **Implement**, **Verify**, or **Confirm**, the custom link type contributes to the implementation or verification status.

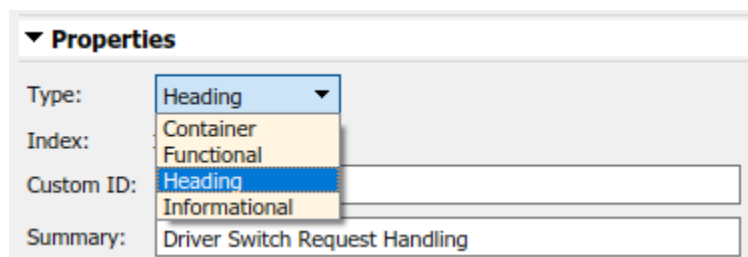
In the example code above, the **Satisfy** and **Solve** custom link types contribute to the implementation and verification statuses for **Functional** requirements because they are subtypes of the **Verify** and **Implement** link types, respectively.

Custom Link Type Impact Direction

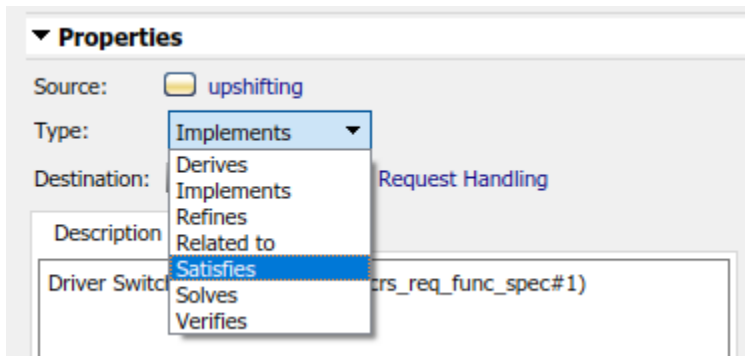
Impact direction describes how changes propagate between nodes in a traceability diagram. For more information, see the “Impact Direction” on page 2-21 section of “Visualize Links with a Traceability Diagram” on page 2-18. Custom link types inherit the impact direction from the built-in type that they are a subtype of. However, because custom link types do not inherit the link direction, use the same link direction as the built-in types for consistency in the traceability diagram. For more information, see the table under “Link Types” on page 2-33.

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 **Details** 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 **Details** pane, under **Properties**, select the custom link type from the **Type** drop-down list.



See Also

More About

- “Requirement Types” on page 1-6
- “Link Types” on page 2-33
- “Map SpecObjectTypes to Requirement Types” on page 1-23

Customize Links with Custom Attributes

In this section...
“Define Custom Attributes for Link Sets” on page 2-43
“Set Custom Attribute Values for Links” on page 2-44
“Edit Custom Attributes” on page 2-45

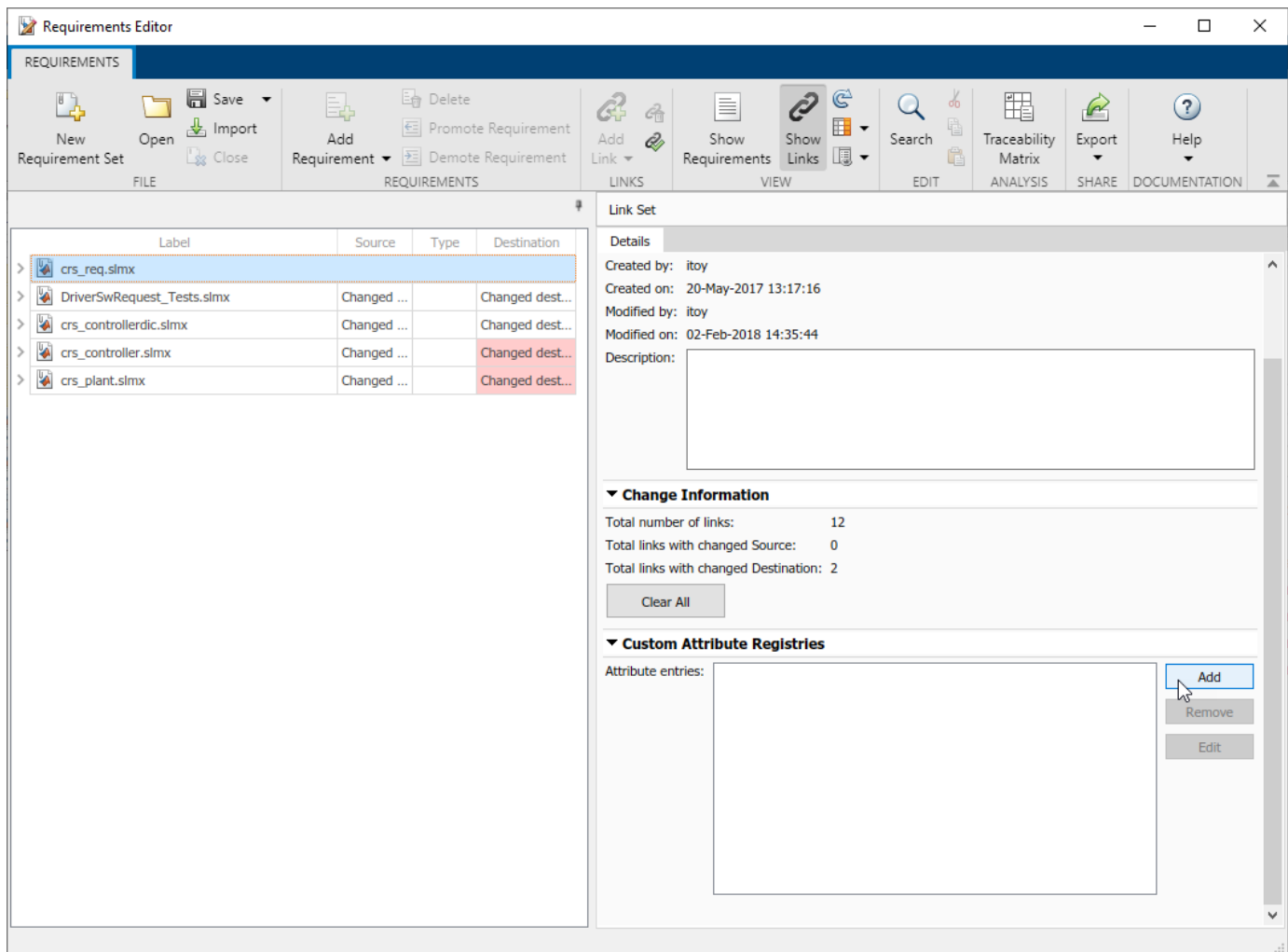
When you create a link set using Simulink Requirements, you can create custom attributes that apply to the links contained in the link set. Custom attributes extend the set of properties associated with your links.

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 “Requirement Links” on page 2-32.
- 3 Click **Show Links**.
- 4 Select the link set.
- 5 In the **Details** 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.



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 **Details** 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.

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 **Details** 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 **Details** pane, under **Custom Attribute Registries**, select the custom attribute you want to edit and click **Edit**.

For custom attributes of any type, you can edit the name and description. For **Combobox** custom attributes, you can also edit the drop-down menu options. You can edit the value of each option in the drop-down menu (excluding **Unset**), or add and remove options. If you edit the value of an option or remove an option, then any links that had been set to that option will be reset to the default value, **Unset**.

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 **Details** pane, under **Custom Attributes**.

See Also

Related Examples

- “Manage Custom Attributes for Links by Using the Simulink® Requirements™ API” on page 2-65

More About

- “Customize Requirements with Custom Attributes” on page 1-48

Requirements Consistency Checks

Check Requirements Consistency in Model Advisor

- “Identify requirement links with missing documents” on page 2-46
- “Identify requirement links that specify invalid locations within documents” on page 2-46
- “Identify selection-based links having description fields that do not match their requirements document text” on page 2-47
- “Identify requirement links with path type inconsistent with preferences” on page 2-48
- “Identify IBM Rational DOORS objects linked from Simulink that do not link to Simulink” on page 2-49

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 Simulink Requirements.

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 Simulink Requirements.

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 Simulink Requirements.

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 Simulink Requirements.

Results and Recommended Actions

Condition	Recommended Action
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>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 Simulink Requirements.

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 1-10.

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, Simulink Requirements 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 manually insert or remove backlinks for your requirements:

- 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**.

Note The backlinks update process does not remove backlinks in IBM DOORS Next. Remove unwanted backlinks in DOORS Next in the DOORS Next interface.

See Also

More About

- “Navigate to Requirements in Microsoft Office Documents from Simulink” on page 6-10

- “Link to Requirements in Microsoft Word Documents” on page 6-2
- “Link to Requirements in Excel Workbooks” on page 6-7
- “Link and Trace Requirements with IBM DOORS Next” on page 7-26

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 Simulink Requirements® 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

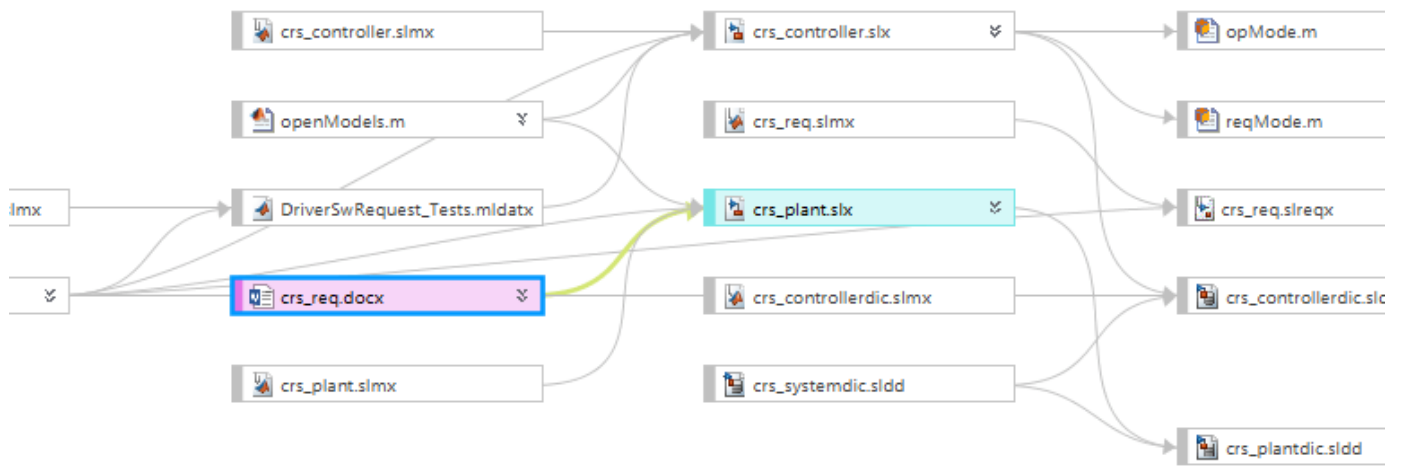
Before you begin, ensure a clean initial state by running `slreq.clear` command. Then type `slreqCCProjectStart` to open the Cruise Control Project example. This will unzip a collection of linked artifact files into a new subfolder under your MATLAB/Projects folder.

```
slreq.clear();  
slreqCCProjectStart();
```

Simulink Model Linked to Requirements

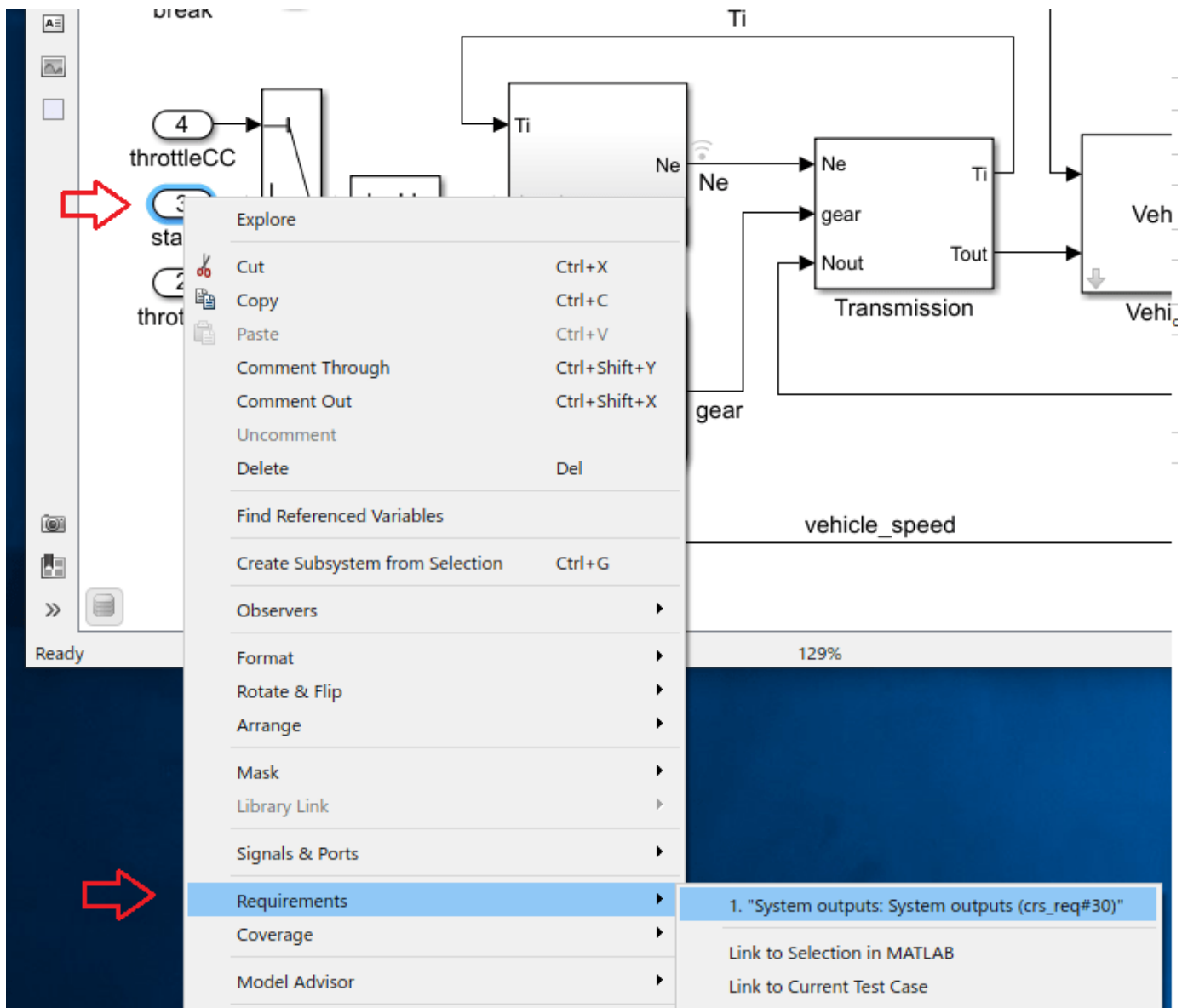
We will focus on a small part of this Project's Dependency Graph: open the `crs_plant.slx` Simulink model, that has several links to an external Microsoft® Word document `crs_reqs.docx`.

```
open_system('crs_plant');
```

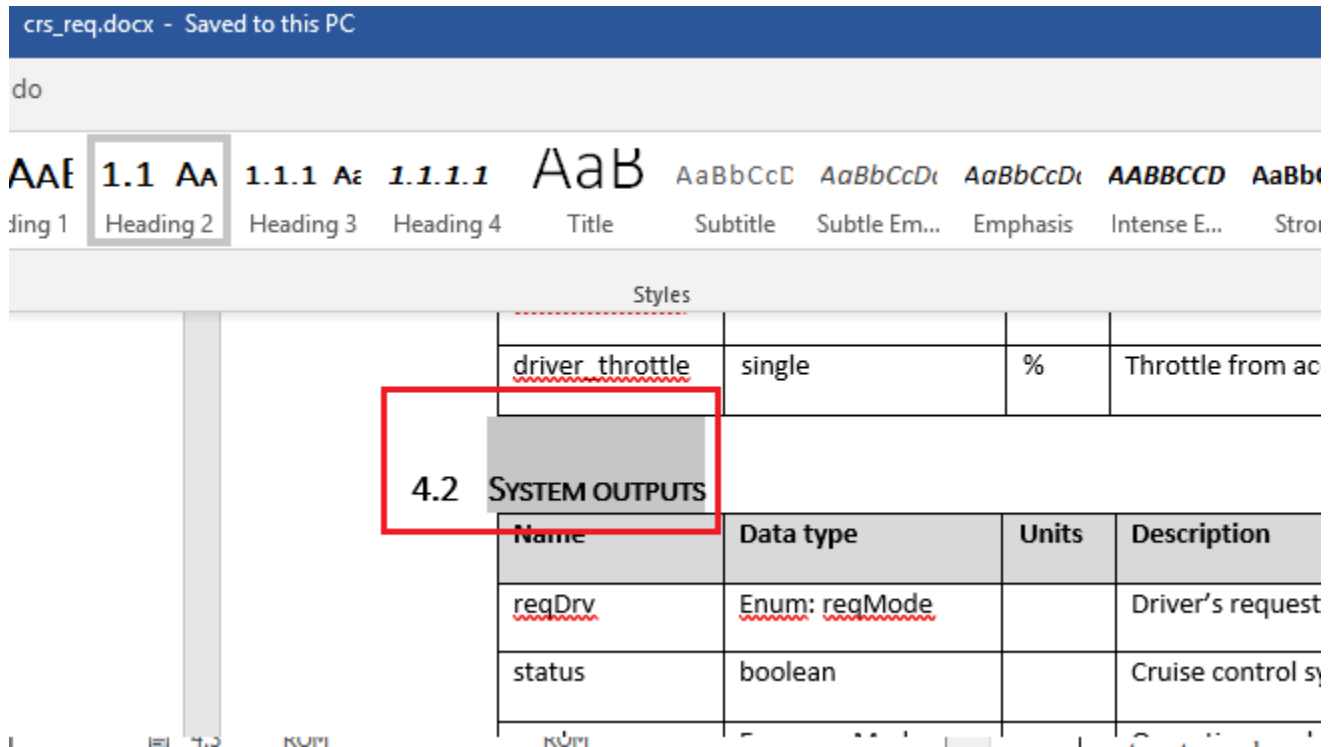


Navigate one of the links to open the linked document.

```
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:');

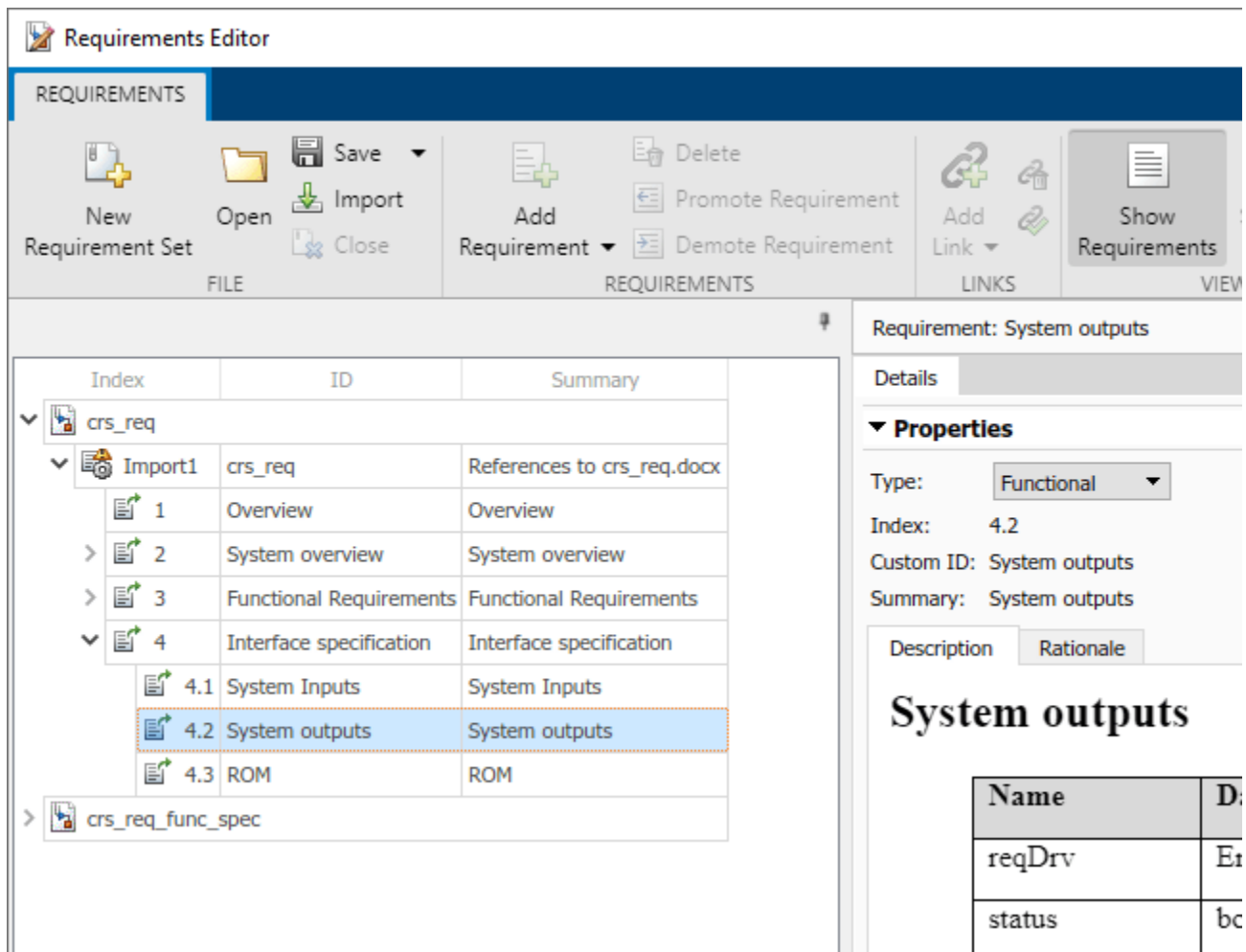
Original Links to Word document:

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 Simulink 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 Simulink 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 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

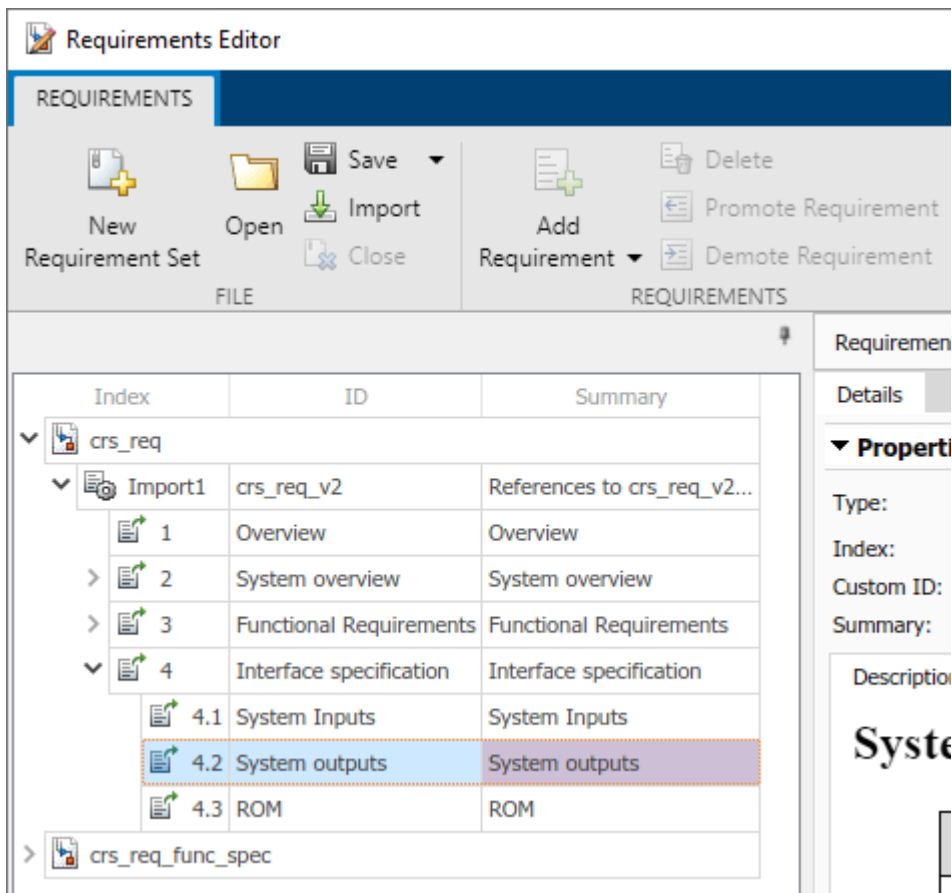
As demonstrated above, when imported references are available in Requirements Editor, navigating a link will select the matching reference object. However, we have just updated links for a new version of the document `crs_req_v2.docx`, and there are no imported references for this document. Navigation from Simulink block in the presence of Requirements Editor brings you directly to the external Word document.

The screenshot shows a Microsoft Word document titled "crs_req_v2.docx - Saved to this PC". The document content includes a table of contents with "4.2 SYSTEM OUTPUTS" highlighted by a red box. Below the table of contents is a table with columns: Name, Data type, Units, and Description. The table contains entries for "reqDrv" (Enum: reqMode) and "status" (boolean).

Name	Data type	Units	Description
<u>reqDrv</u>	<u>Enum: reqMode</u>		Driver's request
status	boolean		Cruise control sy

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 Simulink Requirements 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 "Requirements Definition for a Cruise Control Model" 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
```

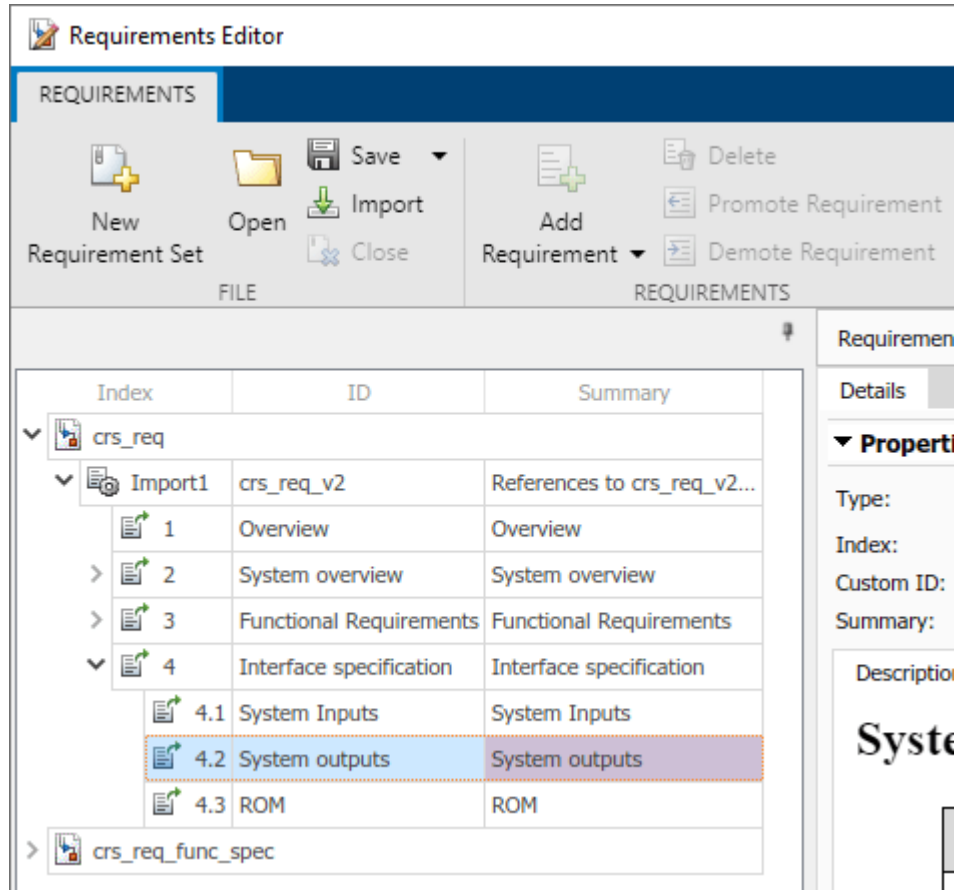
Navigate from the Simulink model to the updated reference.

```
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

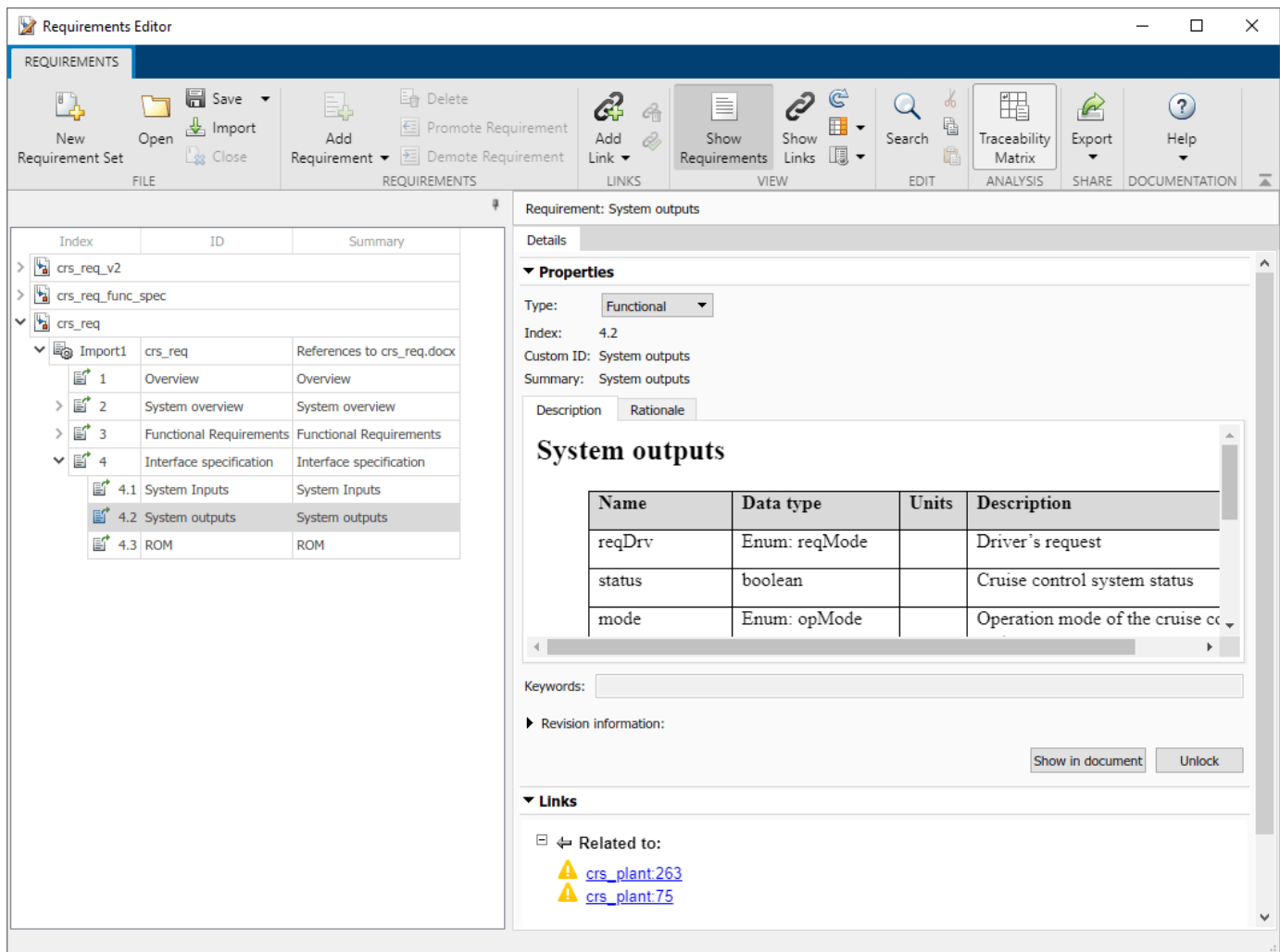
Discard link data changes to avoid prompts on Project close. Close the project (also cleans-up MATLAB path changes). Close Microsoft Word.

```
slreq.clear();  
prj = simulinkproject(); prj.close();  
rmidonet.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 Cruise Control Project from "Requirements Definition for a Cruise Control Model" 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 Cruise Control 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.slmreqx` as the link target, instead of the original `crs_req.slmreqx`. 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.

The screenshot shows the Requirements Editor window. The left pane displays a tree view of requirement sets. The right pane shows the details for the selected requirement, 'System outputs'.

Requirement: System outputs

Properties

Type: Functional
 Index: 4.2
 Custom ID: System outputs
 Summary: System outputs

Description

System outputs

Name	Data type	Units	Description
reqDrv	Enum: reqMode		Driver's request
status	boolean		Cruise control system status
mode	Enum: opMode		Operation mode of the cruise control

Keywords:

Revision information:

Show in document Unlock

Links

Related to:

- status
- throttleCC

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);
```

Cleanup After Use Case 3

Clear the open requirement sets and link sets, and close the open models and projects without saving changes. Close Microsoft Word.

```
slreq.clear();  
bdclose('all');  
prj = simulinkproject(); prj.close();  
rmidotnet.MSWord.application('kill');
```

Manage Custom Attributes for Links by Using the Simulink® Requirements™ API

This example shows how to use the Simulink® Requirements™ 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: ''
      Filename: 'C:\TEMP\Bdoc21b_1757077_3096\ib2EDA31\22\tpd1251b01\slrequirements-ex
      Artifact: 'C:\TEMP\Bdoc21b_1757077_3096\ib2EDA31\22\tpd1251b01\slrequirements-ex
      Domain: 'linktype_rmi_slreq'
      Revision: 8
      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
  Columns 1 through 3
    {'MyCheckboxAttr...'}    {'MyComboboxAttr...'}    {'MyDateTimeAttr...'}
  Column 4
    {'MyEditAttribute'}
```

```
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: 01-Sep-2021 19:06:44
      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
```

```
{'MyCheckboxAttri...'} {'MyComboboxAttri...'} {'MyDateTimeAttri...'}  
{'MyEditAttribute'} {'MyDateAttribute'} {'MyDateAttribute'}
```

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

```
addAttribute(ls,'NewEditAttribute','Edit');
```

```
ls.CustomAttributeNames
```

```
ans = 1x4 cell
```

```
Columns 1 through 3  
{'MyCheckboxAttri...'} {'MyComboboxAttri...'} {'MyDateAttribute'}
```

```

    {'MyCheckboxAttr...'}    {'MyComboboxAttr...'}    {'MyDateTimeAttr...'}
Column 4
    {'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
    {'MyCheckboxAttri...'}    {'MyComboboxAttri...'}    {'MyDateTimeAttri...'}

```

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 Simulink® Requirements™ API” on page 1-79

More About

- “Customize Links with Custom Attributes” on page 2-43

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 Simulink® Requirements™ 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 2-5.

Open the Requirements Definition for a Cruise Control Model project. Load the `crs_req_func_spec` requirement set.

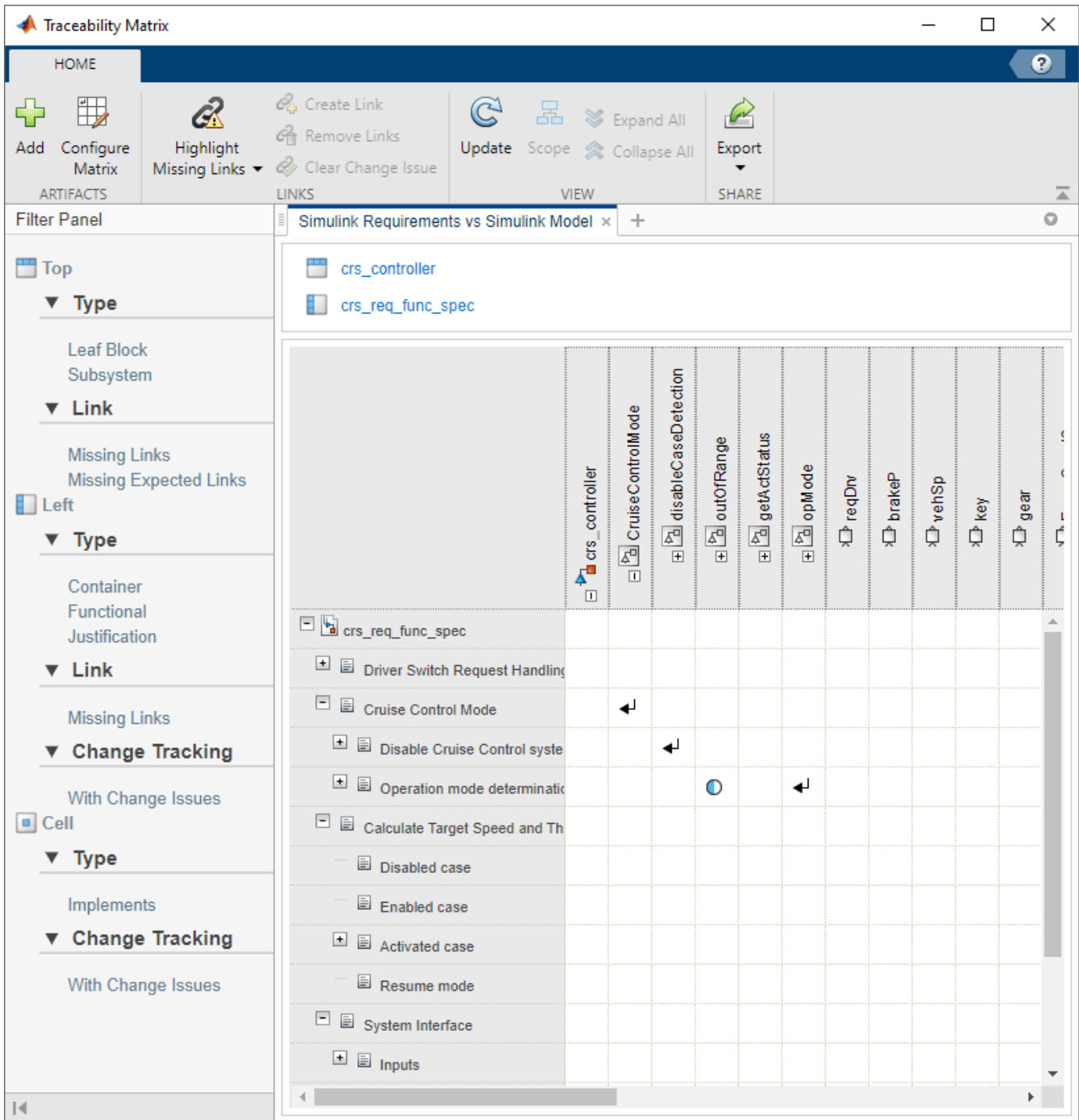
```
slreqCCProjectStart;  
slreq.load('crs_req_func_spec');
```

Generate a Traceability Matrix

Open the Traceability Matrix window.

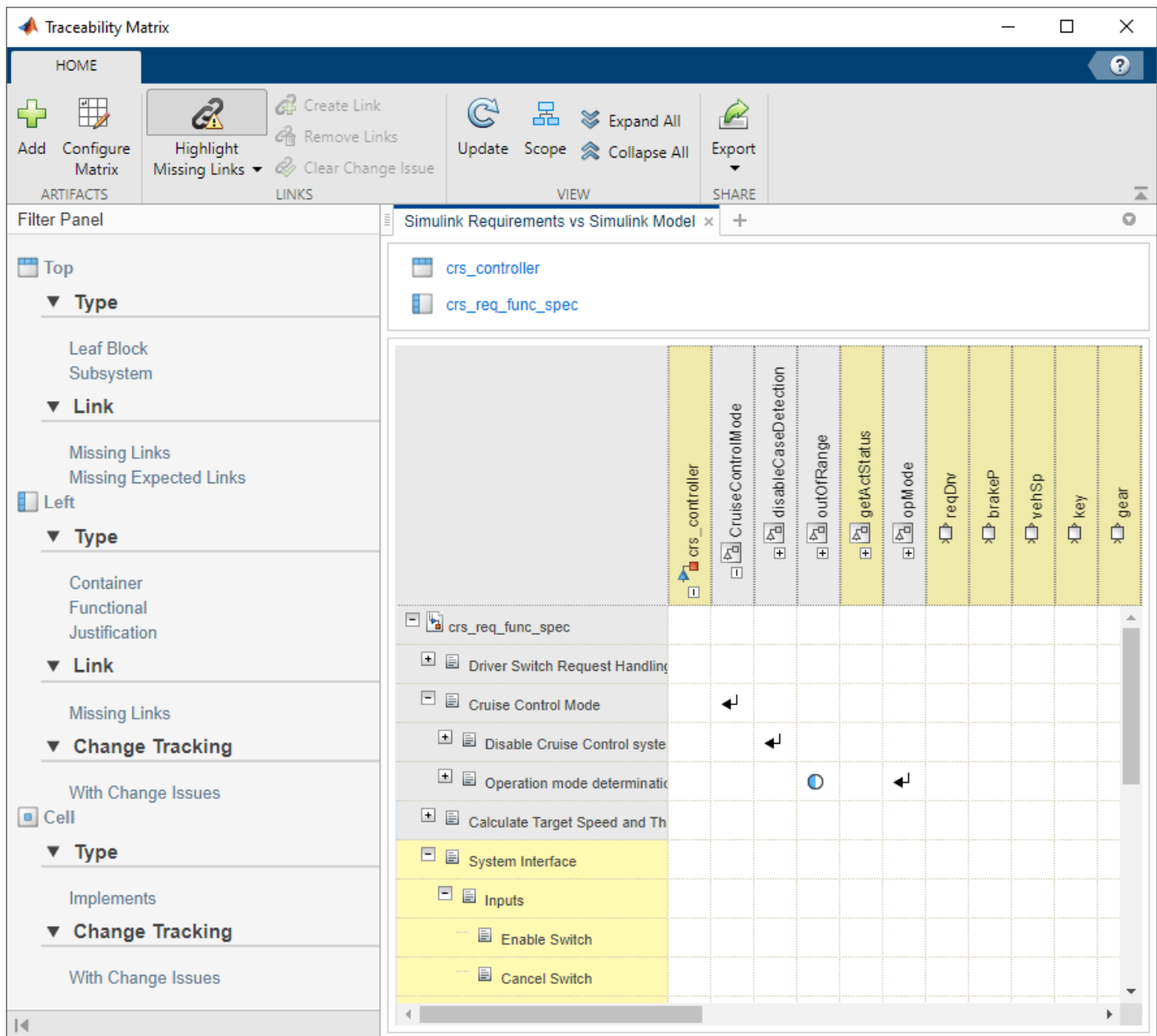
```
slreq.generateTraceabilityMatrix;
```

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.

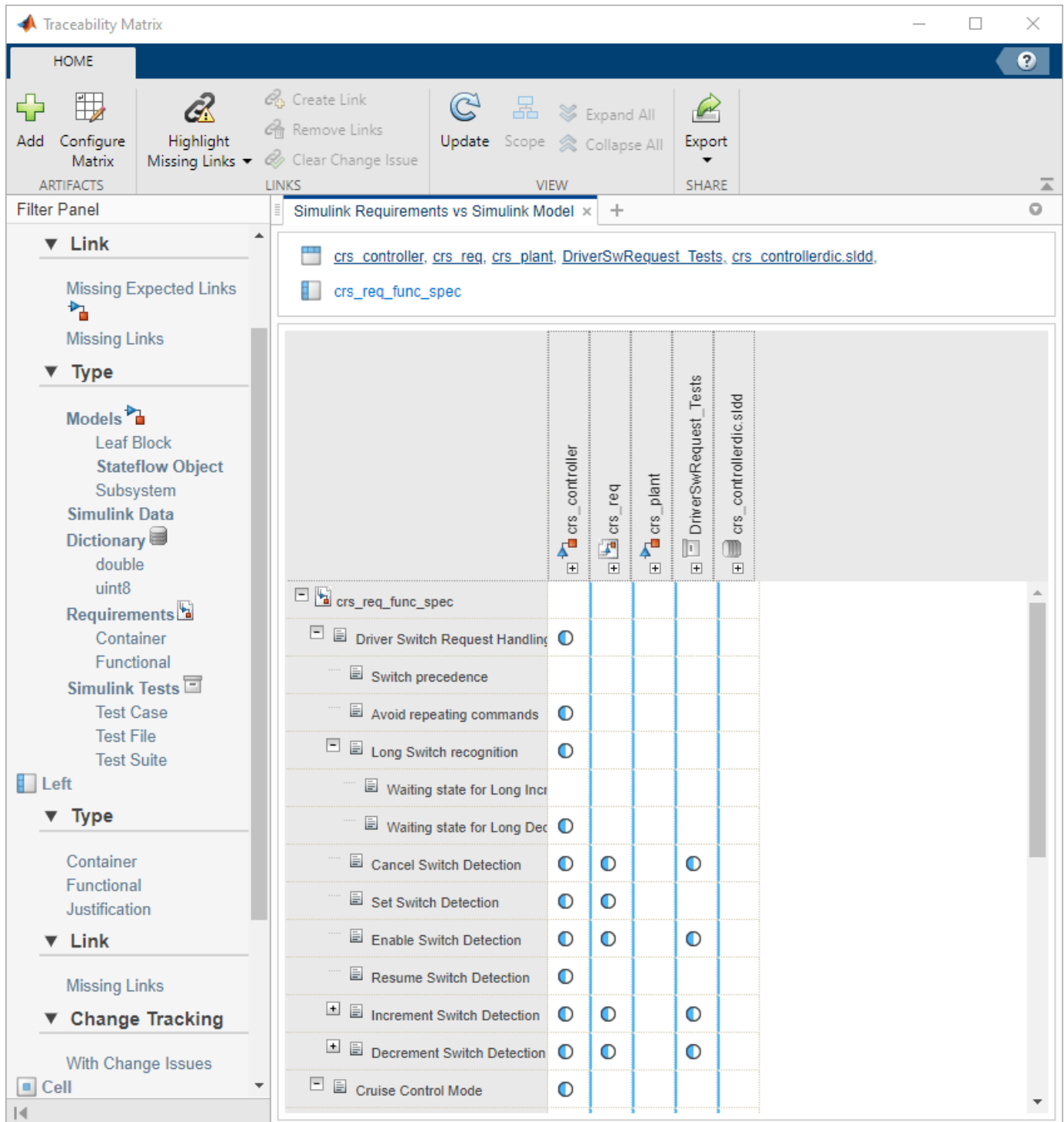
Generate a Traceability Matrix with Multiple Artifacts

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.



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.

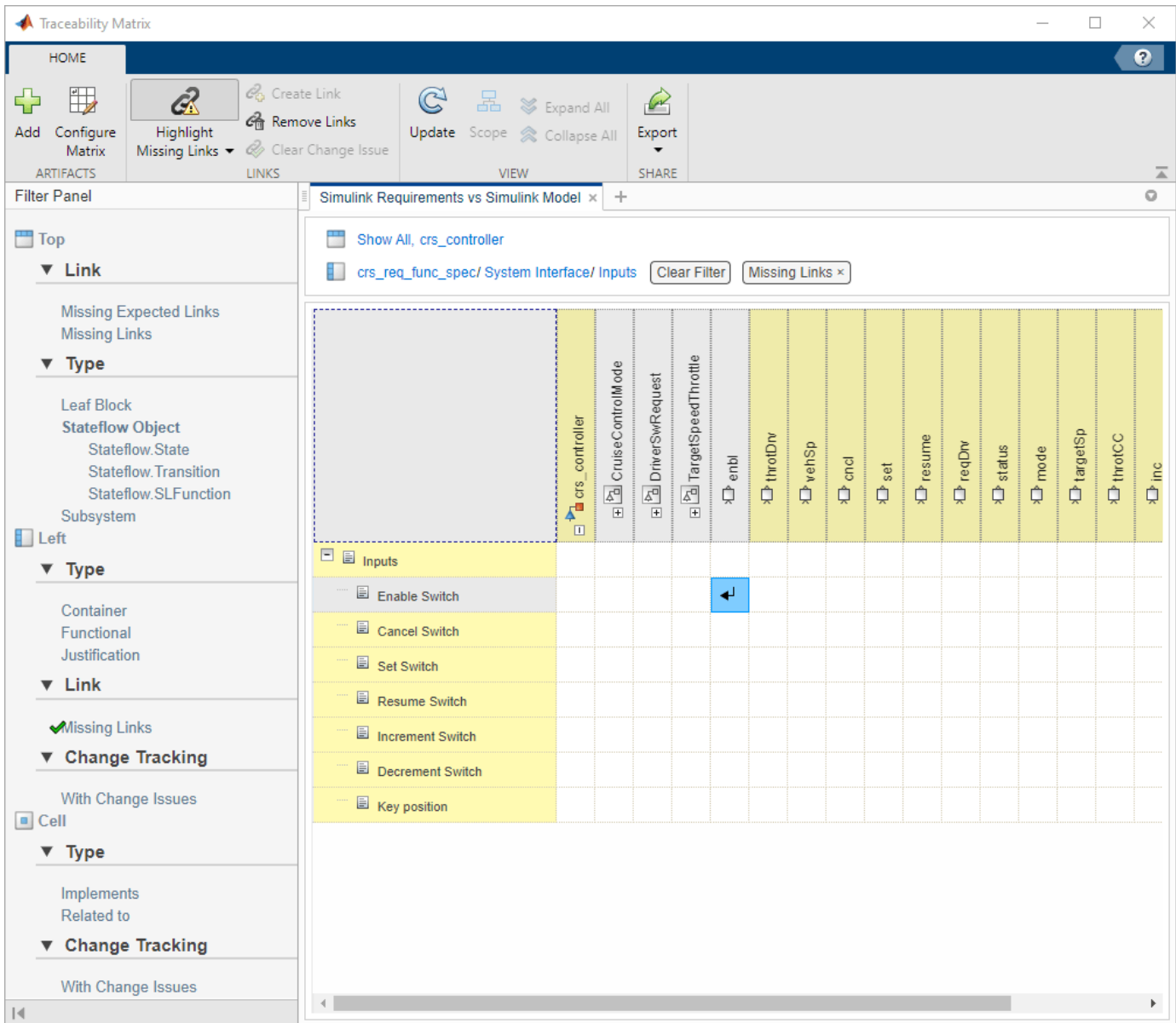
Some of the child requirements link to items in `crs_plant`. Link the remaining unlinked `Inputs` child requirements to model elements in `crs_controller`. Select the the cell corresponding to `crs_controller` and click **Scope**.

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**.

The screenshot shows the Traceability Matrix application window. The top toolbar includes buttons for 'Add', 'Configure Matrix', 'Highlight Missing Links', 'Create Link', 'Remove Links', 'Clear Change Issue', 'Update', 'Scope', 'Expand All', 'Collapse All', and 'Export'. The left sidebar contains a 'Filter Panel' with sections for 'Top', 'Link', 'Type', and 'Cell'. The main area displays a traceability matrix with columns for subsystems and rows for requirements. The 'Inputs' row is highlighted in yellow, and the 'enbl' column is also highlighted. A dashed blue box highlights the cell at the intersection of the 'enbl' column and the 'Inputs' row.

Inputs	crs_controller	CruiseControlMode	DriverSwRequest	TargetSpeedThrottle	enbl	throtDrv	vehSp	cncl	set	resume	reqDrv	status	mode	targetSp	throtCC	inc
Enable Switch																
Cancel Switch																
Set Switch																
Resume Switch																
Increase Switch																
Decrease Switch																
Key position																

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.



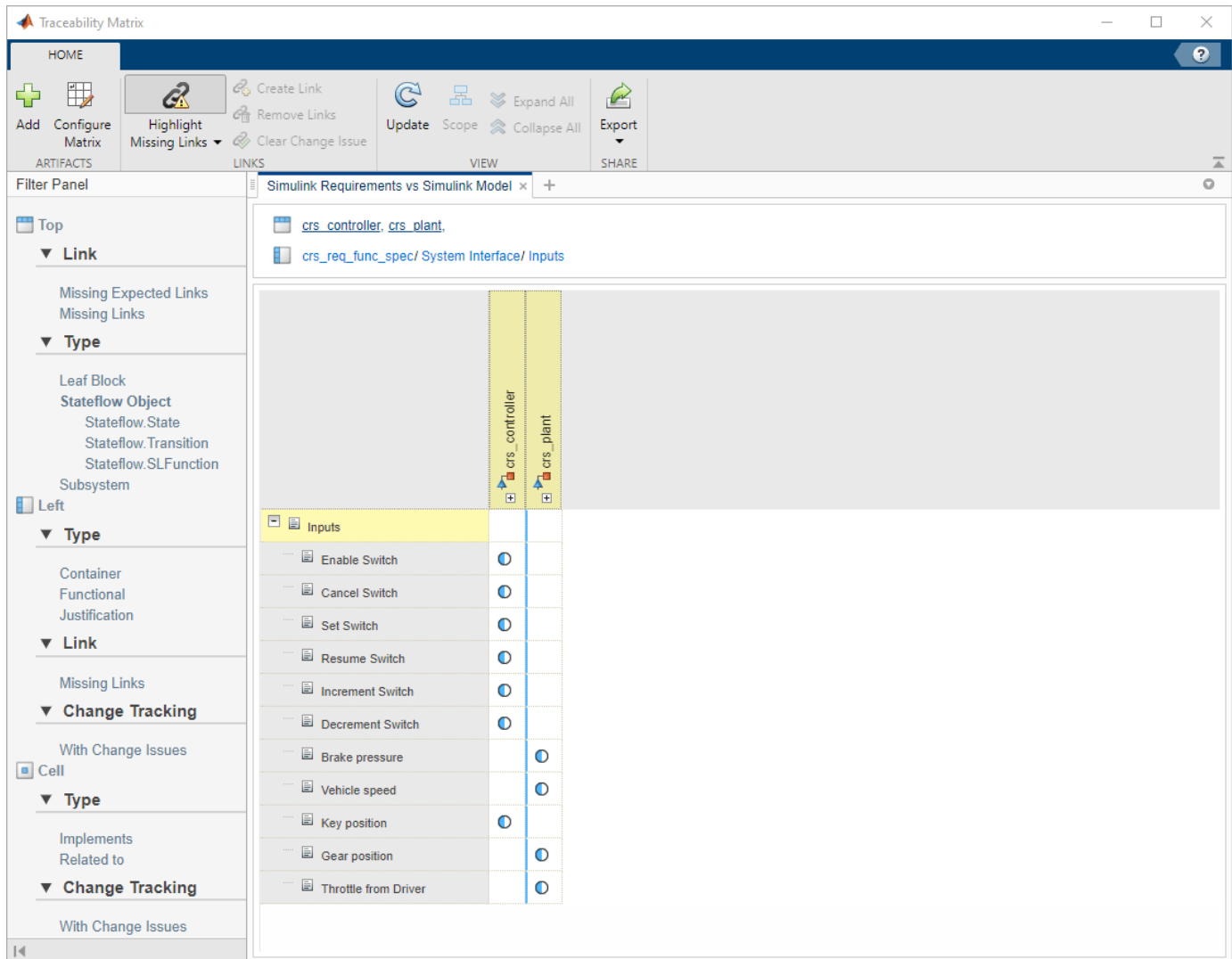
You can create multiple links at a time when you hold **Ctrl**, select the cells where you want to create links, and click **Create Links**. Create links between the remaining requirements and the corresponding model element:

- 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.

The screenshot displays the Traceability Matrix tool interface. The top toolbar contains several icons for actions: Add, Configure Matrix, Highlight Missing Links, Create Link, Remove Links, Clear Change Issue, Update, Scope, Expand All, Collapse All, and Export. The Filter Panel on the left is set to 'Top' and 'Link' type, with 'Missing Expected Links' and 'Missing Links' selected. The main area shows a traceability matrix for 'Simulink Requirements vs Simulink Model'. The top row lists design items: crs_controller, CruiseControlMode, DriverSwRequest, TargetSpeedThrottle, enbl, throlDrv, vehSp, cnd, set, resume, reqDrv, status, mode, targetSp, throlCC, inc, dec, brakeP, key, and gear. The 'Inputs' artifact is expanded, showing child requirements: Enable Switch, Cancel Switch, Set Switch, Resume Switch, Increment Switch, Decrement Switch, and Key position. Blue arrows indicate links from these child requirements to the corresponding design items in the top row.

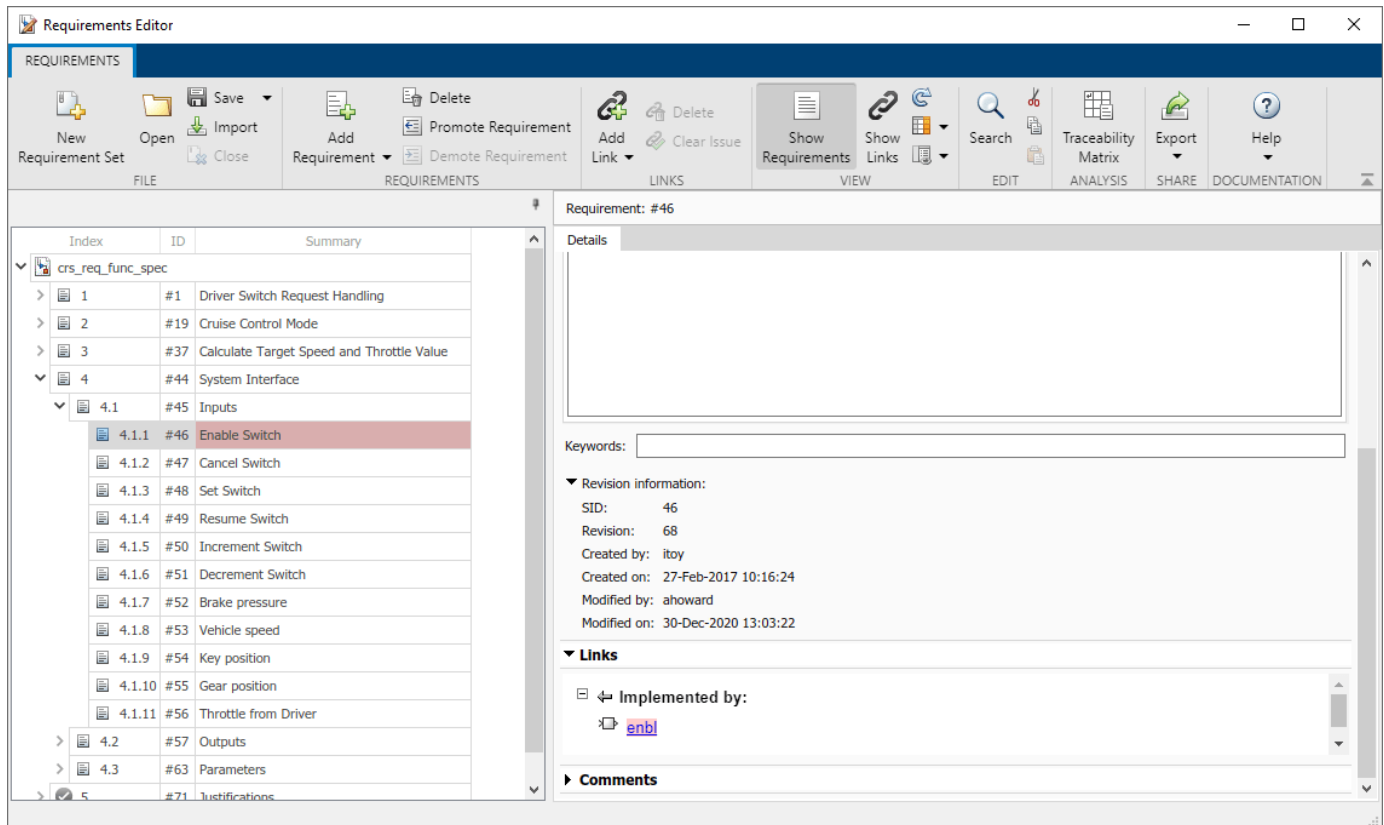
Clear the **Missing Links** filter by clicking **Clear Filter** in the top artifact list. Click **Show All** to show all of the artifacts. All of the Inputs child requirements link to design items, so they are no longer highlighted. Collapse the hierarchies under `crs_controller` and `crs_plant`. The expand icon (🔍) indicates that all of the Inputs child requirements are linked.



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 **Details** 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.

Traceability Matrix

HOME

+ Add Configure Matrix Highlight Missing Links Create Link Remove Links Clear Change Issue Update Scope Expand All Collapse All Export

ARTIFACTS LINKS VIEW SHARE

Filter Panel Simulink Requirements vs Simulink Model +

Top
Link
 Missing Expected Links
 Missing Links
Type
 Leaf Block
 Stateflow Object
 Stateflow.State
 Stateflow.Transition
 Stateflow.SLFunction
 Subsystem
Left
Type
 Container
 Functional
 Justification
Link
 Missing Links
Change Tracking
 With Change Issues
Cell
Type
 Implements
 Related to
Change Tracking
 With Change Issues

[crs_controller, crs_plant,](#)
[crs_req_func_spec/ System Interface/ Inputs](#)

Inputs
 Enable Switch

crs_controller
 DriverSwRequest
 enbl

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 **Details** pane, under **Comments**.

The screenshot displays the Requirements Editor interface. The main table lists various requirements, with 'link #72' highlighted. The right-hand pane shows the details for this link, including its source, type, destination, and a comment.

Label	Source	Type	Desti
#36: Disabling override	Switch	Implem...	Disablr
#37: Calculate Target Speed and Thrott...	TargetSp...	Implem...	Calcula
#38: Disabled case	disabled	Implem...	Disable
#39: Enabled case	enabled	Implem...	Enabler
#40: Activated case	activated	Implem...	Activat
#41: Throttle Value Computation	getThrott...	Implem...	Throttle
#42: Next Target Speed Computation	getNewT...	Implem...	Next T
#43: Resume mode	targetSpe...	Implem...	Resum
#43: Resume mode	From9	Implem...	Resum
#39: Enabled case	Goto1	Implem...	Enabler
#39: Enabled case	From1	Implem...	Enabler
#43: Resume mode	Goto3	Implem...	Resum
#40: Activated case	Goto2	Implem...	Activat
#40: Activated case	From2	Implem...	Activat
#66: Threshold of brake pressure	Compare ...	Implem...	Threshi
#68: Minimum Throttle Value	Saturation2	Implem...	Minimu
#67: Maximum Throttle Value	Saturation2	Implem...	Maximu
#69: Maximum Target Speed	Saturation	Implem...	Maximu
#70: Minimum Target Speed	Saturation	Implem...	Minimu
#64: Proportional of PI	Kp	Implem...	Proport
#65: Integral of PI	DTI	Implem...	Integra
link #72	enbl	Implem...	Enable
link #73	cncl	Implem...	Cancel
link #74	set	Implem...	Set Swi
link #75	resume	Implem...	Resum
link #76	inc	Implem...	Increm
link #77	dec	Implem...	Decrer
link #78	key	Implem...	Key pos

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**.

Cleanup

Clear the open requirement sets and link sets and close the Traceability Matrix window. Close all open models. Close the current project.

```
slreq.clear;  
bdclose all;  
slproject.closeCurrentProject();
```

See Also

`slreq.generateTraceabilityMatrix` | `slreq.getTraceabilityMatrixOptions`

More About

- “Track Requirement Links with a Traceability Matrix” on page 2-5

Modeling System Architecture of Small UAV

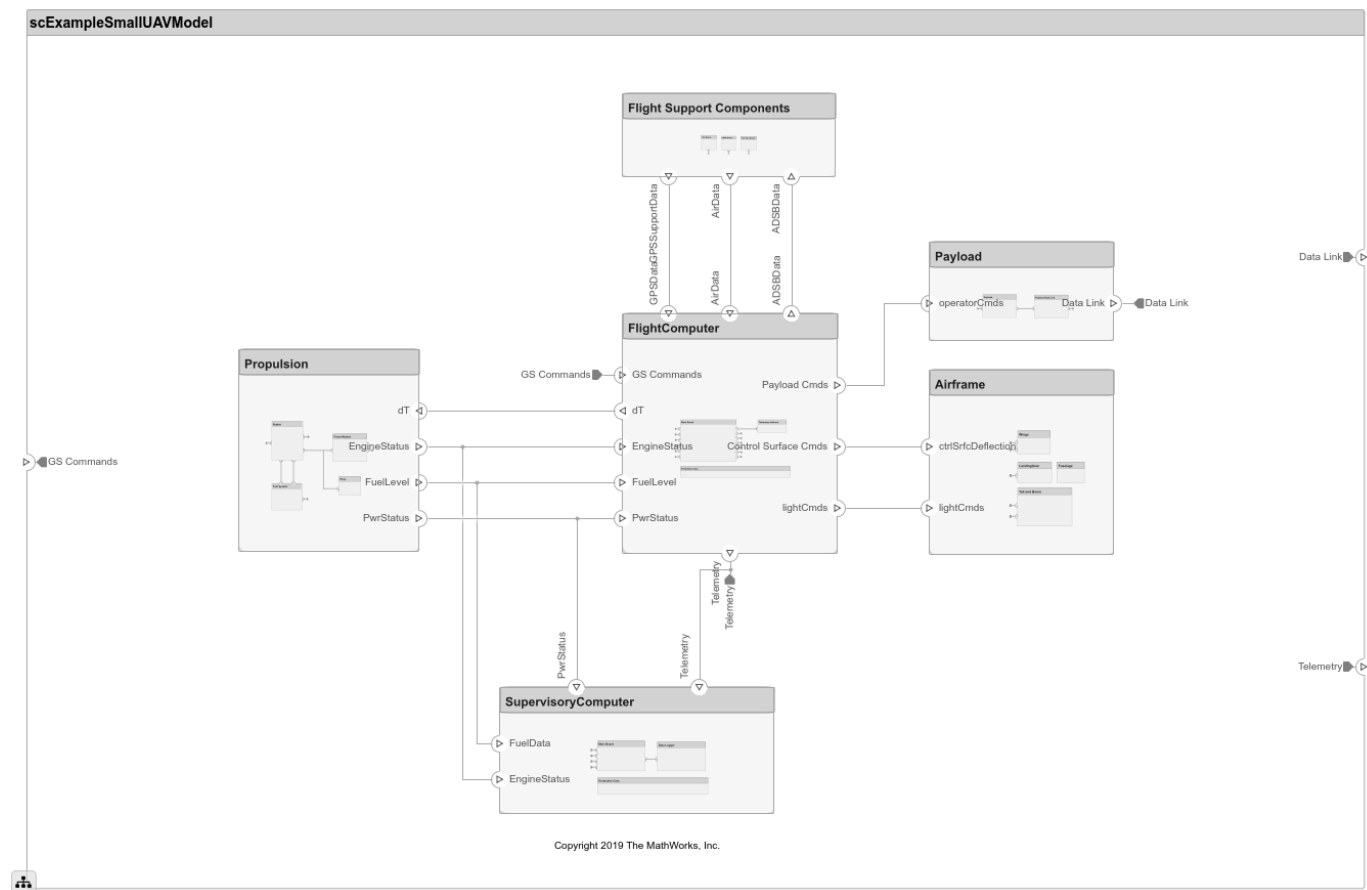
Overview

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

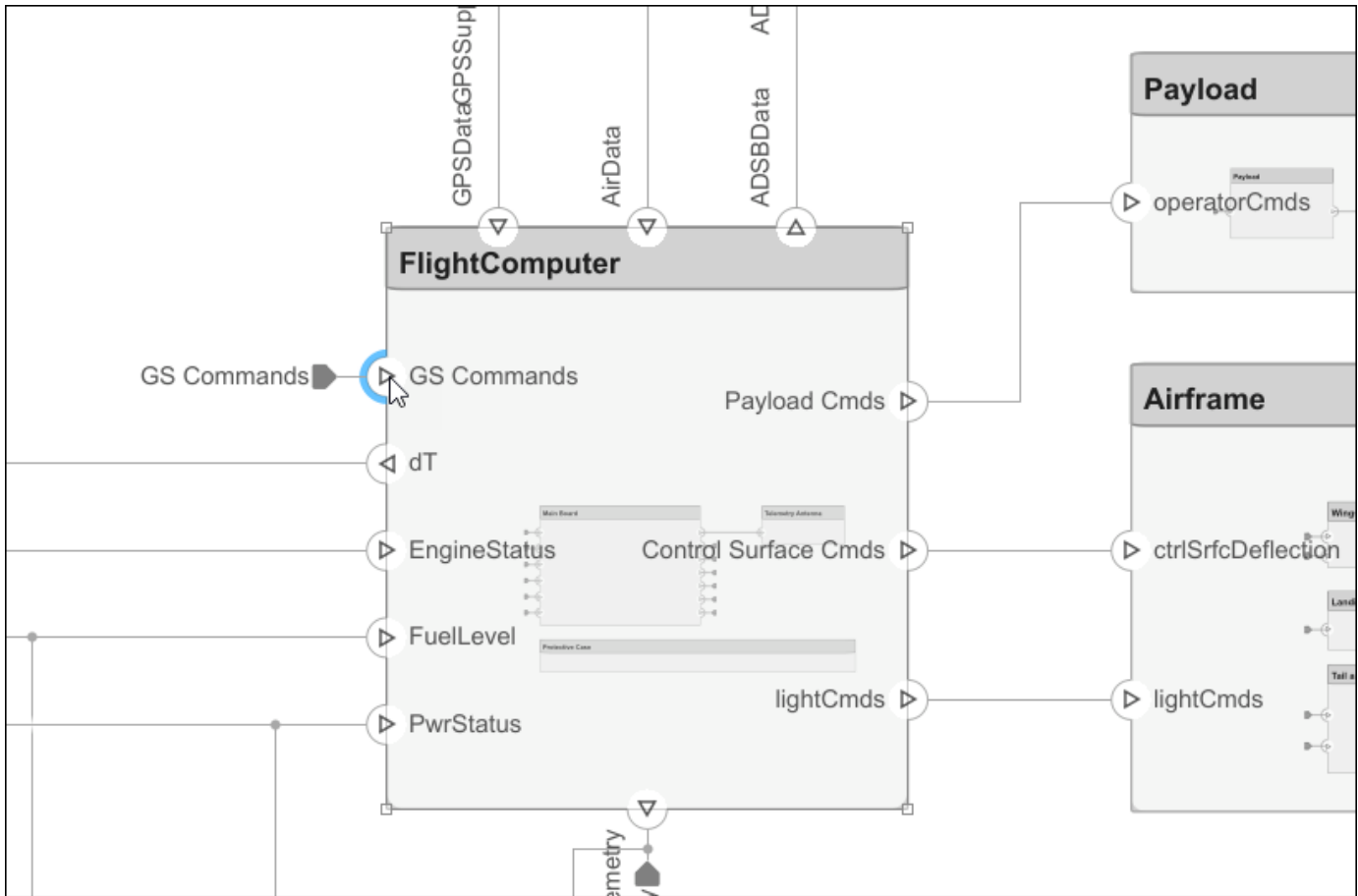


Each top-level component is decomposed into its subcomponents. Navigate through the hierarchy to view the composition for each component. The root component, `scExampleSmallUAVModel`, has input and output ports that represent data exchange between the system and its environment.

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. In the **Modeling** tab in the toolstrip, select **Design**, then 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.

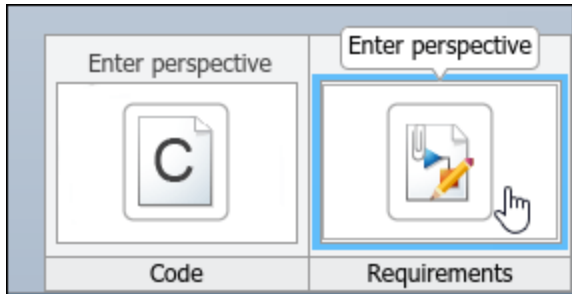


Interfaces							
	Type	Dimensions	Units	Complexity	Minimum	Maximum	Description
scExampleSmallUAVModel.sbx							
architecture_gsCommands							
apConfigParams (param_value_bus)	param_value_bus	1		real	[]	[]	
param_count	uint16	1		real	[]	[]	Total number of onboard parameters
param_id	int8	16		real	[]	[]	Onboard parameter id, terminated by NULL if the length is less than 16 human-r
param_index	uint16	1		real	[]	[]	Index of this onboard parameter
param_type	uint8	1		real	[]	[]	Onboard parameter type: see the MAV_PARAM_TYPE enum for supported data
param_value	single	1		real	[]	[]	Onboard parameter value
gsCommands (gs_commands_bus)	gs_commands_bus	1		real	[]	[]	
RTB	uint8	1		real	[]	[]	Return to Base Command
U_c	single	1		real	[]	[]	Airspeed Commanded by the GS
guidanceMode	uint8	1		real	[]	[]	
h_c_midLevel	single	1		real	[]	[]	Commanded altitude when in Mid Level Commands Mode. Also used as the alti
isManualModeOn	uint8	1		real	[]	[]	
psiDot_c_midLevel	single	1		real	[]	[]	Turnrate command when in mid Level Commands guidance mode

Inspect Requirements

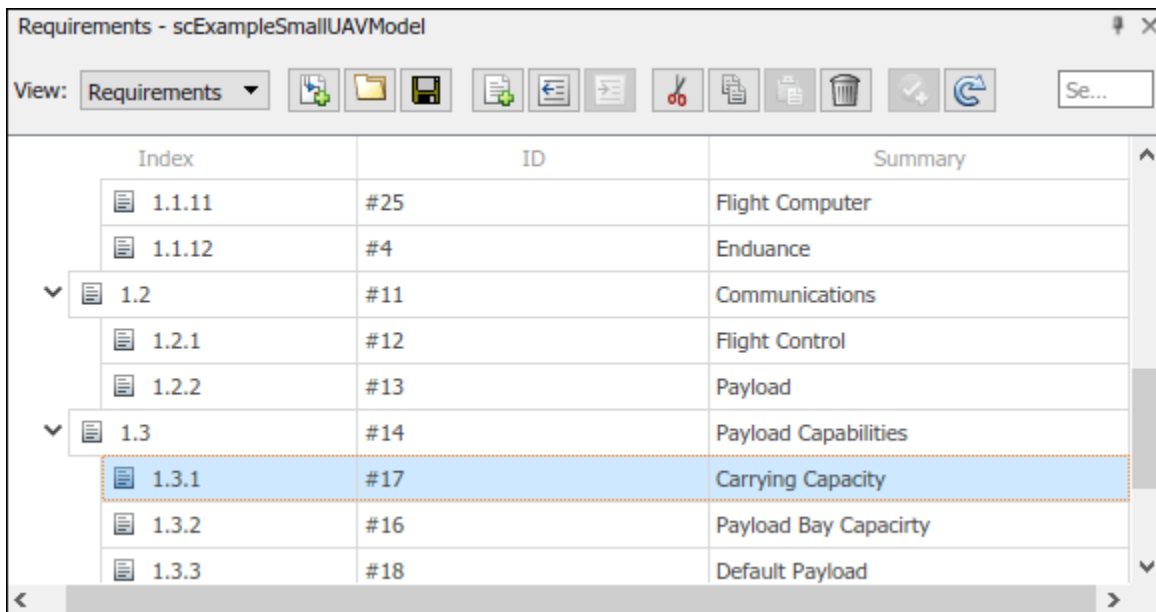
A Simulink Requirements license is required to inspect requirements in a System Composer architecture model.

Components in the architecture model link to system requirements defined in `smallUAVReqs.s1reqx`. Open the **Requirements Manager**. In the bottom right corner of the model pane, click **Show Perspectives**. 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.



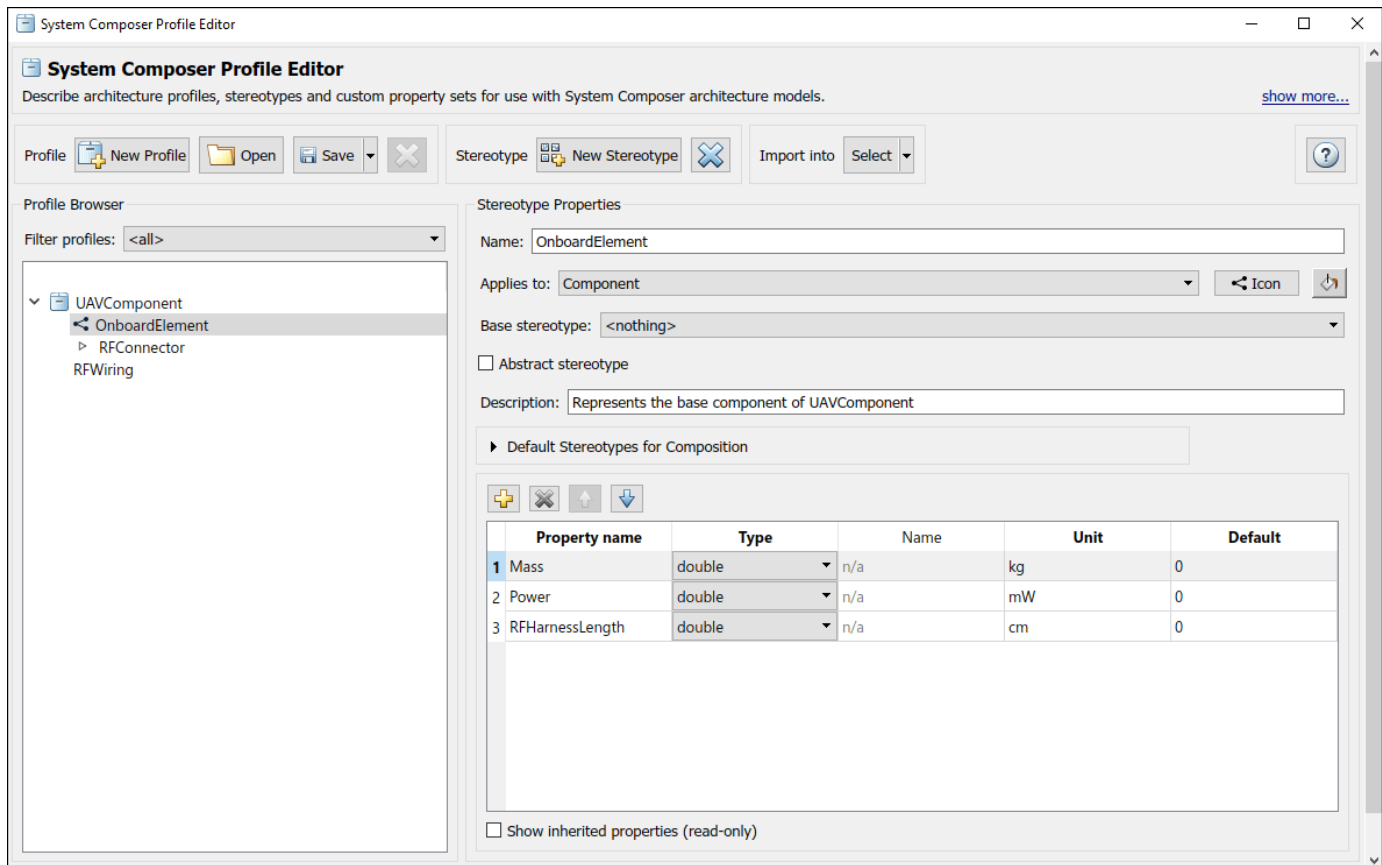
Define Profiles and Stereotypes

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**. On the **Modeling** tab, click **Profile Editor**.

The display appears below.

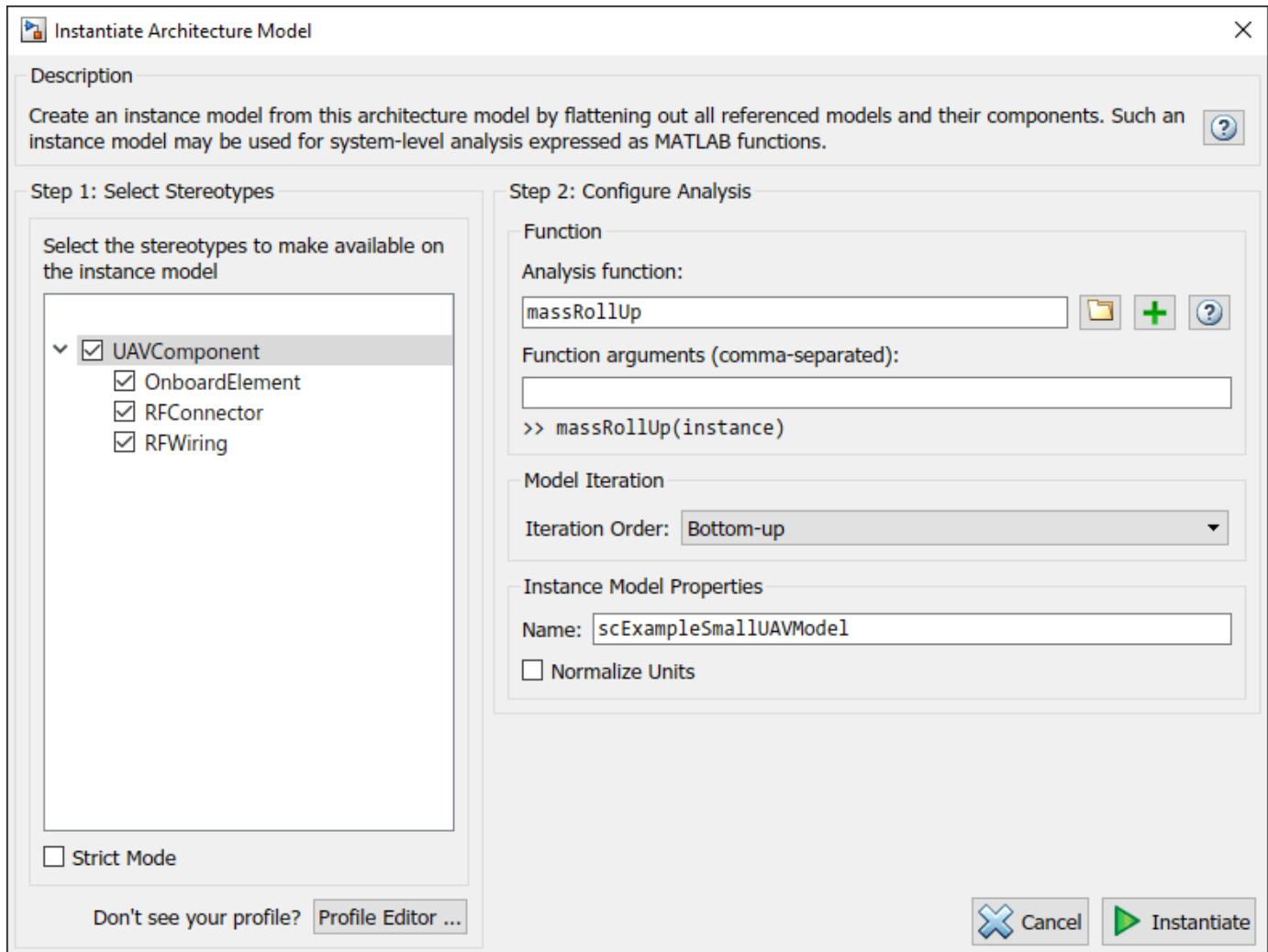


Analyze the Model

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.

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** 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

- “Requirement Links” on page 2-32
- “Link Blocks and Requirements” on page 2-2
- “Review Requirements Implementation Status” on page 3-2

Requirements-Based Verification

- “Review Requirements Implementation Status” on page 3-2
- “Review Requirements Verification Status” on page 3-6
- “Validate Requirements by Analyzing Model Properties” on page 3-9
- “Justify Requirements” on page 3-16
- “Linking to a Test Script” on page 3-19
- “Include Results from External Sources in Verification Status” on page 3-27
- “Linking to a Result File” on page 3-30
- “Integrating Results from a Custom-Authored MATLAB Script as a Test” on page 3-36
- “Integrating Results from an External Result file” on page 3-40
- “Integrating results from a custom authored MUnit script as a test” on page 3-44
- “Fix Requirements-Based Testing Issues” on page 3-48

Review Requirements Implementation Status

In this section...

“Implement Functional Requirements by Linking to Model Elements” on page 3-2
--

“View the Implementation Status” on page 3-3
--

Simulink Requirements 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” on page 2-40. When you select a requirement in the Requirements Editor, the requirement type is displayed in the **Details** 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 2-33 and “Define Custom Requirement and Link Types” on page 2-40.

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 3-16.

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 2-33.

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 **Details** pane, under **Links**.

The screenshot shows the Requirements Editor window with a table of requirements. The table has columns for Index, ID, Summary, Implemented, and Verified. Requirement #3, 'Avoid repeating commands', is highlighted. The right-hand pane shows details for Requirement #3, including a link to 'doNot Repeat' under the 'Implemented by' section.

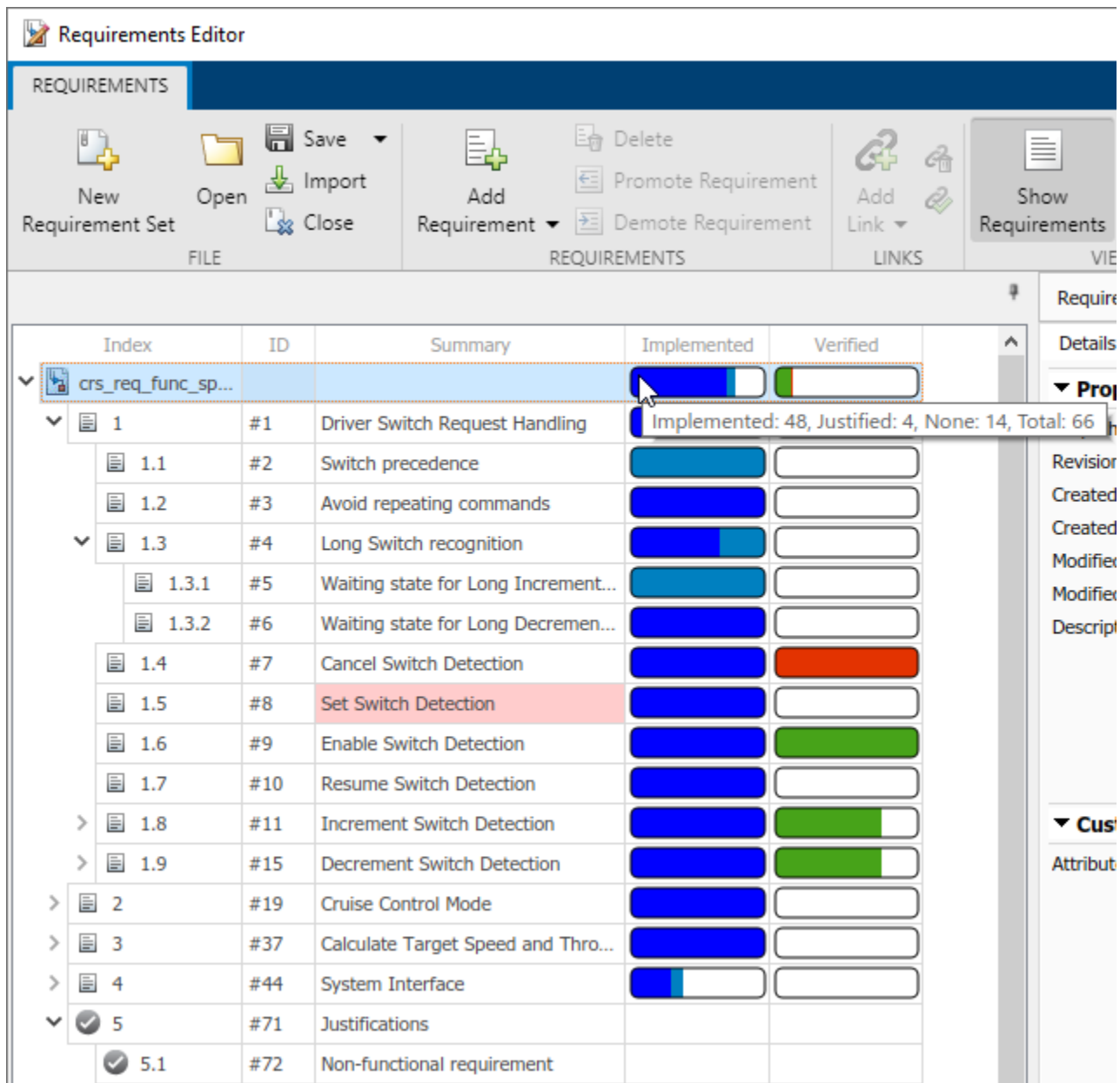
Index	ID	Summary	Implemented	Verified
crs_req_func_spec				
1	#1	Driver Switch Request Handling		
1.1	#2	Switch precedence		
1.2	#3	Avoid repeating commands		
1.3	#4	Long Switch recognition		
1.4	#7	Cancel Switch Detection		
1.5	#8	Set Switch Detection		
1.6	#9	Enable Switch Detection		
1.7	#10	Resume 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.

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 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 3-16.
- **None** (colorless): The requirement does not have any `Implement` type links.

See Also

More About

- “Review Requirements Verification Status” on page 3-6
- “Justify Requirements” on page 3-16
- “Requirement Types” on page 1-6
- “Link Types” on page 2-33

Review Requirements Verification Status

In this section...
“Verify Functional Requirements” on page 3-6
“Display Verification Status” on page 3-7
“Update Verification Status by Running Tests or Analyses” on page 3-8
“Include Verification Status in Report” on page 3-8

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 2-33.

Verify Functional Requirements

The requirement type specifies the role of a requirement. Functional requirements are meant to be implemented and contribute to the verification status, as well as requirements with a custom type that is a subtype of `Functional`. Other requirement types do not contribute to verification status. For more information, see “Requirement Types” on page 1-6 and “Define Custom Requirement and Link Types” on page 2-40.

You can verify functional requirements by linking them with certain verification items with `Verify` type links.

- **Simulation testing:** Requirement verification status reflects the result of the following linkable Simulink Test items after they are run 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” (Simulink Test).

Run-time assessments from verify statements or “Model Verification Blocks” (Simulink Test) can be captured by monitoring those assessments through test cases in the Test Manager. For more information, see Assess Model Simulation Using `verify` Statements (Simulink Test).

Note To view the verification status of a requirement that is linked to a test authored in MATLAB, you must use a MATLAB-based Simulink test. See “Test Models Using MATLAB-Based Simulink Tests” (Simulink Test).

- **Property proving:** Verification status reflects the analysis result 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 “Requirement Links” on page 2-32.

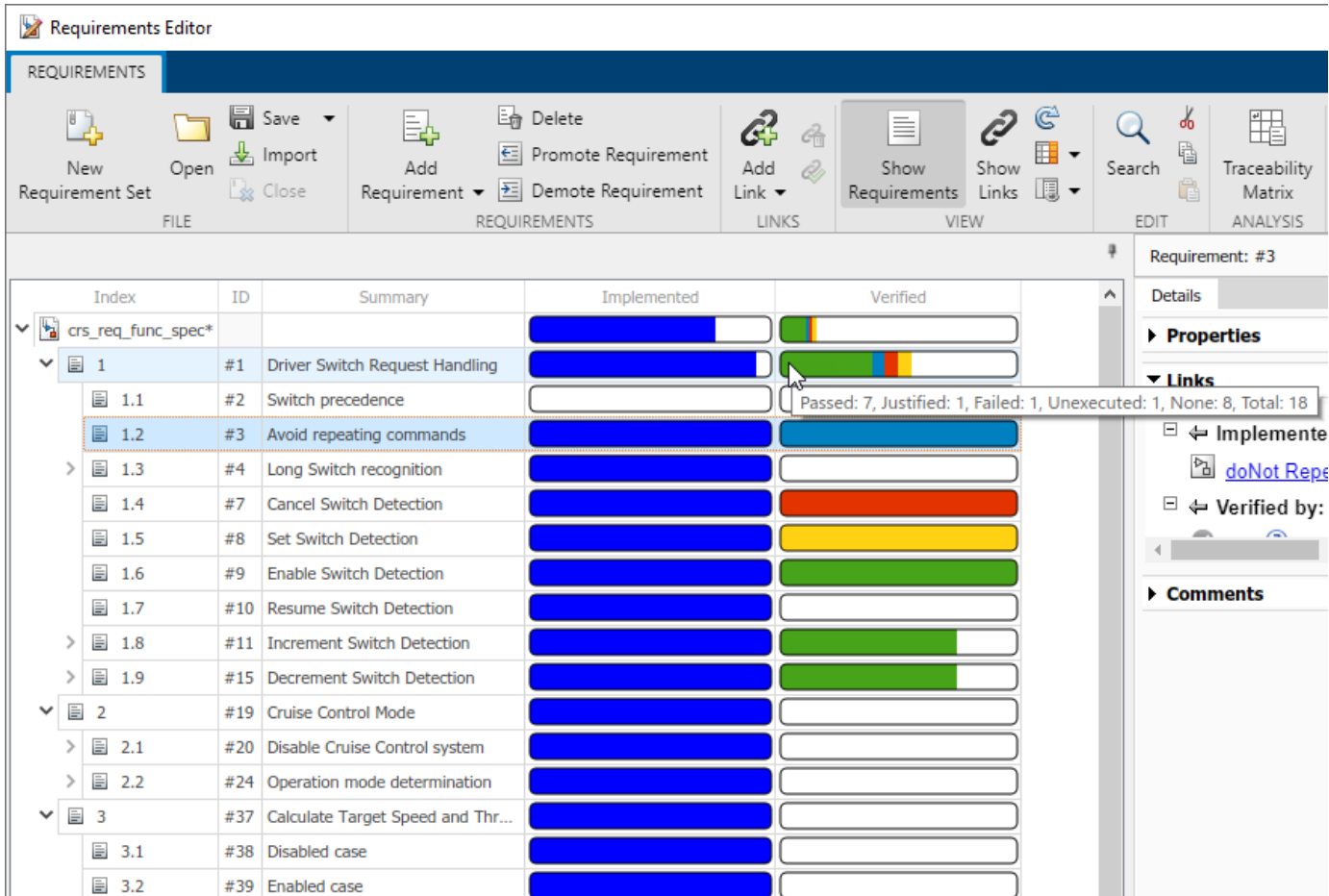
You can also verify requirements by linking to external result sources with Confirm type links. For more information, see “Include Results from External Sources in Verification Status” on page 3-27

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:



The screenshot shows the Requirements Editor interface. The main window displays a table of requirements with columns for Index, ID, Summary, Implemented, and Verified. The 'Verified' column contains progress bars indicating the status of each requirement. Requirement #3, 'Avoid repeating commands', has a bar with green, yellow, and red segments, indicating a failed result. A tooltip for requirement #3 shows the following statistics: Passed: 7, Justified: 1, Failed: 1, Unexecuted: 1, None: 8, Total: 18.

Index	ID	Summary	Implemented	Verified
1	#1	Driver Switch Request Handling	[Progress Bar]	[Progress Bar]
1.1	#2	Switch precedence	[Progress Bar]	[Progress Bar]
1.2	#3	Avoid repeating commands	[Progress Bar]	[Progress Bar]
1.3	#4	Long Switch recognition	[Progress Bar]	[Progress Bar]
1.4	#7	Cancel Switch Detection	[Progress Bar]	[Progress Bar]
1.5	#8	Set Switch Detection	[Progress Bar]	[Progress Bar]
1.6	#9	Enable Switch Detection	[Progress Bar]	[Progress Bar]
1.7	#10	Resume Switch Detection	[Progress Bar]	[Progress Bar]
1.8	#11	Increment Switch Detection	[Progress Bar]	[Progress Bar]
1.9	#15	Decrement Switch Detection	[Progress Bar]	[Progress Bar]
2	#19	Cruise Control Mode	[Progress Bar]	[Progress Bar]
2.1	#20	Disable Cruise Control system	[Progress Bar]	[Progress Bar]
2.2	#24	Operation mode determination	[Progress Bar]	[Progress Bar]
3	#37	Calculate Target Speed and Thr...	[Progress Bar]	[Progress Bar]
3.1	#38	Disabled case	[Progress Bar]	[Progress Bar]
3.2	#39	Enabled case	[Progress Bar]	[Progress Bar]

The fullness of the bar indicates how many requirements in a group (parent + children) are linked to verification items. 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 3-16.
- **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 4-10

See Also

More About

- “Review Requirements Implementation Status” on page 3-2
- “Link to Test Cases from 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:

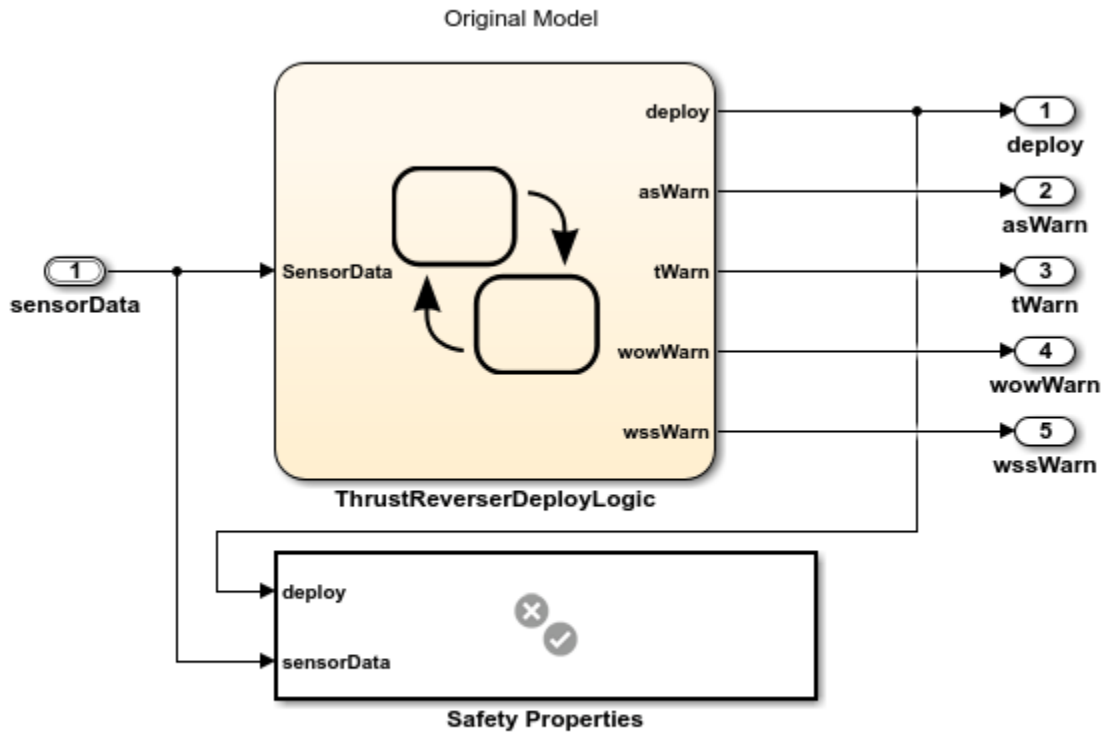
- 1 The thrust reverser shall not deploy if the airspeed is greater than 150 knots.
- 2 The thrust reverser shall not deploy if the aircraft is in the air, as indicated by the value of the weight on wheels sensors. If the aircraft is in the air, the signal value for each of two weight on wheels (WOW) 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	

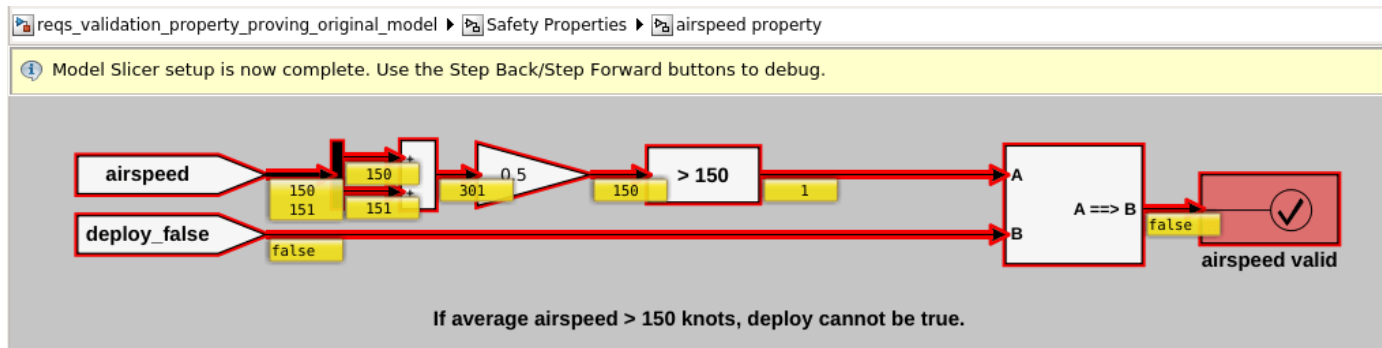
Analyze Model Behavior with Counterexamples

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.

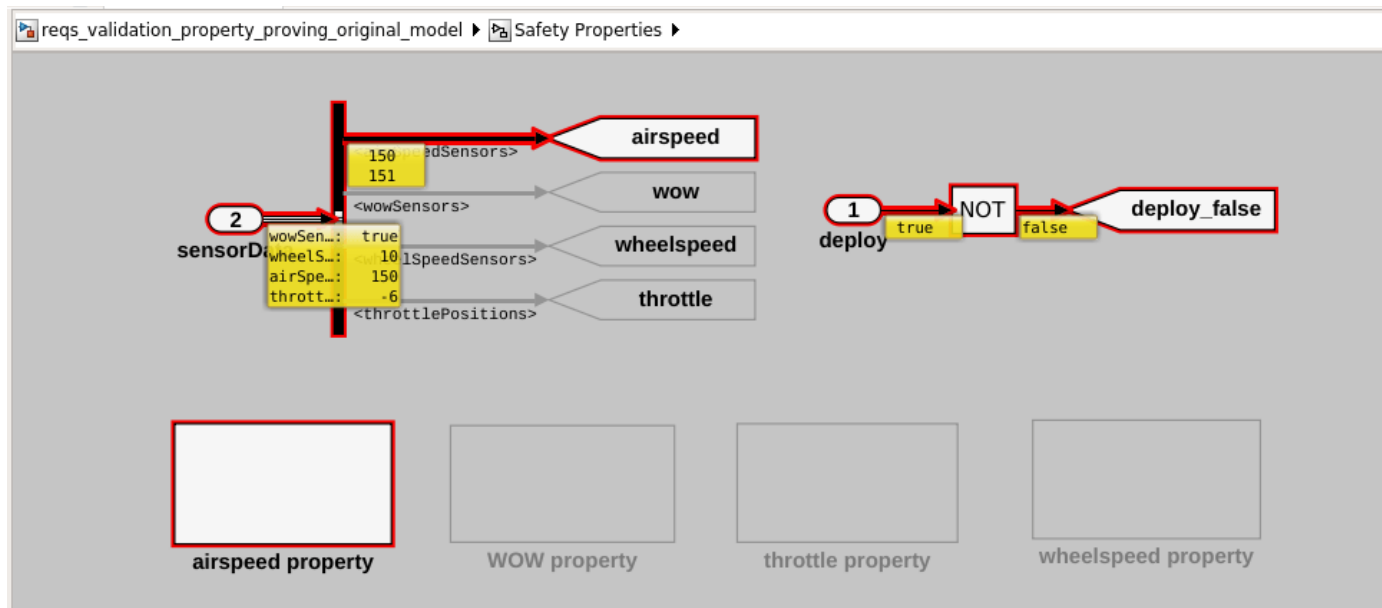
The counterexample time series indicates a condition that might be difficult to encounter in simulation, but nonetheless causes model behavior that violates a requirement. Investigate the behavior by analyzing signal dependencies in the counterexample:

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.



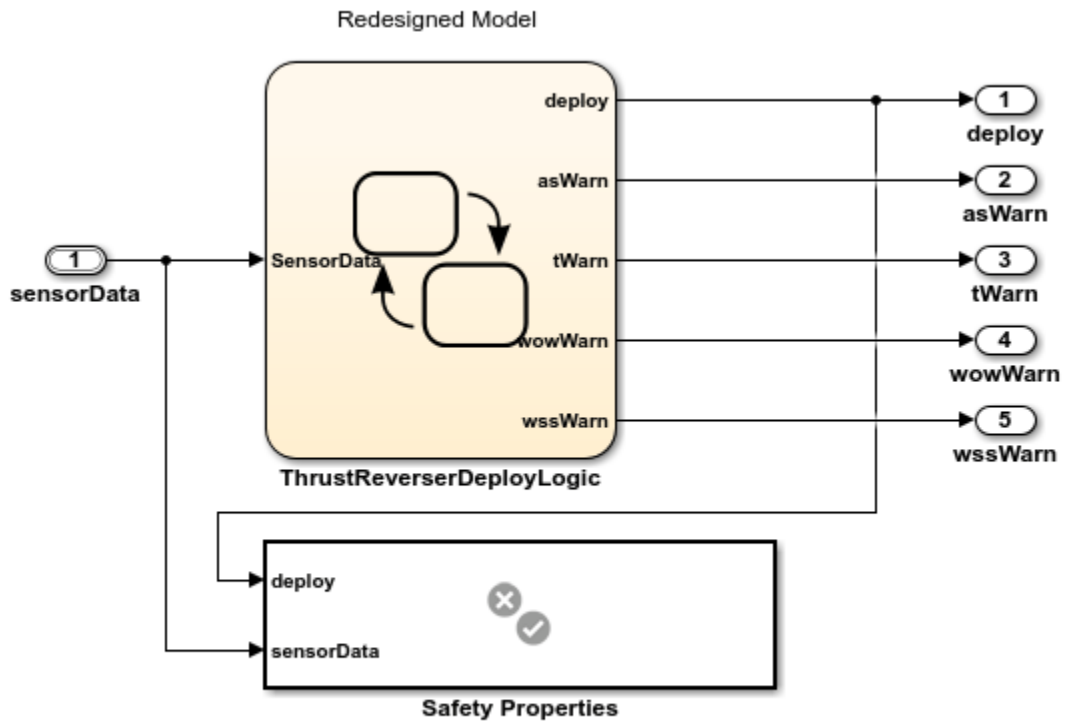
The falsified property and the dependency analysis suggest that the control system algorithm is incompletely designed, and the requirements are incomplete.

Analyze the Redesigned System

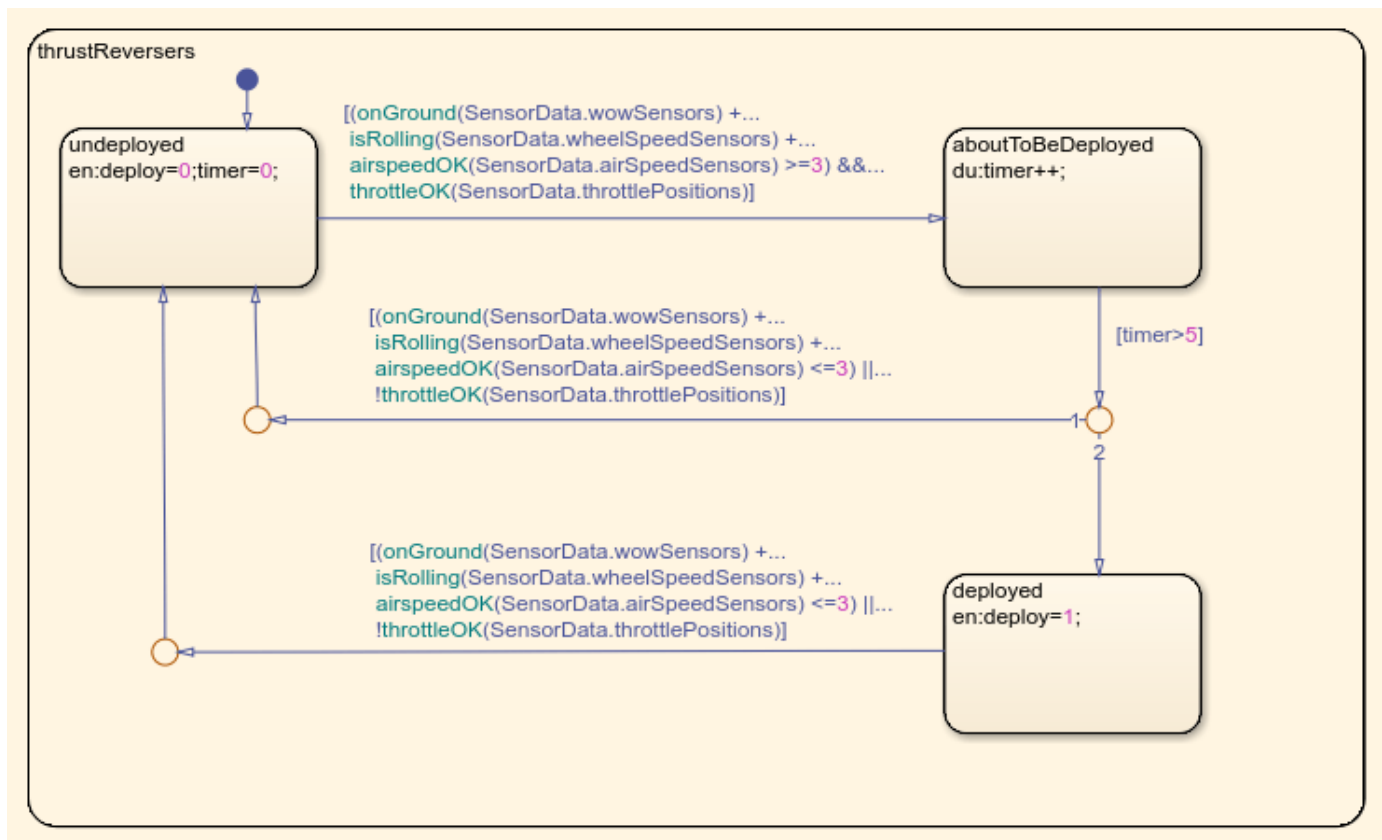
Redesigning a control system requires rethinking requirements. In this case, the lack of an intermediate standby state allows undesirable system behavior when inputs change suddenly. Adding an intermediate deployment mode which delays thrust reverser response resolves the issue.

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:

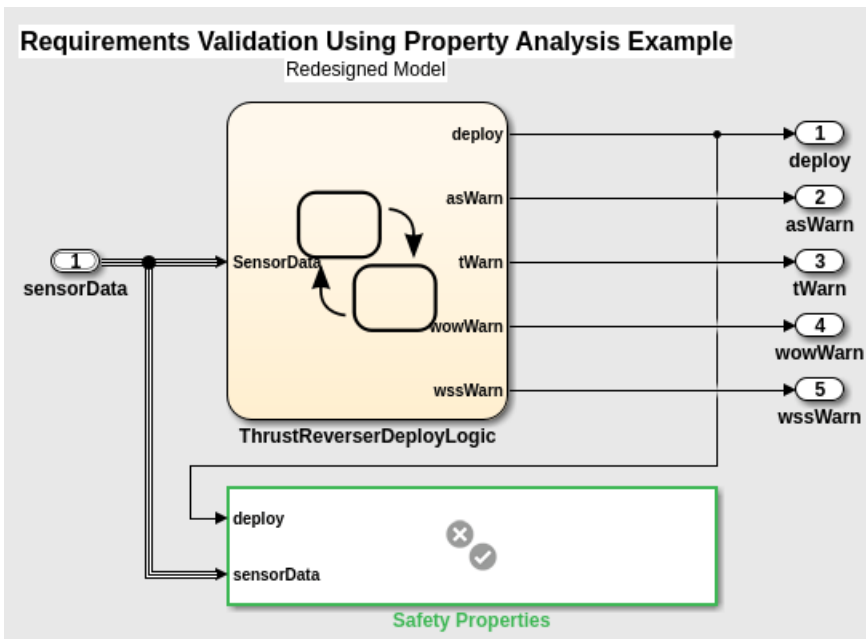
1. In the model, from the **Apps** tab, click **Requirements Manager**.
2. In the **Requirements** tab, click **Requirements Editor**.
3. Open `thrust_reverser_safety_requirements` in the **Requirements Editor**.
4. For requirement 1.1, Airspeed Condition, link to the airspeed valid block in the Safety 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 **Details** 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	
2	R1.2	WOW Condition	
3	R1.3	Throttle Condition	
4	R1.4	Wheelspeed Condition	

The results show that the properties are valid.

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 Simulink Requirements. 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 **Details** 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 3-2
- “Review Requirements Verification Status” on page 3-6

Linking to a Test Script

In this workflow, you link a requirement to a MATLAB script using the “Outgoing Links Editor” on page 10-6 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 3-27.

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 Simulink 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 3-19
- “Create the Link” on page 3-20
- “View the Verification Status” on page 3-21

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 10-2.

- 4** Save `linktype_mymscripttap.m`.
5 Register the link type. At the command line, enter:

```
rmi register linktype_mymscripttap
```

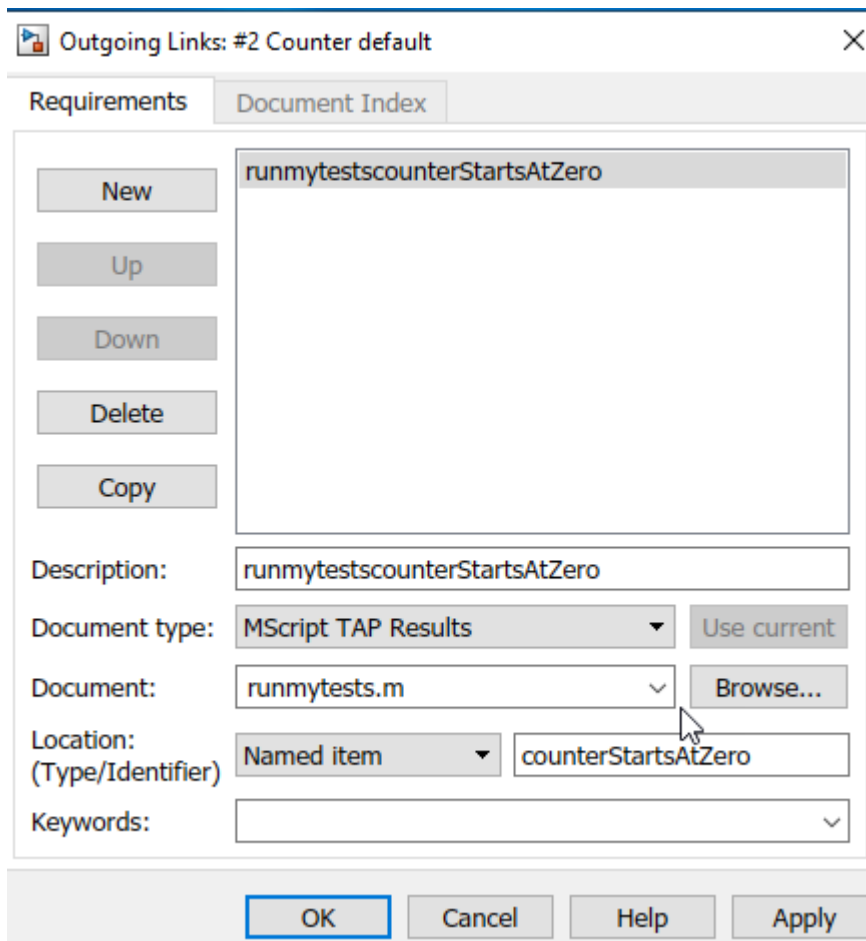
Note If the command returns a warning, then you must unregister the file and follow step 5 again. Unregister the file by entering:

```
rmi unregister linktype_mymscripttap
```


Create the Link

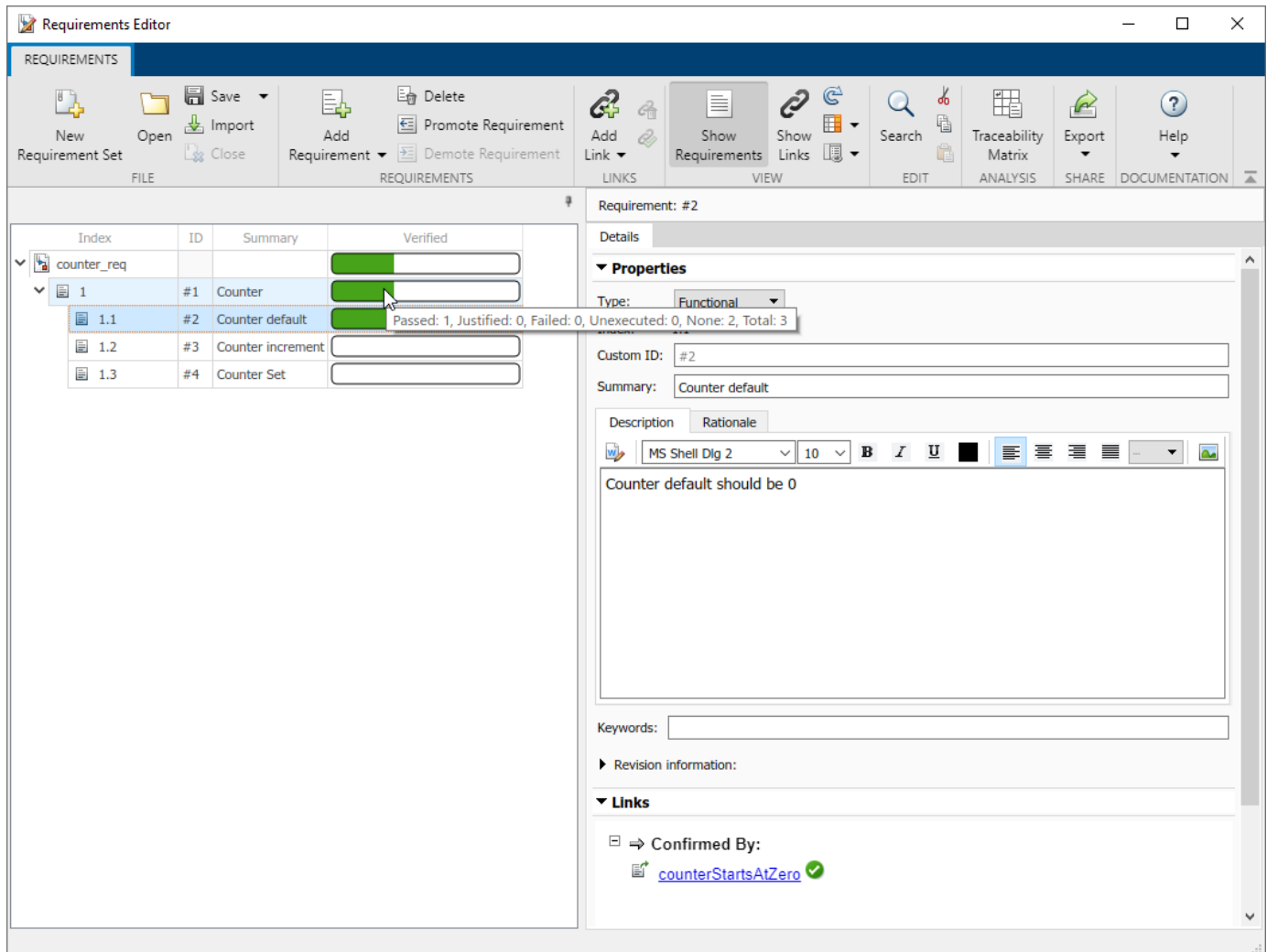
Follow these steps to add the link manually in the Outgoing Links Editor:

- 1** Open the Requirements Editor and, in the `counter_req.slreqx` requirements 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.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 Simulink 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 3-23
- “Create the Link” on page 3-24
- “View the Verification Status” on page 3-24

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 10-2.

- 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:

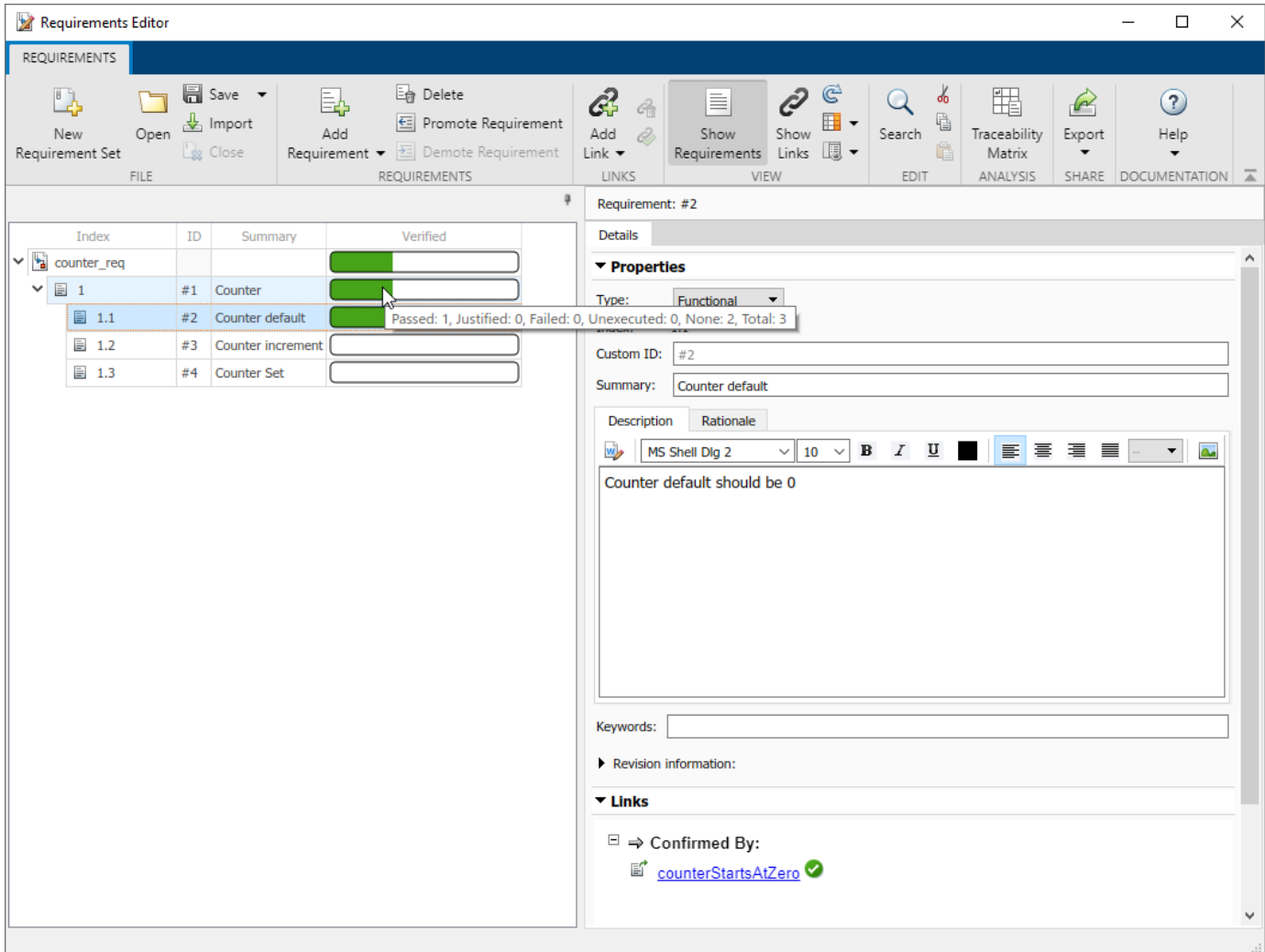
```
reqSet.updateVerificationStatus
```


Fetch the verification status for the requirement by entering :

```
status = reqSet.getVerificationStatus
```

This shows which of the requirements in the requirements 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:

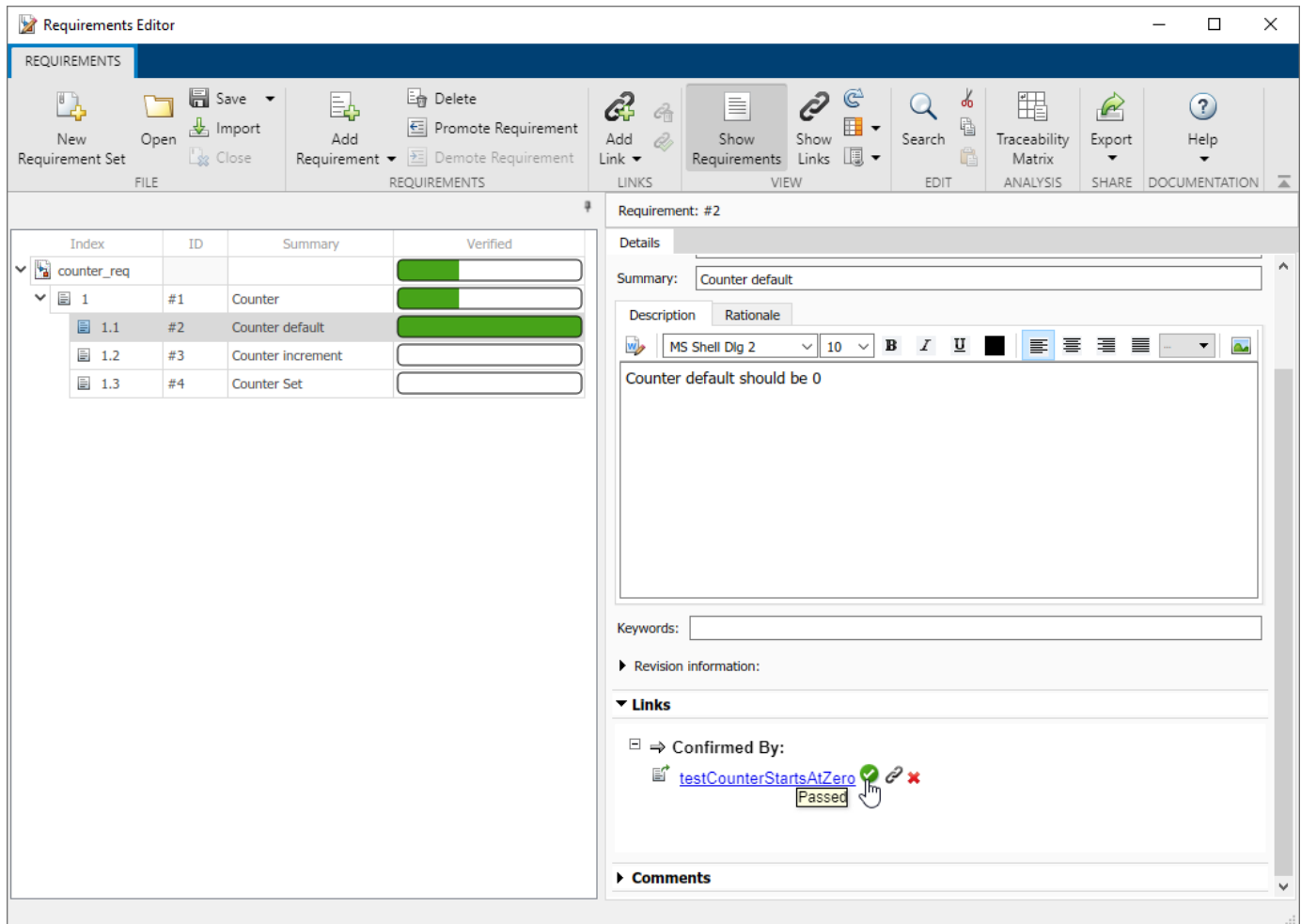
- Details:** Requirement: #2
- Properties:** Type: Functional
- Custom ID:** #2
- Summary:** Counter default
- Description:** MS Shell Dlg 2, 10, Counter default should be 0
- Keywords:**
- Revision information:**
- Links:** Confirmed By: [counterStartsAtZero](#)

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 Simulink Requirements Editor.



See Also

More About

- “Include Results from External Sources in Verification Status” on page 3-27

Include Results from External Sources in Verification Status

Simulink Requirements allows you to include the verification status of results from external sources in the Simulink 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 3-6.

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 10-8.

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:

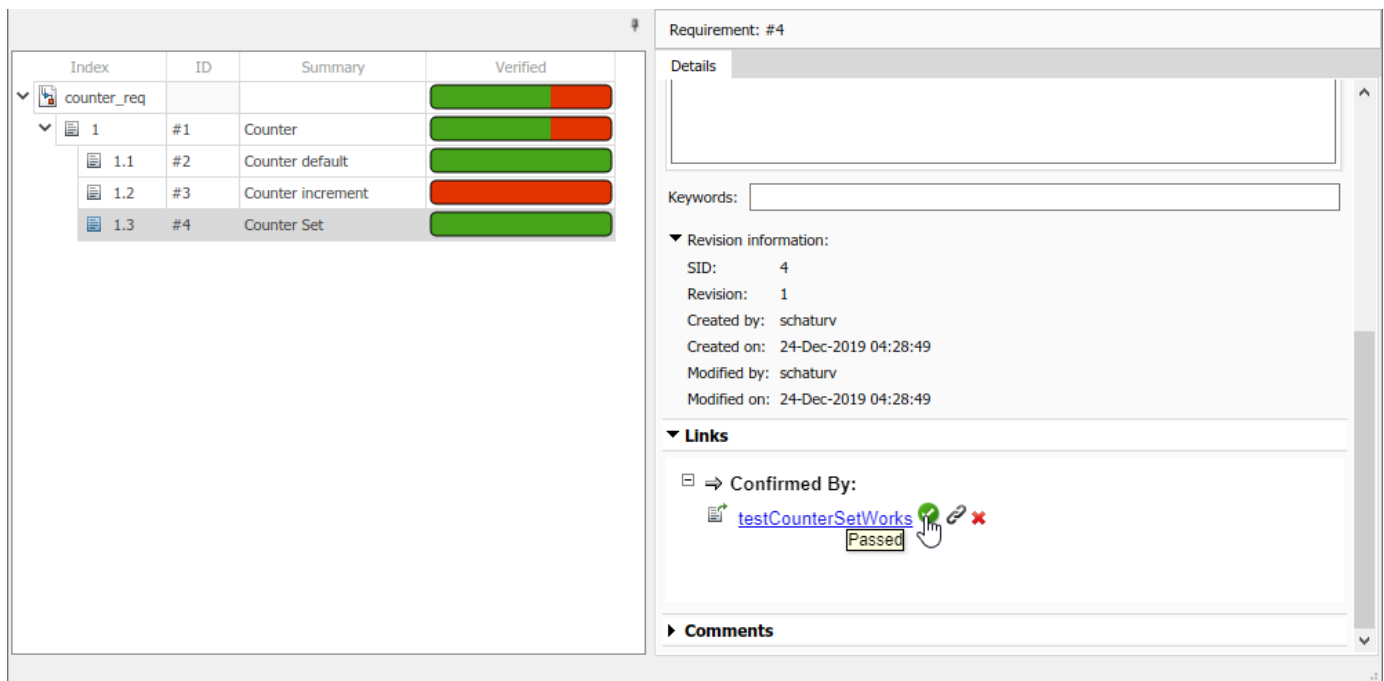
- 1 The file location
- 2 The file format (for example, JUnit or Excel)

For details, see “Linking to a Result File” on page 3-30.

- 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 3-19.

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 10-2.
 - b Save the function on the MATLAB path.
- For details, see “Custom Link Type Registration” on page 10-15.
- 2 **Register the custom link type:** See “Custom Link Type Registration” on page 10-15. 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 3-7.
 - 5 **Refresh the requirements view:** After the tests run, refresh the verification status by clicking the **Refresh**  button.



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 10-2.

- 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 3-36
- “Integrating Results from an External Result file” on page 3-40
- “Integrating results from a custom authored MUnit script as a test” on page 3-44

More About

- “Linking to a Test Script” on page 3-19
- “Linking to a Result File” on page 3-30

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 10-6 and the API. The verification status in the Simulink 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 3-27.

Open Example Files

Open the “Integrating Results from an External Result file” on page 3-40 example.

```
openExample(['slrequirements/' ...
    '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 Simulink 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 Simulink Requirements 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 10-2.

- 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
```

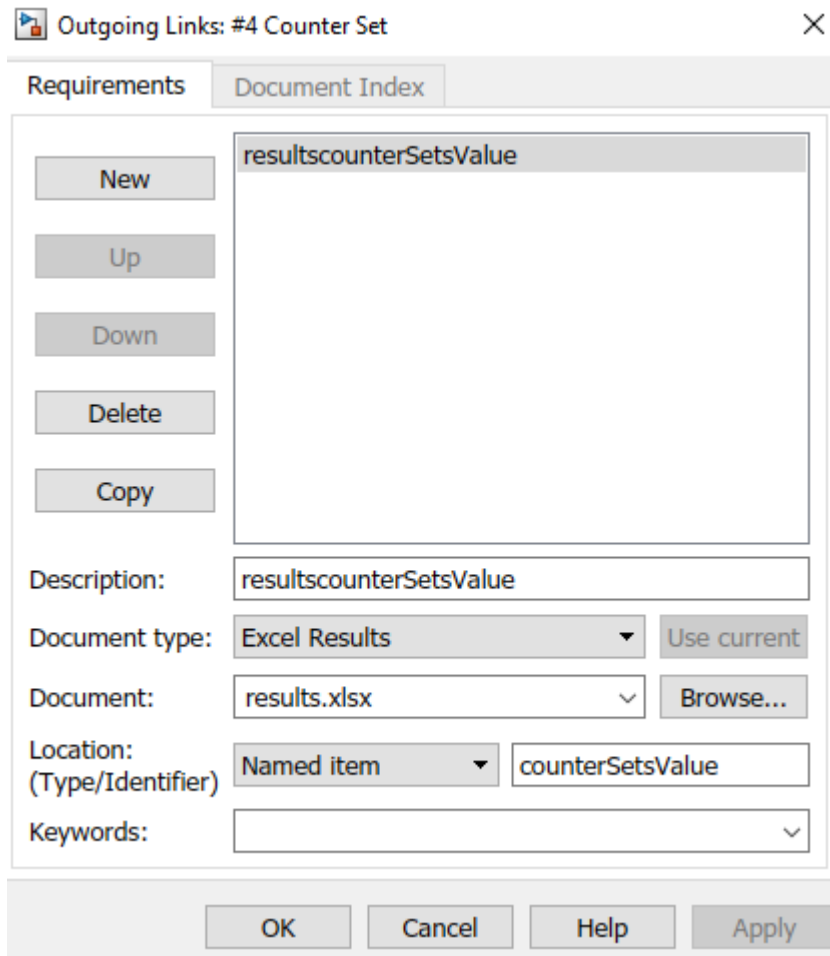
Create a Requirement Link

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 Simulink Requirements 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';
```

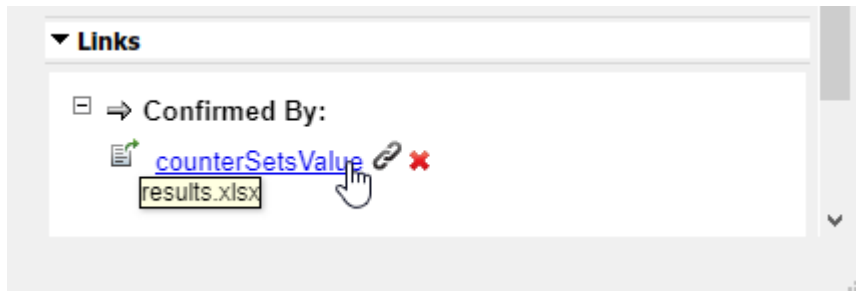
- 2 Open the requirement set and find the requirement related to the link:

```
reqSet = slreq.open('counter_req.slmx');
requirement = find(reqSet, 'Type', 'Requirement', 'SID', 4);
```

- 3 Create the link by entering:



```
link = slreq.createLink(requirement, externalSource);
```

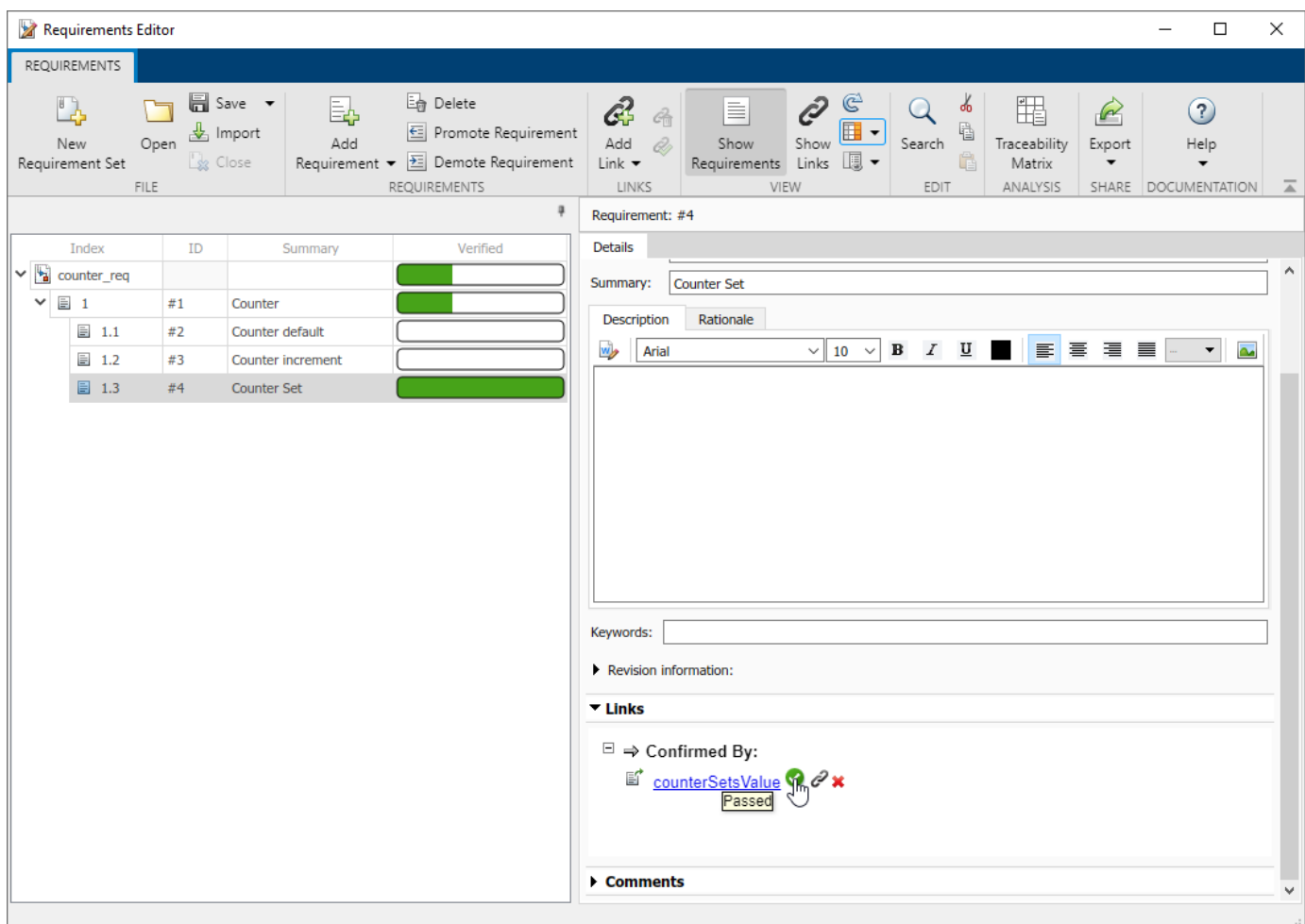
This creates the link from the requirement with SID 4 to the result for the test case in the external result file called counterSetsValue. In Requirements Editor, the link appears in the **Links > Confirmed By** section.



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.



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 3-27

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 Simulink Requirements Editor reflects the test results. This example performs the steps described in “Linking to a Test Script” on page 3-19. 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
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 10-2.

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';
```

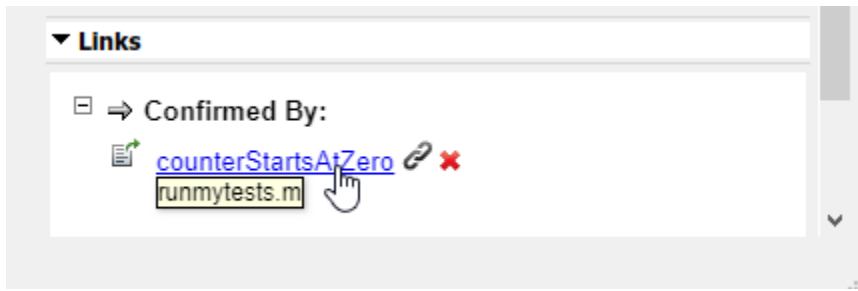
The requirement related to the link has its SID set to 2. To find the requirement related to the link, enter:

```
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

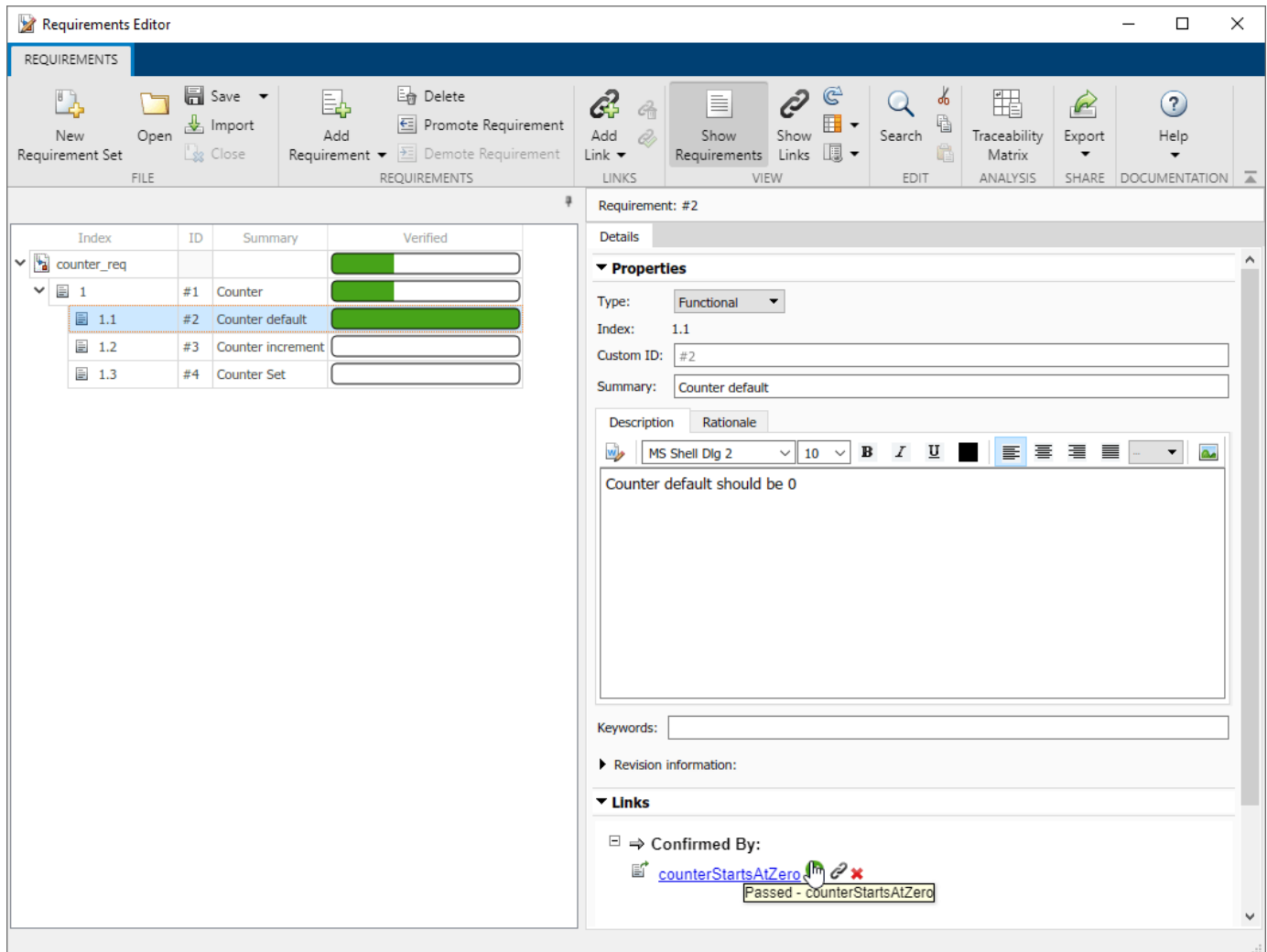
To view the verification status, you need to first update the verification status for the requirement set. At the MATLAB command prompt, type:

```
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;
```

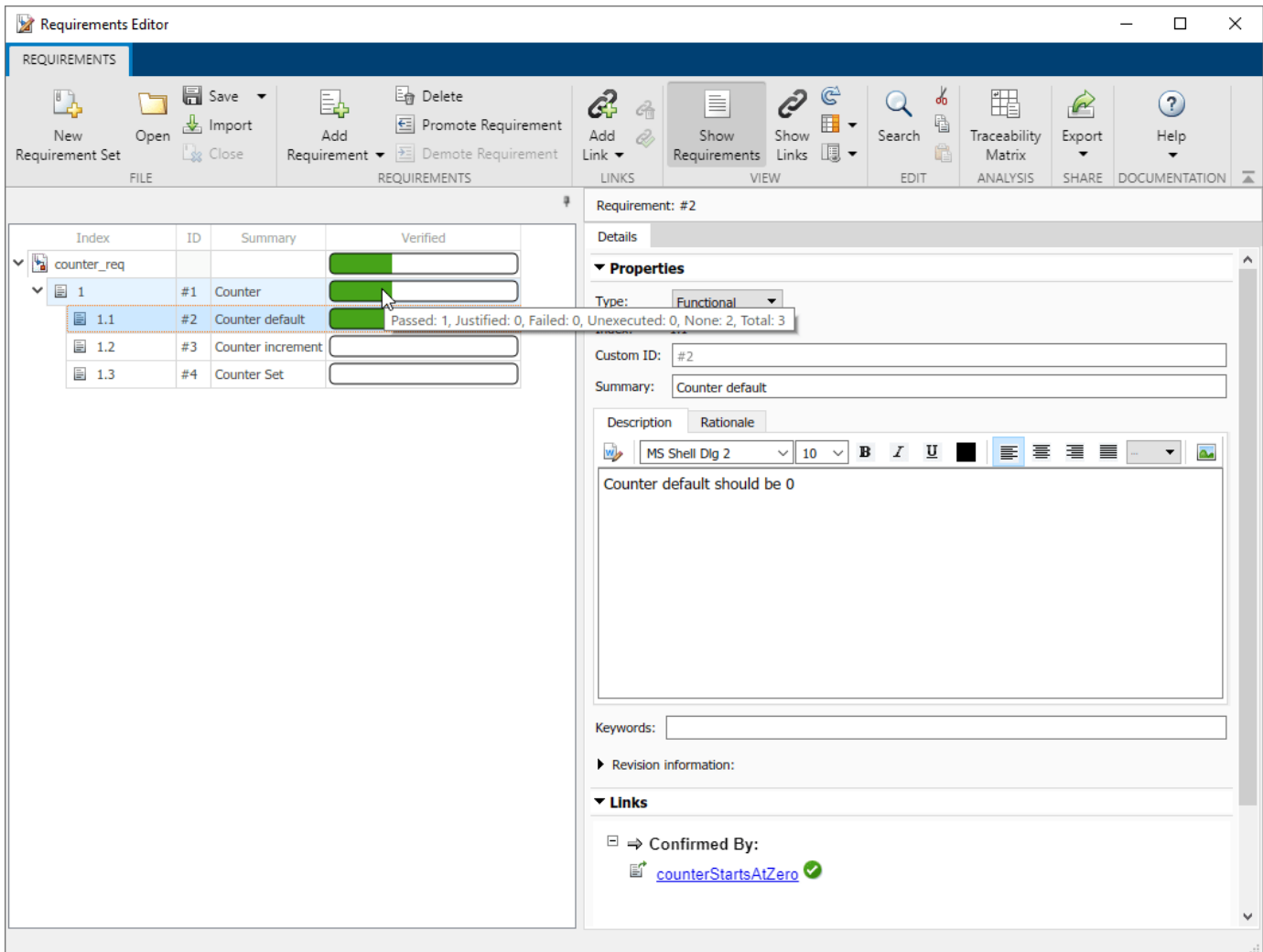
The Requirements Editor shows the verification status for entire requirements set that are passed or failed.



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.



Cleanup

Clear open requirement sets and link sets, and close any open models without saving changes. Unregister the link type.

```
slreq.clear;
bdclose('all');
rmi unregister linktype_mymscripttap;
```

See Also

More About

- “Include Results from External Sources in Verification Status” on page 3-27
- “Linking to a Test Script” on page 3-19

Integrating Results from an External Result file

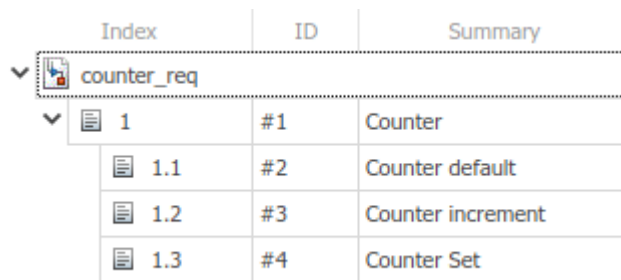
In this example, you link a requirement to a result file in Excel Format. The verification status in the Simulink Requirements Editor reflects the test results. This example performs the steps described in “Linking to a Result File” on page 3-30. To run this example, click **Open Example** and run it. 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`.

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 external results file. The domain registration needed for this example is `linktype_myexcelresults.m`. To register the custom linktype `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 10-2. **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`.

Step 2: 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 **Details** pane under **Links**.



Step 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, select  **Columns > Verification Status**. After the update, fetch the verification status for the requirement:

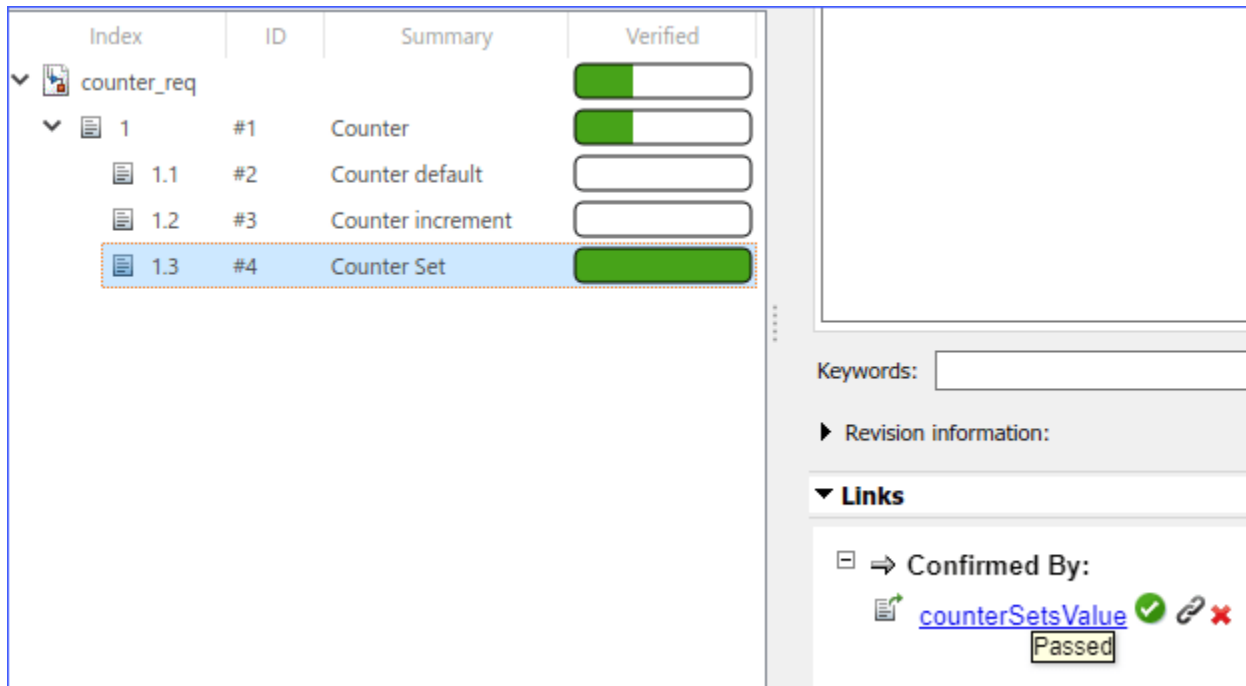
```
status = reqSet.getVerificationStatus
```

```
status = struct with fields:
    total: 3
    passed: 1
    failed: 0
    unexecuted: 0
    justified: 0
    none: 2
```

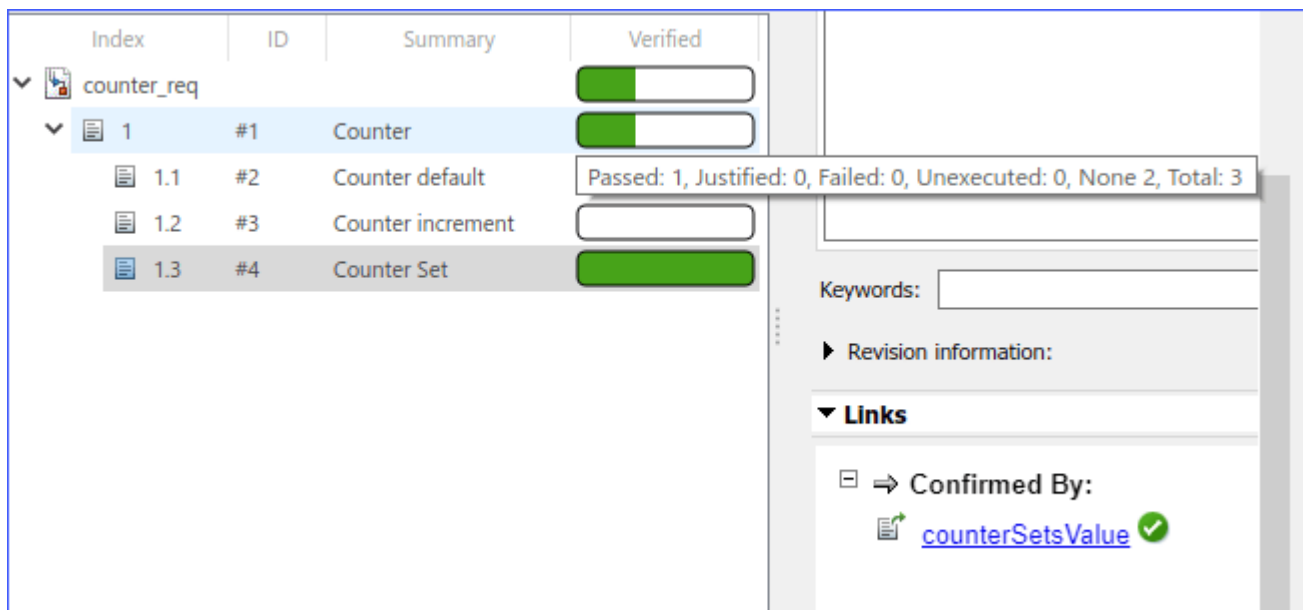
Open the Requirements Editor to see the verification status:


```
reqSet = slreq.open('counter_req.slreqx');
```

The Requirements Editor shows the verification status for each requirement in the requirement set.



The verification status for requirements for the counterSetsValue is fully verified.



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

Clear open requirement sets and link sets, and close any open models without saving changes. Unregister the linktype.

```
slreq.clear;  
bdclose('all');  
rmi unregister linktype_myexcelresults;
```

See Also

More About

- “Include Results from External Sources in Verification Status” on page 3-27
- “Linking to a Result File” on page 3-30

Integrating results from a custom authored MUnit script as a test

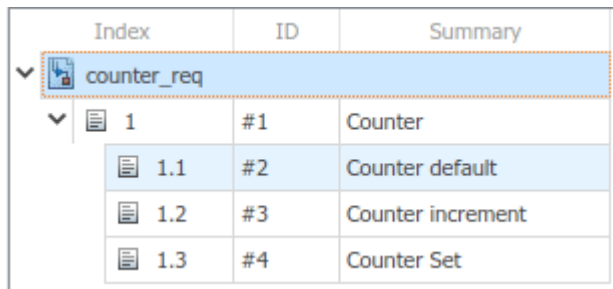
In this example, you integrate the results from a MATLAB xml Unit test by linking to a test script. The verification status in the Simulink Requirements Editor reflects the test results. To run this example, click **Open Example** and run it. This example uses:

- 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 10-2. 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';
```

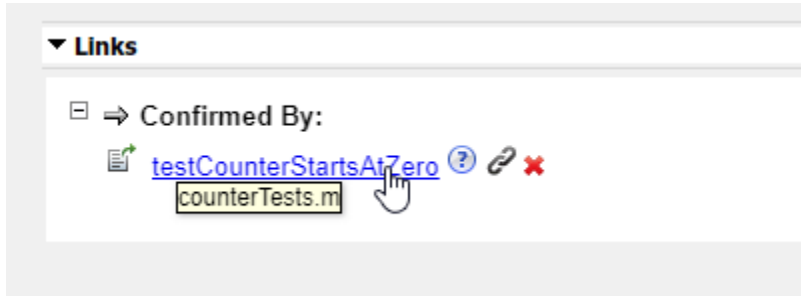
The requirement related to the link has its SID set to 2. To find the requirement related to the link, enter:

```
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 **Details** 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:

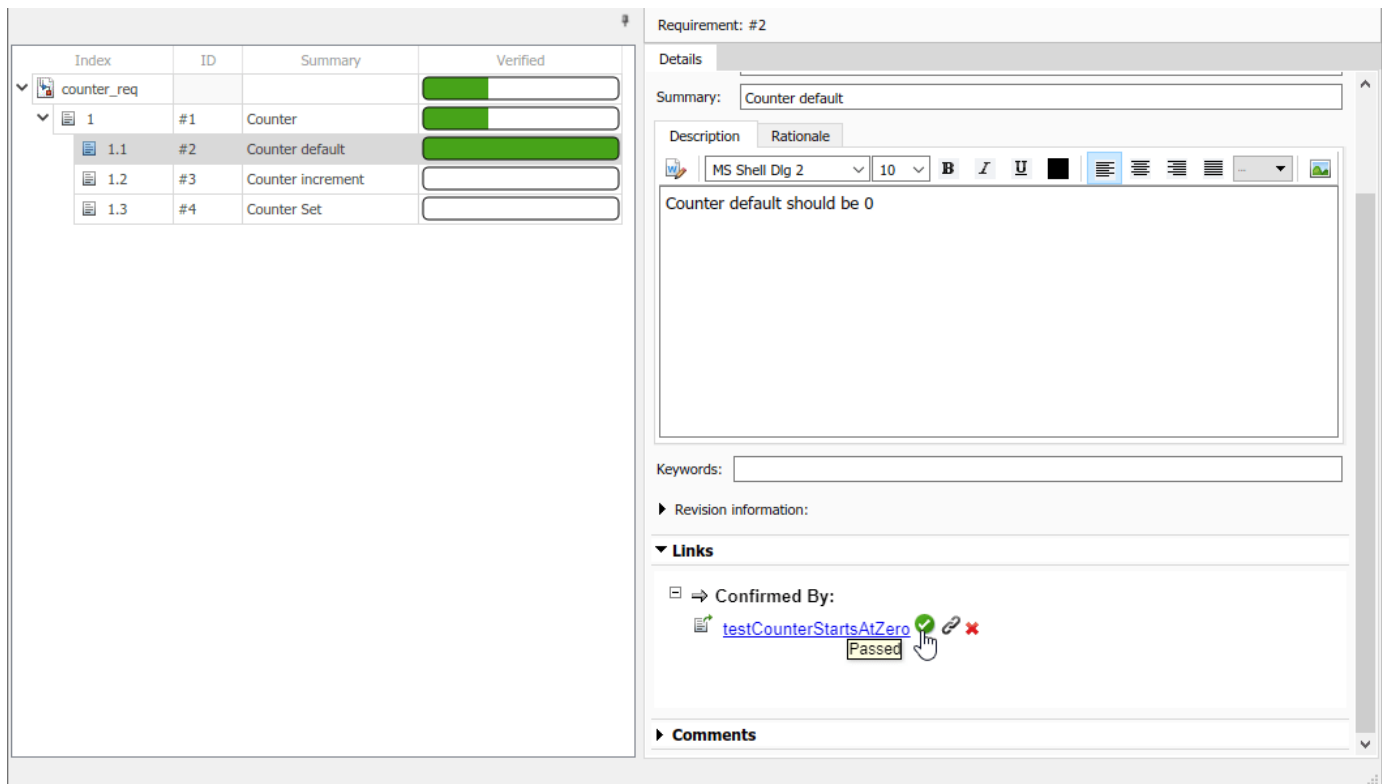
```
reqSet.updateVerificationStatus;
```

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
```

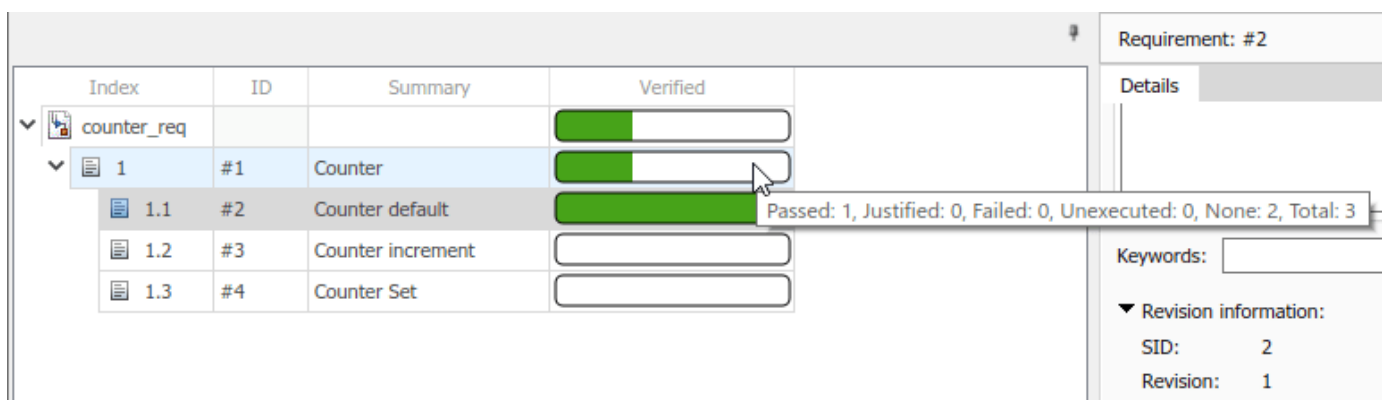
The Requirements Editor shows the verification status for entire requirements set that are passed or failed.



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.



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

Clear open requirement sets and link sets, and close any open models without saving changes. Unregister the link type.

```
slreq.clear;  
bdclose('all');  
rmi unregister linktype_mymljunitresults;
```

See Also

More About

- “Include Results from External Sources in Verification Status” on page 3-27
- “Integrating Results from a MATLAB Unit Test Case” on page 3-25

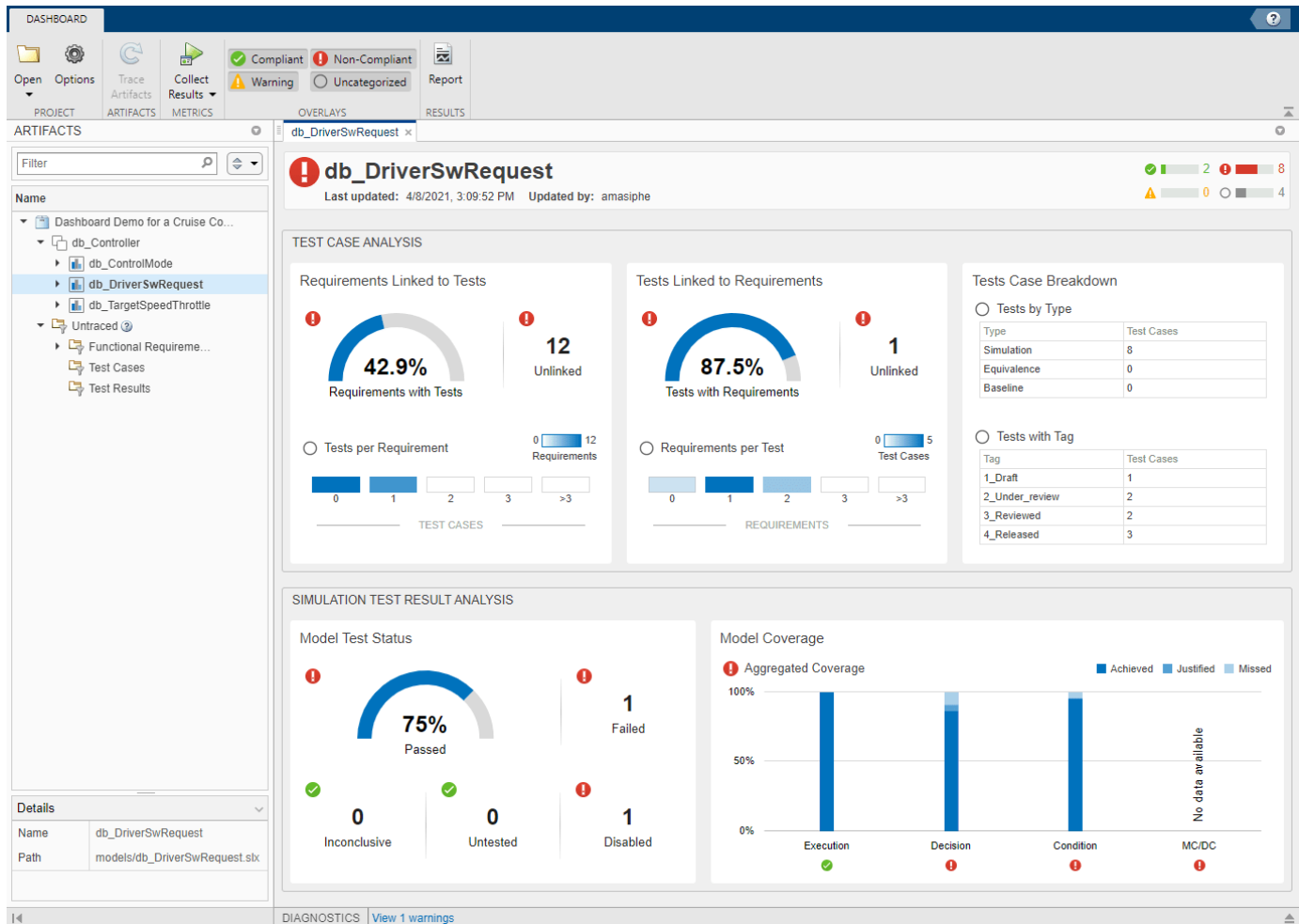
Fix Requirements-Based Testing Issues

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

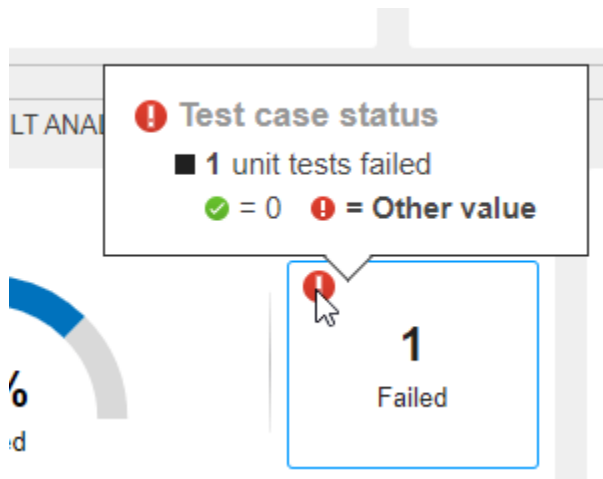
The dashboard displays testing data for a model and the artifacts that the model traces to within a project. For this example, open the project and collect metric data for the artifacts.

- 1 Open the project. At the command line, type `dashboardCCProjectStart`.
- 2 Open the dashboard. On the **Project** tab, click **Model Testing Dashboard**.
- 3 If you have not previously opened the dashboard for the project, the dashboard must identify the artifacts in the project and trace them to the models. To run the analysis and collect metric results, click **Trace and Collect All**.
- 4 In the **Artifacts** pane, the dashboard organizes unit models under the component models that contain them in the model hierarchy. Artifacts such as requirements, test cases, and test results appear under the unit models that they trace to. View the metric results for the model `db_DriverSwRequest`. In the **Artifacts** pane, click the name of the model. The dashboard populates the widgets with data from the most recent metric collection for the model.



You can use the overlays in the Model Testing Dashboard to see if the metric results for a widget are compliant, non-compliant, or generate a warning that the metrics results should be reviewed. Results are compliant if they show full traceability, test completion, or model coverage. In the **Overlays** section, ensure the **Compliant** and **Non-Compliant** buttons are selected. The overlay appears on the widgets that have results in that category. You can see the total number of widgets in each compliance category in the top-right corner of the dashboard.

To see the compliance thresholds for a metric, point to the overlay icon.



You can hide the overlay icons by clicking a selected category in the **Overlays** section.

For more information on the compliance thresholds for each metric, see “Model Testing Metrics” (Simulink Check).

Link a Requirement to its Implementation in a Model

On the **Artifacts** panel, the **Untraced** folder shows artifacts that do not trace to the unit models in the project. You can check the artifacts in this folder to see if there are any requirements that should be implemented by the models but are missing links. For this example, link one of these requirements to the model block that implements it and update the **Artifacts** panel to reflect the link.

- 1 In the **Artifacts** panel, navigate to the requirement **Untraced > Functional Requirements > db_req_func_spec.slreqx > Switch precedence**.
- 2 Open the requirement in the Requirements Editor. On the **Artifacts** panel, double-click **Switch precedence**. This requirement describes the order in which the cruise control system takes action if multiple switches are enabled at the same time. Keep the Requirements Editor open with the requirement selected.
- 3 Open the model `db_Controller`. To open the model from the Model Testing Dashboard, in the **Artifacts** panel, expand the folder **db_Controller > Design** and double-click `db_Controller.slx`.
- 4 The Model block `DriverSwRequest` references the model `db_DriverSwRequest`, which controls the order in which the cruise control system takes action when the switches are enabled. Link this model block to the requirement. Right-click the model block and select **Requirements > Link to Selection in Requirements Browser**.
- 5 Save the model. On the **Simulation** tab, click **Save**.
- 6 Save the requirements set. In the Requirements Editor, click the **Save** icon.
- 7 To update the artifact traceability information, in the Model Testing Dashboard, click **Trace Artifacts**.

The **Artifacts** panel shows the **Switch precedence** requirement under **db_Controller > Functional Requirements > db_req_func_spec.slreqx**. Next, find traceability issues in the artifacts by collecting metrics in the dashboard.

Address Testing Traceability Issues

Open the dashboard for the unit **db_DriverSwRequest** by clicking the name of the unit in the **Artifacts** panel. Because you changed the requirements file by adding a link, the dashboard widgets are highlighted in gray to show that the results might represent stale data. To update the results for the unit, click **Collect Results**.

The widgets in the **Test Case Analysis** section of the dashboard show data about the model requirements, test cases for the model, and links between them. The widgets indicate if there are gaps in testing and traceability for the implemented requirements.

Link Requirements and Test Cases

In the model `db_DriverSwRequest`, the **Requirements Linked to Tests** section shows that some of the requirements in the model are missing links to test cases. Examine the requirements by clicking one of the dashboard widgets. Then, use the links in the table to open the artifacts and fix the traceability issues.

To see detailed information about the unlinked requirements, in the **Requirements Linked to Tests** section, click the widget **Unlinked**. The table shows the requirements that are implemented in the model, but do not have links to a test case. The table is filtered to show only requirements that are missing links to test cases. For this example, link a test for the requirement `Set Switch Detection`.

Requirement linked to test cases

Metric that determines if the requirement is linked to test cases.

Artifact	Source	Test Link Status
Intermediate state	db_req_func_spec.slreqx	Missing linked tests
Resume Switch	db_req_func_spec.slreqx	Missing linked tests
Avoid repeating commands	db_req_func_spec.slreqx	Missing linked tests
Increment Switch	db_req_func_spec.slreqx	Missing linked tests
Set Switch	db_req_func_spec.slreqx	Missing linked tests
Driver Request	db_req_func_spec.slreqx	Missing linked tests
Set Switch Detection	db_req_func_spec.slreqx	Missing linked tests
Decrement Switch	db_req_func_spec.slreqx	Missing linked tests
Waiting state for Long Decrement switch detection	db_req_func_spec.slreqx	Missing linked tests
Enable Switch	db_req_func_spec.slreqx	Missing linked tests
Cancel Switch	db_req_func_spec.slreqx	Missing linked tests
Intermediate state	db_req_func_spec.slreqx	Missing linked tests

- 1 Open the requirement in the Requirements Editor. In the table, click `Set Switch Detection`.
- 2 In the Requirements Editor, examine the details of the requirement. This requirement describes the behavior of the `Set` switch when it is pressed. Keep the requirement selected in the Requirements Editor.
- 3 Check if there is already a test case for the switch behavior. To return to the metric results, at the top of the Model Testing Dashboard, click **db_DriverSwRequest**. The **Tests Linked to Requirements** section shows that one test case is not linked to requirements.

- 4 To see the unlinked test cases, in the **Tests Linked to Requirements** section, click **Unlinked**.
- 5 To open the test in the Test Manager, in the table, click the test case **Set** button. The test case verifies the behavior of the **Resume** switch. If there were not already a test case for the switch, you would add a test case by using the Test Manager.
- 6 Link the test case to the requirement. In the Test Manager, for the test case, expand the **Requirements** section. Click **Add > Link to Selected Requirement**. The traceability link indicates that the test case **Set** button verifies the requirement **Set Switch Detection**.
- 7 The metric results in the dashboard reflect only the saved artifact files. To save the test suite `db_DriverSwRequest_Tests.mldatx`, in the **Test Browser**, right-click `db_DriverSwRequest_Tests` and click **Save**.
- 8 Save the requirements file `db_req_func_spec.slreqx`. In the Requirements Editor, click the **Save** button.

Next, update the metric data in the dashboard to see the effect of adding the link.

Update Metric Results in the Dashboard

Update the metric results in the Model Testing Dashboard so that they reflect the traceability link between the requirement and the test case.

- 1 To analyze the artifact changes in the Model Testing Dashboard, click **Trace Artifacts**. The button is enabled when there are changes in the project artifacts that the dashboard has not analyzed.

2



At the top of the dashboard, the **Stale Metrics** icon indicates that at least one metric widget shows stale data for the model. Widgets that show stale metric data appear highlighted in grey. To refresh the widgets, re-collect the metric data for the model by clicking **Collect Results**.

The **Test Case Analysis** widgets show that there are 11 remaining unlinked requirements. The **Tests Linked to Requirements** section shows that there are no unlinked tests. 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

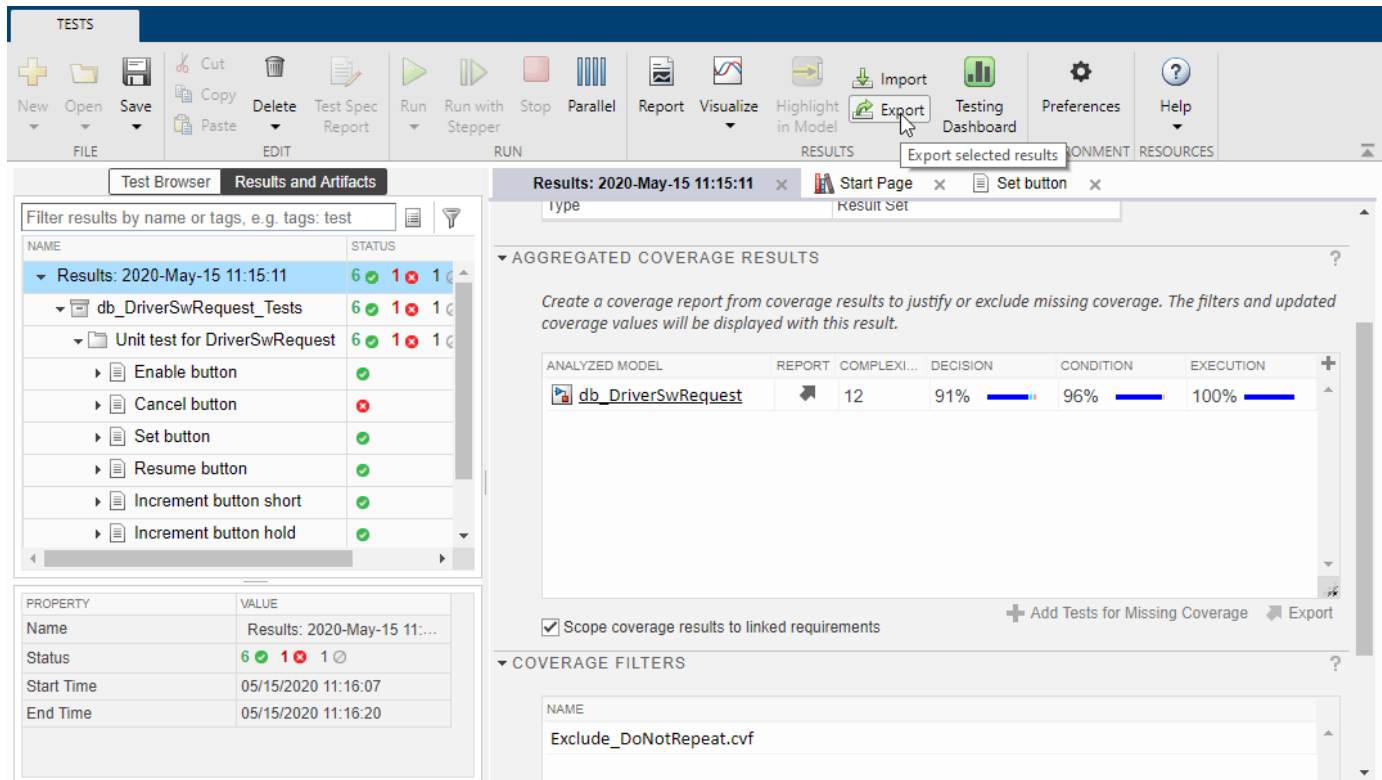
After you create and link unit tests that verify the requirements, run the tests to check that the functionality of the model meets the requirements. To see a summary of the test results and coverage measurements, use the widgets in the **Test Result Analysis** section of the dashboard. The widgets highlight testing failures and gaps. Use the metric results for the underlying artifacts to address the issues.

Perform Unit Testing

Run the test cases for the model by using the Test Manager. Save the results as an artifact in the 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** pane, expand the model `db_DriverSwRequest`. Expand the **Test Cases** folder and double-click the test file `db_DriverSwRequest_Tests.mldatx`.

- 2 In the Test Manager, click **Run**.
- 3 To use the test results in the Model Testing Dashboard, export the test results and save the file in the project. On the **Tests** tab, in the **Results** section, click **Export**. Name the results file `Results1.mdatx` and save the file under the project root folder.



The Model Testing Dashboard detects that you exported the results and automatically updates the **Artifacts** panel to reflect the new results. The widgets in the **Test Result Analysis** section are highlighted in grey to indicate that they are showing stale data. To update the data in the dashboard widgets, click **Collect Results**.

Address Testing Failures and Gaps

In the model `db_DriverSwRequest`, the **Model Test Status** section indicates that one test failed and one test was disabled during the latest test run. Open the tests and fix these issues.

- 1 To view the disabled test, in the dashboard, click the **Disabled** widget. The table shows the disabled test cases for the model.
- 2 Open the disabled test in the Test Manager. In the table, click the test `Decrement button hold`.
- 3 Enable the test. In the **Test Browser**, right-click the test case and click **Enabled**. Save the test suite file.
- 4 To view the failed test, in the dashboard, click the **Failed** widget.
- 5 Open the failed test in the Test Manager. In the table, click the test `Cancel button`.
- 6 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, 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 bar in the widget to see what percent of condition coverage was achieved. For this example, 86.4% of decisions were covered by the tests and 4.55% of the decisions were justified in a coverage filter.

- 1 View the decision coverage details. Click the **Decision** bar.
- 2 In the table, expand the model artifact. The table shows the test case results for the model and the results file that contains them. Open the results file `Results1.mldatx` in the Test Manager.
- 3 To see detailed coverage results, open the model in the Coverage perspective. In the Test Manager, in the **Aggregated Coverage Results** section, in the **Analyzed Model** column, click `db_DriverSwRequest`.
- 4 Coverage highlighting on the model shows the points that were not covered by the test cases. For a point that is not covered, add a test that covers it. Find the requirement that is implemented by the model element or, if there is none, add a requirement for it. Link the new test case to the requirement. If the point should not be covered, justify the missing coverage by using a filter. For this example, do not fix the missing coverage.

Once you have updated the unit tests to address failures and gaps, 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, check if there are downstream testing impacts by updating the tracing data and metric results in the dashboard. Use the 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 4-2
- “Track Changes to Requirement Links” on page 4-3
- “Compare Requirements Sets” on page 4-8
- “Compare Link Sets” on page 4-9
- “Report Requirements Information” on page 4-10
- “Three-way AutoMerge Solution for Requirement Set and Link Set” on page 4-13
- “Merge Requirement Set and Link Set Files” on page 4-15

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 “What Are 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 requirements 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.slreqx	•	✓	
tests	•	✓	
baseline_test.mldatx	•	✓	Test
baseline_test.slmx	•	✓	Test
baseline_test_data.xlsx	•	✓	Test
baseline_test_result_criteria.xlsx	•	✓	Test
simulation_tests.mldatx	•	✓	Test
simulation_tests.slmx	•	✓	Test
test_utilityfn.m	•	✓	Design

If your project includes shared library models, requirements 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 `slreq.refreshLinkDependencies` command.

Track Changes to Requirement Links

After you “Author Requirements in Simulink” on page 1-2 and create links between design elements and your requirements, Simulink Requirements tracks the links and detects when linked requirements change. Track change information from the Requirements Editor or in the Traceability Matrix. You can 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, at the MATLAB command prompt, enter:

```
slreq.editor
```

- 2 Open a requirement set.

- 3 Ensure that  **Information** > **Change Information** is selected.


When you enable **Change Information**, this setting stays enabled even after you close the Requirements Editor.

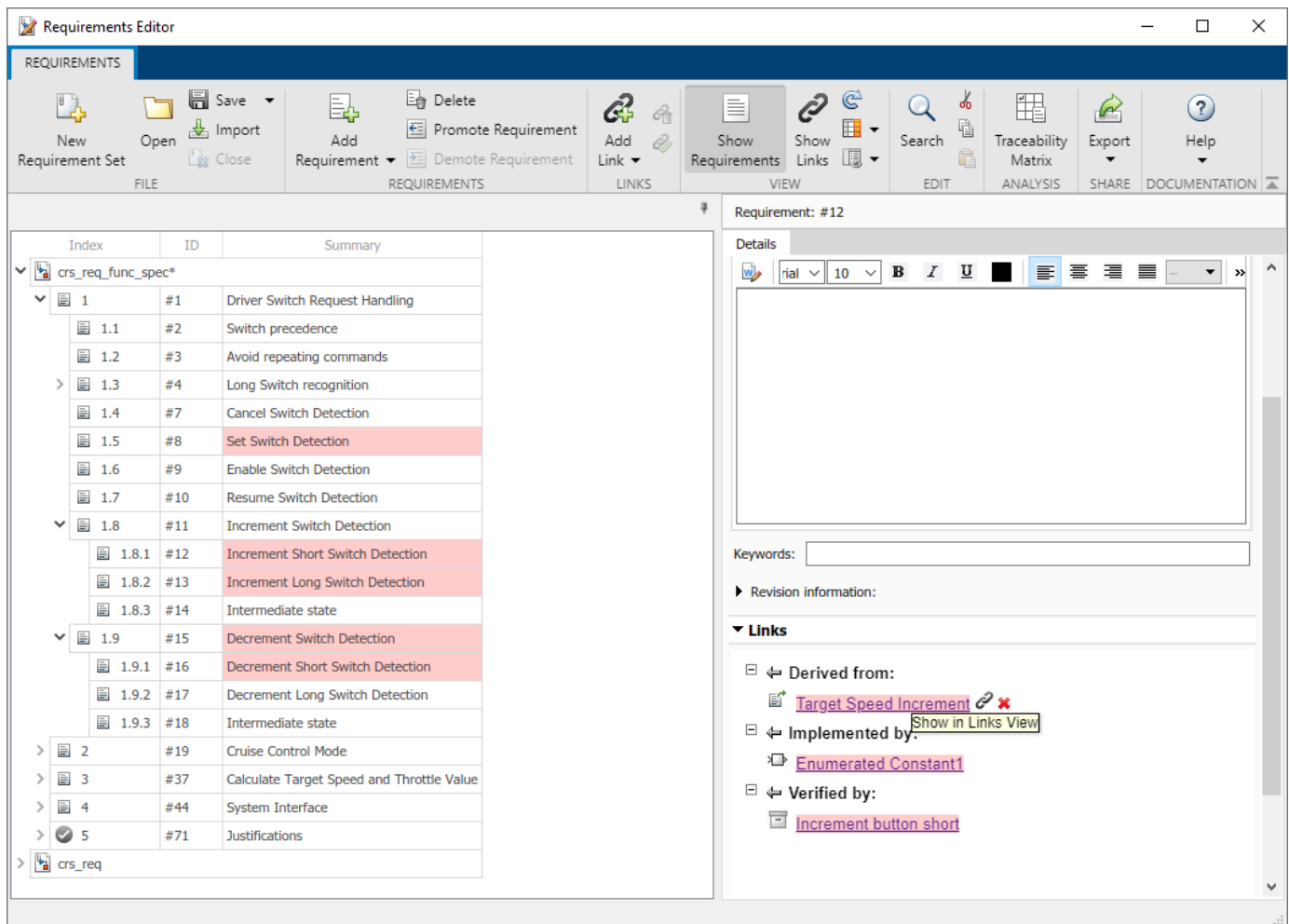
Alternatively, you can enable change tracking for requirement links from the Requirements Perspective. Right-click an item in the Requirements Perspective and select **Change Information**.

Review Changes to Requirements

Requirements can be linked to other types of items. For a full list of linkable items, see “Linkable Items” on page 2-32. When you change a requirement that is linked to another item, the link is highlighted in the Requirements Editor and Traceability Matrix to indicate that it has a change issue. After you “Enable Change Tracking for Requirement Links” on page 4-3, you can view the change issues associated with a particular requirement from the Requirements Editor or the Traceability Matrix.

Note Simulink Requirements only provides change tracking information for unresolved links if the linked requirement is valid. For more information on why a link might become unresolved, see “Resolve Links” on page 2-36.

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 **Details** 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.



In the Traceability Matrix, click **Highlight Missing Links > Highlight Changed Links** to highlight in red the row, column, and cell associated with the linked requirement that was changed. To view changes to the linked requirement in the Requirements Editor, 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 to find change issues, see “View and Clear Change Issues for Links” on page 2-15.

The screenshot shows the Requirements Editor interface. At the top, there is a 'HOME' button and a help icon. Below that, a tab is labeled 'Simulink Requirements vs Simulink Model'. A 'FILTER PANEL' on the left lists 'crs_controller' and 'crs_req_func_spec'. The main area displays a traceability matrix with columns for 'crs_controller', 'DriverSwRequest', 'decrement', 'Enumerated Constan', 'Enumerated Constant2', and 'Switch2'. The rows are 'crs_req_func_spec', 'Driver Switch Request Handling', 'Long Switch recognition', and 'Set Switch Detection'. A tooltip is open over the 'Set Switch Detection' cell, showing the following information:

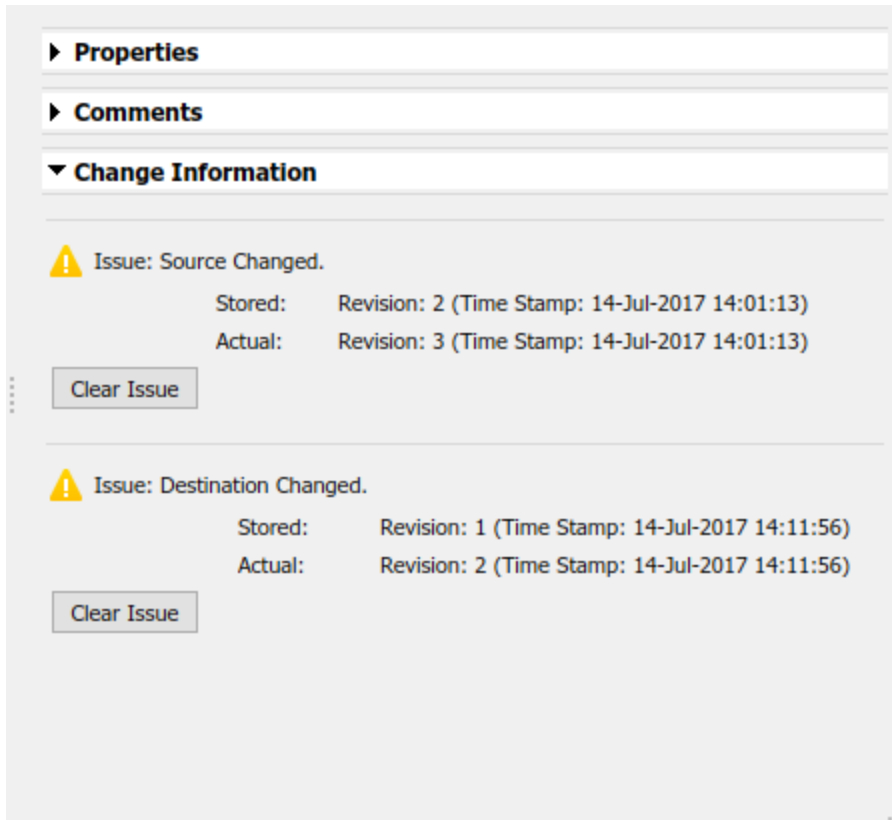
Source	Switch2
Destination	Set Switch Detection
Link	#8: Set Switch Detection (Implement)

Resolve Change Issues

The Requirements Editor displays change information, including change issues, for each link. Click **Show Links** and, in the **Details** pane, expand **Change Information**. Simulink Requirements compares the stored timestamp and revision to the current timestamp and revision for the linked requirement. If you change 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 **Show Links** is selected. To clear a change issue, select the link and, in the **Details** 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 **Details** 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 action that you took. 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 **Details** pane, under **Comments**, click **Add Comment**.

Manually Check for Using Links Change Tracking

Change tracking information is automatically updated in the Requirements Editor, but it can also be manually refreshed. To refresh the change tracking information:

- In the Requirements Editor, click  **Refresh**.
- In the Traceability Matrix, click **Update**.

In the Traceability Matrix, change tracking information must be refreshed manually.

See Also

More About

- “Create a Project from a Model”
- “Track Requirement Links with a Traceability Matrix” on page 2-5

Compare Requirements Sets

To compare differences between two requirements sets, use the “Compare Revisions” tool.

Compare Two .slreqx Simulink Requirements Sets

If you have two versions of a .slreqx Simulink requirements set file, use the Simulink “Compare Revisions” tool to find any differences between the two files.

Select Two Requirements 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 Simulink Requirements Comparison as the **Comparison type**.

The Simulink comparison tool shows the differences between the two .slreqx requirements 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 .slreqx requirements 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 use the Simulink comparison tool with your source-controlled Simulink Requirements files. For more information, see “Compare Revisions”.

Compare Link Sets

If you have two versions of a .slmx Simulink link set file, use the Simulink “Compare Revisions” tool to find any 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 **Simulink Requirements Comparison** as the **Comparison type**.

The Simulink 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

Empty Sections

Group Links By:

Artifact Link Type

Change Information

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 requirements 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 Simulink Requirements 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 a Project to Source Control”.

Select and Merge Branches in Git

To select a branch and merge the changes:

- 1 From within your Git repository folder, select **Branches** from the toolbar.
- 2 From **Branches** drop-down list, select a branch from which you want to merge the changes.
- 3 Click **Merge** to merge from the selected branch.

Note 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

- 1 Git is the only supported source control tool.
- 2 You must resolve merge conflicts manually.

See Also

More About

- “Branch and Merge with Git”
- “Create a New Project From a Folder”

See Also

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 comparisons. `ExternalSCMLink.setupGitConfig` command.

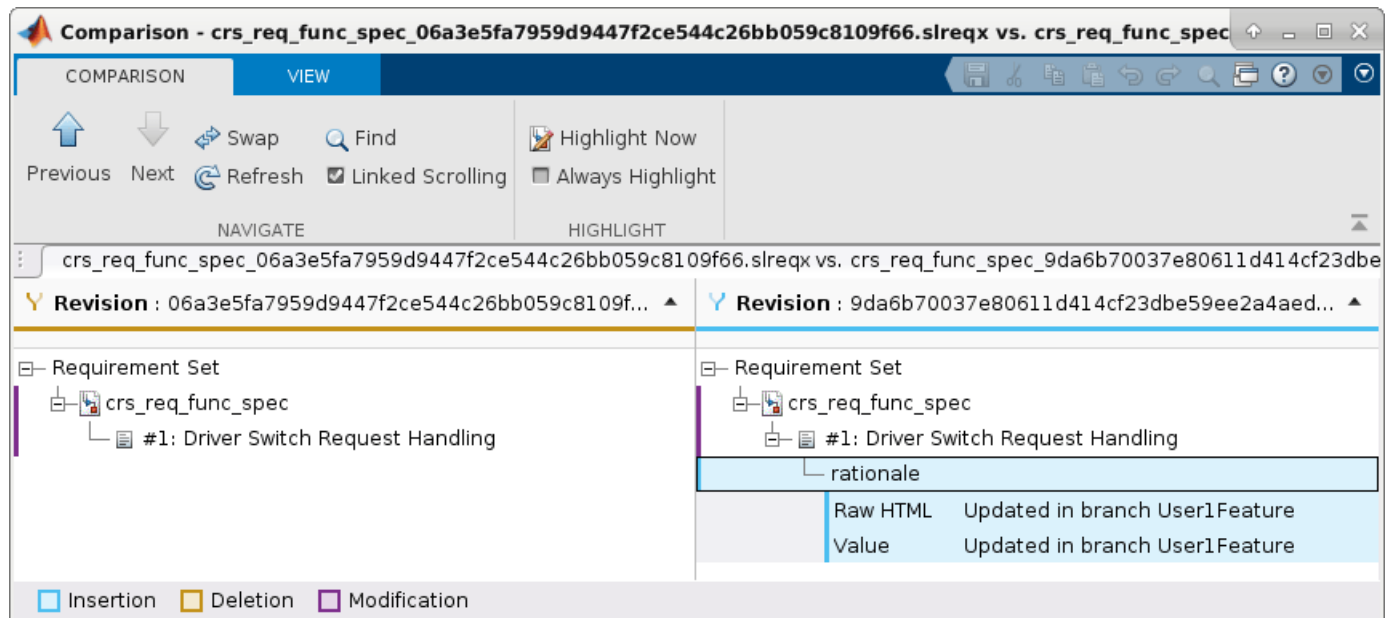
To merge a file without any conflict:

1. At the MATLAB® command prompt, enter:

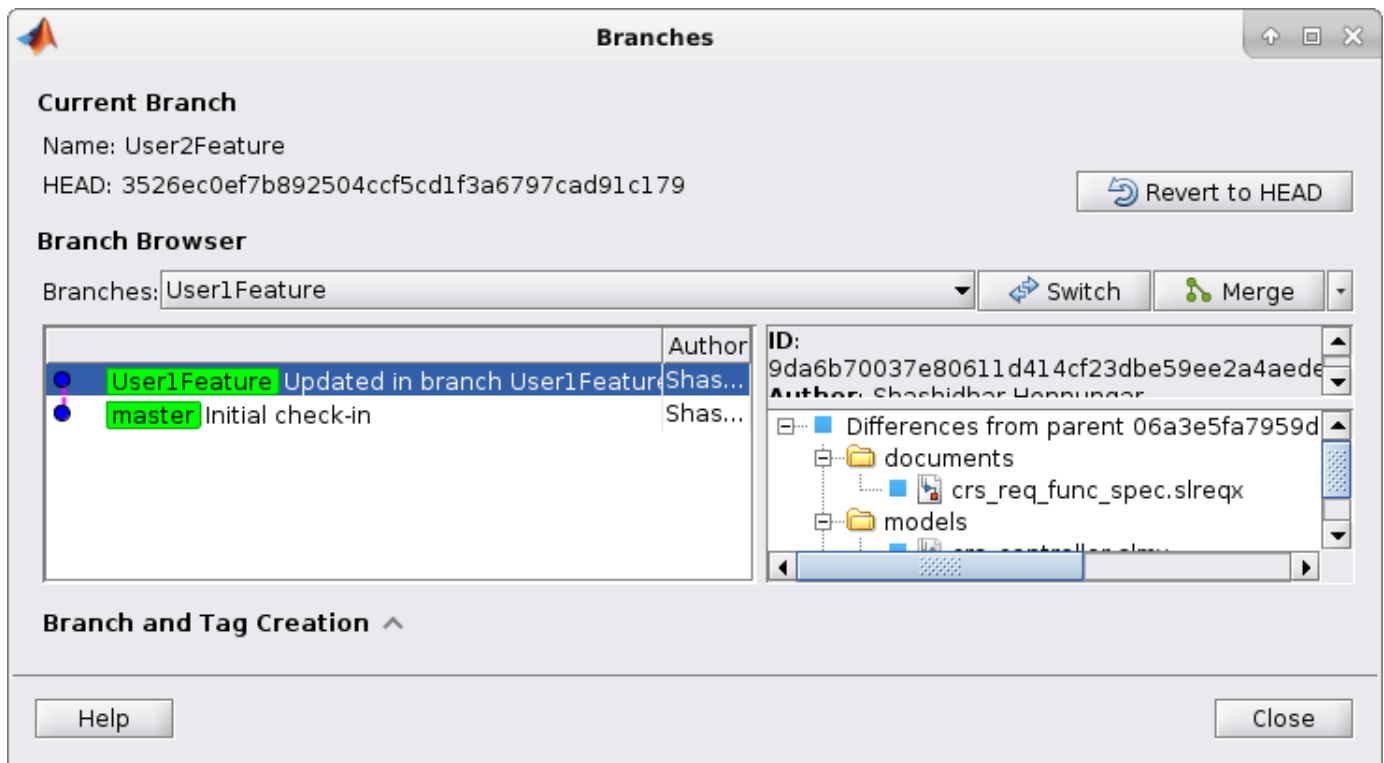
```
slreqCCMergeSetup
```

This helps you to set up two branches, `User1Feature` and `User2Feature`, where `User2Feature` is the current active branch.

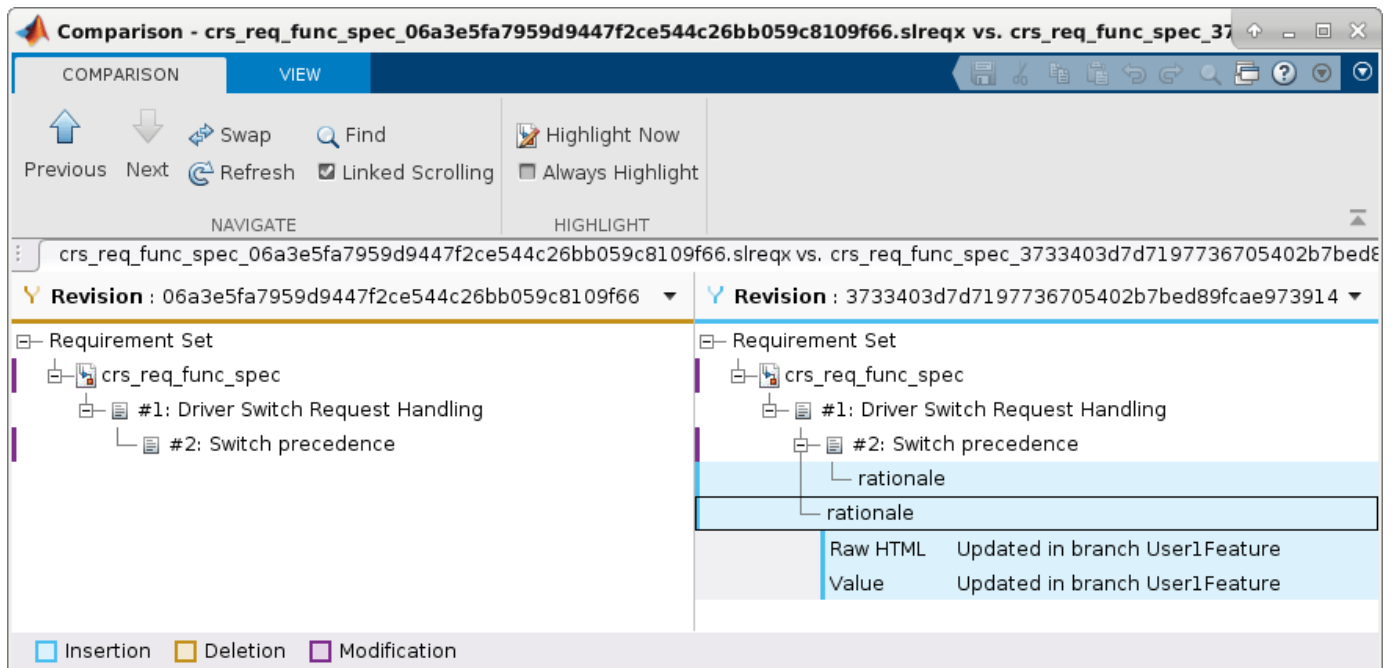
2. To inspect the changes in each branch, switch to that branch and right-click the `crs_req_func_spec.slreqx` file in the current folder browser and select **Source Control > Compare To Revision** and select latest two revisions for comparison.



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 `crs_req_func_spec.slrqx` file in current folder browser and click **Source Control > 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:

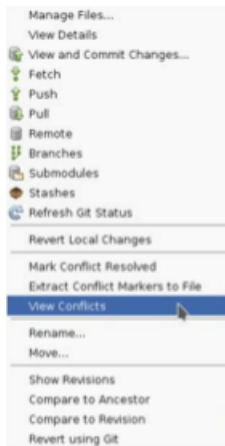
```
slreqCCMergeConflictSetup
```

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 `crs_req_func_spec.slreqx` file in the current folder browser and select **Source Control > Compare To Revision** and select latest two revisions for comparison.

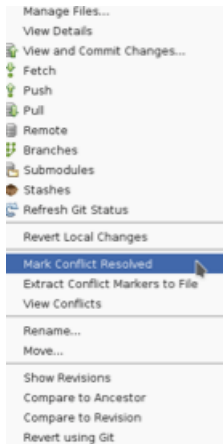
3. Select the **User1Feature** branch and click **Merge** to execute merge command. Observe that MATLAB reports a conflict.

4. The Merge tool automatically merges non-conflicting changes. To view the conflicting changes, right-click the `crs_req_func_spec.slreqx` file in current folder browser and click **Source Control > View Conflicts**.



5. To manually resolve conflicts, open the requirement set in **Requirements Editor** and make the changes.

6. Right-click the `crs_req_func_spec.slreqx` file and select **Source Control > Mark Conflicts Resolved**.



7. Click **Commit** to merge the changes.

8. Right-click the `crs_req_func_spec.slreqx` file and select **Source Control > Compare to Ancestor** to observe the merged changes.

See Also

- “Three-way AutoMerge Solution for Requirement Set and Link Set” on page 4-13

Requirements Management Interface Setup

- “Configure Simulink Requirements for Interaction with Microsoft Office and IBM Rational DOORS” on page 5-2
- “Requirements Link Storage” on page 5-4
- “Supported Requirements Document Types” on page 5-8
- “Requirements Settings” on page 5-10
- “Migrating Requirements Management Interface Data to Simulink® Requirements™” on page 5-16

Configure Simulink Requirements for Interaction with Microsoft Office and IBM Rational DOORS

Simulink Requirements communicates with external tools such as Microsoft Office and 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.

You can configure MATLAB and Simulink to:

- Use ActiveX® controls for navigation from Microsoft Office documents to Simulink models (PC only).
- Use Simulink Requirements with IBM Rational DOORS software (Windows only).
- Use Simulink Requirements with IBM DOORS Next web server.

Configure Simulink Requirements 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 Simulink Requirements 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 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 Simulink Requirements still does not communicate after performing this setup, try the setup process described in “Configure Simulink Requirements for IBM Rational DOORS Software” on page 7-2.

Configure Simulink Requirements for IBM DOORS Next

You do not need to use `rmi setup` for DOORS Next integration. However, if you use `rmi setup` for Microsoft Office or DOORS 9 integration, you can continue with the setup for DOORS Next. The utility prompts you to enter your DOORS Next server address and port number. You can enter the

values at that point, and you can easily adjust those in each future MATLAB session. To read more about configuring MATLAB and Simulink with IBM DOORS Next, see “Link and Trace Requirements with IBM DOORS Next” on page 7-26.

Install the Simulink Requirements Widget in IBM DOORS Next

The **Simulink Requirements** widget enables you to propagate selection information from IBM DOORS Next.

- 1 In the Windows File Explorer, navigate to the folder `toolbox\slrequirements\slrequirements\resources` in your MATLAB installation.
- 2 Copy the `dngsllink_config` 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 then restart the server. For details on this process, see the IBM RM Extensions Hosting Guide
- 4 After copying the `dngsllink_config` folder to your server, add the **Simulink Requirements** 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\dngsllink_config` 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/dngsllink_config/dngsllink_config.xml`.

- 6 Click **Add Widget**. The **Mini Dashboard** now displays the Simulink Requirements widget.

See Also

`slreq.dngConfigure`

More About

- “Configure Simulink Requirements for IBM Rational DOORS Software” on page 7-2

Requirements Link Storage

When you create a link from a Model-Based Design item to a requirement, Simulink Requirements stores the link information in an external `.slmx` file with the same base file name and in the same folder as the artifact that contains the link source.

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.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.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.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 10-8.

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 10-6 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 5-4.
- **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 5-10.
- **Filters** tab to set the options for filtering requirements in a model. For filtering information, see “Configure Requirements Filtering” on page 5-15.
- **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 11-19.

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 11-35.
Apply this keyword to new links	Enter text to attach to the links you create. For more information about user tags, see “Filter Requirements with User Tags” on page 5-11.
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 Tags

- “User Tags and Requirements Filtering” on page 5-11
- “Apply a User Tag to a Requirement” on page 5-11
- “Filter, Highlight, and Report with User Tags” on page 5-12
- “Apply User Tags During Selection-Based Linking” on page 5-14
- “Configure Requirements Filtering” on page 5-15

User Tags and Requirements Filtering

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

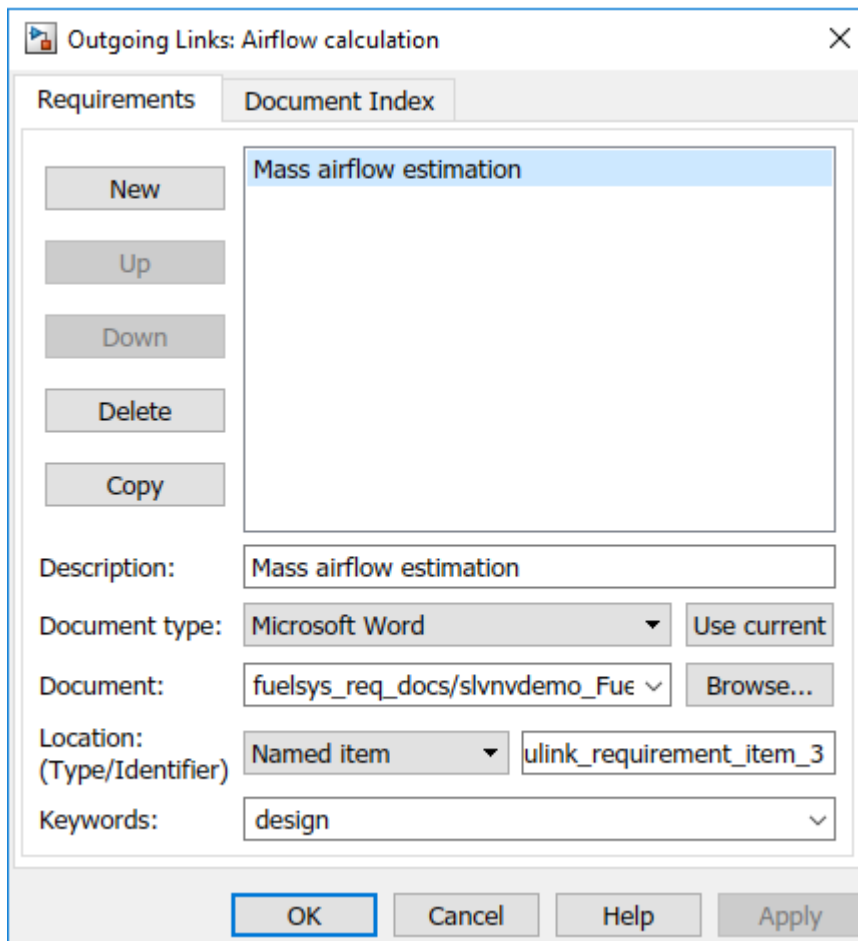
- Highlight or report only those requirements that have a specific user tag.
- Highlight or report only those requirements that have one of several user tags.
- Do not highlight and report requirements that have a specific user tag.

Apply a User Tag to a Requirement

To apply one or more user tags 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 tag `test` to specify a second user tag for this requirement.

User tags:

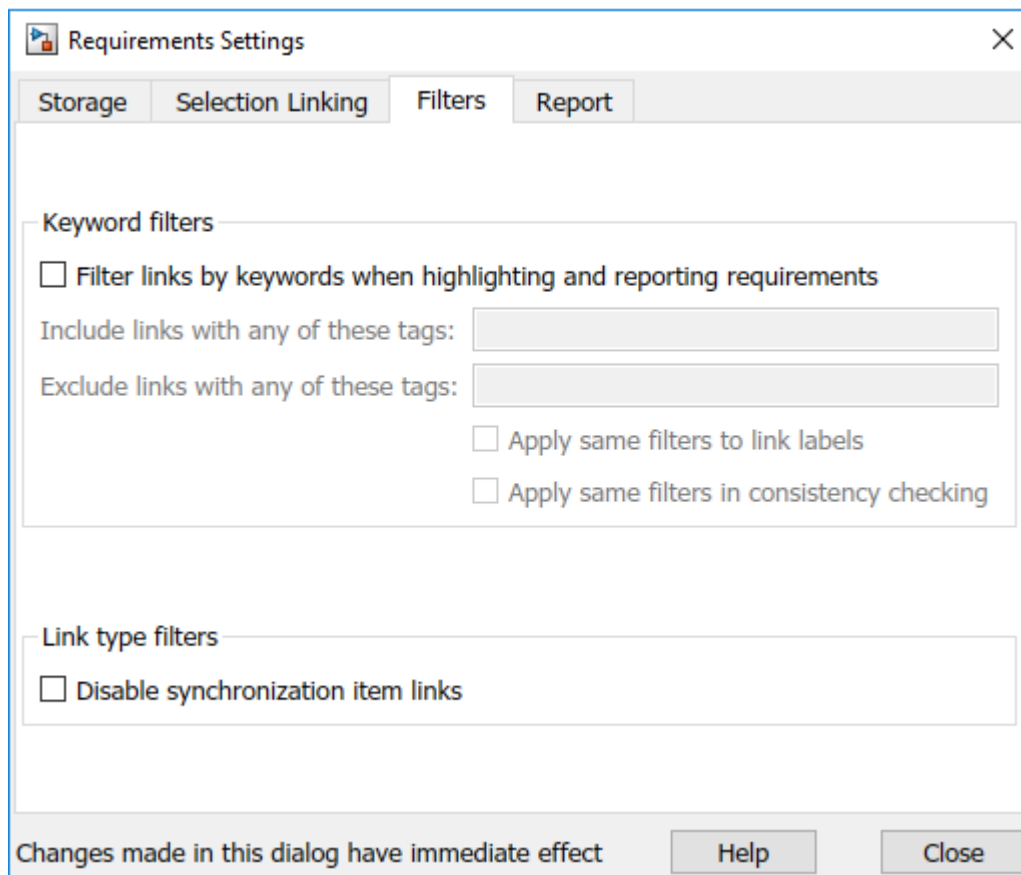
- Are not case sensitive.
- Can consist of multiple words. For example, if you enter `design requirement`, the entire phrase constitutes the user tag. Separate user tags with commas.

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

Filter, Highlight, and Report with User Tags

The `slvnvdemo_fuelsys_officereq` model includes several requirements with the user tag `design`. This section describes how to highlight only those model objects that have the user tag, `test`.

- 1 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 tags, click the **Filter links by user tags when highlighting and reporting requirements** option.
- 5 To include only those requirements that have the user tag, `test`, enter `test` in the **Include links with any of these tags** field.
- 6 Click **Close**.
- 7 In the **Requirements** tab, click **Highlight Links**.

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

- 8 Reopen the Requirements Settings dialog box to the **Filters** tab.
- 9 In the **Include links with any of these tags** field, delete `test`. In the **Exclude links with any of these tags** field, add `test`.

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

- 10 In the **Requirements** tab, select **Share > Generate Model Traceability Report**.

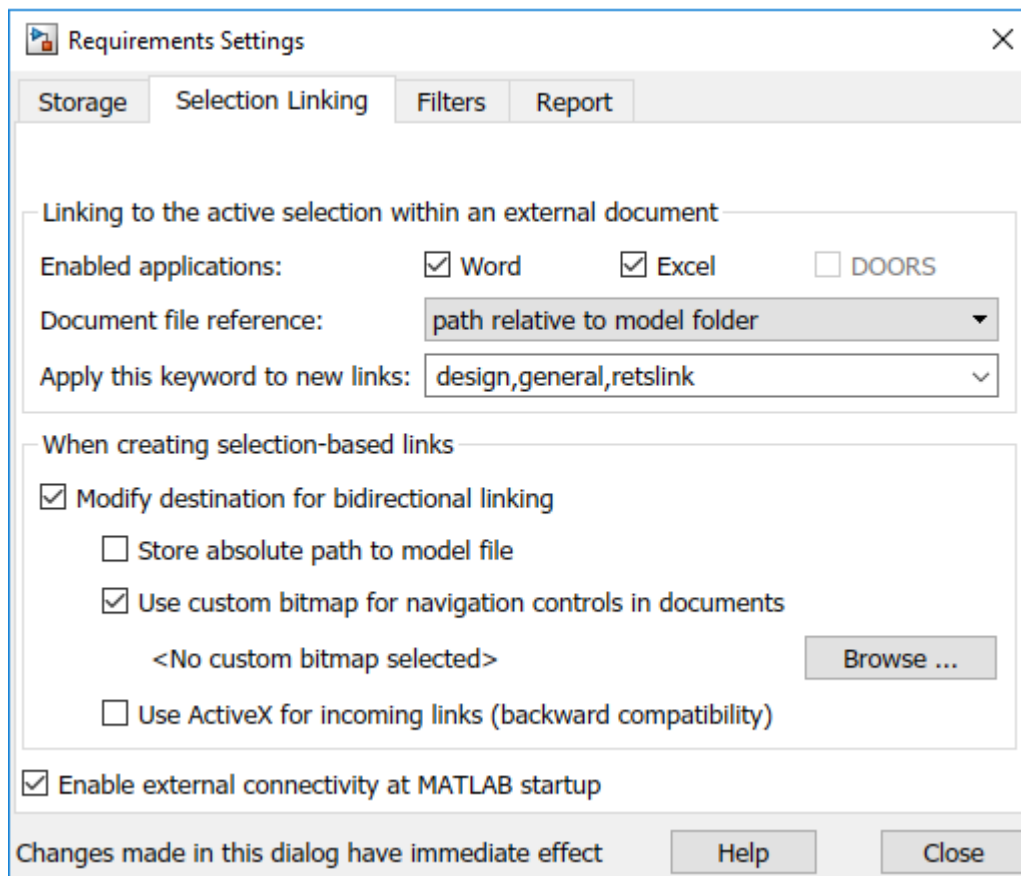
The report does not include information about objects whose requirements have the `test` user tag.

Apply User Tags During Selection-Based Linking

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

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

- 1 In the **Requirements Viewer** tab, click **Link Settings**.
- 2 Select the **Selection Linking** tab.

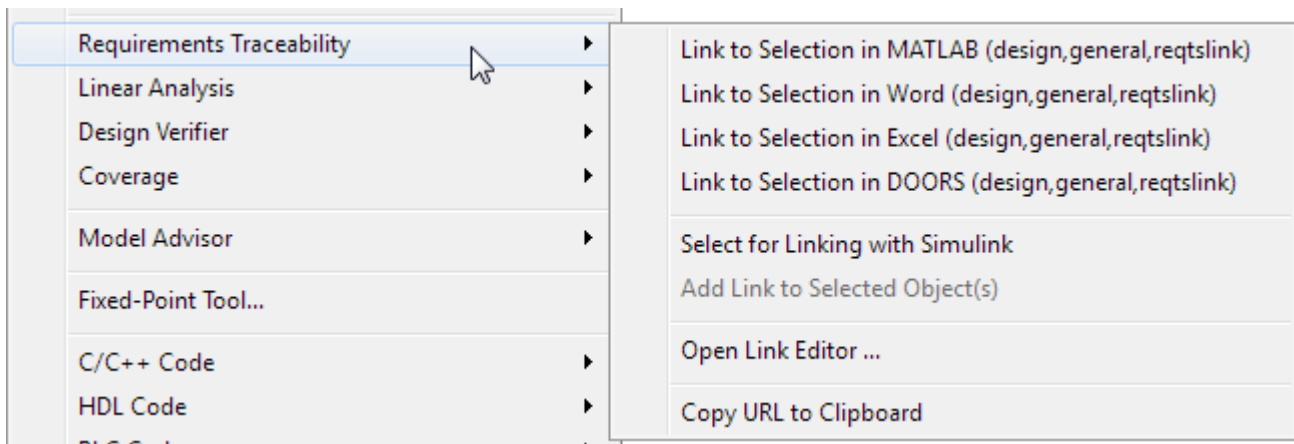


- 3 In the **Apply this keyword to new links** field, enter one or more user tags, separated by commas.

The RMI applies these user tags 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 tags the RMI applies to the link that you create. In the following example, you can apply the user tags `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 tags.
Include links with any of these tags	Includes information about requirements that have the specified user tags. Separate multiple user tags with commas.
Exclude links with any of these tags	Excludes information about requirements that have the specified user tags. Separate multiple user tags 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 tag.
Apply same filters in consistency checking	Includes or excludes requirements with specified user tags 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 tag filters.

Migrating Requirements Management Interface Data to Simulink® Requirements™

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 Simulink Requirements, 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 toolstrip, 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 Requirements Management Interface Setup

The screenshot displays the Requirements Editor application window. The interface is divided into several sections:

- Toolbar:** Located at the top, it includes buttons for 'New Requirement Set', 'Open', 'Save', 'Import', 'Close', 'Add Requirement', 'Promote Requirement', 'Demote Requirement', 'Delete', 'Add Link', 'Show Requirements', 'Show Links', 'Search', 'Traceability Matrix', 'Export', and 'Help'.
- Requirement Tree:** A hierarchical view on the left showing a tree structure under 'FuelSysRequirements'. The selected item is '1.1 1. Interface Definition'.
- Requirement Details:** A panel on the right showing the details for the selected requirement. It includes:
 - Properties:** Type: Functional, Index: 1.1, Custom ID: 1. Interface Definition, Summary: Interface Definition.
 - Description:** A text area containing the requirement text:

1. Interface Definition

```
graph LR; 1((1 throttle)) --> throttle[throttle];
```
 - Keywords:** A text input field.
 - Revision information:** A section with a 'Show in document' button and an 'Unlock' button.
 - Links:** A section showing 'No links'.

5. You can navigate to the document. In the **Properties** pane, click **Show in document**.

1. Interface Definition

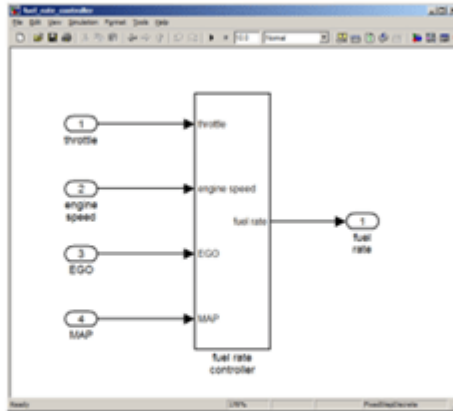


Image 1: High level block diagram of the System

1.1. Input Signals

1.1.1. Throttle Sensor

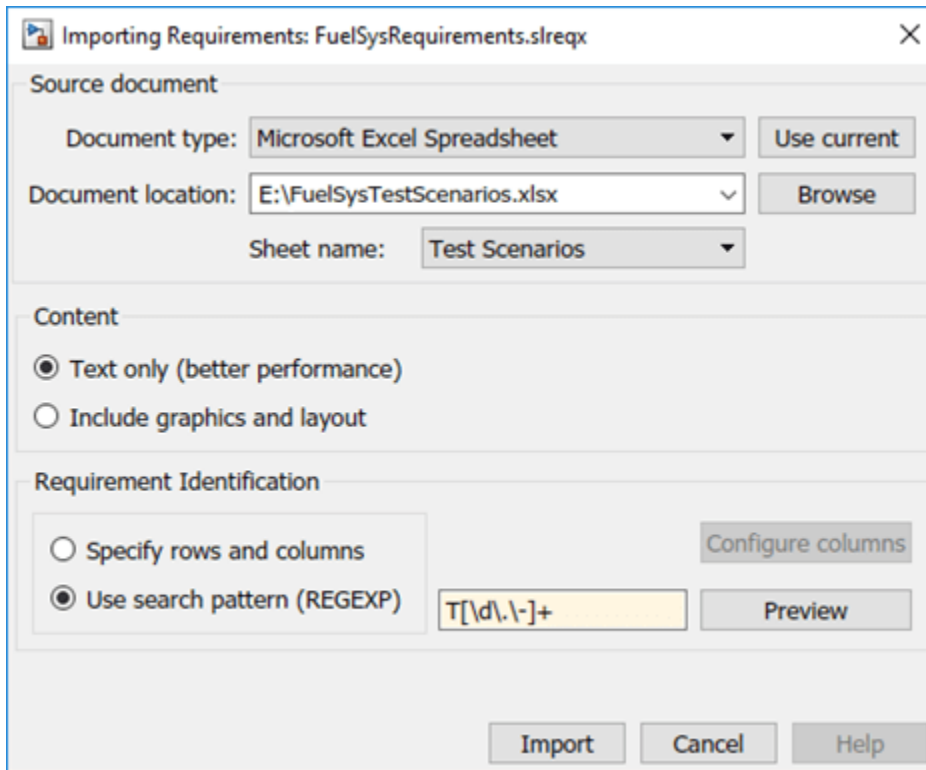
Now, import requirements from a Microsoft® Excel® document. When importing from Excel, you specify which columns to import. You can map the columns to either **Summary**, **Keywords**, or **Custom Attribute** fields in the imported data. You can also locate specific ranges in tables by specifying a regular expression pattern of requirements identifiers.

1. Open the FuelSysTestScenarios.xlsx file from the working folder.

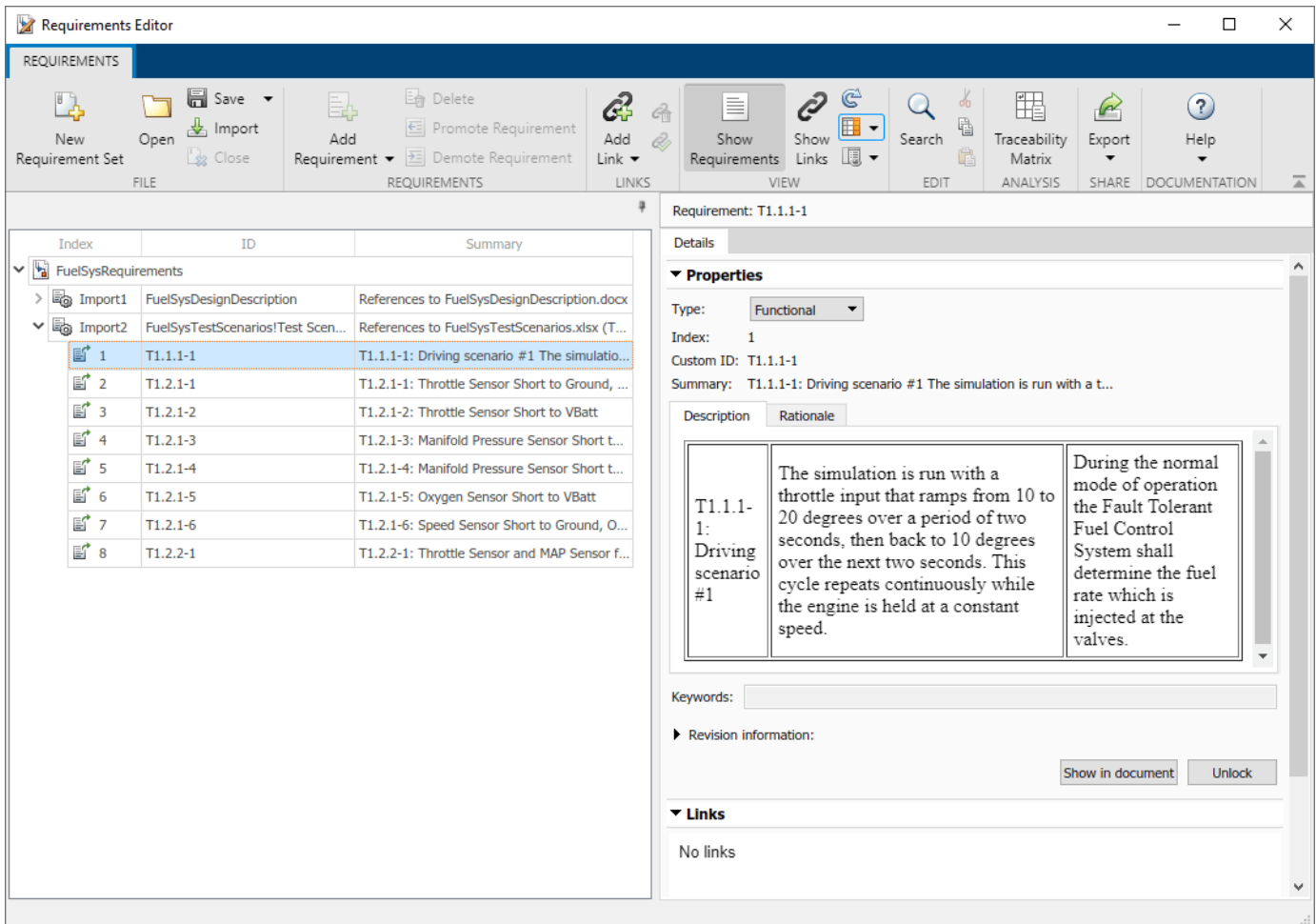
	A	B	C	D	E
1	Test Scenarios for the Fault Tolerant Fuel Control System				
2	<i>This Test Scenario Document provides a more detailed description on how the System is going to be verified against the high level requirements</i>				
3	<i>Typically such a document is a part of a more elaborate Test Plan.</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`.

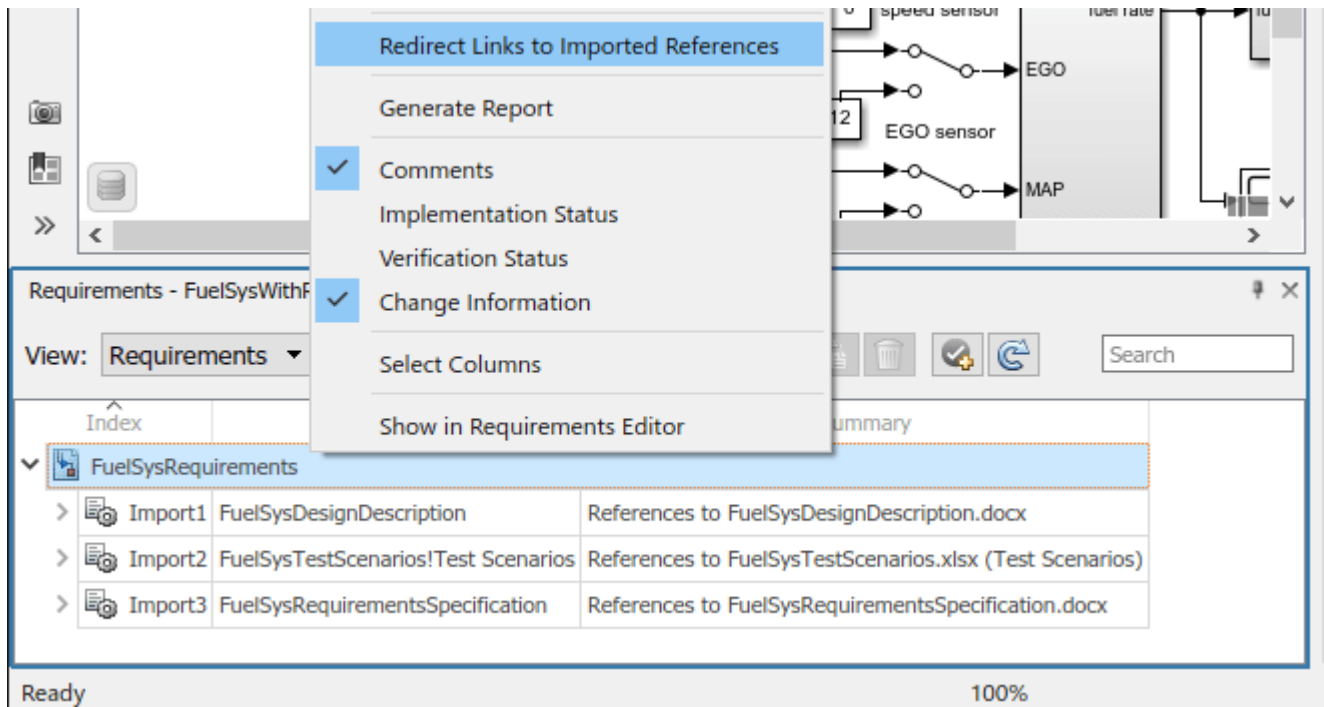
Update Model Link Destinations

Update the model link destinations to the imported requirements. Open the `FuelSysWithReqLinks` model from the working folder.

```
open_system("FuelSysWithReqLinks.slx")
```

2. Enable the Requirements Perspective of 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.

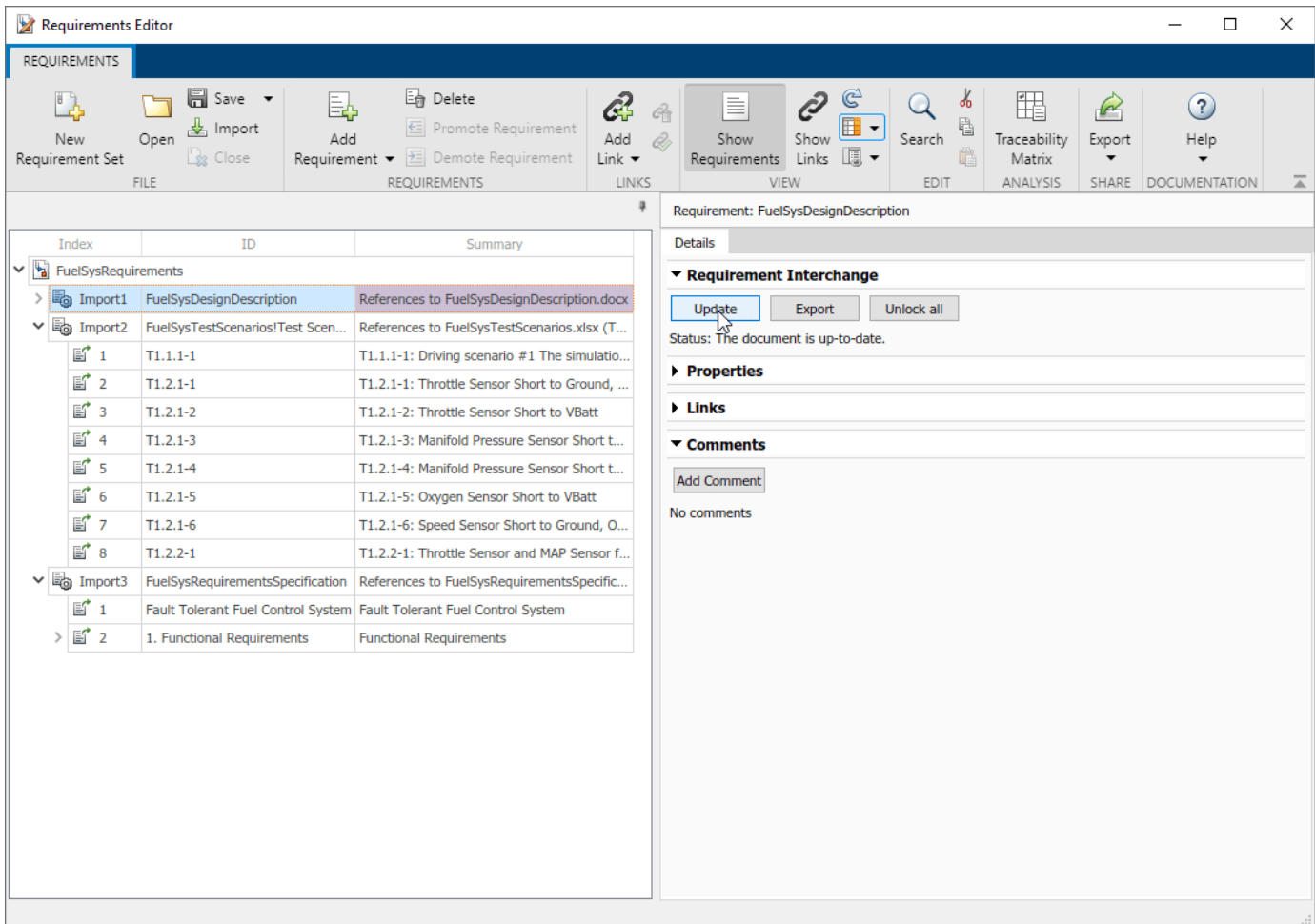
3. Right-click the `FuelSysRequirements` line item and select **Redirect Links to Imported References**.



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:

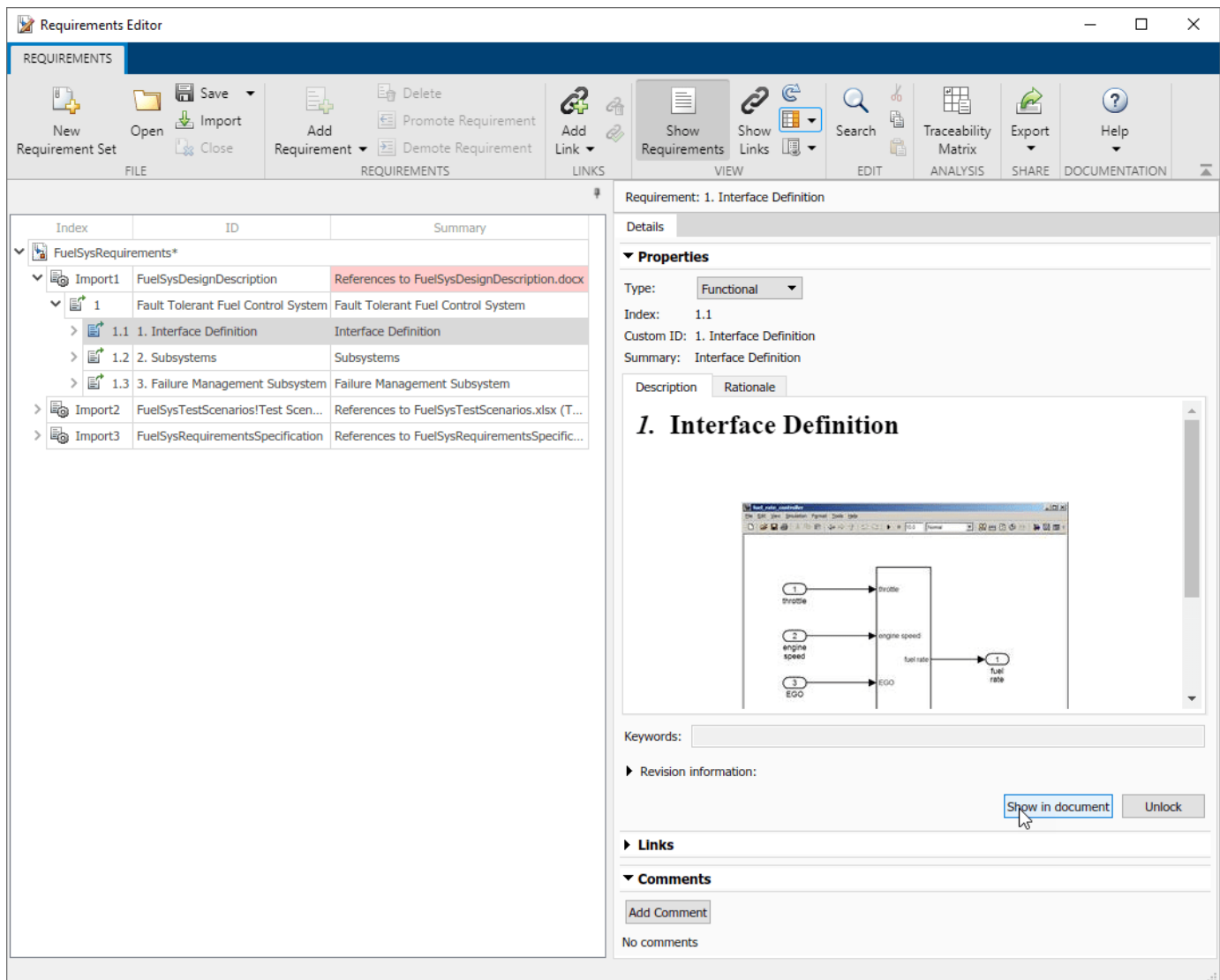
1. Select the top-level Import node in the Requirements Editor.
2. In the right pane, under **Requirement Interchange**, click the **Update** button.



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.



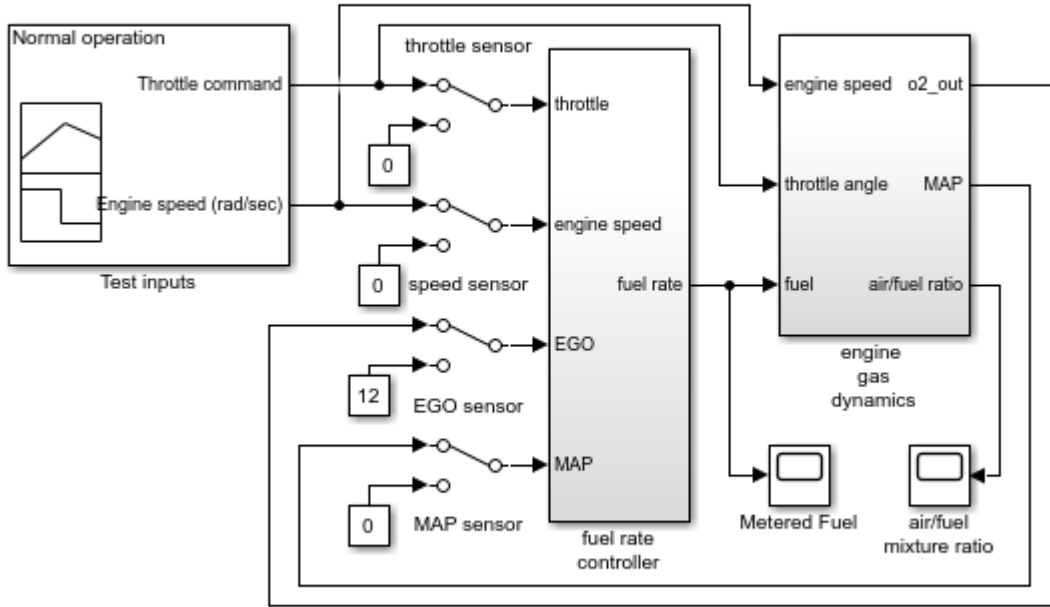
Load a Model with Links to Imported Documents

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 Simulink Requirements license prompts you save the requirements data in the updated Simulink Requirements 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.

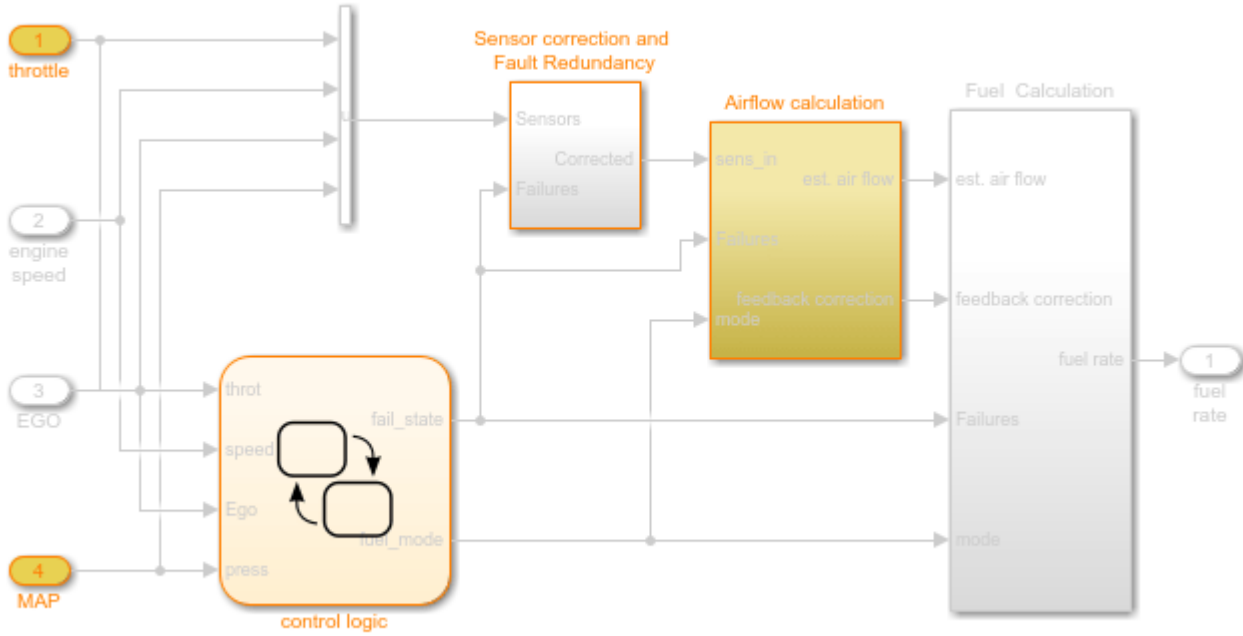
Fault-Tolerant Fuel Control System



Copyright 1997-2010 The MathWorks, Inc.

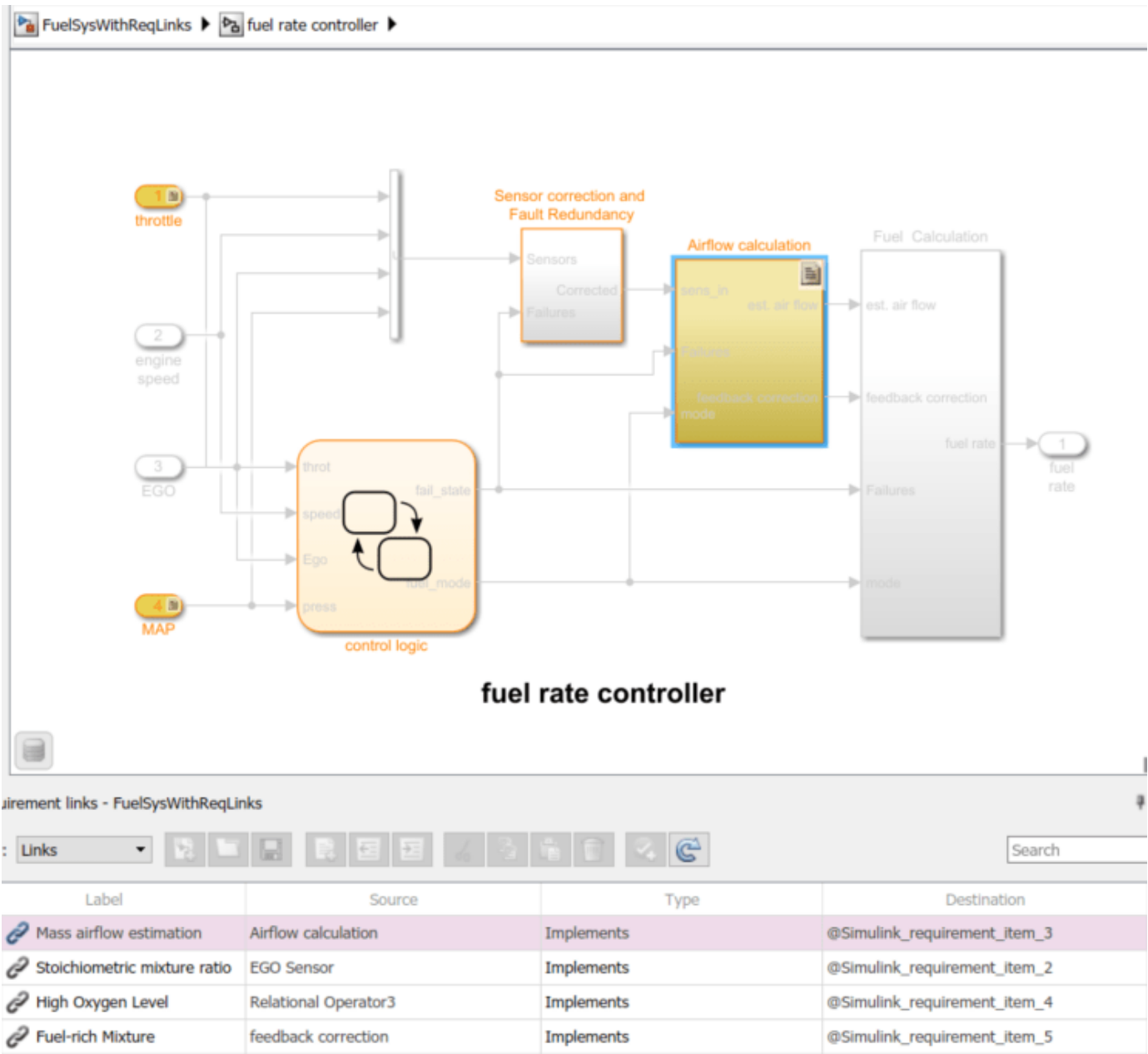
To highlight blocks with requirement links, in the **Requirements Viewer** tab, click the **Highlight Links** button.

FuelSysWithReqLinks ▶ fuel rate controller ▶

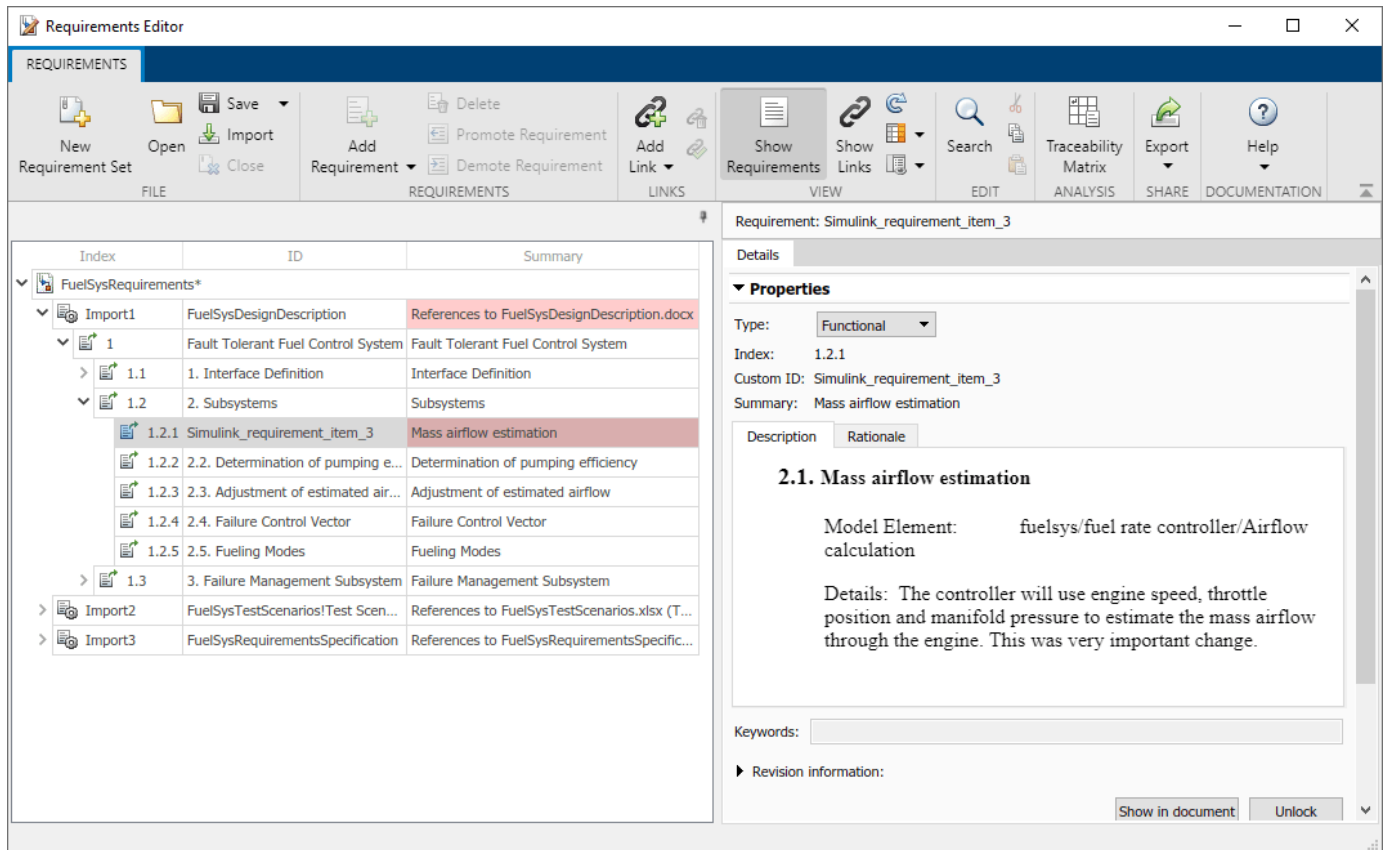


fuel rate controller

To show the linked requirement, open the **Requirements Manager** tab and select a block.



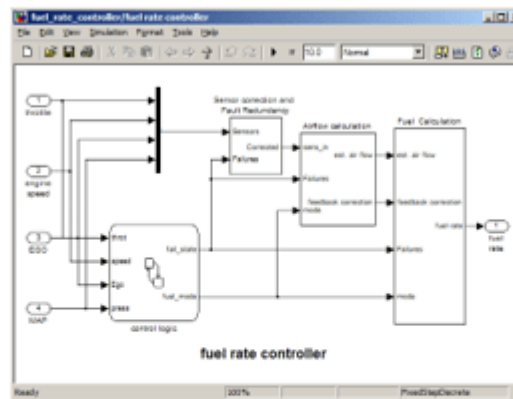
If the Requirements Editor is open, the reference item is highlighted. You can review the contents of the requirement in the **Details** pane.



The incoming link is displayed in the **Links** pane. With Simulink Requirements, 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 Simulink Requirements 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.

2. Subsystems



2.1. Mass airflow estimation

Model Element: fuelsys/fuel rate controller/Airflow calculation

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

Cleanup

Clear the open requirement sets and link sets. Close all open models.

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


Microsoft Office Traceability

- “Link to Requirements in Microsoft Word Documents” on page 6-2
- “Link to Requirements in Excel Workbooks” on page 6-7
- “Navigate to Requirements in Microsoft Office Documents from Simulink” on page 6-10
- “Managing Requirements for Fault-Tolerant Fuel Control System (Microsoft Office)” on page 6-14

Link to Requirements in Microsoft Word Documents

With Simulink Requirements, you can create direct links from linkable items on page 2-32 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 `slvnvdemo_fuelsys_officereq` model.

```
open_system('slvnvdemo_fuelsys_officereq');
```

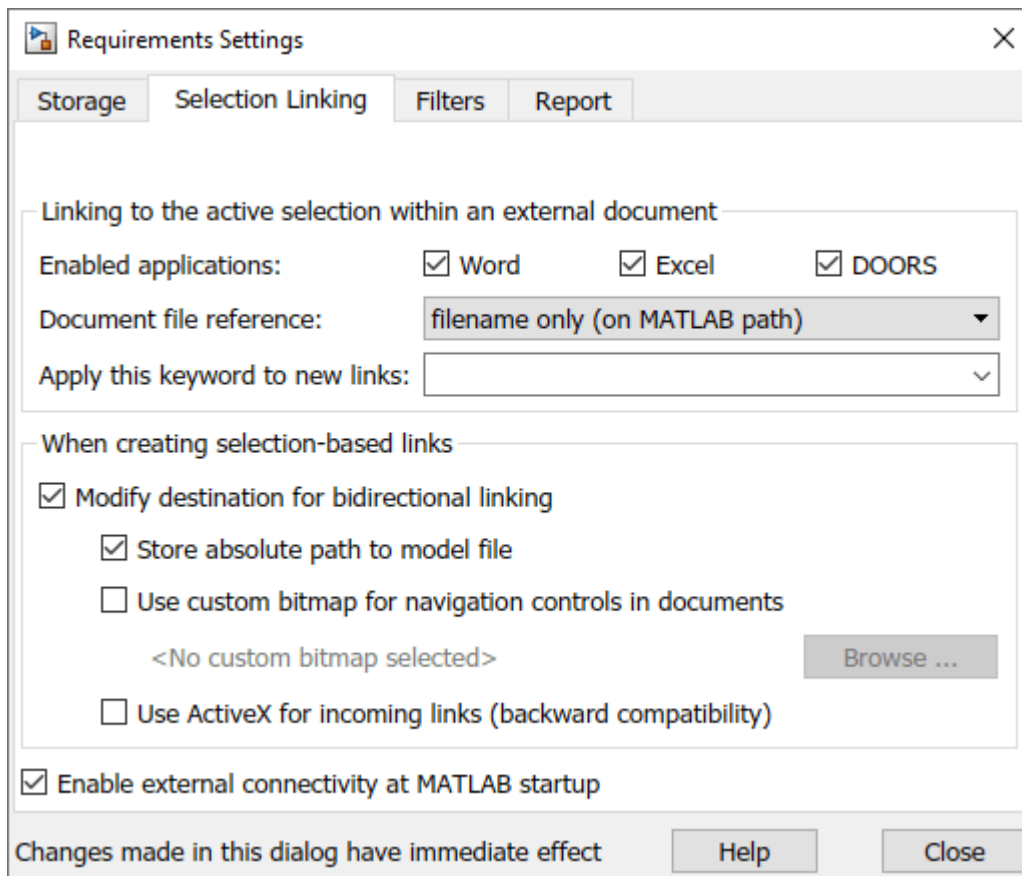
Open the `slvnvdemo_FuelSys_DesignDescription.docx` requirements document from the working directory, or at the MATLAB® command line by entering:

```
open('slvnvdemo_FuelSys_DesignDescription.docx')
```

Configure Selection Link Settings

First, ensure that Simulink Requirements 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.

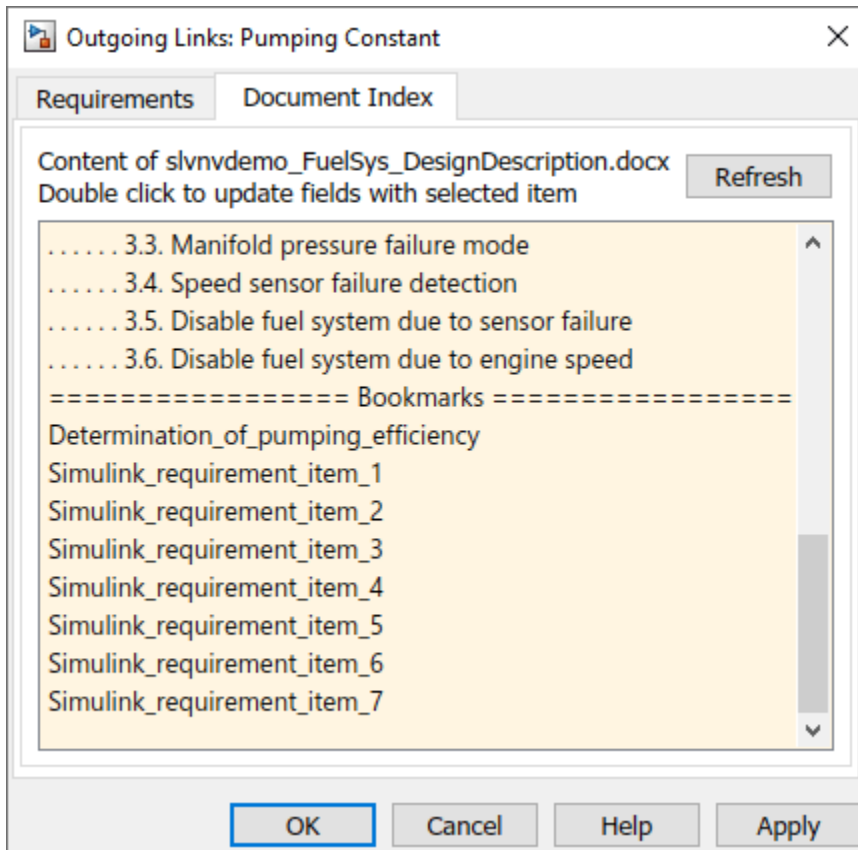
Create a Link to a Bookmark in a Microsoft Word Requirements Document

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

- “Requirement Links” on page 2-32
- “Link to Requirements in Excel Workbooks” on page 6-7
- “Import Requirements from Microsoft Office Documents” on page 1-11
- “Import and Update Requirements from a Microsoft Word Document” on page 1-54

Link to Requirements in Excel Workbooks

With Simulink Requirements, you can create direct links from linkable items on page 2-32 such as Simulink blocks or test cases to requirements in Microsoft Excel. You can only create direct links to requirements in Microsoft Excel on Windows platforms.

Navigate from a Model Object to Requirements in an Excel Workbook

- 1 Open the example model. At the command line, enter:
`slvndemo_fuelsys_officereq`
- 2 In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Highlight Links** to highlight the model objects with requirements.
- 3 Right-click the Test inputs Signal Builder block and select **Requirements > 1. "Normal mode of operation"**.

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

Keep the example model and workbook open.

For information about creating requirements links in Signal Builder blocks, see "Link Signal Builder Blocks to Requirements and Simulink Model Objects" on page 8-9.

Create Requirements Links to the Workbook

- 1 At the top level of the `slvndemo_fuelsys_officereq` model, right-click the speed sensor block and select **Requirements > Open Outgoing Links dialog**.

The Requirements Traceability Link Editor opens.

- 2 To create a requirements link, click **New**.
- 3 In the **Description** field, enter:

`Speed sensor failure`

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

- 4 When you browse and select a requirements document, the RMI stores the document path as specified by the **Document file reference** option on the Requirements Settings dialog box, **Selection Linking** tab.

For information about which setting to use for your working environment, see "Document Path Storage" on page 11-35.

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

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

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

- 7 Click **Apply** or **OK** to create the link.
- 8 To confirm that you created the link, right-click the speed sensor block and select **Requirements > 1. "Speed sensor failure"**.

The workbook opens, with cells B22:E22 highlighted.

Keep the model and Excel file open.

Link Multiple Model Objects to a Microsoft Excel Workbook

You can use the same technique to link multiple model objects to a requirement in a Excel workbook. Follow this workflow:

- 1 In the model window, select the objects to link to a requirement.
- 2 Right-click one of the selected objects and select **Requirements > Open Outgoing Links dialog**.
- 3 When you browse and select a requirements document, the RMI stores the document path as specified by the **Document file reference** option on the Requirements Settings dialog box, **Selection Linking** tab.

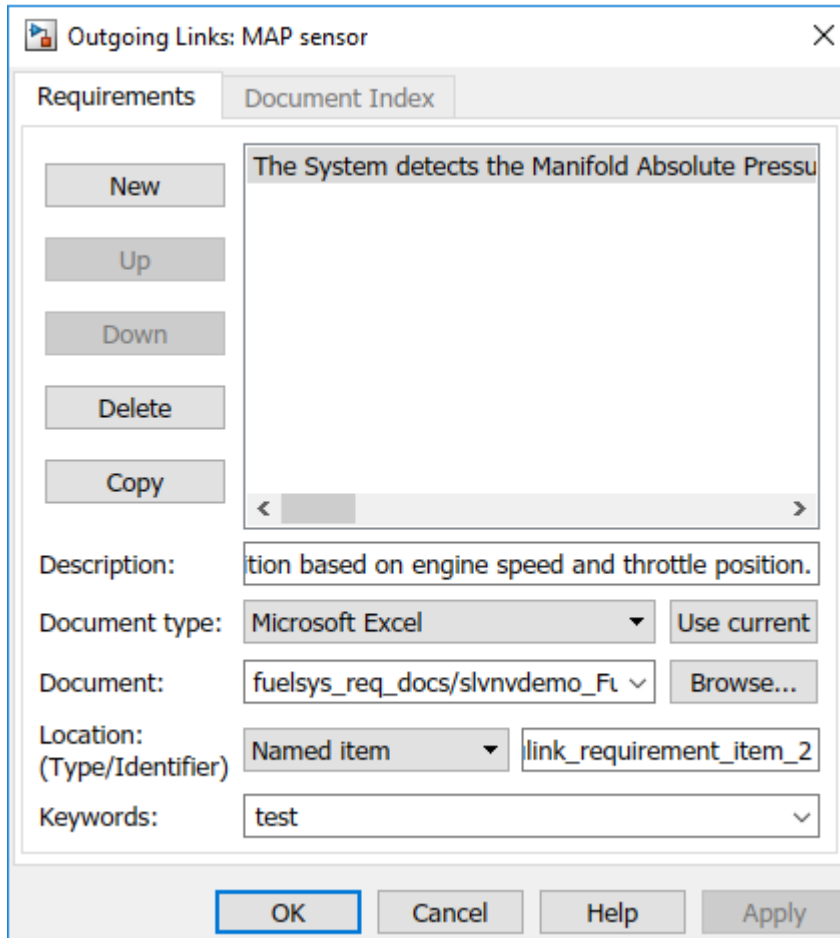
For information about which setting to use for your working environment, see "Document Path Storage" on page 11-35.

- 4 Use the Link Editor to specify information about the Excel requirements document, the requirement, and the link.
- 5 Click **Apply** or **OK** to create the link.

Change Requirements Links

- 1 In the slvnvdemo_fuelsys_officereq model, right-click the MAP sensor block and select **Requirements > Open Outgoing Links dialog**.

The Requirements Traceability Link Editor opens displaying the information about the requirements link.



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

MAP sensor test scenario

The **Keyword** field contains the tag `test`. User tags filter requirements for highlighting and reporting.

Note For more information about keywords, see “Filter Requirements with User Tags” on page 5-11.

- 3 Click **Apply** or **OK** to save the change.

Keep the example model open.

Navigate to Requirements in Microsoft Office Documents from Simulink

With Simulink Requirements, 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 6-14.

```
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 5-10.

- 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 Tags and Requirements Filtering” on page 5-11.

- 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 6-10, 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 6-10.
- 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 6-11, 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 6-11.

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 7-50 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 6-0 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

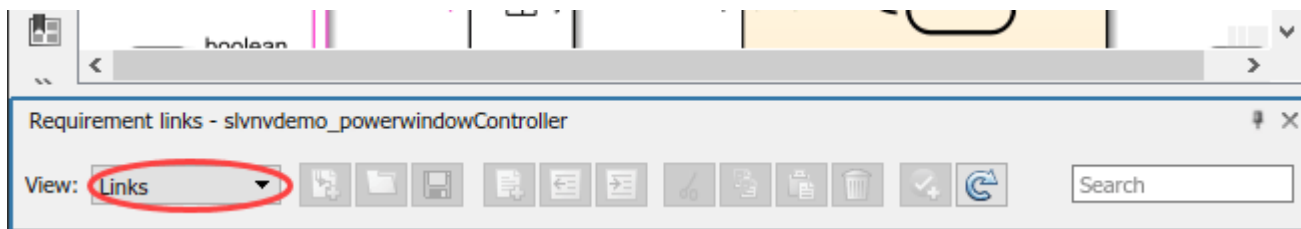
Requirements management features are demonstrated using an example model of a fault-tolerant fuel control system. You can open this model by evaluating the following code.

```
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.

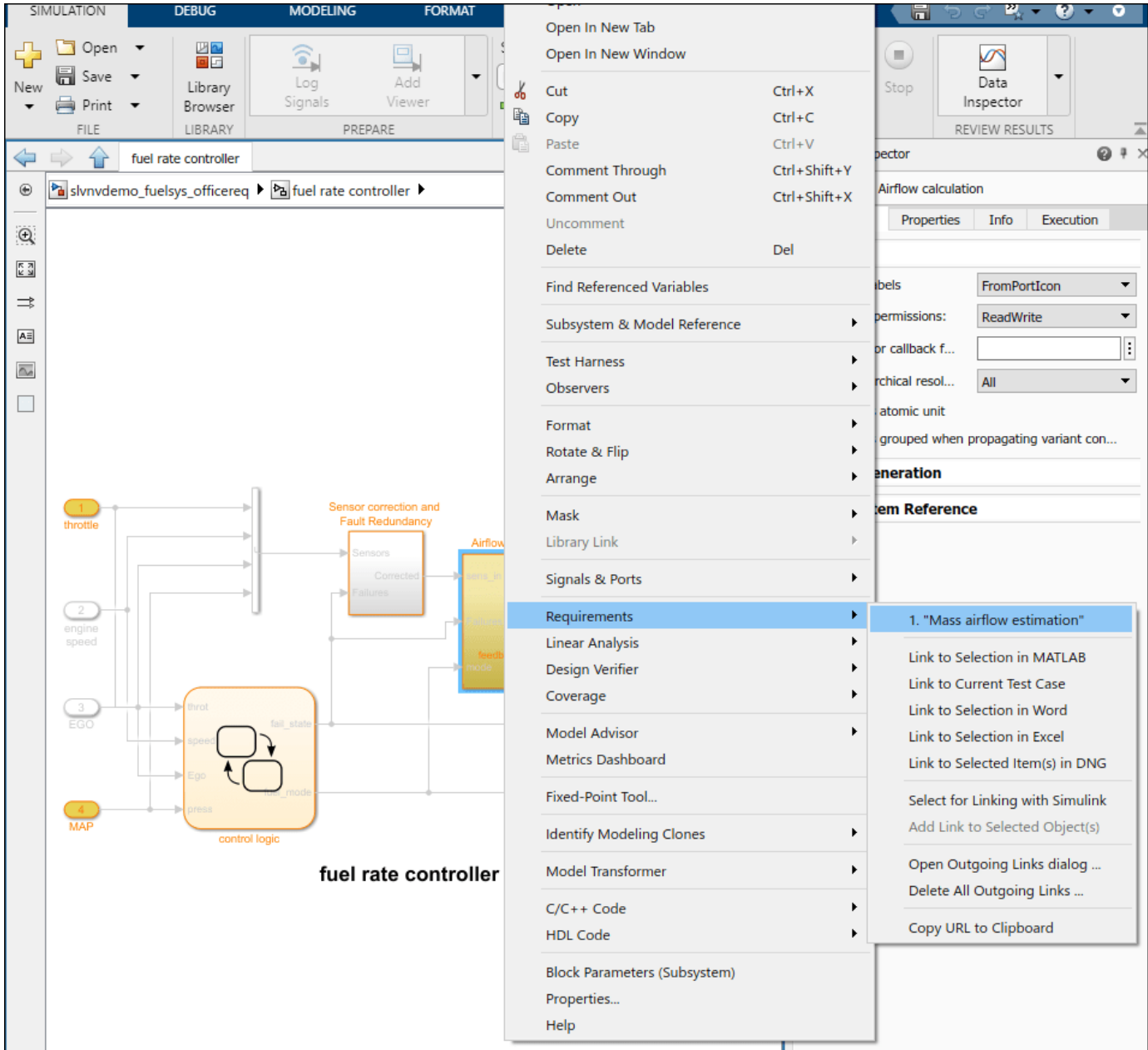
```
rmi('highlightModel', 'slvndemo_fuelsys_officereq');
```

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.

```
rmidemo_callback('locate','slvnvdemo_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:

```
rmidemo_callback('view','slvnvdemo_fuelsys_officereq/fuel rate controller/control logic:26',1)
```

Navigate from Requirements Document to Model Objects

In the slvnvdemo_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.

The screenshot shows a Microsoft Word document with the following content:

2.5. Fueling Modes


The controller shall provide access to the fueling mode indicator that will identify when the controller is using operating in a stoichiometric mode, a programmed enrichment mode, or the fuel disabled mode.

3. Failure Management Subsystem

3.1. Oxygen sensor disabled during warmup


Model Element: State: Oxygen_Sensor_mode.O2_warmup
Trans: [t > o2_t_thresh]

Details: During a calibratable warm up period the oxygen sensor correction will be disabled.

3.2. Enriched mixture usage 

Model Element: State: Fueling_Mode_Running_Rich_Mixture

Details: The fuel system will use an enriched mixture whenever a sensor has failed.

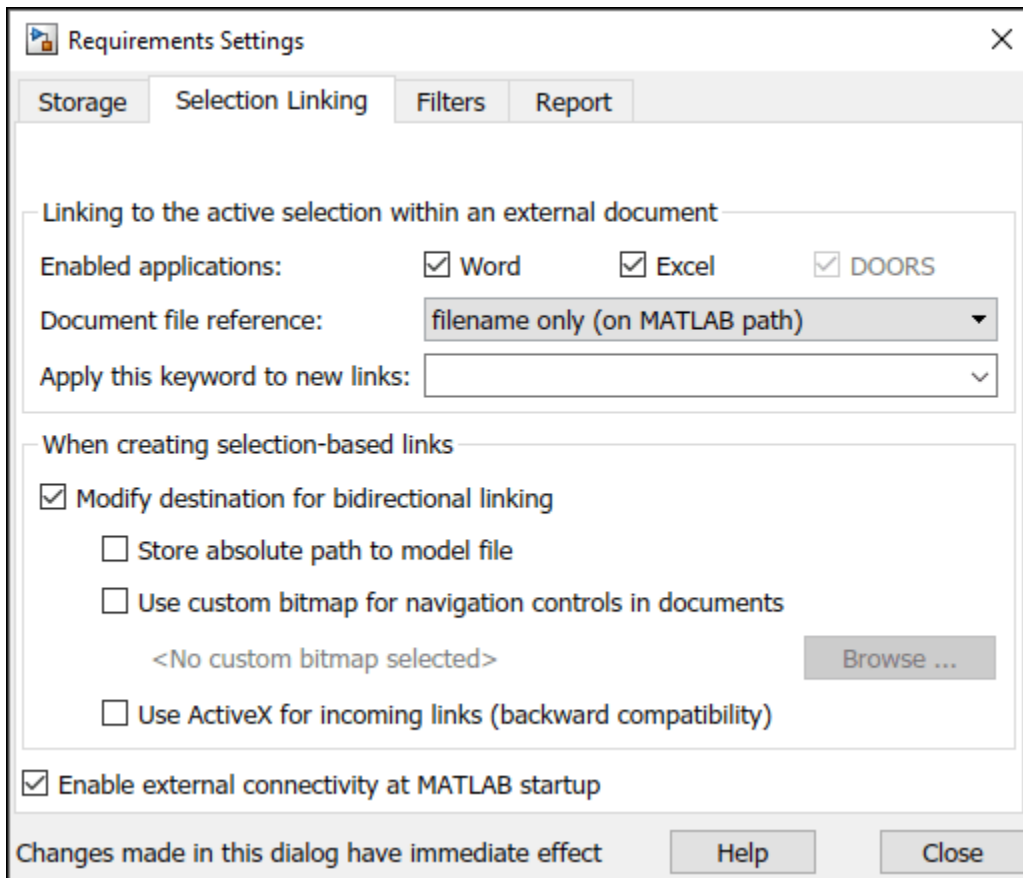
3.3. Manifold pressure failure mode 

Model Element: fuelsys/fuel_rate_controller/Sensor_correction_and_Fault

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.

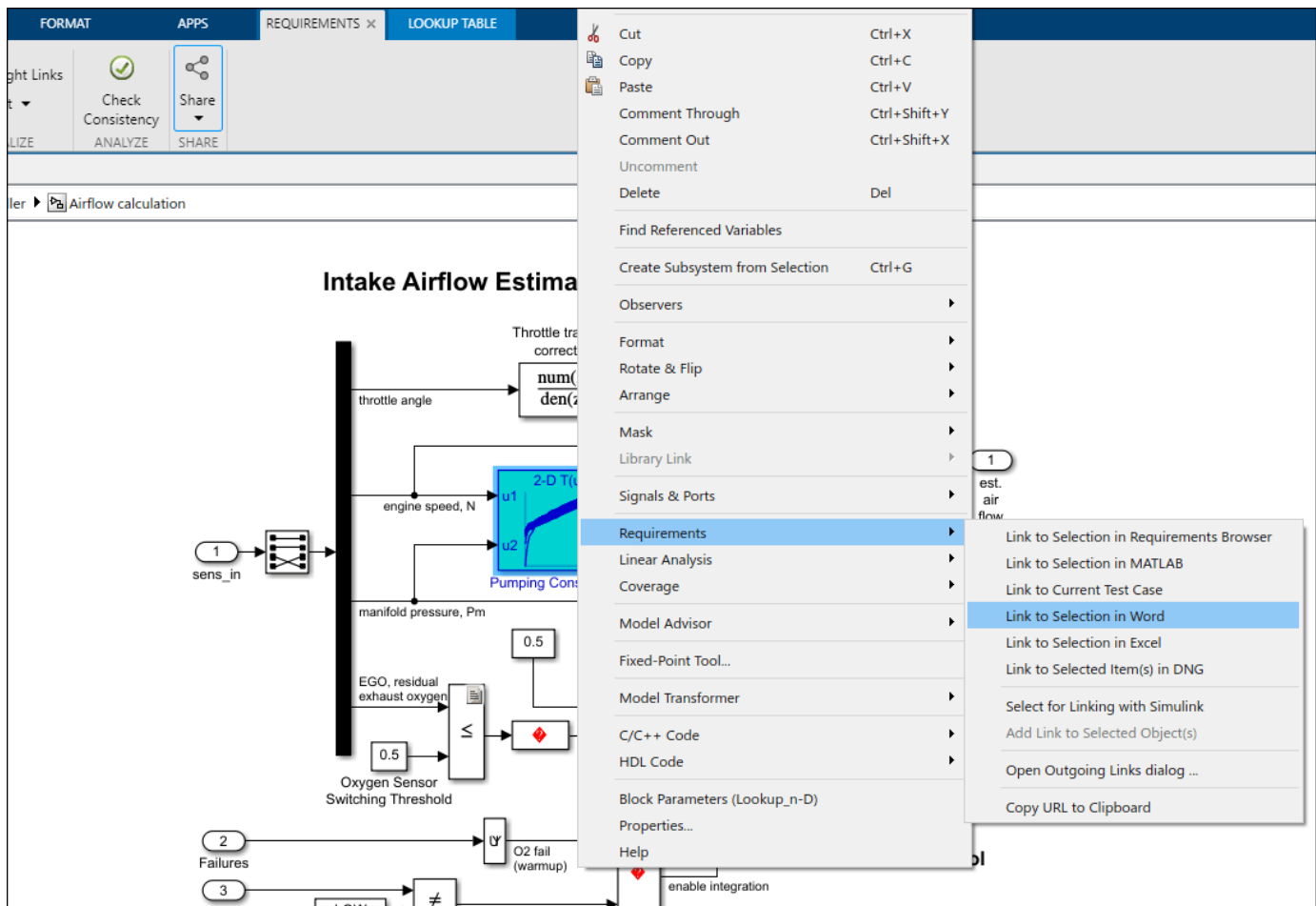


Now create new links similar to the ones you've just navigated. Note: Microsoft Word will not allow you to create the link when the document is *ReadOnly*. For the next step of this example, consider saving your own local copy of the document and using it instead of the installed document.

- 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

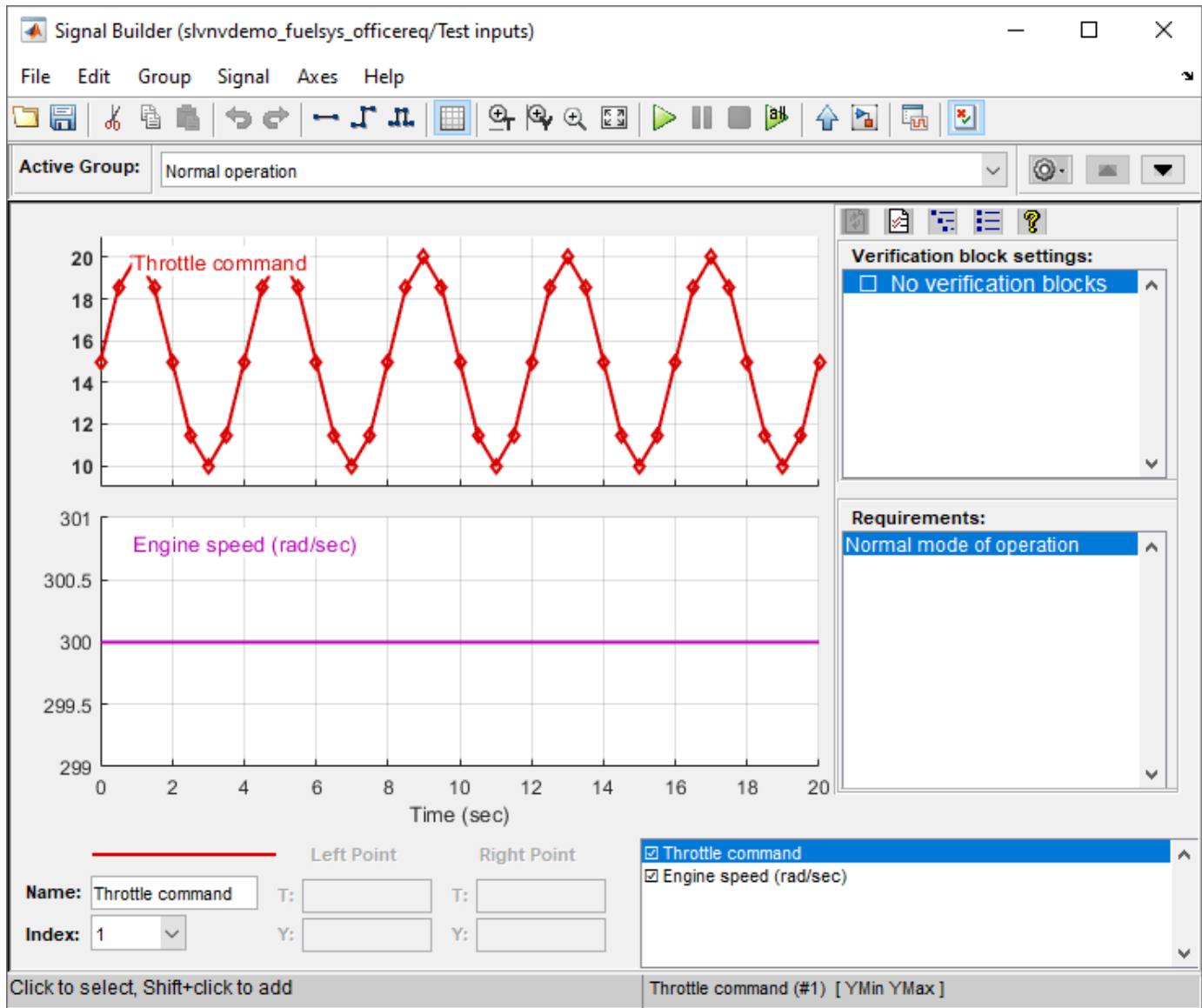
Signal links are attached to individual groups of signals, not to the Signal Builder block as a whole. Use this sort of links for test cases that are defined as Signal Builder groups.

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');
```

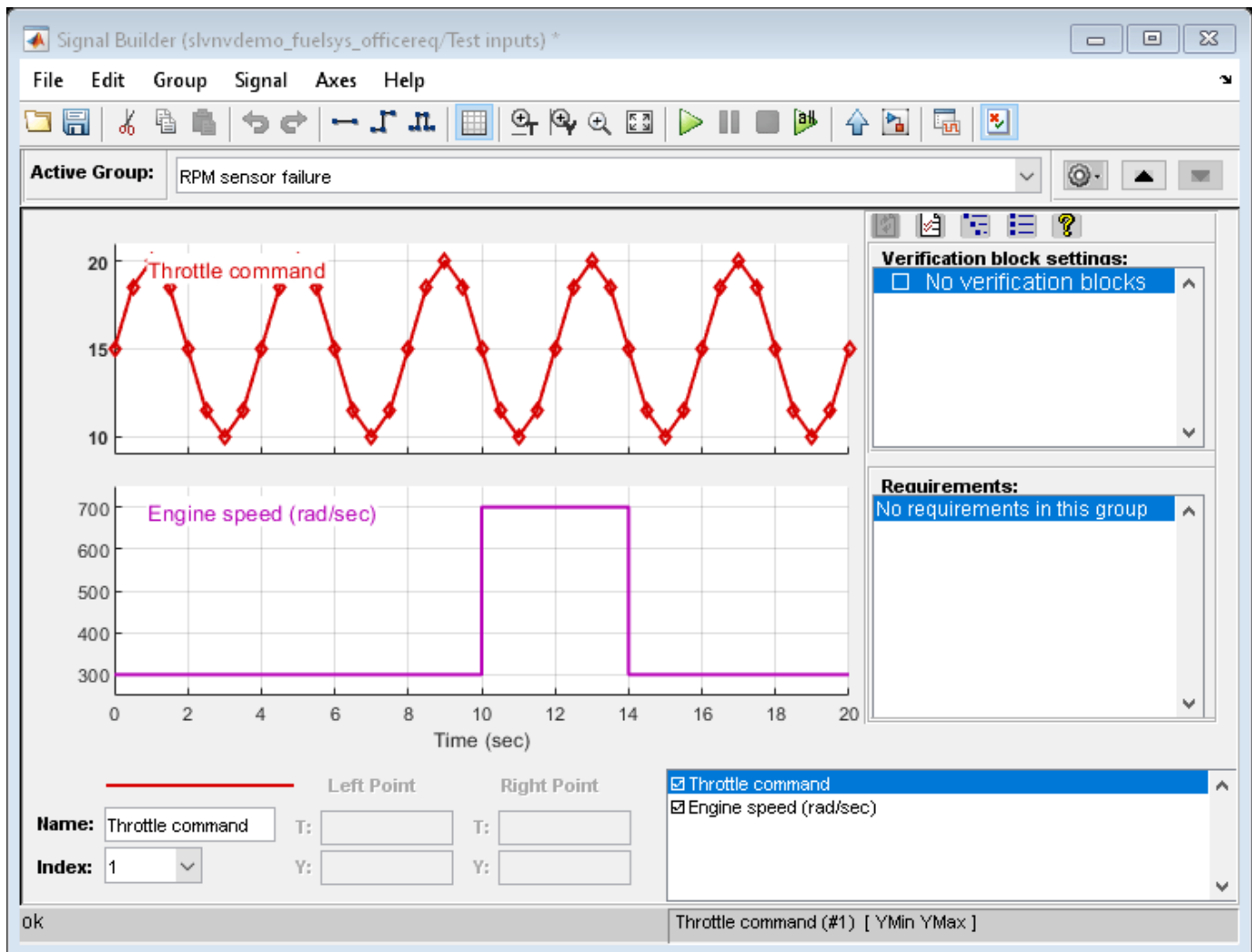
- Click the **Show verification settings** button  at the end of toolbar to display the Verification panel.
- If you do not see the **Requirements** panel below the **Verification block settings**, click the **Requirements display**  button at the top of the panel.
- Right-click the link label under **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)
```

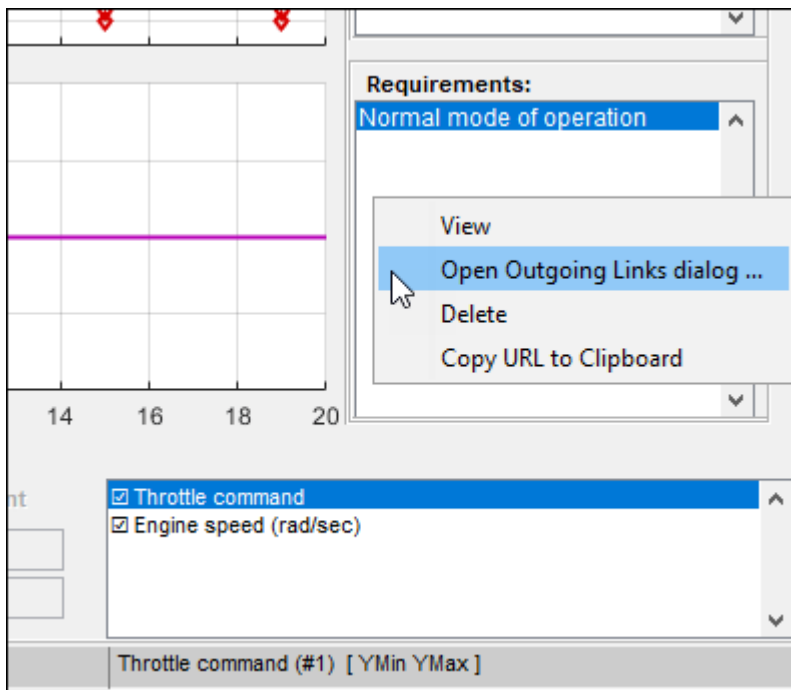


A transient RPM instability is modeled by a rectangular pulse on **Engine speed** data in the second group of signals:

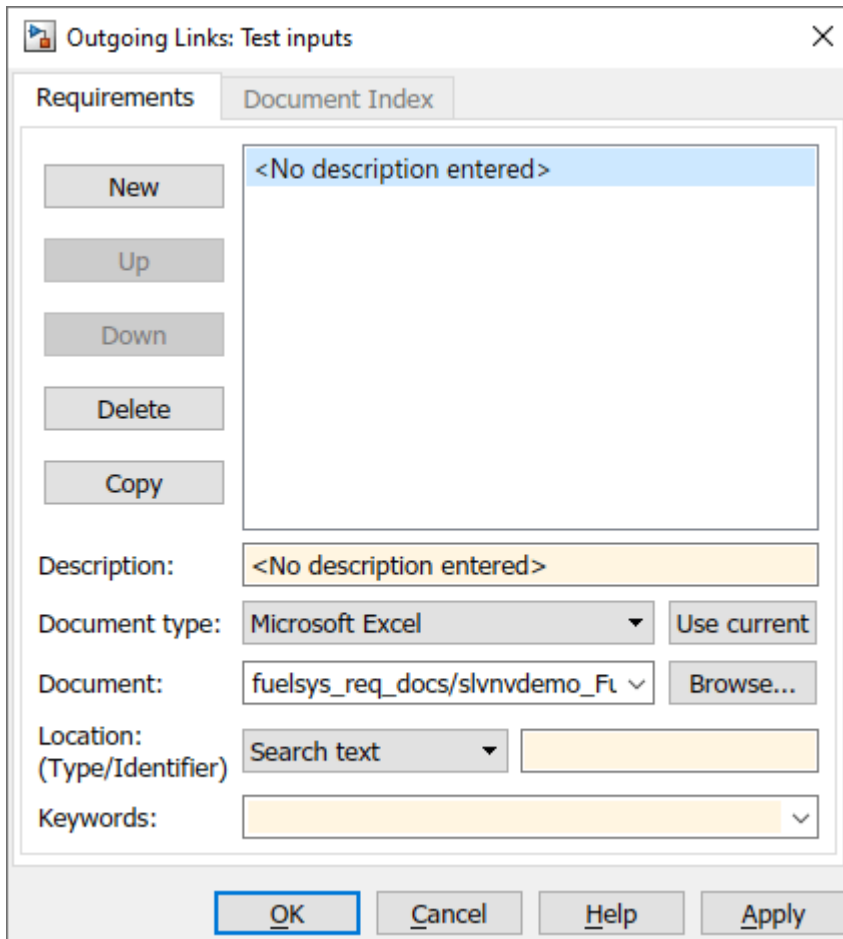
```
rmdemo_callback('signalgroup','slvndemo_fuelsys_officereq/Test inputs',2)
```



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.

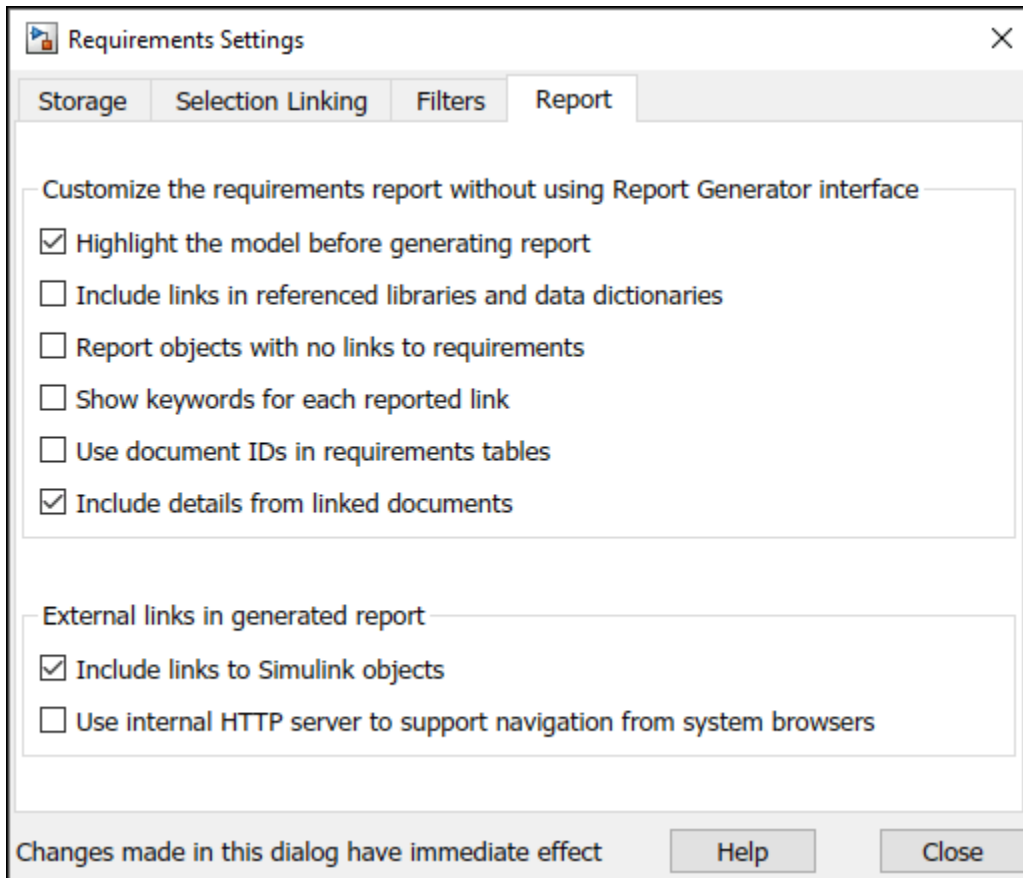
```
rmdemo_callback('report', 'slvndemo_fuelsys_officereq')
```

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.

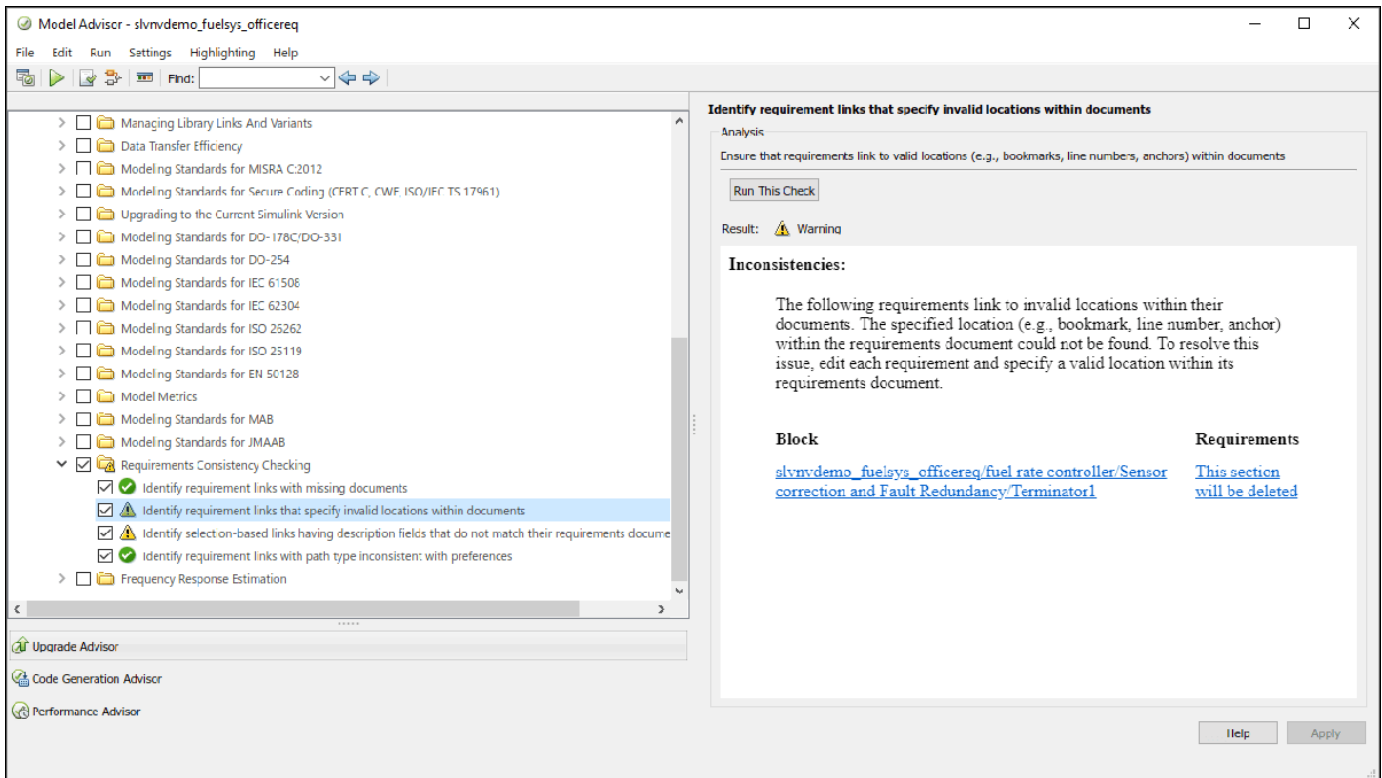


Requirements Consistency Checking

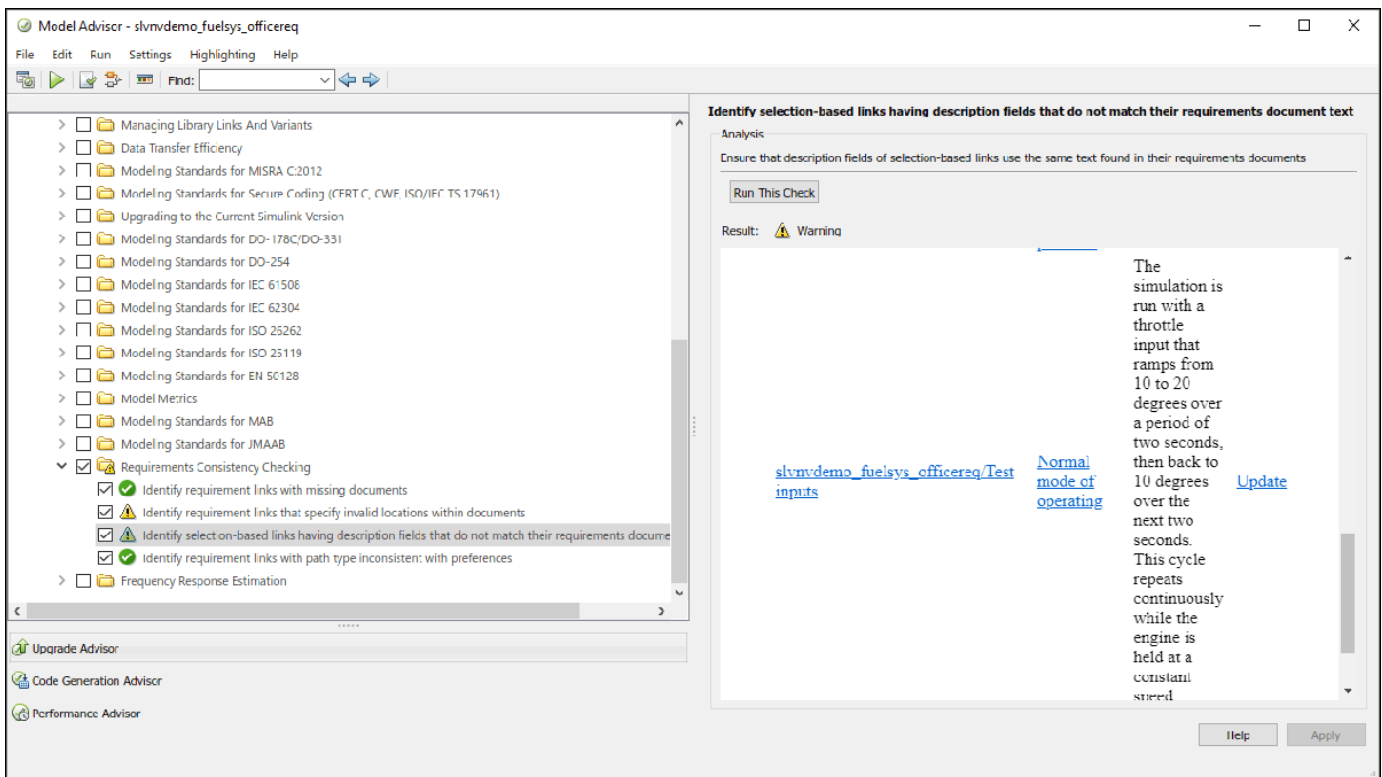
Use Model Advisor to automatically detect and fix inconsistencies in requirements links data. Click **Check Consistency** in the **Requirements** tab to open Model Advisor with only the RMI check points activated. The links are checked for missing documents, unmatched locations in documents, unmatched 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:



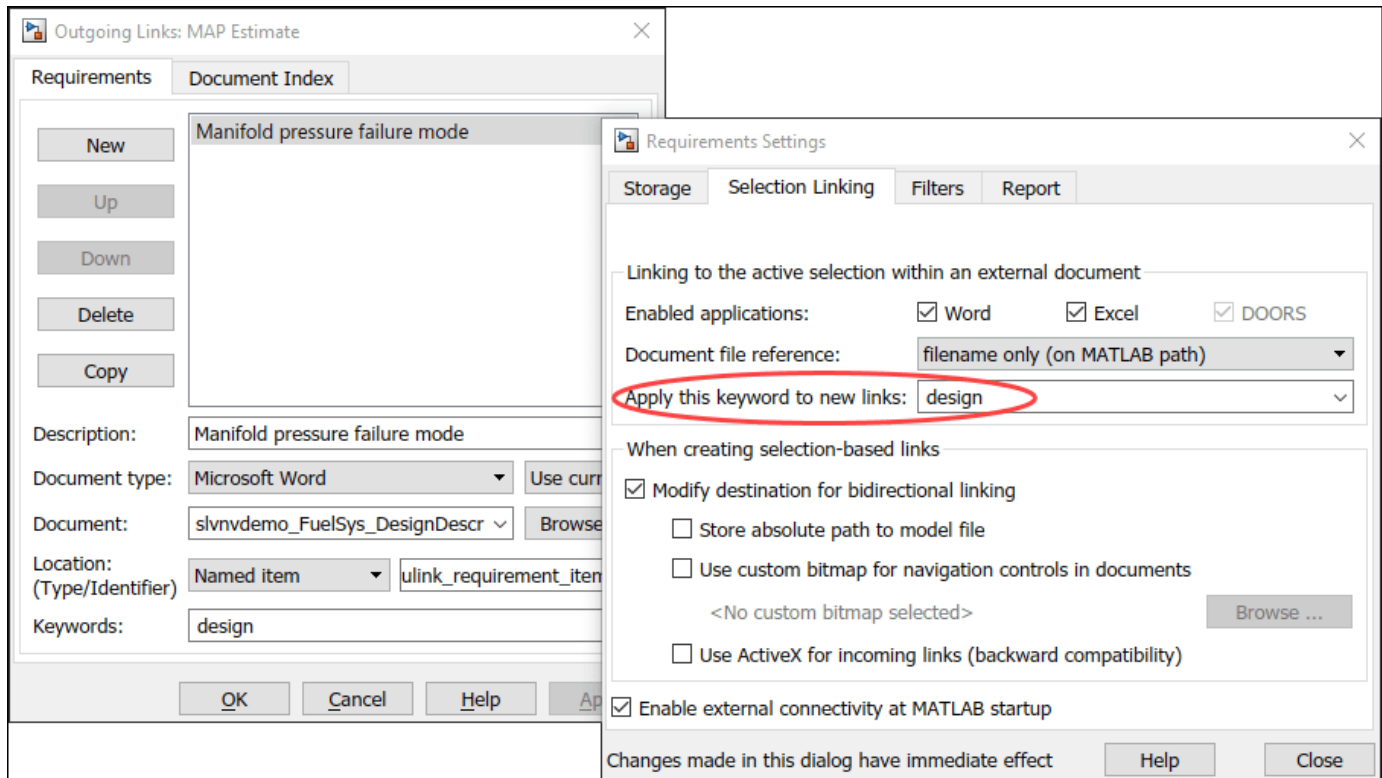
Another link has a label that does not match the original selection when the link was created:



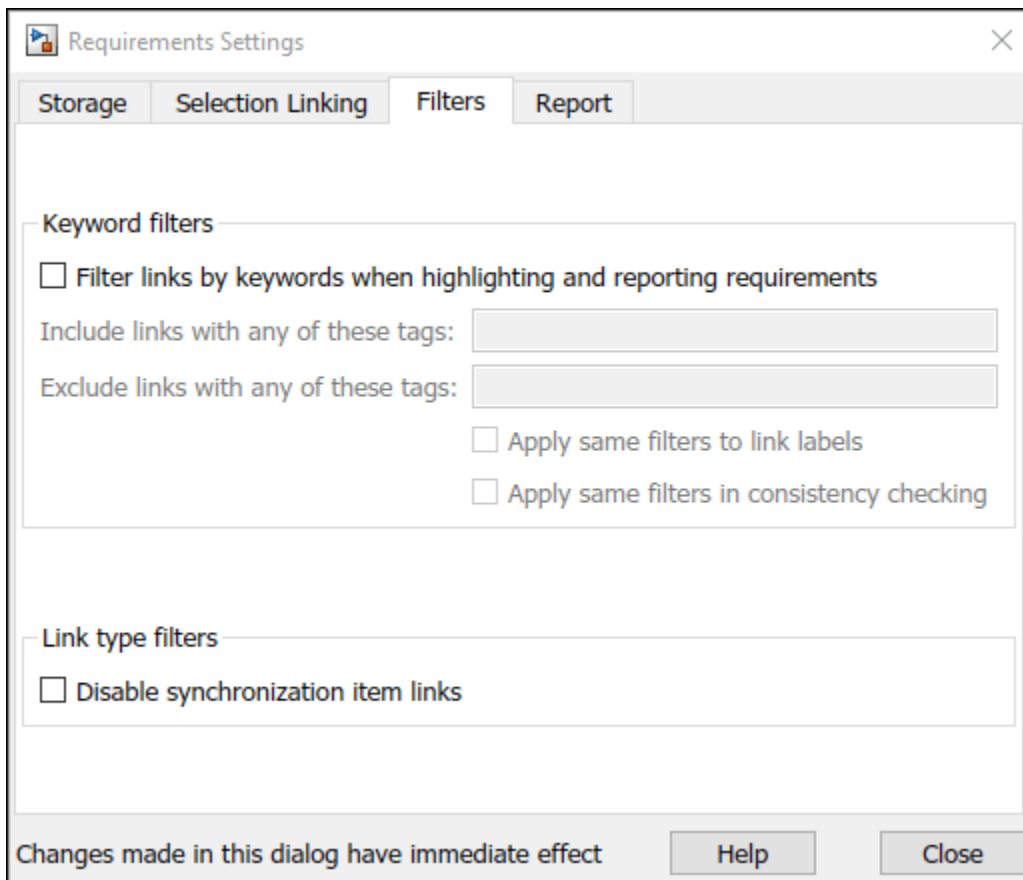
Click **Fix** or **Update** in Model Advisor report to automatically resolve reported inconsistencies. Rerun the checks to ensure reported problem is resolved.

Filtering Requirements on User Tag Property

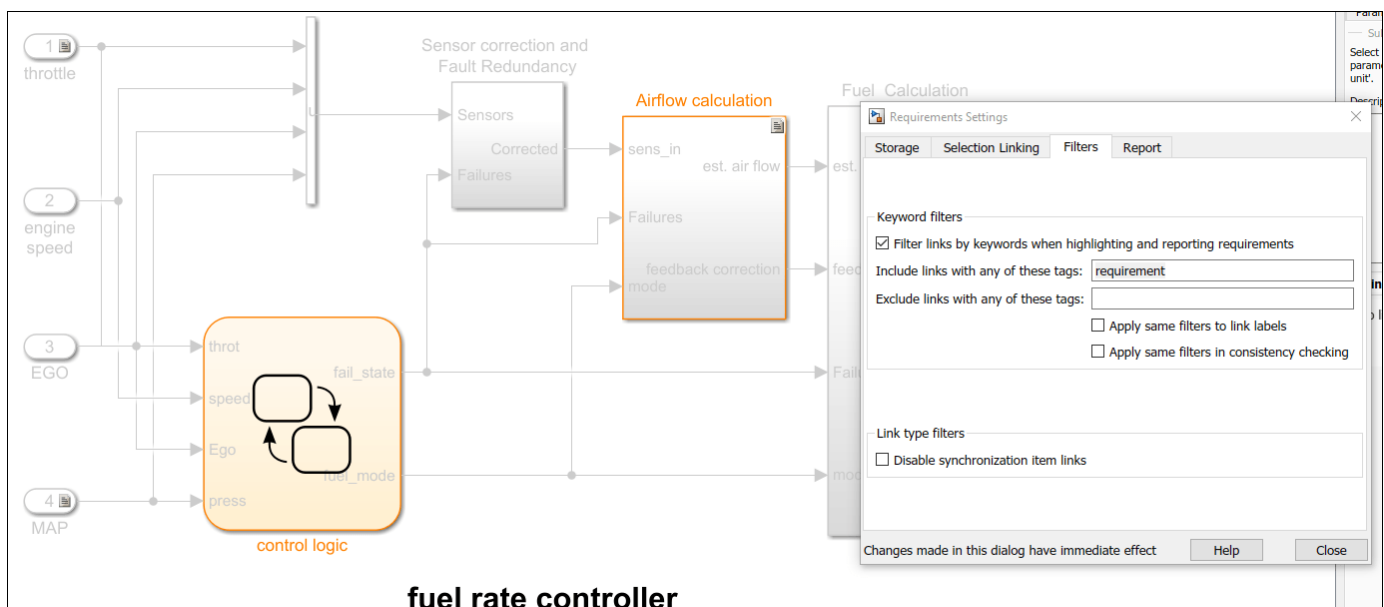
Requirements links in Simulink support an optional **User Tag** property that can store any comma-separated string values. Use these tags to distinguish between different types of links, for example, functional requirements links, design description links or testing details links. You can specify the 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 Update model requirements document paths and file names.
RMIDOCRENAME(MODEL_HANDLE, OLD_PATH, NEW_PATH)
RMIDOCRENAME(MODEL_NAME, OLD_PATH, NEW_PATH)
```

```
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:

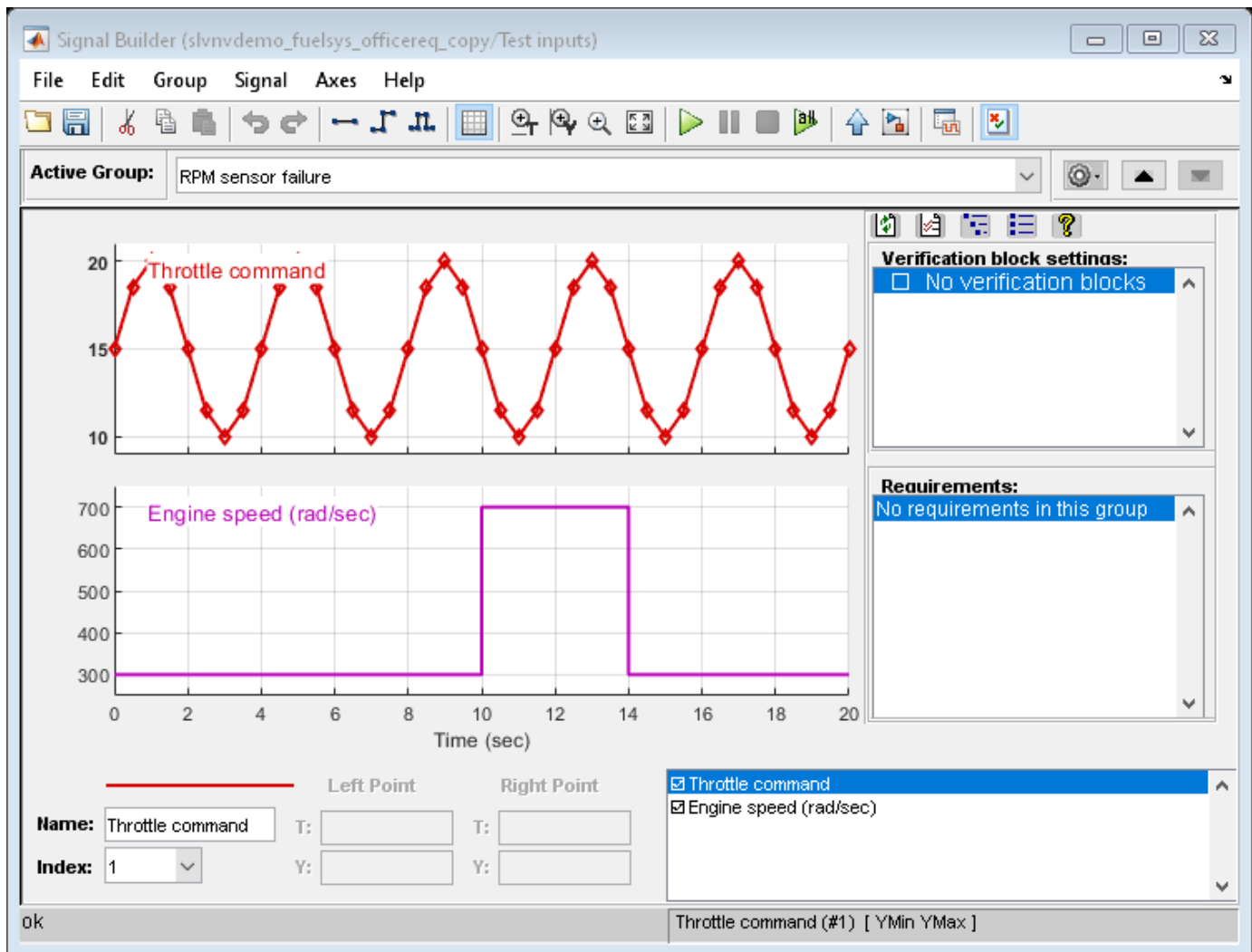
```
For the current Simulink(R) model, update all links to requirements
files whose names contain the string 'project_0220', replacing
with 'project_0221':
    rmidocrename(gcs, 'project_0220', 'project_0221');
```

```
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 RMI RMITAG

- Create a writable copy of the example model. For example, open the original system and save as `slvnvdemo_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:
`rmdocrename(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:

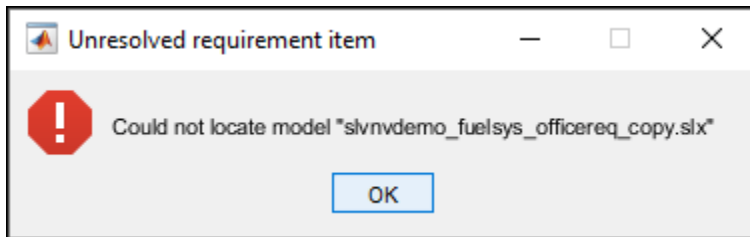
- Make sure `slvndemo_FuelSys_DesignDescription.doc` is open and is your current Microsoft Word document.

- 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 Simulink Requirements for IBM Rational DOORS Software” on page 7-2
- “Link with Requirements in IBM DOORS Next” on page 7-4
- “Link and Trace Requirements with IBM DOORS Next” on page 7-26
- “Navigate to Requirements in IBM Rational DOORS Databases from Simulink” on page 7-35
- “Synchronize Simulink Models with IBM Rational DOORS Databases by using Surrogate Modules” on page 7-39
- “Working with IBM Rational DOORS 9 Requirements” on page 7-50
- “Managing Requirements for Fault-Tolerant Fuel Control System (IBM Rational DOORS)” on page 7-59

Configure Simulink Requirements for IBM Rational DOORS Software

Simulink Requirements 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 Simulink Requirements and Rational DOORS on Windows platforms.

First perform the setup described in “Configure Simulink Requirements for Interaction with Microsoft Office and IBM Rational DOORS” on page 5-2. If Simulink Requirements 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 Simulink Requirements 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 Simulink Requirements for Interaction with Microsoft Office and IBM Rational DOORS” on page 5-2, and restart the DOORS software.

See Also

More About

- “Configure Simulink Requirements for Interaction with Microsoft Office and IBM Rational DOORS” on page 5-2
- “Import Requirements from IBM Rational DOORS” on page 1-33

Link with Requirements in IBM DOORS Next

This example shows how to link with requirements in IBM® DOORS® Next® and compares two ways to create links. For more information, see “Import Requirements from IBM DOORS Next” on page 1-27 and “Link and Trace Requirements with IBM DOORS Next” on page 7-26.

IBM Engineering Requirements Management DOORS Next (formerly known as DOORS Next Generation, or DNG) is a requirements management tool in IBM Collaborative Lifecycle Management platform. Traceability of file-based MBD artifacts (Simulink blocks, Test Cases, Data Dictionary entries) with items managed on a shared server and accessed by means of web browser can be accomplished in a few different ways. Your choice of workflow will depend on the needs and constraints of a given project. This example gives a review of DOORS Next integration features supported by Simulink Requirements, it includes step-by-step setup instruction and compares alternative workflows. Working with IBM® DOORS® Next® is supported on Microsoft Windows®.

Overview

Simulink Requirements allows two different ways to integrate with DOORS: *direct linking* between design elements and web-based requirements, and *live cache* approach where you establish traceability natively in Simulink Requirements workspace. Each approach has unique advantages and challenges.

Direct Linking approach allows to establish one-click navigation from your design to associated requirements in DOORS, and from DOORS requirements back to linked design elements. Because this approach requires system web browser interactions, DOORS server-side configuration is needed, as well as system browser customization by the user. Because link destinations are 'external' to Simulink Requirements, you can only use a limited subset of Simulink Requirements product features. For example, additional scripting will be required for Implementation and Verification status analysis, and for Change Tracking. On the other hand, you have a choice to store link information in DOORS server data, so that links can be viewed and analyzed on DOORS side without MATLAB/Simulink session.

Live Cache approach relies on importing a snapshot of DOORS Requirements into Simulink Requirements. Navigation from design or test to a related requirement in DOORS is still possible but involves an intermediate proxy entry in Simulink Requirements set. Because both ends of your created links belong to Simulink family domains, you can take full advantage of traceability features in Simulink Requirements product. Cached content can be updated anytime when there are changes on DOORS server side. Also, because links are stored only on MATLAB/Simulink side, this approach avoids the problems of stale or conflicting links on the server. This approach does not necessarily require server-side configuration.

Details below will help you decide which approach is best for your project.

Direct Linking

You can link *directly* with DOORS Next artifacts, using the **Link to Selected Item(s) in DNG** shortcut in **Requirements** menu for Simulink objects, MATLAB code, or Simulink Test Cases. This capability implies that MATLAB session is aware of your selection in DOORS Next web browser, which is why extra setup steps are required:

- 1 Simulink Requirements *custom widget* files must be copied to DOORS Next server, to make the widget available for DOORS Next users.
- 2 Each user interested in linking with Simulink must add the widget to **Mini Dashboard** in DOORS Next web interface.

- 3 System browser should be allowed to communicate with MATLAB's embedded HTTPS server (localhost:31515).

Each required setup step is detailed below.

Server-side configuration

This step is performed by DOORS Next server administrator once per IBM server installation. You will need to copy the `dnsslk_config` subfolder from `MATLAB_INSTALL_DIR/toolbox/slrequirements/slrequirements/resources/` into the DOORS Next server's custom extensions folder. The location of custom extensions folder depends on the particular Jazz server version. For example, if you are running Jazz server on a Windows computer, your extensions folder could be here:

```
C:\Program Files\IBM\JazzTeamServer\server\liberty\servers\clm\dropins\war\extensions
```

You may also need to "enable dropins" in DOORS Next server configuration. Instruction below is based on the following IBM's page: <https://jazz.net/wiki/bin/view/Main/RMExtensionsHostingGuide605>

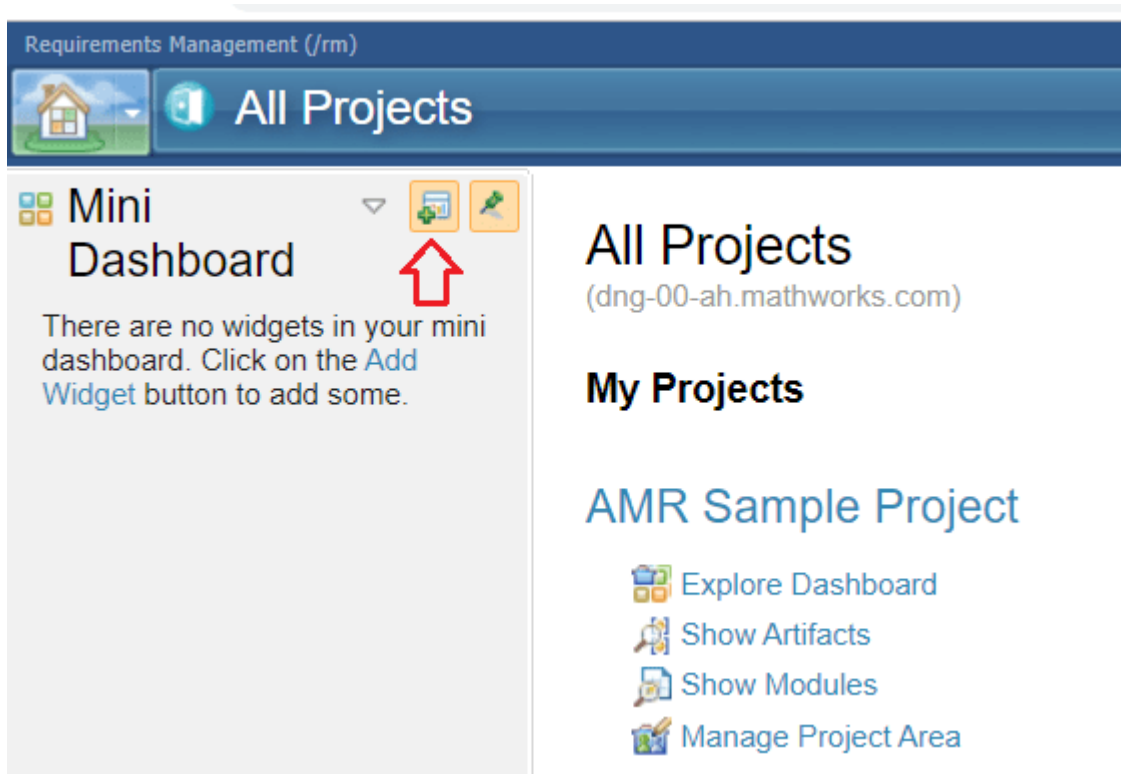
- 1 Locate `server.xml` file in the `C:\[JAZZ_INSTALL_DIR]\server\liberty\servers\clm` folder.
- 2 Open this file in a text editor, and locate this line: `<applicationMonitor dropinsEnabled='false' pollingRate='10s' updateTrigger='mbean' />`
- 3 Change `dropinsEnabled` to `'true'`.
- 4 Restart the server.

Please refer to IBM's page for more detailed instructions for hosting custom extensions.

Client Browser Configuration

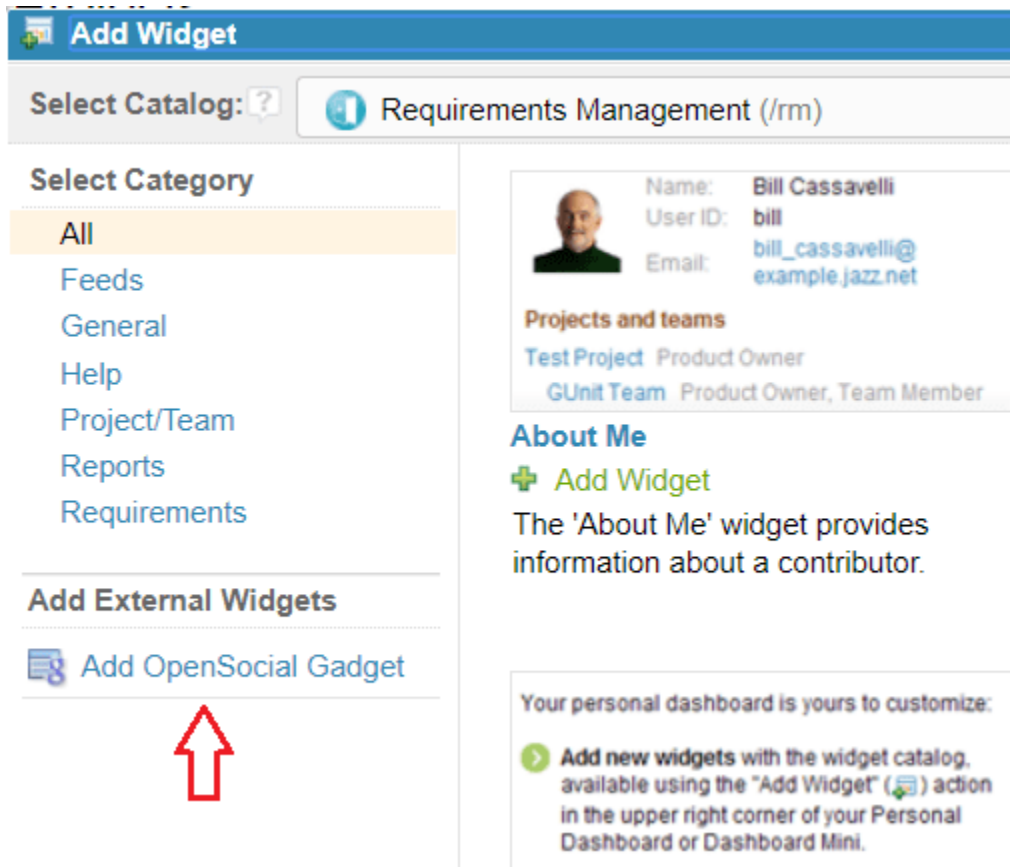
This step needs to be performed once by each DOORS Next user interested in linking with MATLAB/Simulink. After `dnsslk_config` custom extension files are available on your DOORS Next server, follow these steps to add this *custom widget* to the **Mini Dashboard** in DOORS Next web interface. Ensure that your DOORS Next server is secure (HTTPS), or the widget won't function correctly. After logging into DOORS Next:

1. In **Mini Dashboard**, click the **Add Widget** button:



Custom gadgets menu will open.

2. Click **Add OpenSocial Gadget**:



Add Widget

Select Catalog: Requirements Management (/rm)

Select Category

- All
- Feeds
- General
- Help
- Project/Team
- Reports
- Requirements

Add External Widgets

[Add OpenSocial Gadget](#)

Name: Bill Cassavelli
User ID: bill
Email: bill_cassavelli@example.jazz.net

Projects and teams

Test Project Product Owner
 GUnit Team Product Owner, Team Member

About Me

[Add Widget](#)

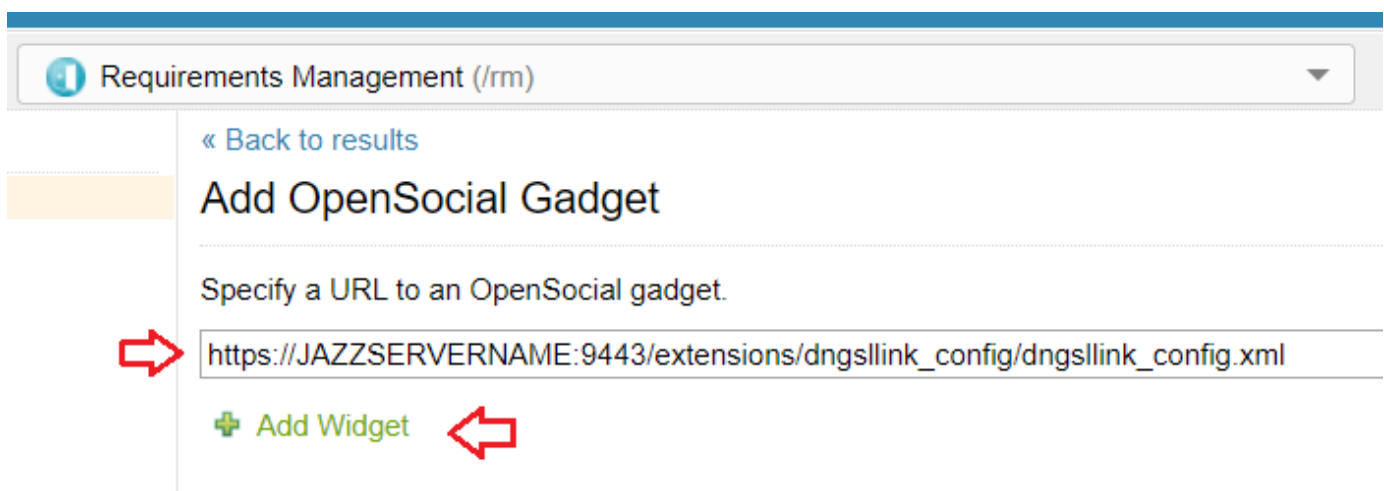
The 'About Me' widget provides information about a contributor.

Your personal dashboard is yours to customize:

[Add new widgets](#) with the widget catalog, available using the "Add Widget" action in the upper right corner of your Personal Dashboard or Dashboard Mini.

3. Specify the URL that matches the location of Simulink Requirements widget code on your server. For example:

`https://JAZZSERVERNAME:9443/extensions/dngsllink_config/dngsllink_config.xml:`



Requirements Management (/rm)

[« Back to results](#)

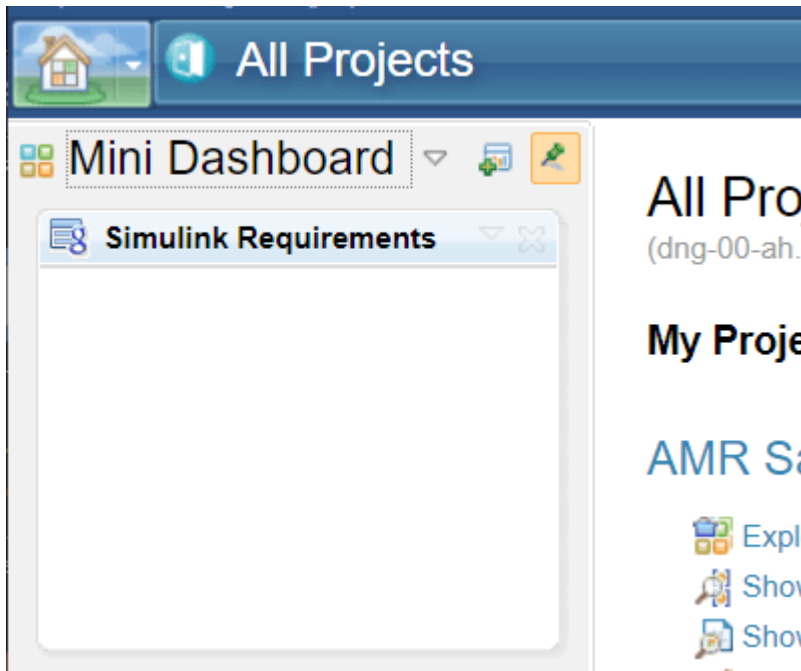
Add OpenSocial Gadget

Specify a URL to an OpenSocial gadget.

https://JAZZSERVERNAME:9443/extensions/dngsllink_config/dngsllink_config.xml

[Add Widget](#)

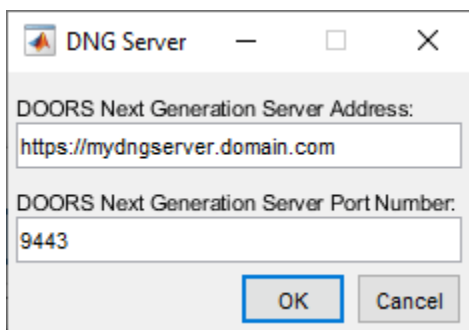
4. Click **Add Widget**. Your **Mini Dashboard** should now display the **Simulink Requirements** widget:



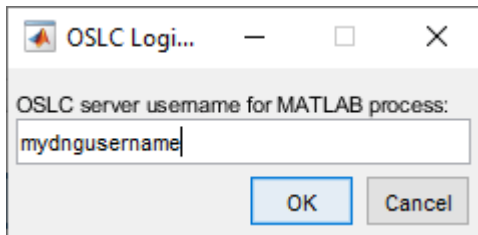
Configuring MATLAB session

This step needs to be performed each time MATLAB session is restarted. Use the `slreq.dngConfigure` command to prepare your MATLAB session for linking with DOORS Next. Follow the prompts and provide the requested values. The server URL, port number, and username is stored in your personal user preferences. However, you have to enter the DOORS Next password each time.

1. When prompted, enter your DOORS Next server domain name and the port number. If you do not see any port number displayed in the address bar of your system browser when viewing DOORS Next pages, enter the default value of "9443".

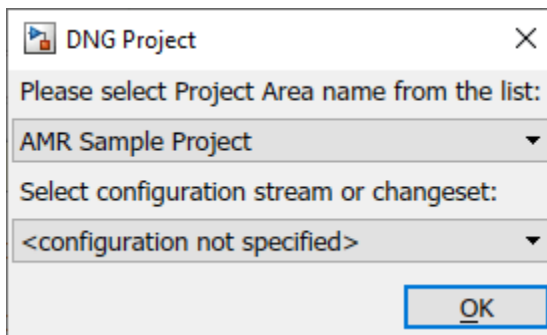


2. Enter your DOORS Next user name, which may be different from your computer login user name:

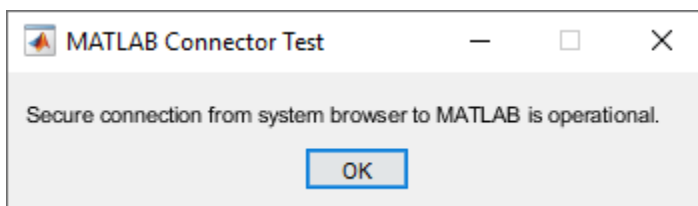


3. When prompted, enter your DOORS Next password and press **Enter**. It is normal to see a few warnings in MATLAB's command window when establishing connection with DOORS Next. The feature will operate, unless there are errors.

4. After successful connection to the server is established, a dialog box appears to allow you select a DOORS Next projects from the list, as well as the preferred configuration stream (if enabled for the selected project).



5. A browser-to-localhost connection test runs automatically. This communication channel is required for your MATLAB session to receive messages when you select DOORS Next item in the web browser. You may see an empty browser page, and a popup from MATLAB indicating that you are ready for linking:



6. If you do not see the confirmation message as shown above, your system browser may be blocking HTTPS connections to `https://localhost:31515`. To resolve this issue, allow the connection. The exact steps depend on your web browser. For example:



Your connection is not private

Attackers might be trying to steal your information from **localhost** (for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR_CERT_AUTHORITY_INVALID

Help improve Safe Browsing by sending some [system information and page content](#) to Google. [Privacy policy](#)

Advanced

Back to safety

In this case, click **Advanced** and then click the hyperlink to allow the connection:

Hide advanced

Back to safety

This server could not prove that it is **localhost**; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

[Proceed to localhost \(unsafe\)](#)



If you still do not get the popup from MATLAB, your MATLAB session may be listening on a different (non-default) port number, which can happen when starting more than one instance of MATLAB on

the same host. To quickly check the active port number, run this command:
`connector.securePort`. If this command returns anything other than 31515, make sure that you do not have any other MATLAB instances running on the same host, then restart MATLAB. Repeat `connector.securePort` command to confirm the correct port number. Rerun `slreq.dngConfigure` setup steps.

Once you see the confirmation dialog, do not close the browser window. It is best to reuse this same browser window for your DOORS Next session when linking with Simulink Requirements, because you have just authorized this instance of the web browser application to communicate with MATLAB. If you open a new browser window, depending on web browser type and version, secure communication with MATLAB may be blocked again. If this happens, you can simply copy-paste the following URL into your browser's address bar: `https://localhost:31515/matlab/oslc/inboundTest` then, again, click on "Proceed to localhost" to allow connection with MATLAB.

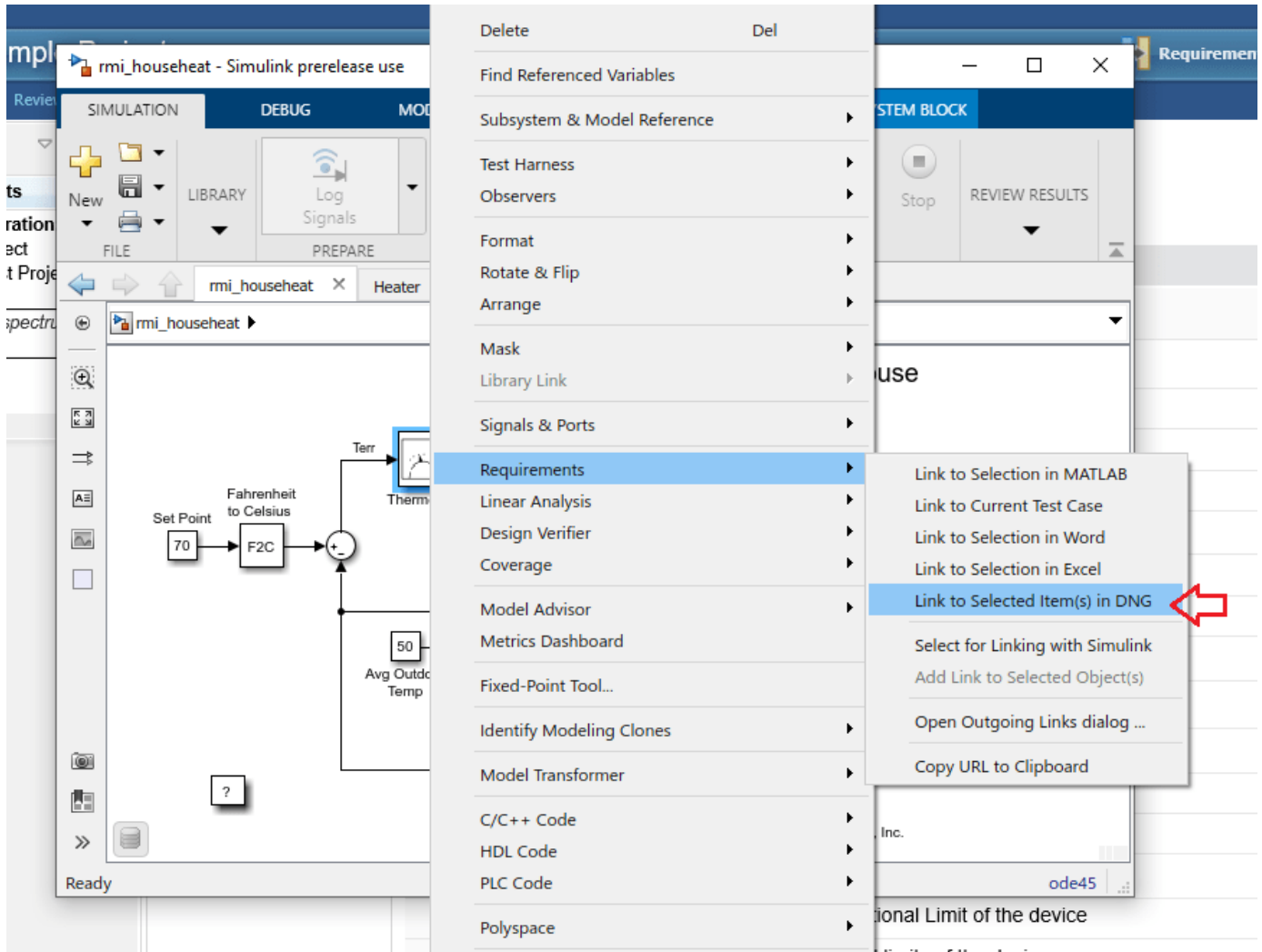
One-way Links from MATLAB/Simulink to DOORS Next

In DOORS Next, open **Show artifacts** view for the requirements collection of interest, and select the checkbox for the item you want to link with. You will notice that the **Simulink Requirements** widget is updated to confirm the ID and label of the selected item. This information is sent to MATLAB when you interact with DOORS Next item checkboxes.

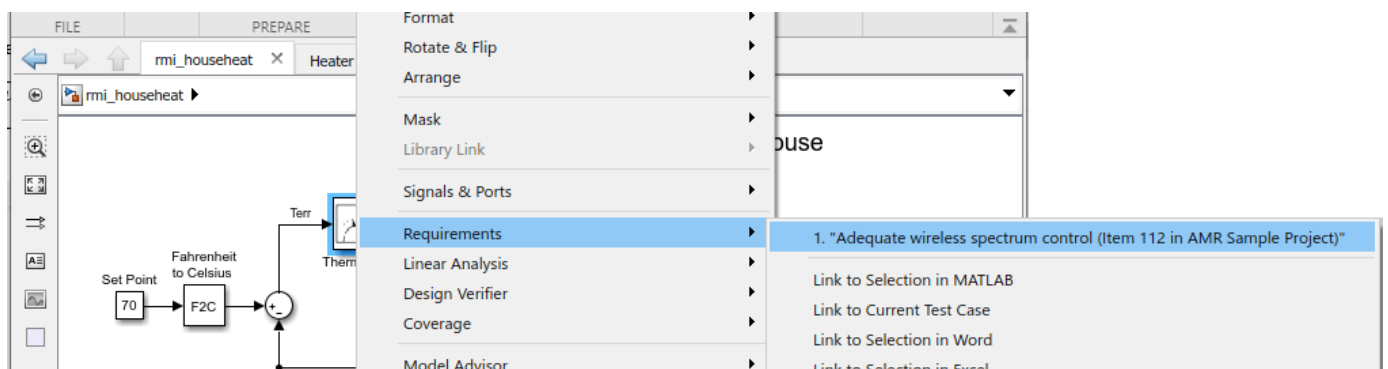
The screenshot shows the IBM DOORS Next interface. On the left, a 'Mini Dashboard' contains a 'Simulink Requirements' widget. This widget displays 'Simulink session configuration' for 'Project: AMR Sample Project' and 'Context: Requirements Test Project Initial Stream'. A table within the widget shows a selected requirement with ID '245' and the label 'Temperature Operational Limit of the device'. A red circle highlights this entry. Below the table is a link that says 'Query links from SL'. The main area of the interface displays a list of requirements artifacts under the heading '01 Requirements > | 3 AMR Hazards and Risks'. The list has columns for 'Views', 'ID', and 'Contents'. A red circle highlights the row for ID '245', which is 'Temperature Operational Limit of the device'. The right-hand side of the interface shows a 'Module' panel for '3: AMR Haz and Risks', displaying details such as 'Description', 'Project: Requirements', 'Team Ownership: AMR', 'Content Folder: _AMR Haz and Risks artifacts', 'Created On: May 7, 20...', 'Created By: Kot Matros', 'Modified On: May 7, 20...', 'Modified By: Kot Matros', 'Is Suspect', 'Type: Hazard and F', 'Format: Module', 'Approved By:', and 'Approver Position:'.

Views	ID	Contents
<input type="checkbox"/>	336	-1 System Hazards
<input type="checkbox"/>	263	-1.1 Physical Hazards
<input type="checkbox"/>	43	Sharp Edges
<input type="checkbox"/>	297	Pinch Areas
<input type="checkbox"/>	340	Electrocution
<input type="checkbox"/>	152	-1.1.1 Size Limitations
<input type="checkbox"/>	280	Future AMR handheld size growth
<input type="checkbox"/>	347	Future AMR handheld mass growth
<input type="checkbox"/>	268	1.1.2 Fire/Explosion
<input type="checkbox"/>	55	-1.2 Environmental Hazards
<input type="checkbox"/>	13	-1.2.1 Animals
<input type="checkbox"/>	89	Leashed Pets
<input type="checkbox"/>	30	Stray Animals
<input type="checkbox"/>	70	-1.2.2 External Weather
<input checked="" type="checkbox"/>	245	Temperature Operational Limit of the device
<input type="checkbox"/>	60	Humidity operational limits of the device
<input type="checkbox"/>	212	-1.3 External Communications

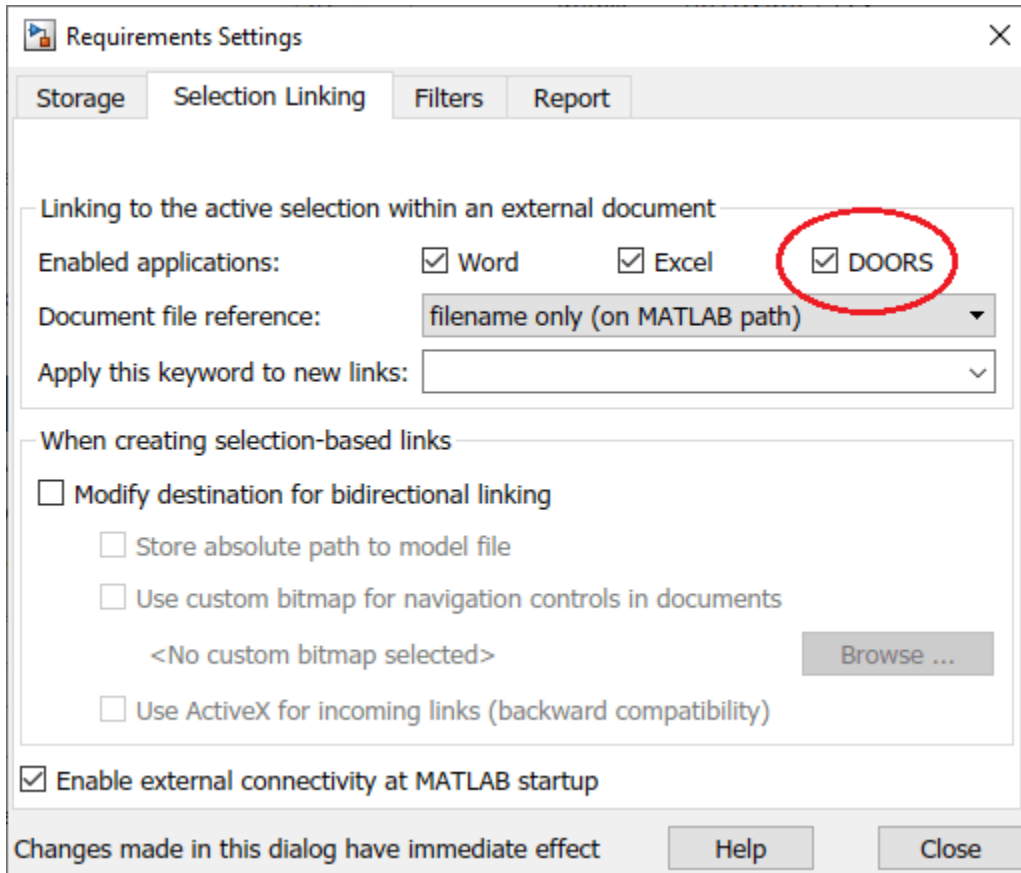
In Simulink, right-click a block you want to link from, then select **Link to Selected Item(s) in DNG** under **Requirements** context menu. It may take a few seconds for MATLAB to retrieve additional data from DOORS Next and create the link.



Click the same block again to see the new link at the top of **Requirements** submenu. Click the link label to navigate from Simulink to DOORS Next:



Note: if you do not see the **Link to Selected Item(s) in DNG** shortcut in the **Requirements** context menu, you may need to enable DOORS linking option in **Selection Linking** tab of **Requirements Settings** dialog:



Alternatively, you can control this setting via the command-line API:

```
rmipref('SelectionLinkDoors', true);
```

Reviewing MATLAB/Simulink Links from the DOORS Next Side

DOORS Next integration feature in Simulink Requirements allows you to query MATLAB/Simulink links from DOORS Next context. When you select an item from the artifacts list in DOORS Next page, the **Simulink Requirements** widget displays information about the selected item, and provides a hyperlink for querying links as stored in Simulink Requirements. Click **Query Links from SL** to get a popup with the list of incoming links for the selected DOORS Next item.

The screenshot shows the IBM Rational DOORS Requirements Management interface for the 'AMR Sample Project'. The 'Mini Dashboard' on the left displays 'Simulink Requirements' with a session configuration for 'Project: AMR Sample Project' and 'Context: Requirements Test Project In'. A requirement with ID 239 is highlighted, with the text 'The AMR system is used to determine water...' and a link 'Query links from SL' indicated by a red arrow. The main panel shows a list of requirements with columns for 'Views', 'ID', and 'Contents'. The requirement with ID 239 is selected and highlighted in orange.

Views	ID	Contents
Search View	241	-1 Int
Gold Plating	271	-1.1
Trace to Stakeholder Require	53	This d system and f
Upstream and Downstream T	357	-2 Ge
	191	-2.1
<input checked="" type="checkbox"/>	239	The A 79,00 squa
	235	is ha

The screenshot shows a browser window titled 'https://localhost:31515/?id=239 - Links from MATLAB/Simulink id=239 - Internet Explorer'. The page content is titled 'Links from MATLAB/Simulink id=239' and displays a table of linked artifacts.

Source Artifact	Source Object	Linked Document	Version	Item
rmi_househeat.slx	Thermostat (SubSystem)	AMR Sample Project	Requirements Test Project Initial Stream	239

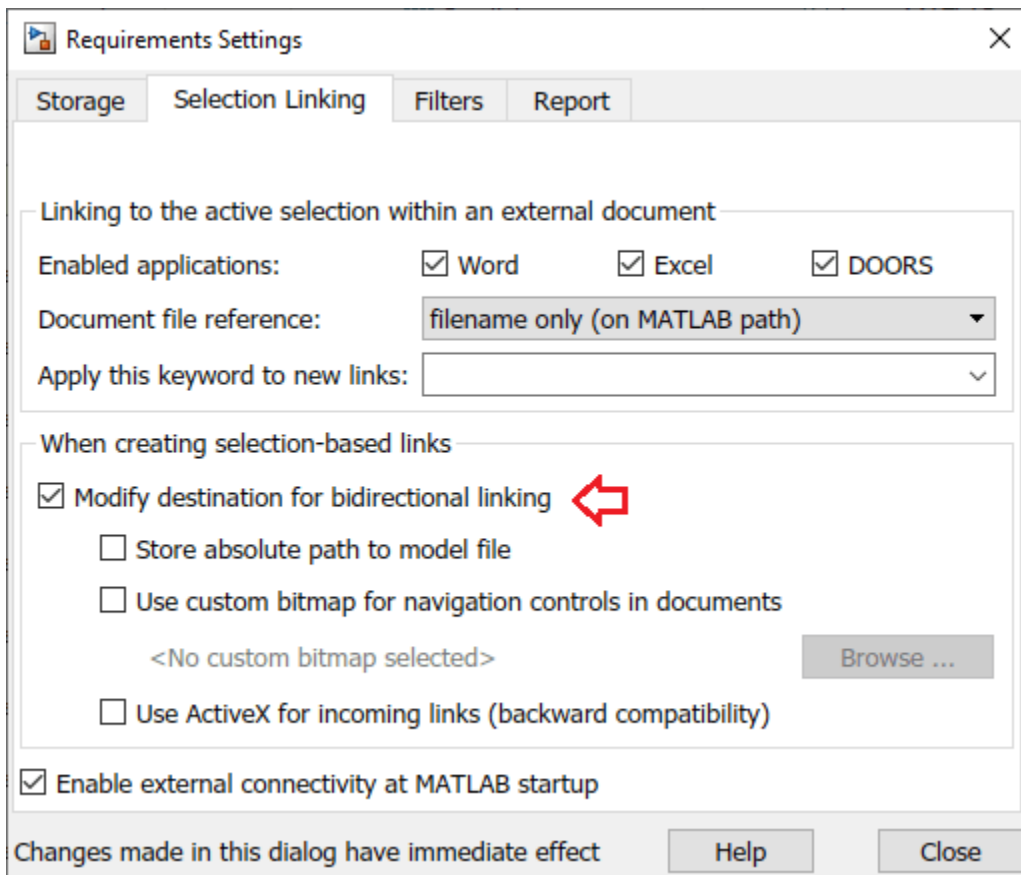
One should keep in mind that these links cannot be discovered when MATLAB is not running, or when the corresponding data files are not loaded on Simulink side. For example, the link we created above is stored in .slmx file for the linked Simulink model. If this .slmx file is not loaded in the current MATLAB/Simulink session, no links will be reported in the browser popup. When relying on **Query Links from SL** to review links, you must first ensure that:

- 1 MATLAB is running
- 2 MATLAB session is configured for DOORS Next linking (slreq.dngConfigure step was completed)
- 3 all related linked design artifacts are loaded

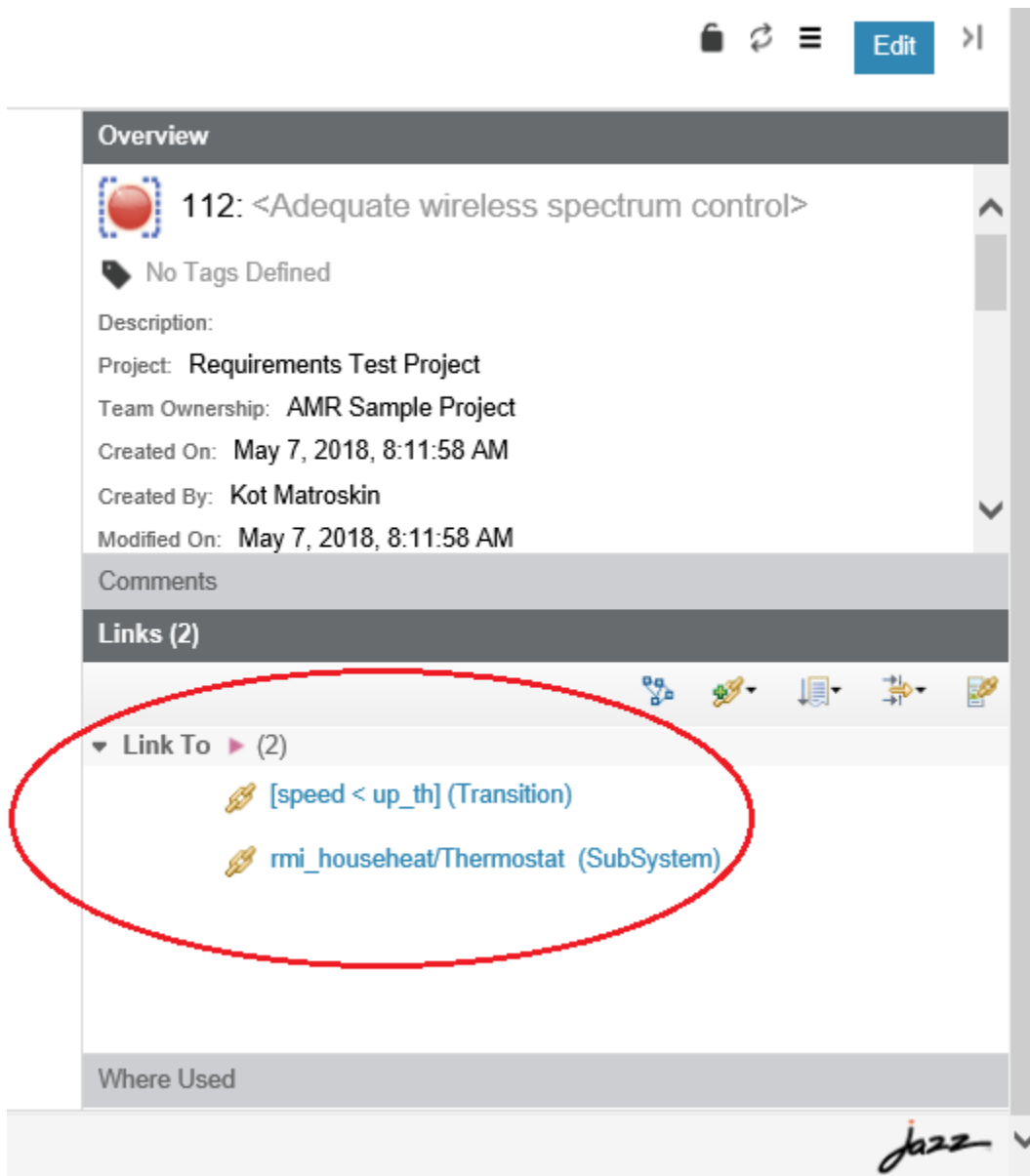
You can review the list of loaded Link Sets by opening the **Requirements Editor** by entering `slreq.editor` at the MATLAB command line. In the Requirements Editor, click **Show Links**.

Store Links in DOORS Next for Two-Way Traceability

If you prefer to always find your MATLAB/Simulink links in DOORS Next context, independent from whether Simulink is running or whether the linked MBD artifacts are loaded, you have an option of truly bi-directional linking. Re-open the **Requirements Settings** dialog to the **Selection Linking** tab and enable the **Modify destination for bi-directional linking** checkbox.



Alternatively, you can use the command-line API `rmipref('BiDirectionalLinking',true)` to toggle the option. Once bi-directional linking is enabled, each new link you create will not only add an entry in the Simulink Requirements Link Set, but will also insert an *External Web Link* from DOORS Next, which you can see in the **Links** panel for the linked DOORS Next item. You can use the hyperlinks in the **Links** pane to navigate from DOORS Next item to linked objects in MATLAB/Simulink.



When enabling **Modify destination for bi-directional linking** option in **Requirements Settings**, consider the following:

1. Every DOORS Next user will see these links when working with same version of this DOORS Next project, even if they do not use Simulink or do not have access to linked MBD artifacts.
2. Navigation from DOORS Next will fail, unless MATLAB is running, and linked artifact is either already loaded or can be found on MATLAB path.
3. Links inserted into DOORS Next by Simulink Requirements do not synchronize automatically. If you delete a link on Simulink side, link in DOORS Next remains, and you need to remove it manually.
4. Simulink Requirements product does not check for conflicts in DOORS Next links. For example, if Simulink user A linked a DOORS Next requirement to a block in a Simulink model, the links inserted

in DOORS Next will behave consistently for this user, but user B will see the link from DOORS Next, and can navigate it to the same block in his version of same Simulink model, even if his copy of the Simulink model does not store a link from Simulink block to DOORS Next. If user B decides to create his own link from same block to same DOORS Next item while his personal preference is configured for bi-directional linking, this will insert a duplicate link on DOORS Next side. If user A later changed his mind and deleted a link from block to Simulink, and then tries to cleanup the *backlinks* from DOORS Next, both links will need to be deleted on DOORS Next side, and now user B will be left with only a one-way link from Simulink to DOORS Next. Using personal streams and changesets in DOORS Next should minimize this sort of problems.

```
rmipref('BiDirectionalLinking', true);
```

Working with Cached Requirements Collections

As can be seen from the above, both 1-way and 2-way *direct linking* solutions have disadvantages:

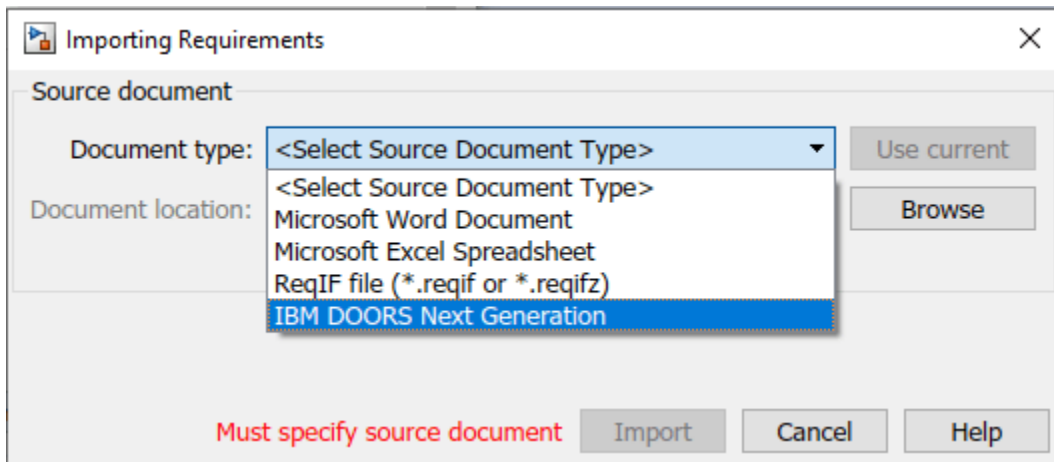
- *direct linking* requires that you modify the DOORS Next server configuration and install the **Simulink Requirements custom gadget**,
- *direct linking* requires HTTPS communication between your system browser and the local MATLAB process, which could present a security risk when using this same browser for external web pages,
- 1-way links are difficult to discover from DOORS Next side, and are entirely hidden unless linked artifacts are loaded in current MATLAB session,
- 2-way links may become difficult to manage in large multi-user projects or when switching between DOORS Next streams and changesets,
- you cannot control the Type of links from DOORS Next to MATLAB/Simulink, the links are always of generic "Link To" type,
- built-in analysis in Simulink Requirements product do not cover the *direct* links.

To resolve these limitations and to bypass most of the complications, Simulink Requirements offers an entirely different workflow option: you can cache a subset of DOORS Next requirements into an internally managed Simulink Requirements Set, then perform all linking and analysis in Simulink Requirements environment as you would do with the usual internally managed or imported entries.

You will not be able to edit cached DOORS Next contents locally, and you will not immediately see the updates when the sourced requirements are updated on the server, but you get the advantage of native linking support between Simulink artifacts, including drag-drop linking with Simulink objects, without disturbing the server side, and you can use all the features of Simulink Requirements product for reviewing and analysing links, including implementation and verification status, as well as change impact detection and management.

Capture DOORS Next Collections into Simulink Requirements Set

In the Requirements Editor, click **Import**. Select "IBM DOORS Next Generation" in **Document type** selector:



As before, you will be prompted for the DOORS Next login password. If this is your initial connection for the current MATLAB session, you will also be prompted to confirm the server URL and the username.

Document location selector will populate with names of all DOORS Next project available on the specified server. Once you select the Project to import from, additional option controls will appear:

Importing Requirements

Source document

Document type: IBM DOORS Next Generation Use current

Document location: AMR Sample Project Browse

DNG Module: AMR System Requirements Specification

Get requirements from:

Full module hierarchy (takes time if large module)

Filter by query (flat list of matched items)

Query Builder

Query history:

<Reuse from history>

Raw query string:

Destination(s)

Requirement Set: L:\OSLC_proxy\AMR_Sample_Project_3.slreqx Browse

Allow updates from external source

Import Cancel Help

Two different modes are supported for capturing DOORS Next contents into Simulink Requirements. You can import the specified module, including the hierarchical relationships between DOORS Next requirements, or you can switch into the **Filter by query** mode, which produces a flat list of matched requirements.

Importing Requirements

Source document

Document type: IBM DOORS Next Generation Use current

Document location: AMR Sample Project Browse

DNG Module: <Select module>

Get requirements from:

Full module hierarchy (takes time if large module)

Filter by query (flat list of matched items) ←

Query Builder ←

Query history:

<Reuse from history>

Raw query string:

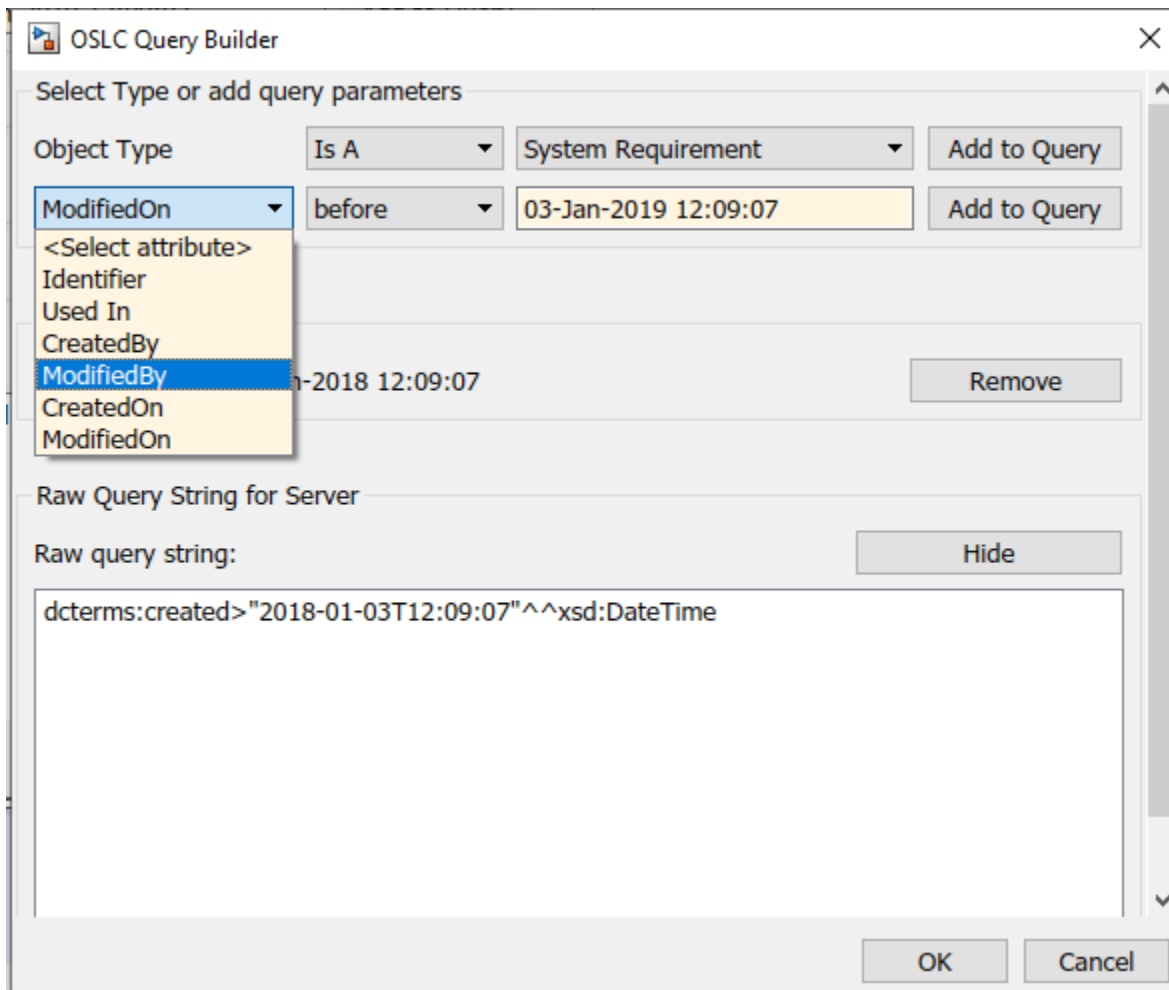
Destination(s)

Requirement Set: L:\OSLC_proxy\AMR_Sample_Project_3.slreqx Browse

Allow updates from external source

DNG query string must be specified Import Cancel Help

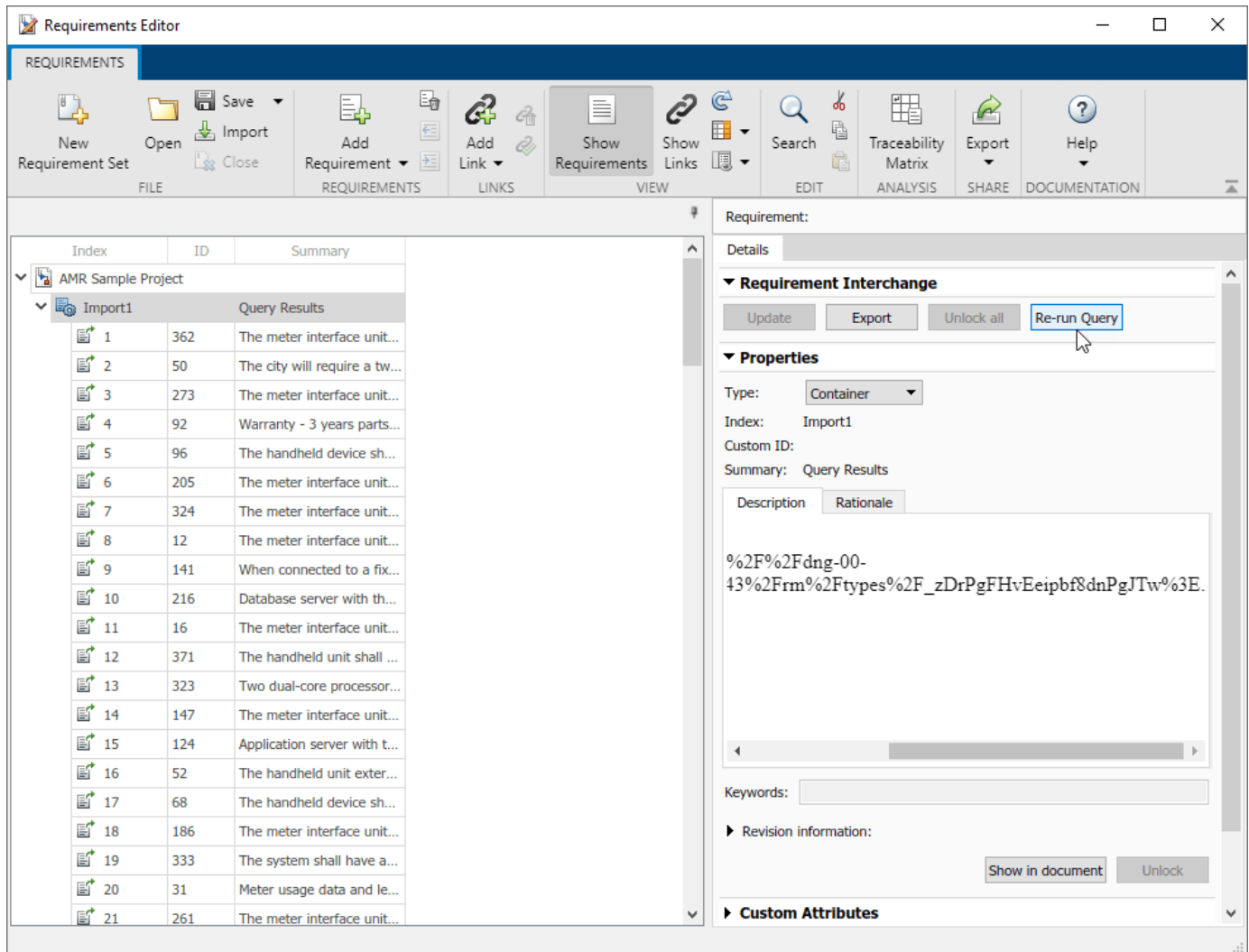
When using the **Filter by query** option, in most cases, you will not need to type the query expression manually, but use the **Query Builder** dialog to configure the filter:



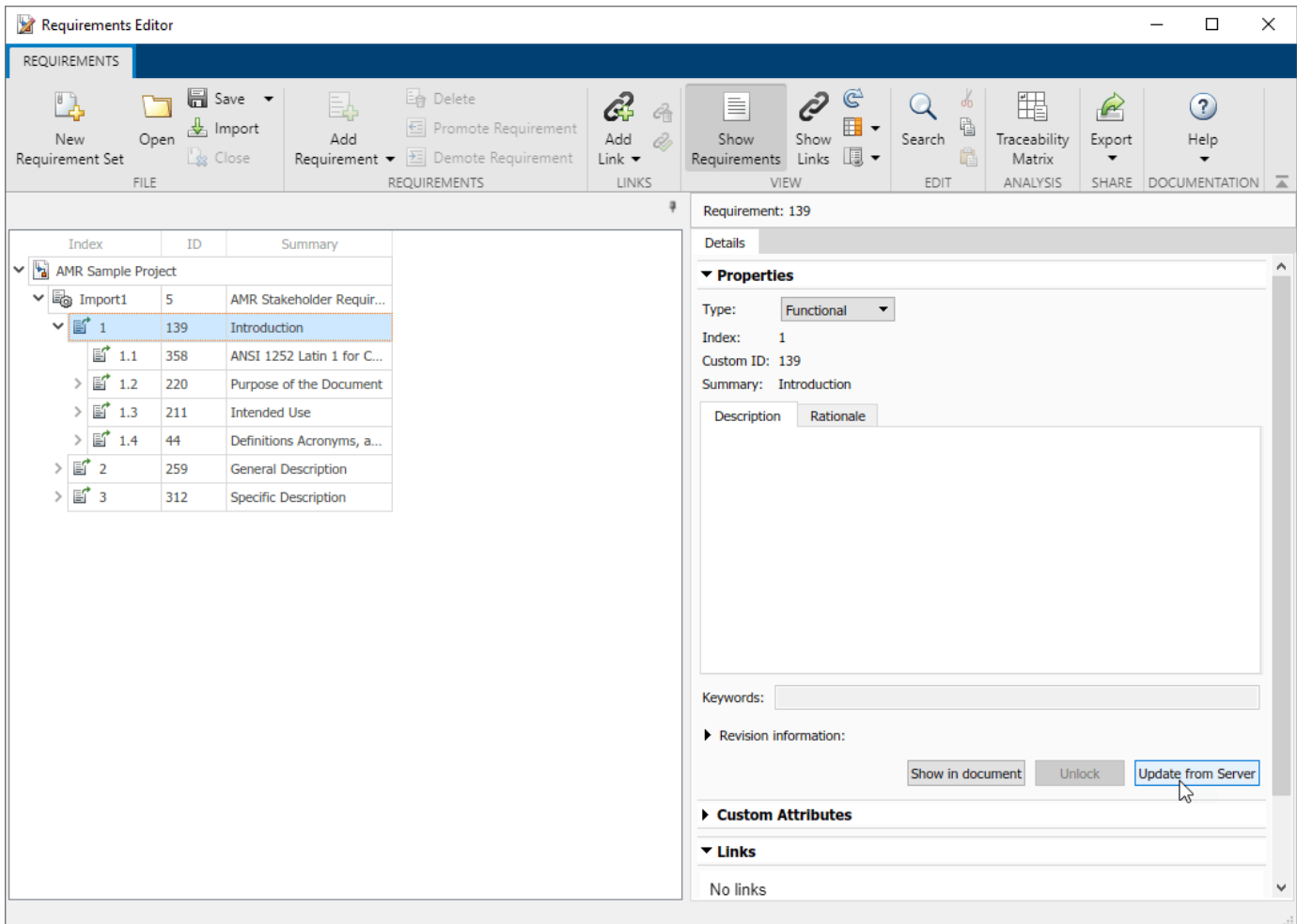
When you import using **Filter by query**, you can only use one filter at a time.

In both cases you get a top-level *Import node* with the ID that matches the name of your DOORS Next project. The Summary text of the *Import node* will indicate the parameters used when capturing data from DOORS Next. You can now work with the imported items as you would with the usual entries in Simulink Requirements:

- create links with related MBD artifacts and use all the built-in analysis capabilities.
- navigate to the original requirements in DOORS Next by clicking the **Show in document** button,
- refresh the captured content using the **Update from Server** button,
- when you save to a `.slreqx` file, the links are saved to a corresponding `.slmx` file.



The one essential difference, however, is that you cannot unlock and modify cached requirements: all the needed updates should happen on the server side. You then use the **Re-run query** button for the *Import node* (or the **Update from Server** button for a single requirement) to pull-in updates from the server.



You cannot import images or tables from DOORS Next to Simulink Requirements.

Linking with Captured DOORS Next Items

Now that you have captured DOORS Next requirements collection of interest into a Requirement Set and saved it to .slreqx file, you can easily established traceability between Requirements and design, then manage your Link Sets together with the rest of MBD artifacts, without affecting other users of the same DOORS Next project. For example, you can switch your Simulink model into **Requirements perspective** view, then open the captured set of DOORS Next requirements in requirements browser, and create links by drag-drop between the requirements browser and the blocks in your Simulink diagram. You will see linked blocks highlighted together with linked cached DOORS Next items in the requirements browser.

The screenshot displays the IBM Rational DOORS interface for a Simulink project named 'rmi_househeat'. The main workspace shows a Simulink block diagram with various components like 'F2C', 'Thermostat', 'House', and 'Cost Calculator'. A red circle highlights a requirement link labeled '370: Environmental Considerations' within the 'House' block. Below the diagram is a 'Requirements - rmi_househeat' panel with a 'View: Requirements' dropdown and a search bar. A table lists requirements with columns for Index, ID, and Summary. The row for Index 2.2 (ID 370) is highlighted in red, with a red arrow pointing to the 'Environmental Considerations' summary. To the right, a 'Property Inspector' panel shows details for Requirement 370, including its Type (Functional), Index (2.2), Custom ID (370), and Summary (Environmental Considerations). The 'Description' and 'Rationale' tabs are visible, and the 'Revision information' section shows SID: 370.

Index	ID	Summary
Import1	AMR Sample Project	
1	241	Introduction
2	357	General Description
2.1	191	Functions and Purpose
2.2	370	Environmental Considerations
3	167	System Requirements

Reviewing and Analyzing Traceability Data

As with links to internally-managed requirements, you can access more details about links when you select **Links** from the **View** drop-down. You can edit link properties such as Type, Description, Rationale, keywords, and Comments.

As with all other Simulink Requirements links, you enable display of Implementation and Verification status to check which requirements lack coverage, and which tests need to be rerun or updated.

When DOORS Next requirements on the server are updated or removed, you perform the automated update of cached requirements subsets in Simulink Requirements, and you check the **Links** view for flagged *stale* or *broken* links, to quickly identify the needed design or testing changes.

The screenshot displays the IBM DOORS Next interface with a Simulink model titled "rmi_househeat" and its associated requirement links. The main workspace shows a Simulink diagram with various blocks like "F2C", "Thermostat", "House", and "370: Environmental Considerations". A red circle highlights the "House" block in the diagram. Below the diagram is a table of requirement links, with the last row highlighted in blue. To the right, the "Property Inspector" shows the details for the selected link, with a red circle around the "Source" and "Destination" fields.

Requirement links - rmi_househeat

Label	Source	Type	Destination
top-level diagram in ...	Sum2	Implements	fuelsys_fewReqs
Two-stage furnace g...	HeatFlow	Implements	@Simulink_requirement_item_8
Two-stage furnace g...	HeatFlow	Implements	@Simulink_requirement_item_8
Adequate wireless sp...	Thermostat	Implements	Item 112 in AMR Sample Project
370: Environmental C...	House	Implements	370 Environmental Considerations

Property Inspector

Link: 370: Environmental Considerations (A...)

Details

Properties

Source: [House](#)

Type: **Implements**

Destination: [370 Environmental Consid...](#)

Description

Rationale

370: Environmental Considerations (AMR_S

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.

Simulink Requirements imports IBM DOORS Next requirements as `slreq.Reference` objects, which are also 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 1-27. You can then link MATLAB or Simulink Model-Based Design artifacts or other linkable items on page 2-32 with the referenced requirements in the Simulink canvas or Requirements Editor. You can also navigate from the referenced requirement in Simulink Requirements to the original requirement in DOORS Next.

With direct linking, you link directly from MATLAB or Simulink to 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 Simulink Requirements 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. At the MATLAB command prompt, enter:

```
slreq.dngConfigure
```

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. If you do not see a port number, enter the default value of 443. In the Server Login Name and Server Login Password dialog boxes, enter your login credentials. 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>**. For more information about configurations, see “Specifying and Updating the IBM DOORS Next Configuration” on page 7-32.

MATLAB then tests the connection 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 section of `slreq.dngConfigure`.

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 1-27.

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, select a model element, then select the referenced requirement in the Requirements Editor and click **Add Link > Link from Selected Simulink Object**. See “Requirement Links” on page 2-32 for more information.

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 you want to use to insert 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 any 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 7-32.

Install the Simulink Requirements widget in IBM DOORS Next. For more information, see “Install the Simulink Requirements Widget in IBM DOORS Next” on page 5-3. To confirm the widget is operating as expected, 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 Rational DOORS Next interface. At the top, there is a header for 'Requirements Management (/rm)' and 'AMR Sample Project'. Below the header, there are navigation tabs: 'Project Dashboard', 'Artifacts', 'Reviews', and 'Reports'. The 'Artifacts' tab is active, showing a table of artifacts. A 'Mini Dashboard' widget is pinned to the left side of the page, displaying 'Simulink Requirements' configuration. The configuration shows 'Project: AMR Sample Project' and 'Context: Requirements Test Project'. A table within the widget shows a selected requirement with ID '96' and the text 'The handheld device shall interfaces with...'. Below the table is a button labeled 'Query links from SL'. The main 'Artifacts' table has columns for 'Folders', 'Views', 'ID', and 'Name'. The selected requirement (ID 96) is highlighted in orange. The table content is as follows:

Folders	Views	ID	Name
Sample Project		92	Warranty - : parts on-site next busine
Admin			
_AMR Information Architect		96	The handhe shall interfa the city's ba software.
Requirements			
_AMR Hazards and Risks a		324	The meter i unit shall be

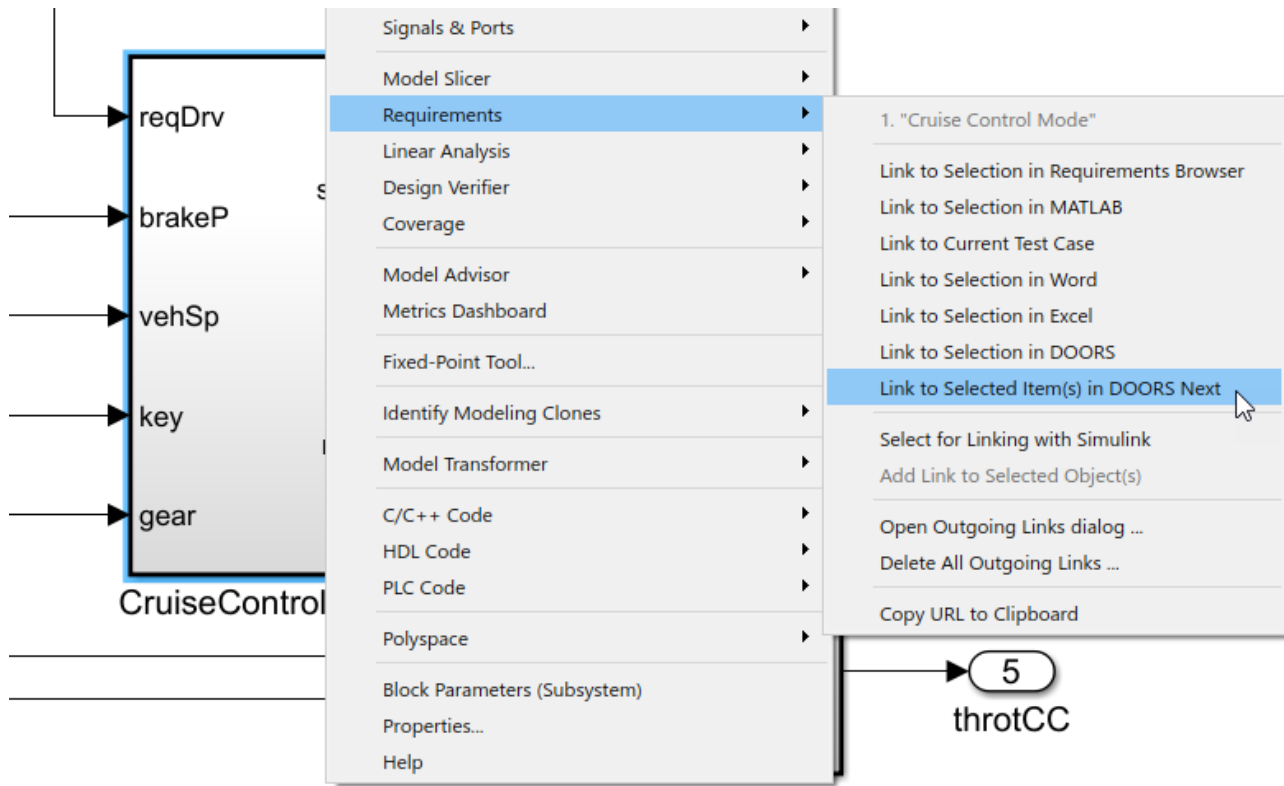
You can verify that MATLAB is receiving information about your selection in DOORS Next. At the MATLAB command prompt, enter:

```
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 as expected, you can create links between Simulink linkable items and DOORS Next in one click 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 Simulink Requirements widget in the **Mini Dashboard**.
- 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 Simulink Requirements 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:

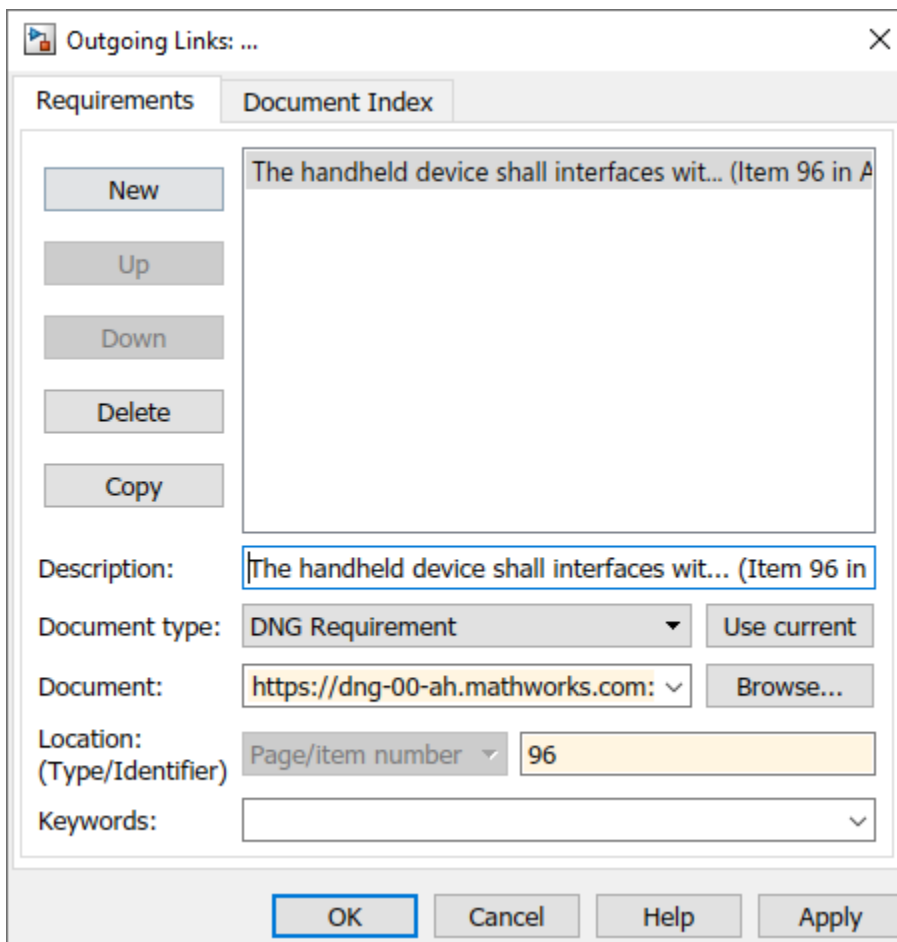
- 1 In Simulink, right-click the Simulink model element that you want to link to the selected IBM DOORS Next requirements. Select **Requirements > Link to Selected Item(s) in DOORS Next** from the context menu.
- 2 The DOORS Link Target dialog box appears, but no information is populated. Set the **Project 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.
- 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 DNG Requirement.

- 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.
-
- 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.
-
- 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. You can post-insert backlinks in your DOORS Next project. See “Insert Missing Backlinks” on page 7-31.

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 all 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 Simulink Requirements. 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 Simulink Requirements.

Additionally, the backlinks in DOORS Next will be 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 2-50.

Navigate Between DOORS Next Requirements and Directly Linked Items

Once you've directly linked a linkable item on page 2-32 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 **Details** 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**. Backlinks are listed under **Links to**.
- 5 Click the backlink to navigate to the linked item in MATLAB or Simulink.

The screenshot displays the IBM DOORS Next interface for a project named "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 "Contents" for 926 is "The meter interface unit shall take readings of pressure with a maximum interval of 1 second".

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

The right-hand pane shows the details for the selected artifact (ID 926). It includes the module name "744: Meter Interface Subsystem Requirements Specification", a description, component "AMR with CM", team ownership "AMR with CM", and creation date "Sep 29, 2020, 10:28:35". Below this, there is a section for "Artifact Links (1)" which contains a link to "sampleTime in crs_controller.slx".

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). Simulink Requirements 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`.

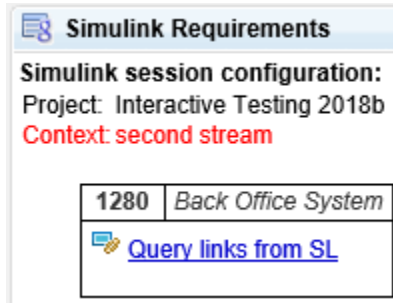
Updating Stored Stream or Changeset by API

Simulink Requirements 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 Simulink Requirements Widget to Synchronize and Update Session Context

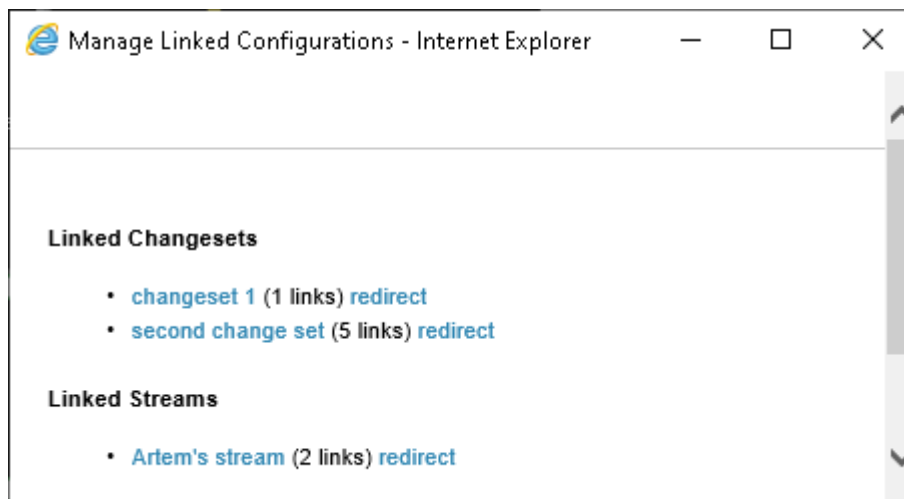
The Simulink Requirements widget displays information about the current configuration stream context in Simulink Requirements. The widget indicates a mismatch between the active configuration stream contexts in Simulink Requirements and in IBM DOORS Next by displaying and highlighting the active Simulink Requirements 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 7-32 or by using the **Query Links from SL** hyperlink in the Simulink Requirements widget.

- 1 In DOORS Next, under the Simulink Requirements 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 is updated, 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.dmgConfigure

Related Examples

- “Link with Requirements in IBM DOORS Next” on page 7-4

More About

- “Import Requirements from IBM DOORS Next” on page 1-27
- “Manage Navigation Backlinks in External Requirements Documents” on page 2-50

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 5-10.

- 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 tags to apply to the links that you create.

For more information about user tags, see “User Tags and Requirements Filtering” on page 5-11.

- 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 7-35, 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**.

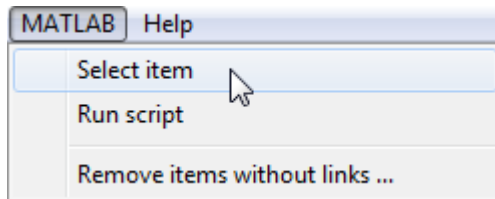
A single navigation object is inserted at the selected requirement.

- 5 Double-click the navigation object in DOORS to highlight the Simulink objects that are linked to that requirement.

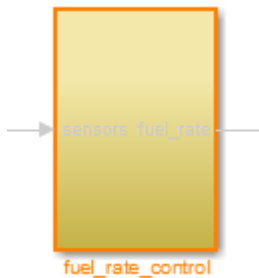
Navigate Between IBM Rational DOORS Requirement and Model Object

In “Insert Navigation Objects into IBM Rational DOORS Requirements” on page 7-35, 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**.

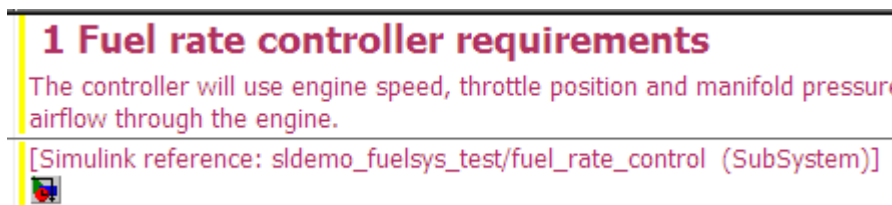


Your version of the `sldemo_fuelsys` model opens, with the `fuel_rate_control` subsystem highlighted.



- 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 7-35, 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 7-43.

- 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 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

Red triangles are visible next to the '1.1.1 engine torque' and '1.2 transmission' rows, indicating links.

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

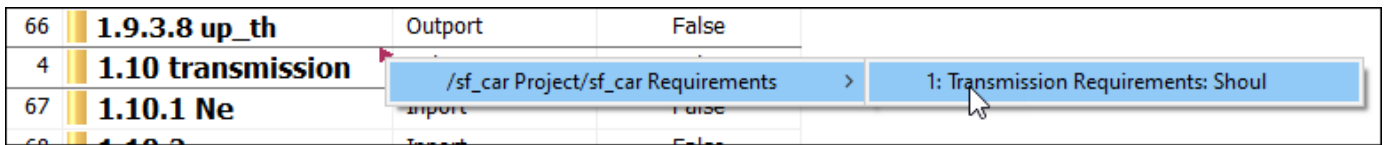
Navigate Between Requirements and the Surrogate Module in the DOORS Database

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.

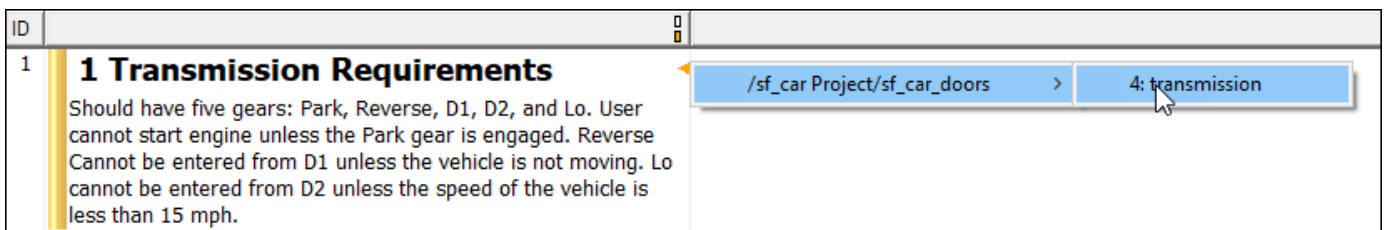


- 2 Select the requirement name.

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 11-35.
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 7-45.
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

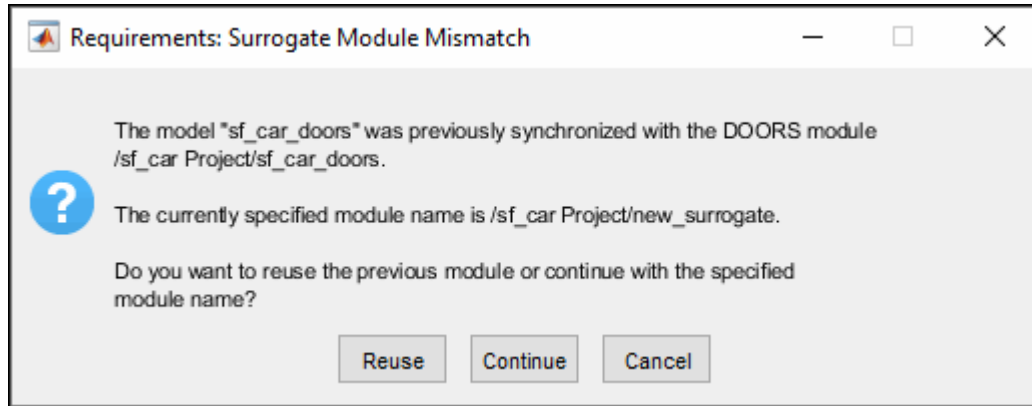
You can synchronize the same Simulink model with a new DOORS surrogate module. For example, you might want the surrogate module to contain only objects that have requirements to DOORS, rather than all objects in the model. In this case, you can change the synchronization options to reduce the level of 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 7-39, 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 7-39 and “Create Links Between Surrogate Module and Formal Module in an IBM Rational DOORS Database” on page 7-40.

- 1 Open the `sf_car_doors` model that you synchronized in “Synchronize a Simulink Model to Create a Surrogate Module” on page 7-39 and again in “Create Links Between Surrogate Module and Formal Module in an IBM Rational DOORS Database” on page 7-40.

- 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 7-39.

For information about how the RMI resolves the path to the requirements document, see “Document Path Storage” on page 11-35.

- 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 7-46:

- 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.

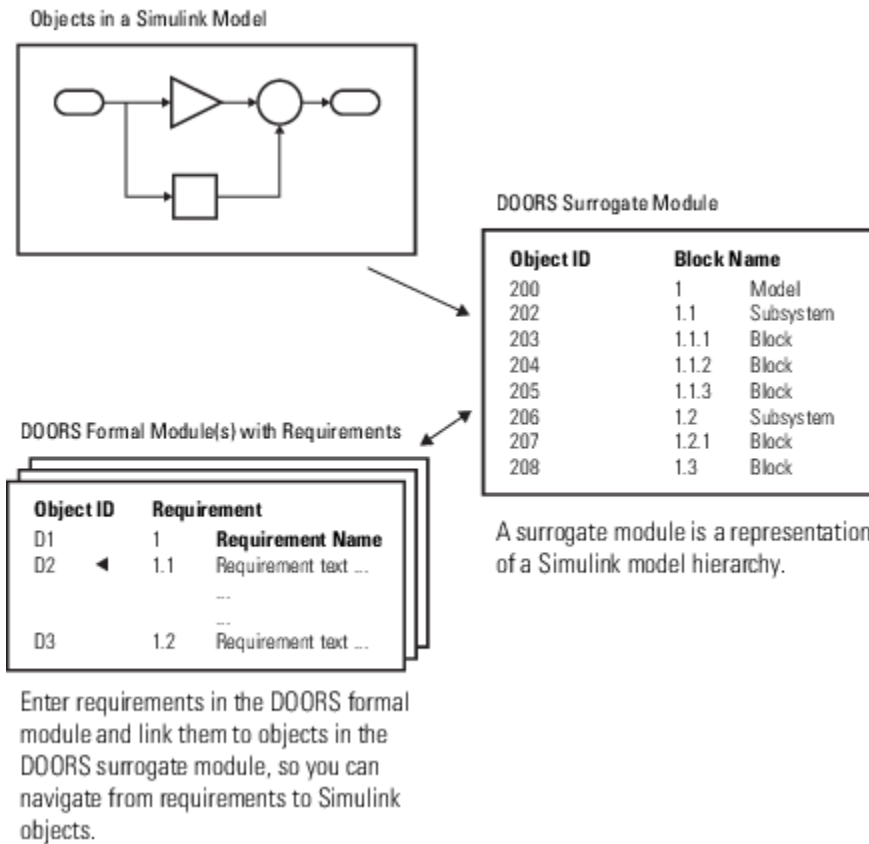
When you synchronize a model for the first time, the DOORS software creates a surrogate module. The surrogate module contains a representation of the model, depending on your synchronization settings. (To learn how to customize the links and level of detail in the synchronization, see “Customize IBM Rational DOORS Synchronization” on page 7-43.)

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.



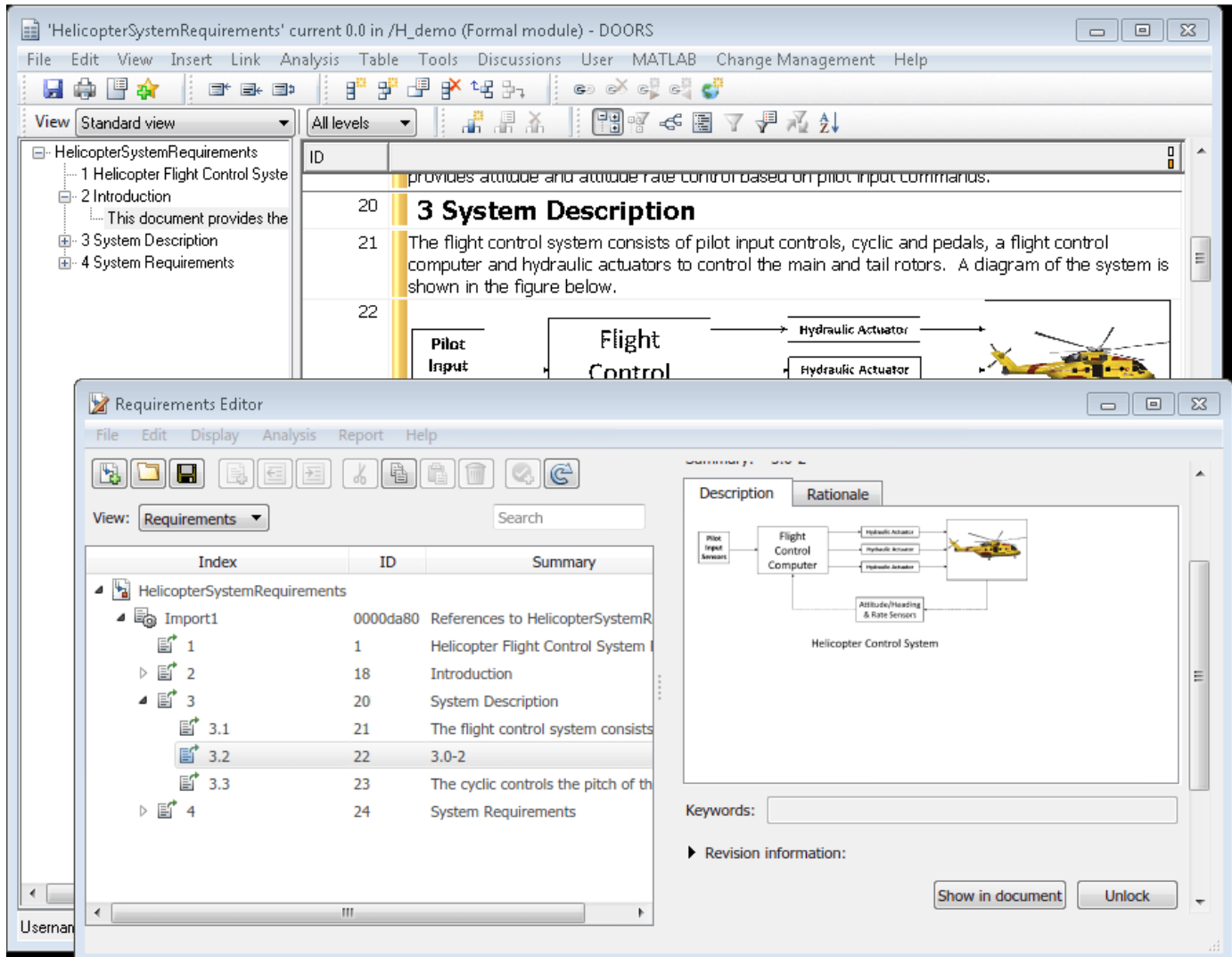
Advantages of Synchronizing Your Model with a Surrogate Module

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 Simulink Requirements for Interaction with Microsoft Office and IBM Rational DOORS” on page 5-2.

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 Simulink Requirements 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 Simulink Requirements.
- 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 Simulink® Requirements™” on page 5-16.

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 1-33.

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 7-59.

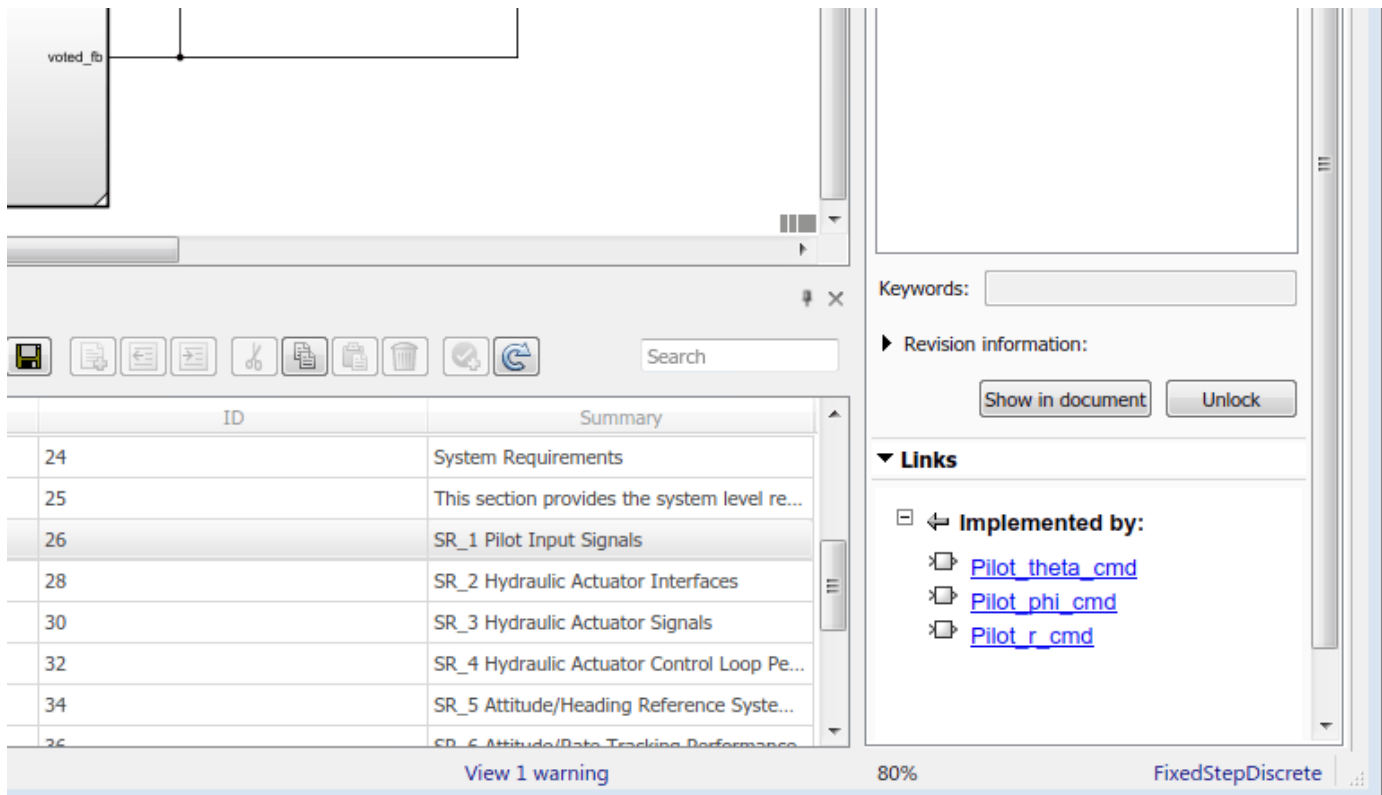
Before you import your DOORS module, be sure that you've added all desired requirements attributes. You cannot import additional attributes to Simulink Requirements after the original import.

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 to Test Cases from Requirements” and “Working with IBM Rational DOORS 9 Requirements” on page 7-50.

Update Requirements to Reflect DOORS Changes

If the source requirements in DOORS change, you can update the imported references in Simulink Requirements.

- Select the top-level node that corresponds to updated DOORS module.
- Click the **Update** button.

Follow the steps in “Update Imported Requirements” on page 1-52.

If you have added attributes to your DOORS module since the original import to Simulink Requirements, 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 Simulink Requirements.

Synchronizing Links and Navigation from DOORS

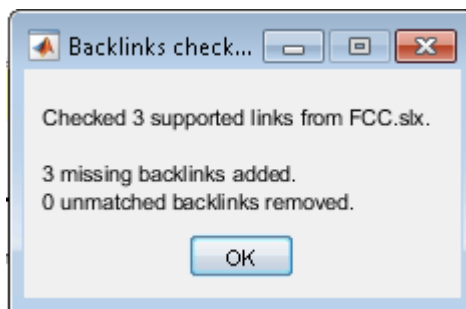
You can bring traceability data into DOORS for easier navigation from original requirements to design and tests. To synchronize your Simulink Requirements links into DOORS:

- Select Links from the **View** drop-down.
- Locate and right-click the Link Set that has new links.

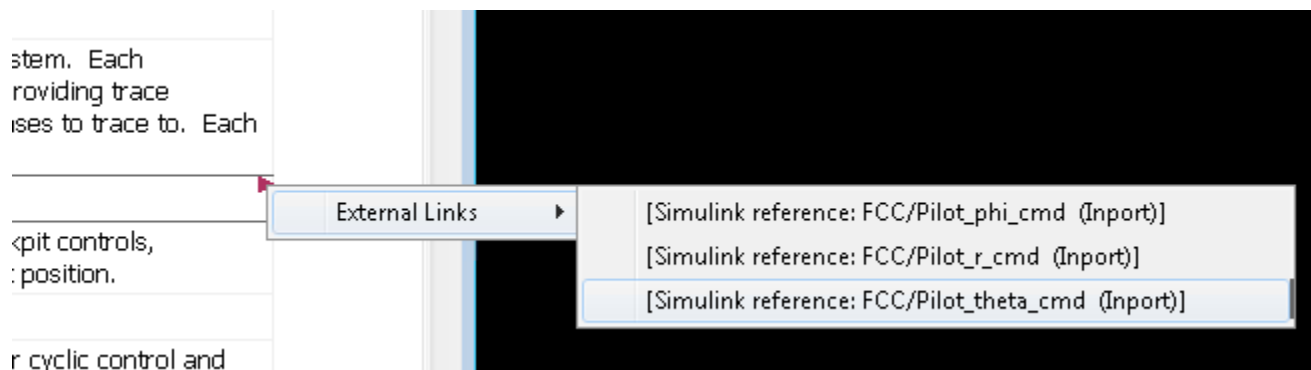
- Select **Update Backlinks** shortcut at the bottom of context menu.

Simulink Requirements 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:

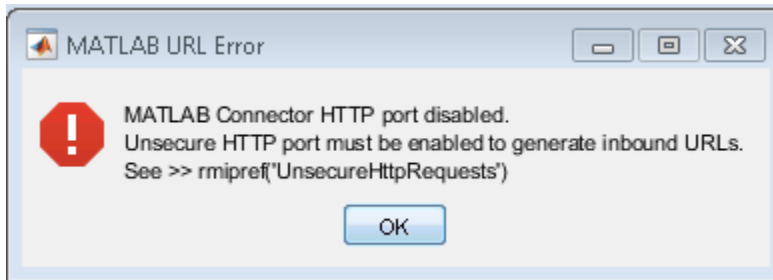


See “Manage Navigation Backlinks in External Requirements Documents” on page 2-50 for general information about managing links from external documents.

Embedded HTTP Connector

Navigation from external applications to MATLAB/Simulink relies on the built-in HTTP server in MATLAB. Simulink Requirements 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

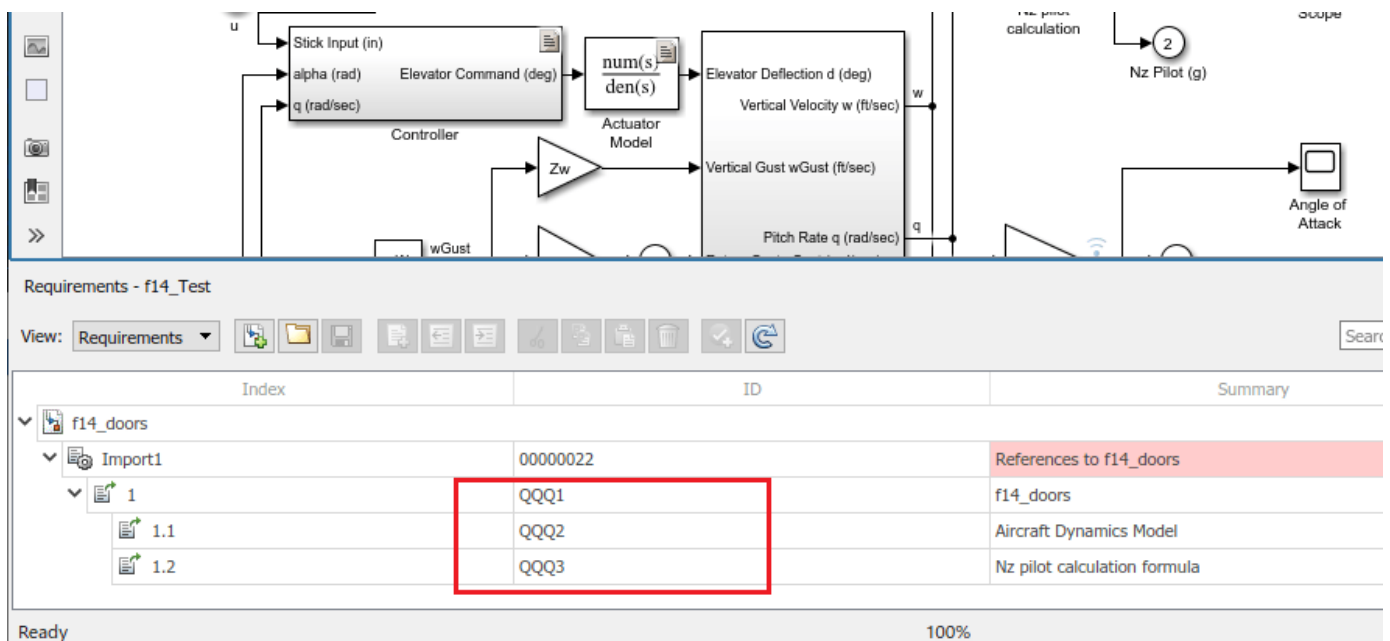
At some point after linking MBD artifacts with requirements in DOORS, you may have created Baselines for linked modules. By default, your links stored in Simulink Requirements will still navigate to the current version of the linked modules. If you want to lock your design version to a baseline version of requirements, Simulink Requirements 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 Simulink Requirements Set. Also, when updates received from DOORS do not include some Custom IDs that are present in Simulink Requirement Set, the corresponding items are assumed to be deleted in DOORS, and will be cleaned-up from Simulink Requirements Set. With this comes the following danger: if DOORS user has modified the module prefix in DOORS before performing the **Update** for Simulink Requirements Set, none of the existing Custom IDs will match, because DOORS module prefix is a part of ID, and all IDs known on Simulink Requirements 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
f14_doors		
Import1	0000022	References to f14_doors
1	QQQ1	f14_doors
1.1	QQQ2	Aircraft Dynamics Model
1.2	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:

The screenshot displays a system model and a Requirement links table. The model shows a Controller block with inputs for Stick Input (in), alpha (rad), and q (rad/sec), and an output for Elevator Command (deg). It is connected to a Model block, which outputs Vertical Gust wGust (ft/sec) and Pitch Rate q (rad/sec). The Actuator Model block outputs f14_doors.slreqx:15 and f14_doors.slreqx:16. The Requirement links table below shows the following data:

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

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 a Simulink model interface with a 'Requirement links - f14_Test' window open. The window shows a table of resolved links. The table has columns for Label, Source, Type, and Destination. The 'Label' column contains requirement IDs and their updated labels. The 'Source' column lists the source components (Actuator Model, Controller). The 'Type' column shows the relationship (Implements). The 'Destination' column lists the target components and requirement IDs.

Label	Source	Type	Destination
f14_Test.slmx*	Changed source: 0/4		Changed destination: 4/4
00000022: References to f14_doors (f14_...)	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 (f1...)	Controller	Implements	00000022 References to f14_doors

See Also

Related Examples

- “Managing Requirements for Fault-Tolerant Fuel Control System (IBM Rational DOORS)” on page 7-59
- “Import Requirements from IBM Rational DOORS by using the API” on page 1-87

More About

- “Configure Simulink Requirements for IBM Rational DOORS Software” on page 7-2
- “Import Requirements from IBM Rational DOORS” on page 1-33

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 6-14.

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 Simulink Requirements for Interaction with Microsoft Office and IBM Rational DOORS” on page 5-2.

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.

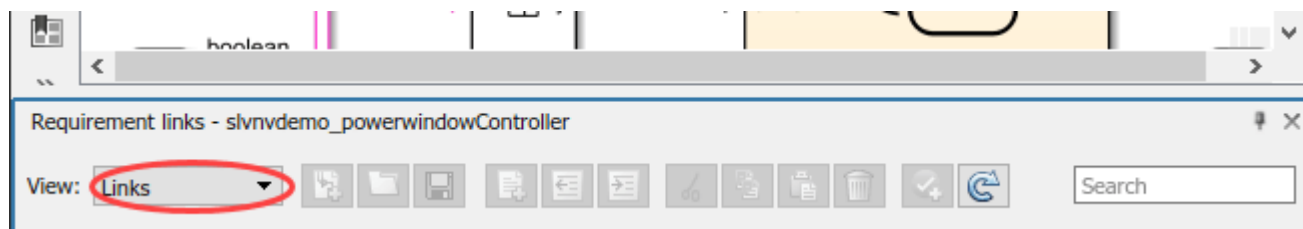
```
open_system('slvndemo_fuelsys_doorsreq');
```

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.



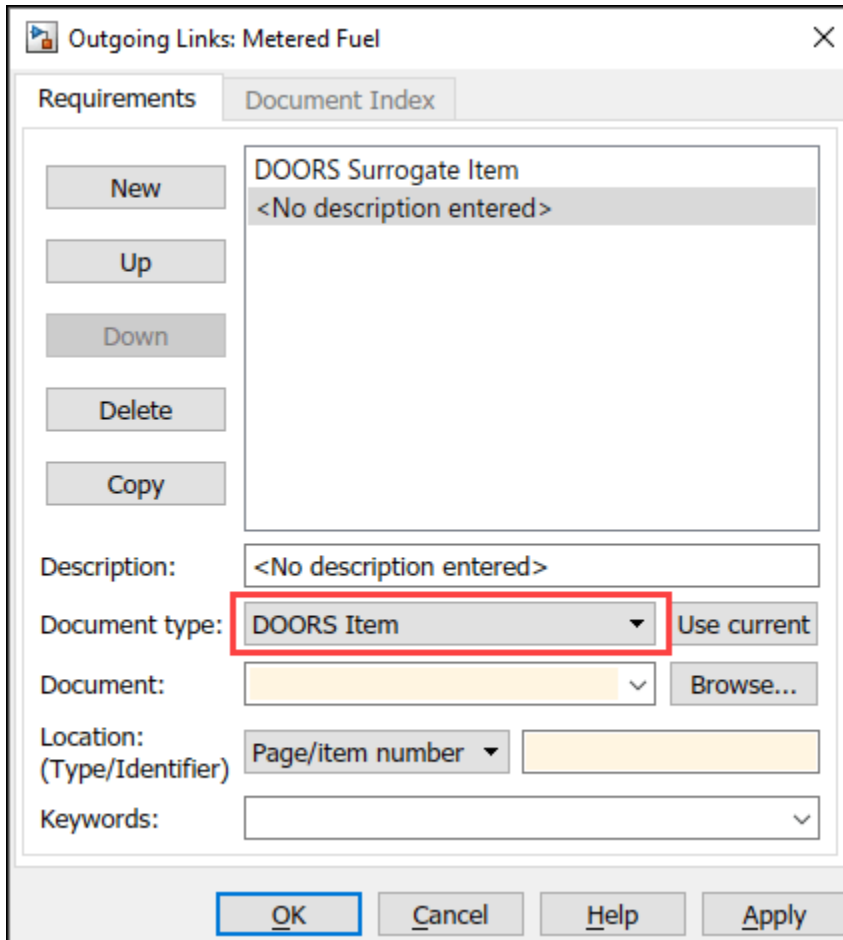
Linking via Outgoing Link Dialog

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');
```

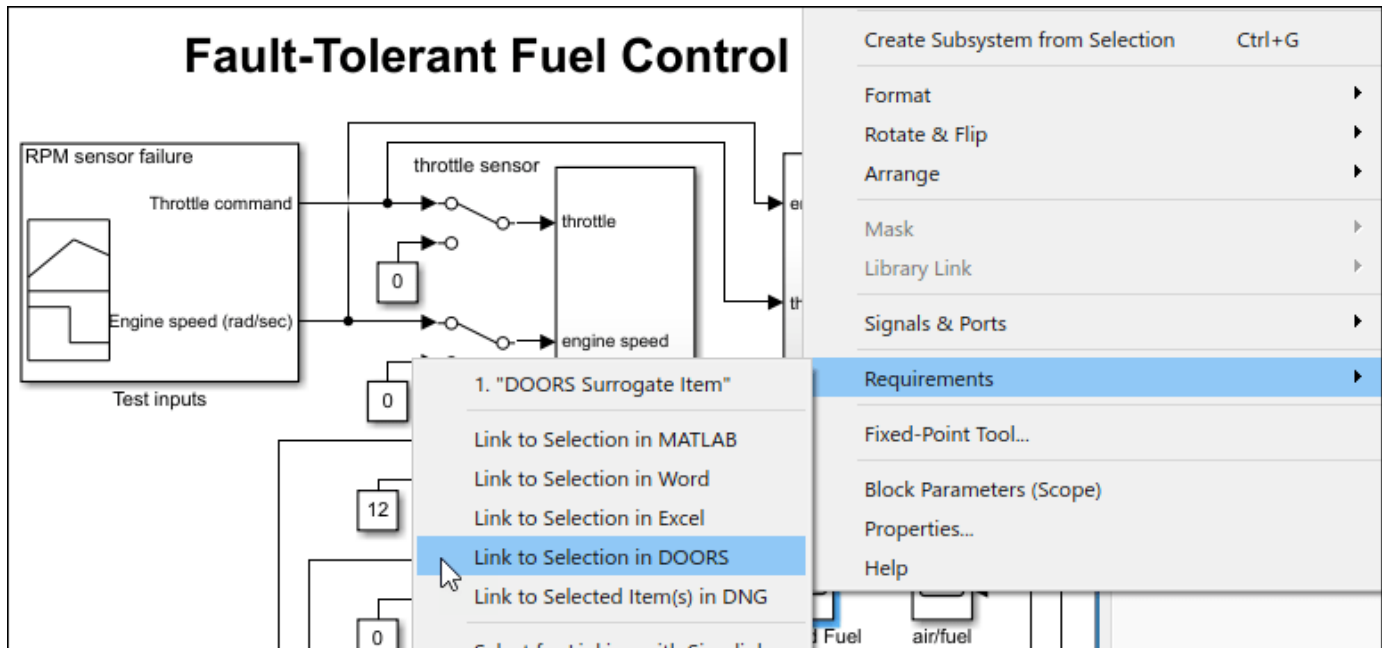
- Right-click the block and select **Requirements > Open Outgoing Links dialog...** from the context menu. The Outgoing Link dialog opens.
- Click **New** to create a new requirement.
- Select **DOORS Item** in the **Document type** drop-down box.



- 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.

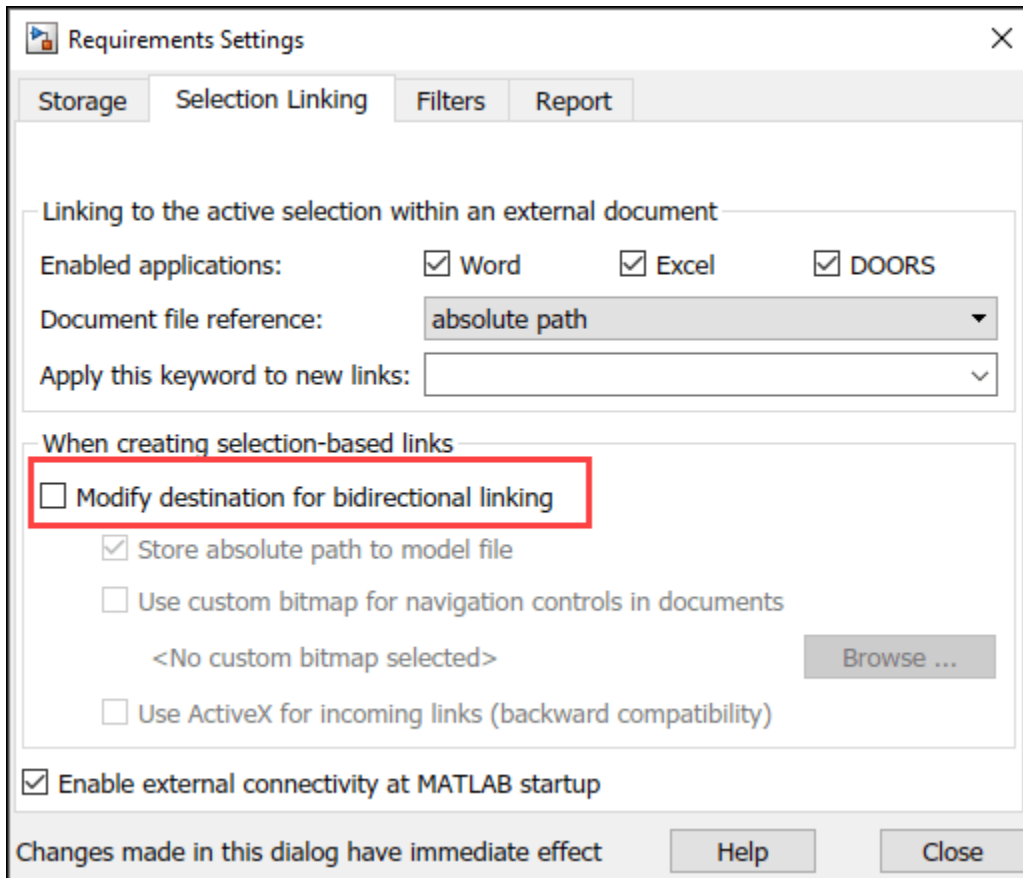
Linking via Context Menu Shortcuts

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.

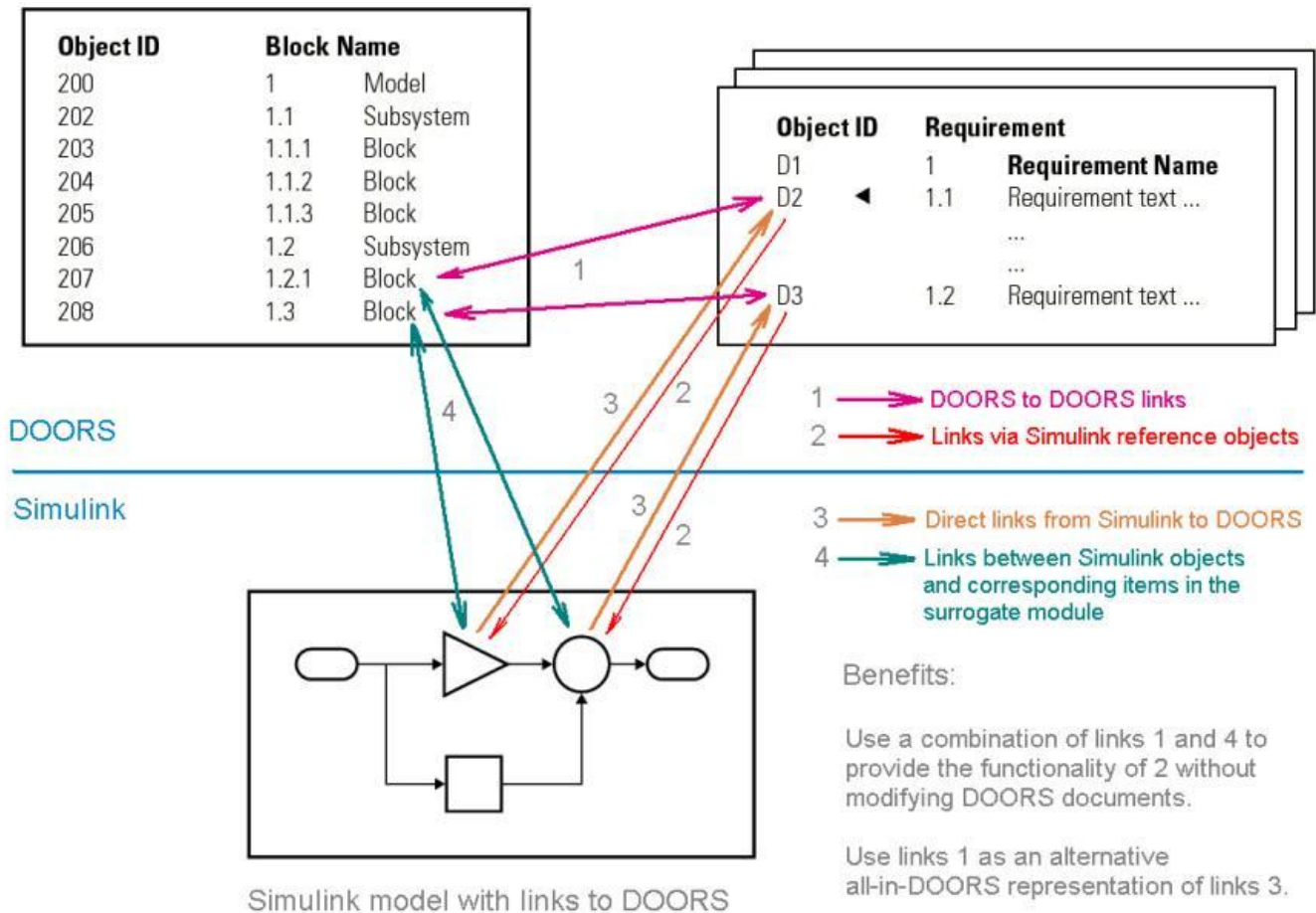
About Surrogate Modules and Synchronization

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:

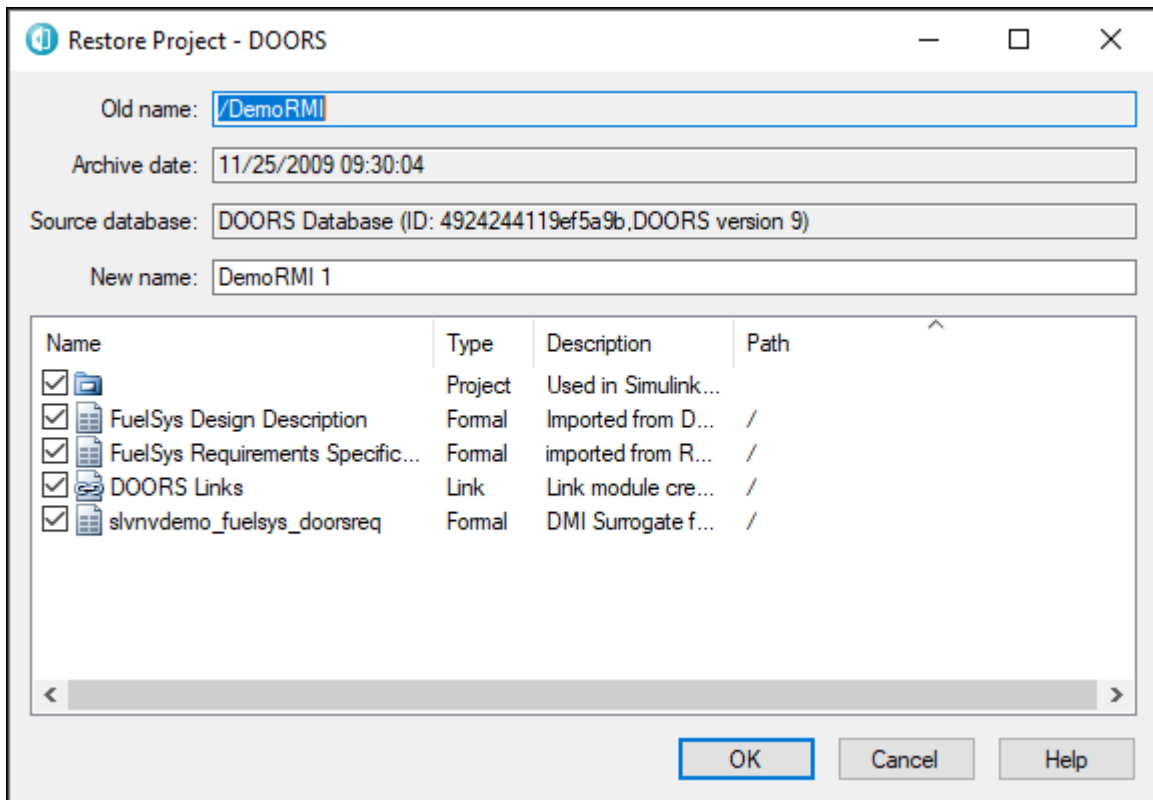
ID	Block Name	Block Type	Block Deleted?
52	1.11.5.6.11 f(theta)	Fcn	False
53	1.11.5.6.12 g(pratio)	Fcn	False
54	1.11.5.6.13 threshold = 0.5	Switch	False
55	1.11.5.6.14 Throttle Flow, mdot (g/s)	Outport	False
56	1.11.5.7 Mass Airflow Rate	Outport	False
57	1.11.5.8 MAP (bar)	Outport	False
58	1.11.6 o2_out	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 6-14.



- 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:

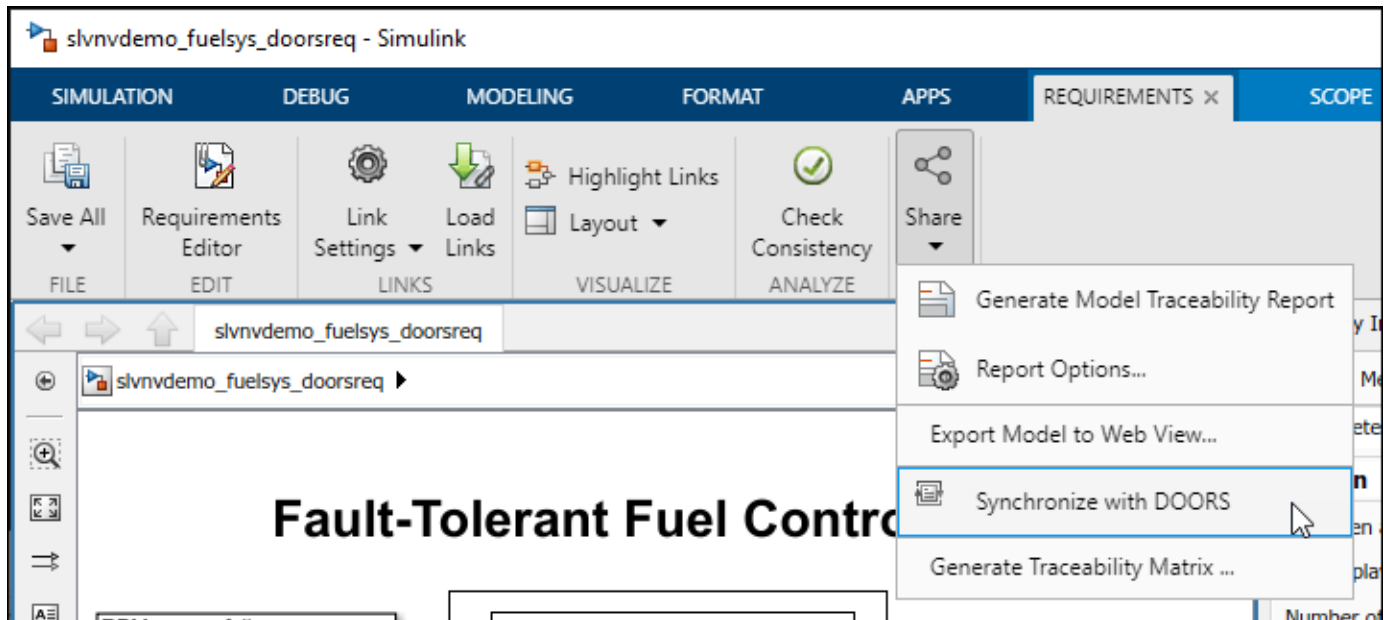
ID	Block Name	Block Type	Block Deleted?
13	1.11.1 engine speed	Import	False
14	1.11.2 throttle angle	Import	False
15	1.11.3 fuel	Import	False
16	1.11.4 Mixing & Combustion	SubSystem	False
17	1.11.4.1 fuel rate	Import	False
18	1.11.4.2 air flow	Import	False
19	1.11.4.3 Constant4	Constant	False
20	1.11.4.4 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:

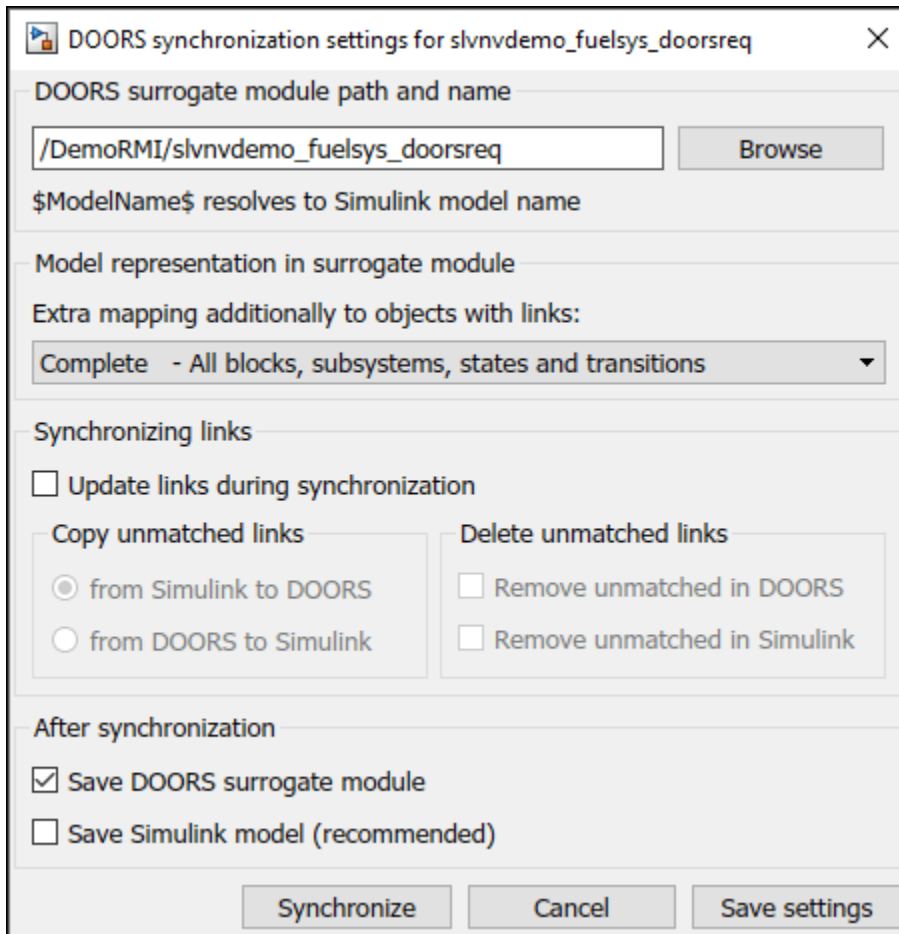
- Click **1.11.4.1 fuel rate** in the **slvnvdemo_fuelsys_doorsreq** surrogate module.
- In main menu of the module window, click **MATLAB > Select Item**. A correct subsystem diagram opens and the corresponding input is highlighted.

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.

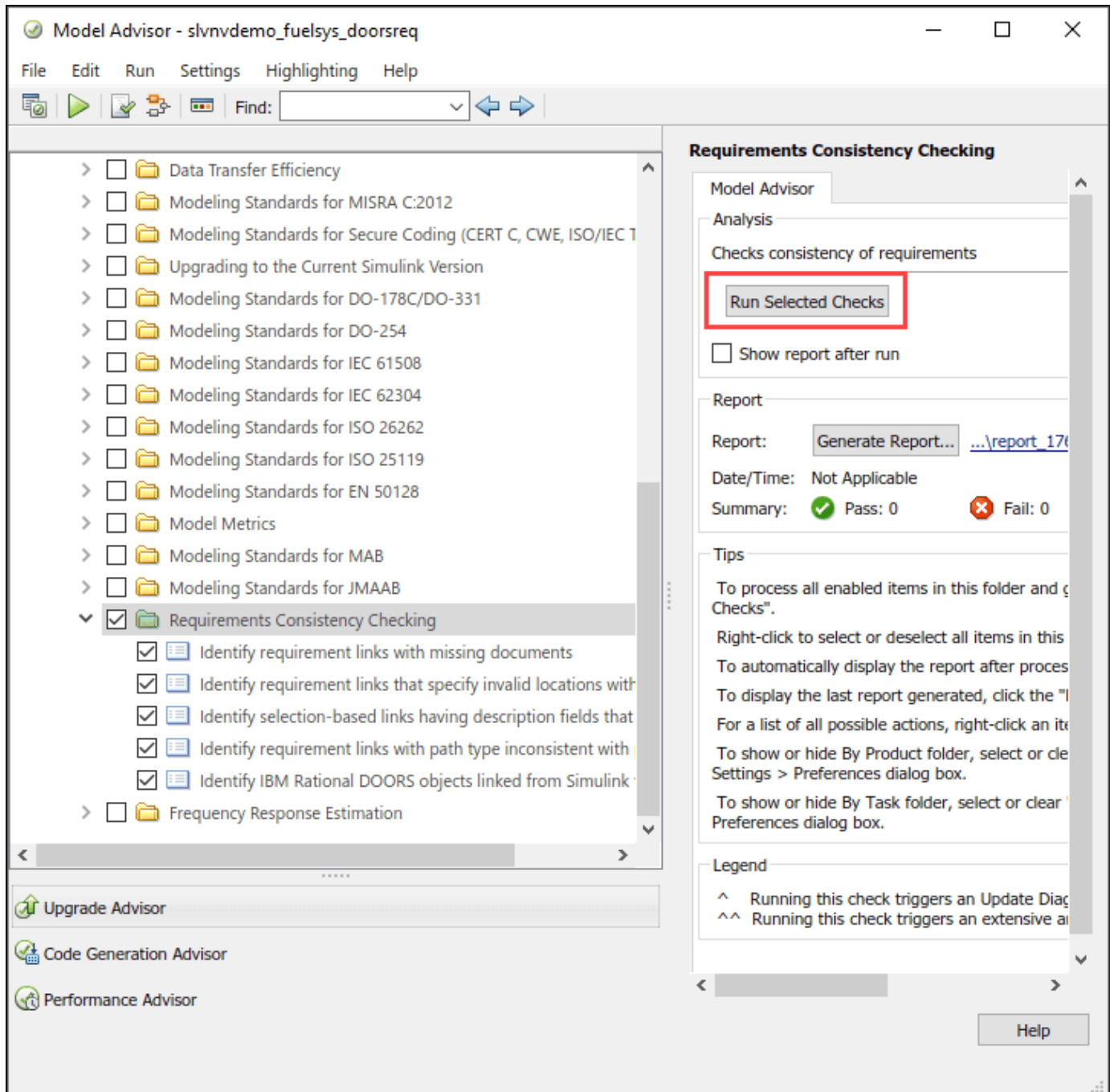
Using Model Advisor for RMI Consistency Checking

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.

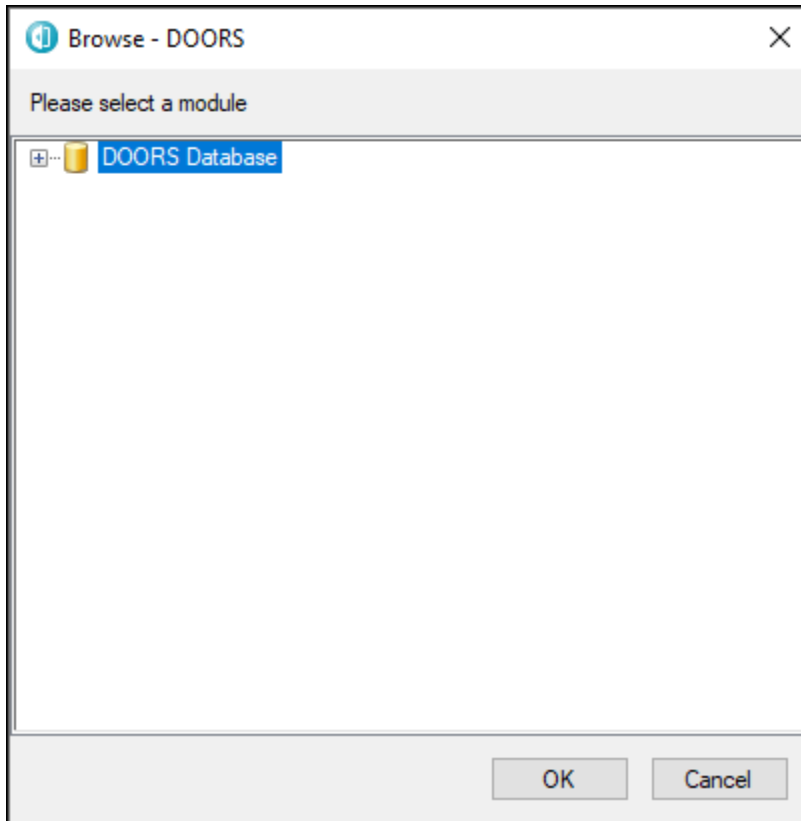
```
rmidemo_callback('locate', ['slvndemo_fuelsys_doorsreq/fuel_rate_controller/' ...
    'Airflow calculation']);
```

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.

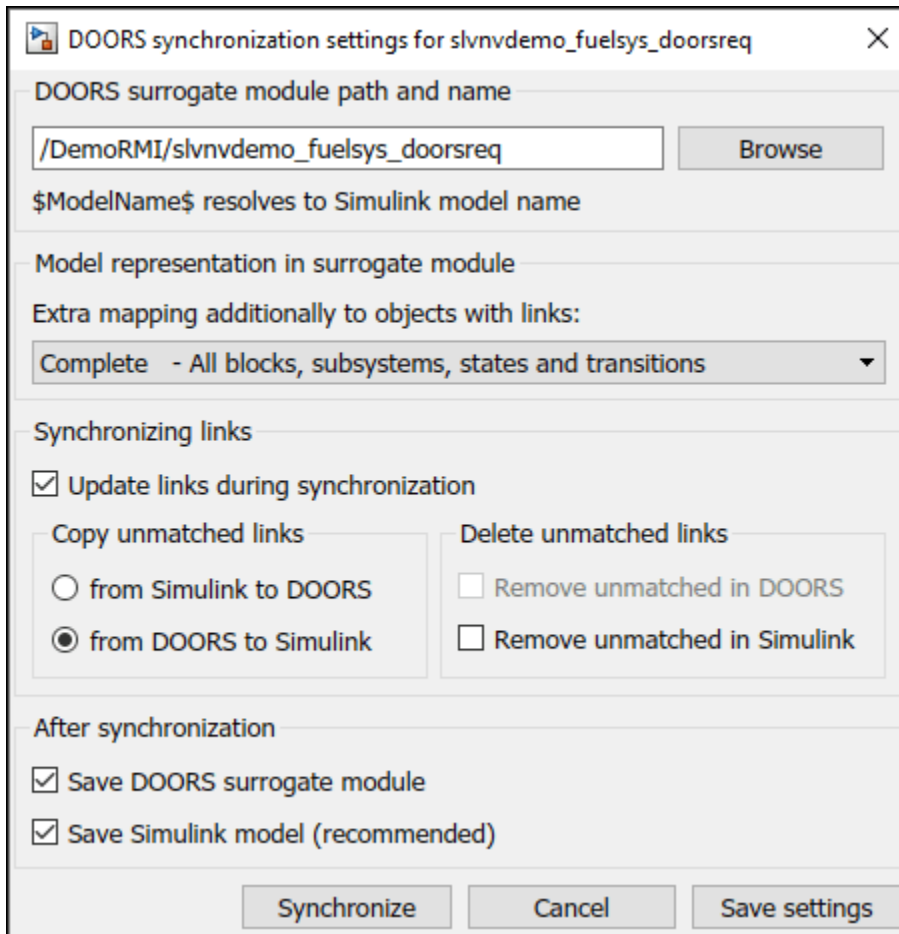


- Locate the restored **FuelSys Design Description** module in your database and select it with a mouse.
- Click **OK** to close DOORS database browser.
- Click **Run This Check** again. The check should now pass.
- Re-try navigation: right-click the **Airflow** calculation subsystem in the model and 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

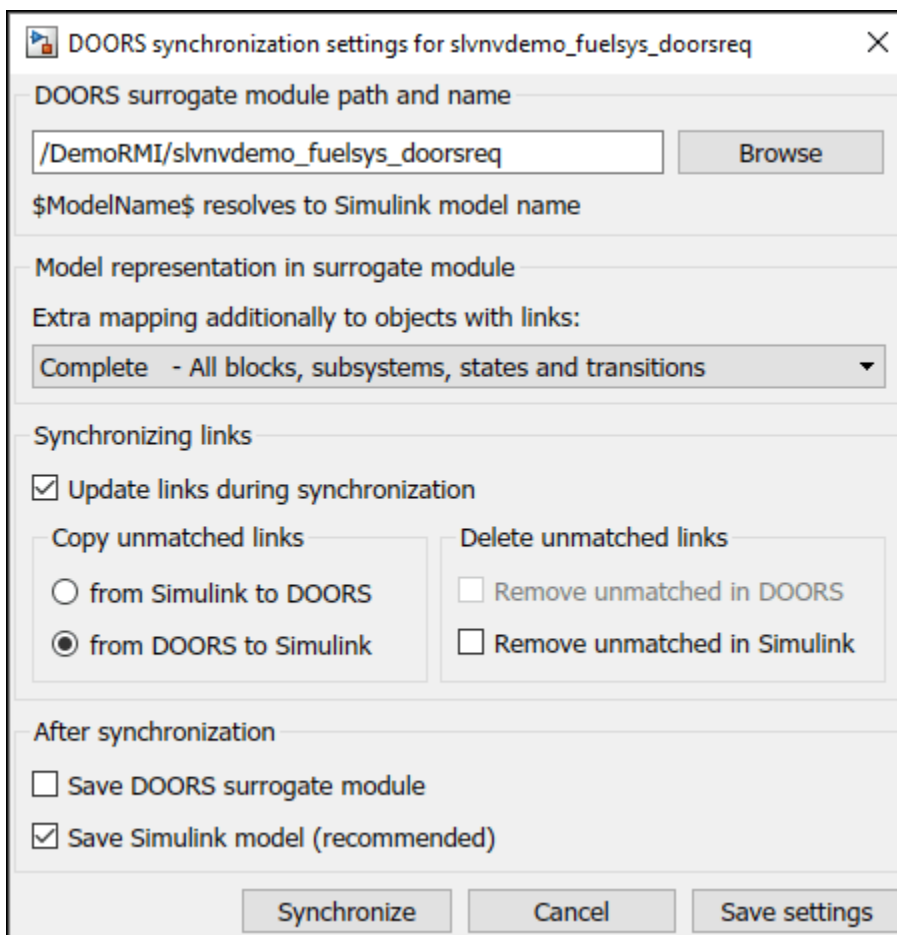
Synchronization via surrogate module provides a convenient way to propagate system requirements updates in DOORS to corresponding Simulink implementation elements. To demonstrate this workflow, the restored project contains DOORS links from the surrogate module to the **FuelSys Requirements Specifications** 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.

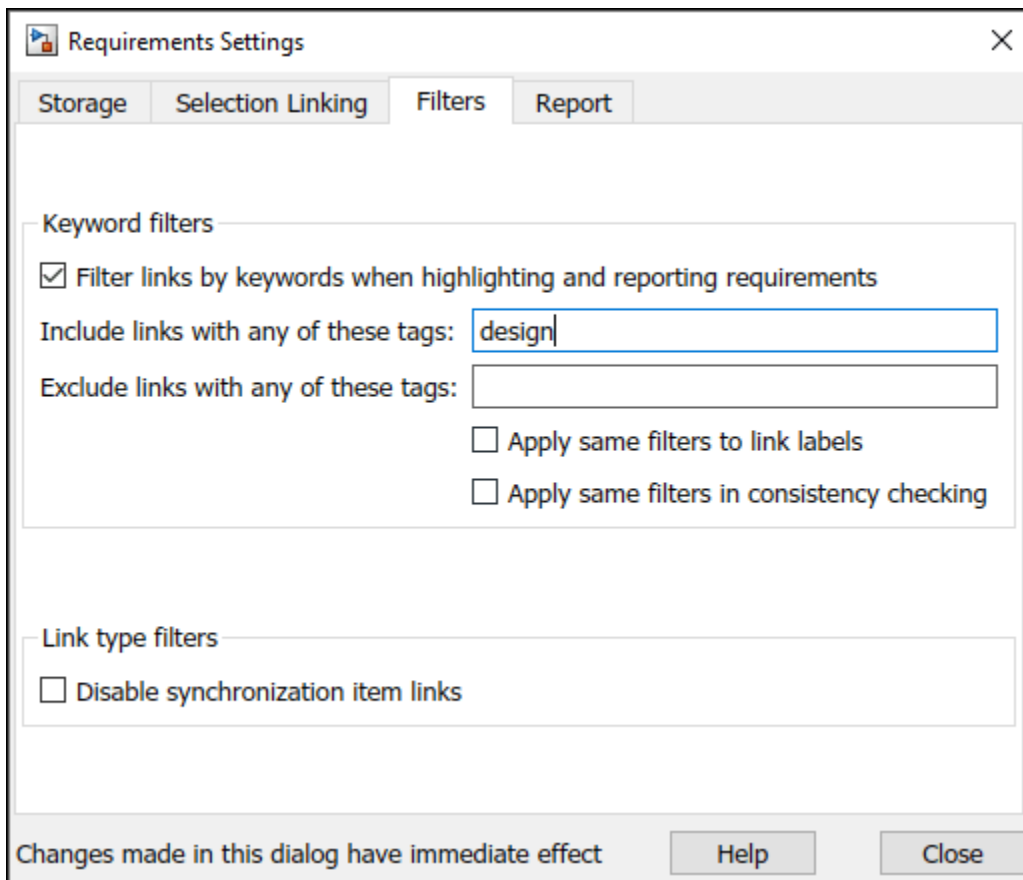
Review Your Changes Using User Tags

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:

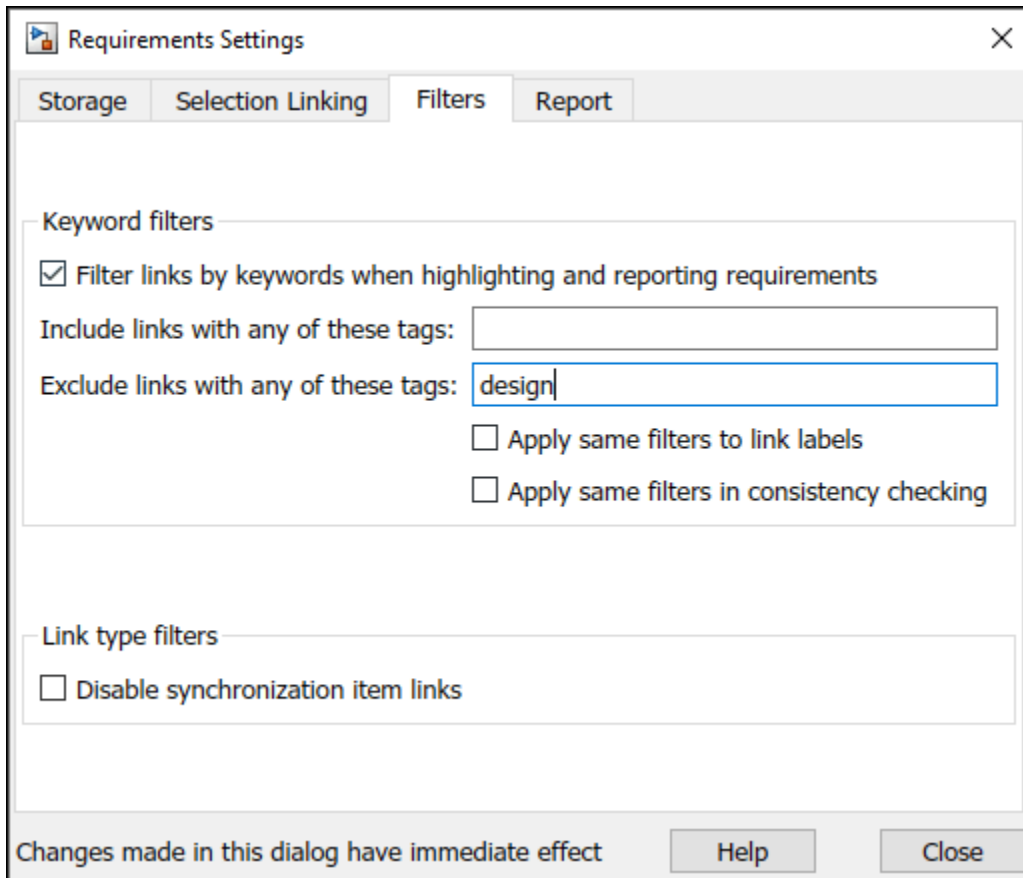
- Navigate back to the fuel rate controller subsystem and in the **Requirements** tab, click **Highlight Links**.

```
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.

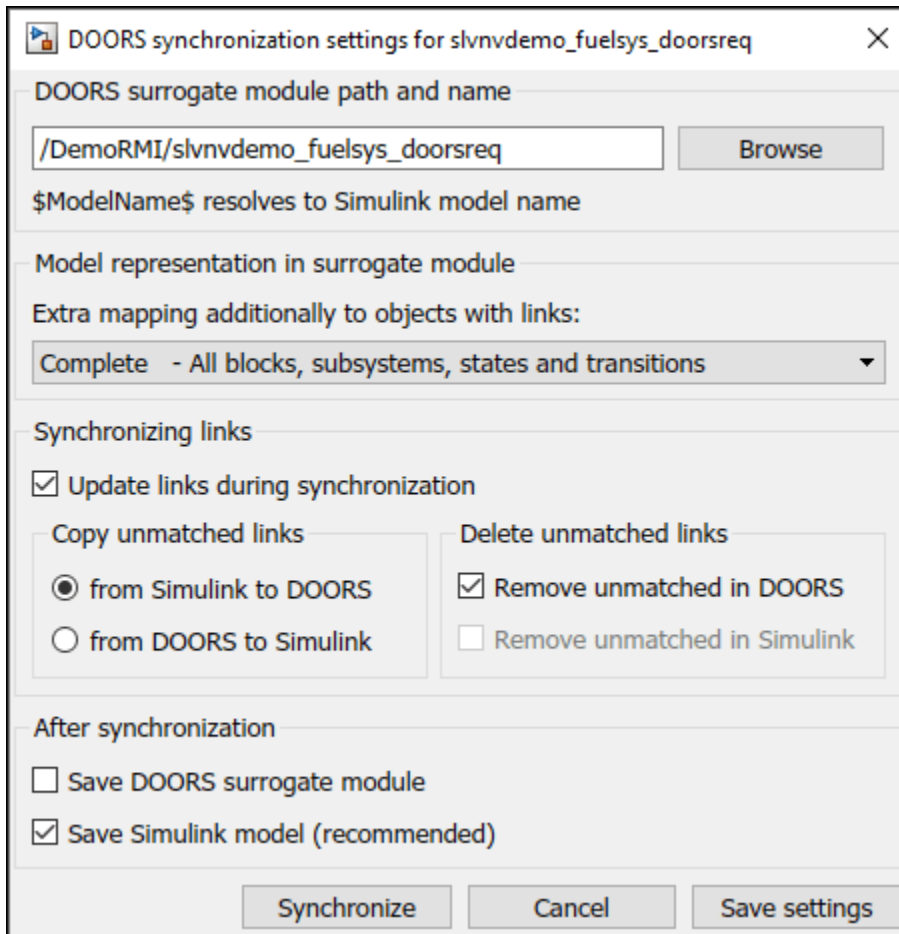
Removing Links in Simulink and DOORS

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.

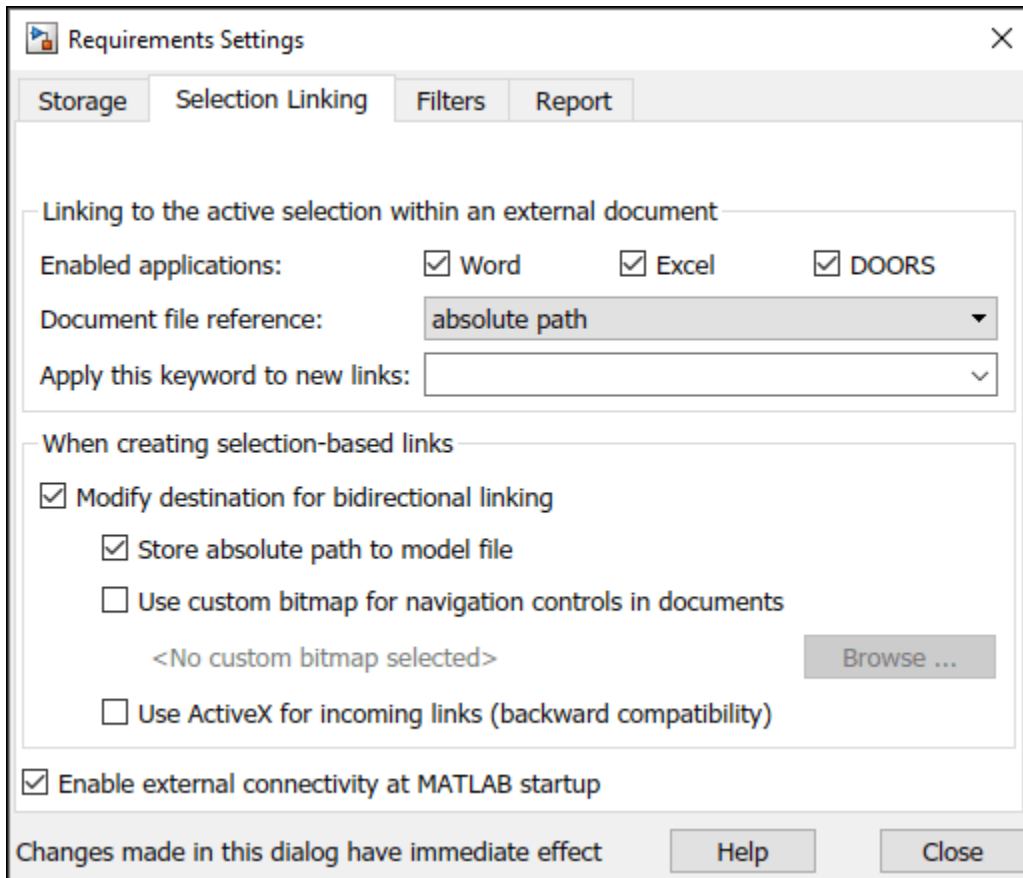
Similarly, when links are removed in DOORS and you need to propagate the changes to Simulink, rerun synchronization with the **DOORS to Simulink** option selected and **Remove unmatched in Simulink** checkbox enabled.

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.

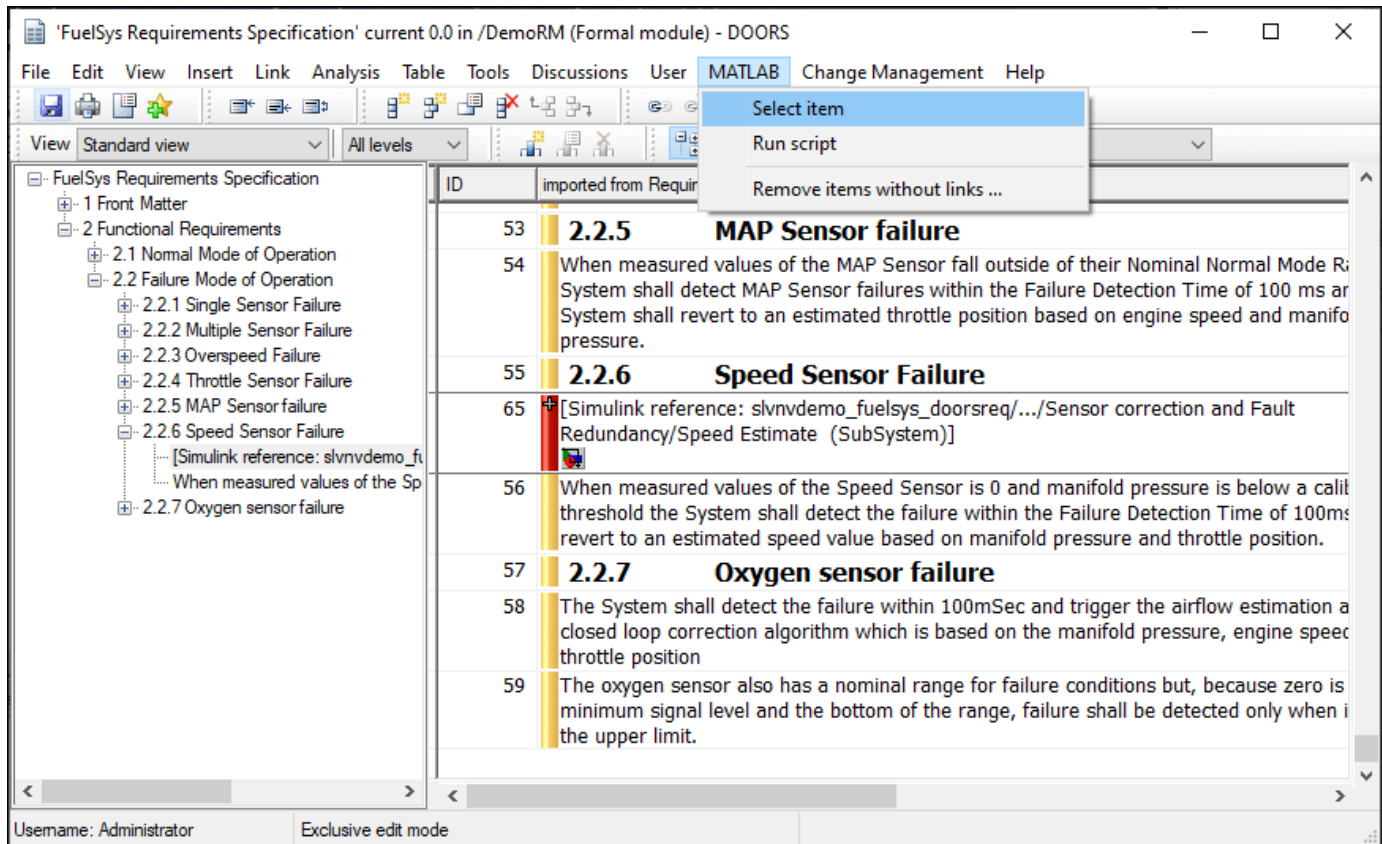


Now, when you use selection linking, Simulink creates navigation objects. There are two types of references to choose from. When **Use ActiveX for incoming links** option at the bottom of **Selection Linking** tab is ON, RMI will insert new DOORS objects with Simulink icon and destination object label 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 8-2
- “Link Test Cases to Requirements Documents” on page 8-3
- “Link Simulink Data Dictionary Entries to Requirements” on page 8-7
- “Link Signal Builder Blocks to Requirements and Simulink Model Objects” on page 8-9
- “Requirements Links for Library Blocks and Reference Blocks” on page 8-13
- “Navigate to Requirements from Model” on page 8-16
- “Link to Requirements Modeled in Simulink” on page 8-18

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 Simulink Requirements license, you can link requirements to test harnesses, test sequences, and test cases. Before adding links, review “Supported Requirements Document Types” on page 5-8.

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**  .

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 Simulink Requirements.

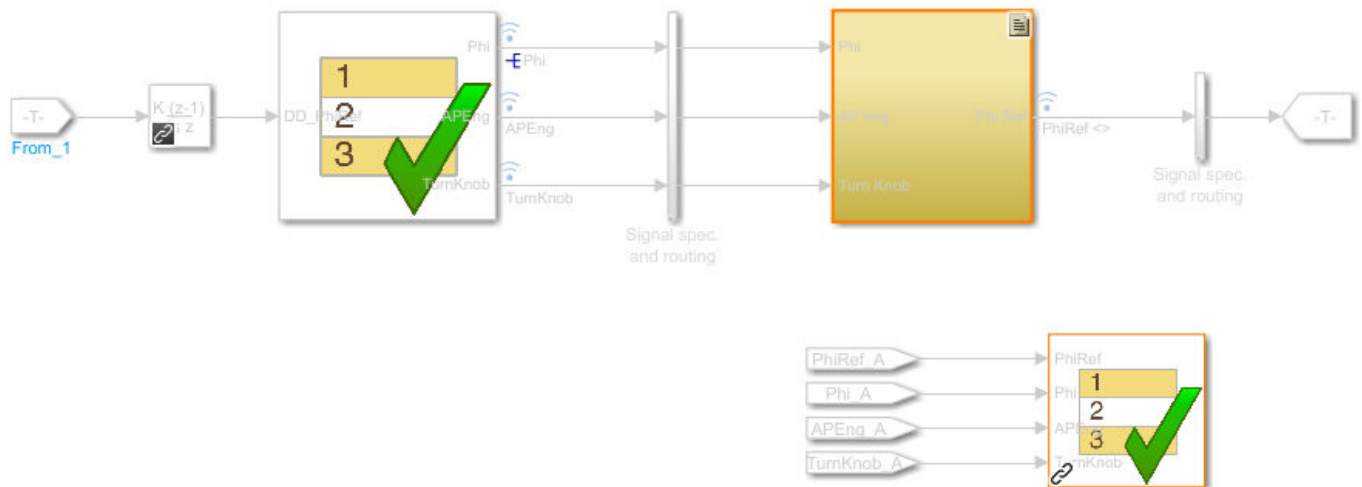
- 1 Open the test file, the model, and the harness.

```
open AutopilotTestFile.mldatx
open_system RollAutopilotMdlRef
sltest.harness.open('RollAutopilotMdlRef/Roll Reference',...
'RollReference_Requirement1_3')
```

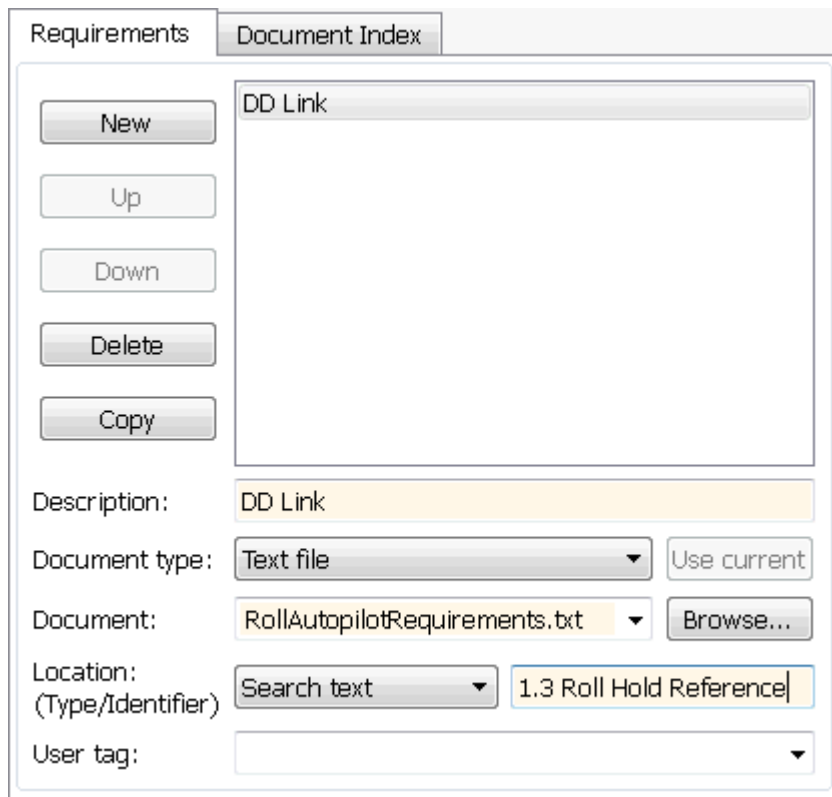
- 2 In the test harness, on the **Apps** tab, under Model Verification, Validation, and Test, click

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
 - 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 to Test Cases from Requirements”

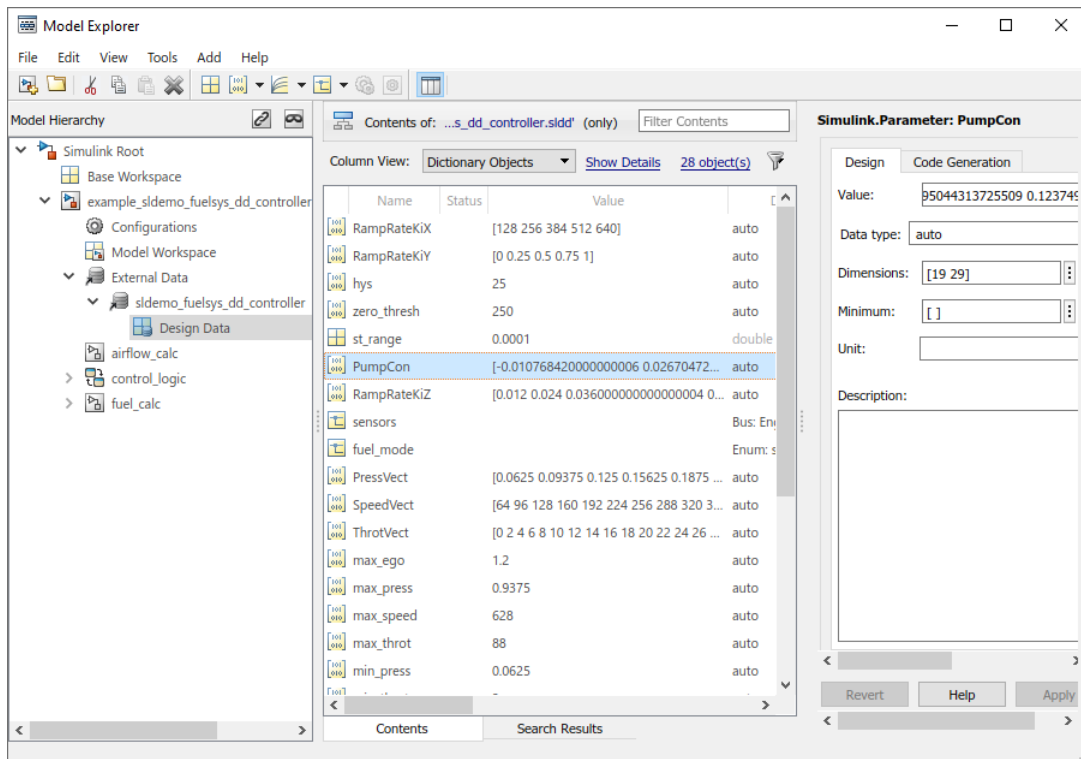
Link Simulink Data Dictionary Entries to Requirements

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 Builder Blocks to Requirements Documents

This example shows how to create links from a signal group in a Signal Builder block to a requirements document.

Open the `sf_car` model.

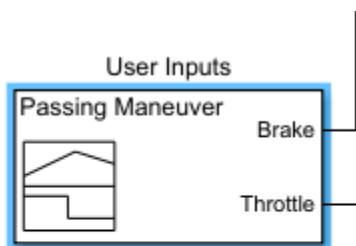
```
open_system("sf_car")
```

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 11-35.

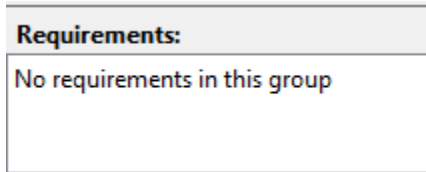
Create Links from the Signal Builder Block

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 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 Builder dialog box.



Create the link to this signal group by using the Outgoing Links dialog.

- 1 In the Signal Builder 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:

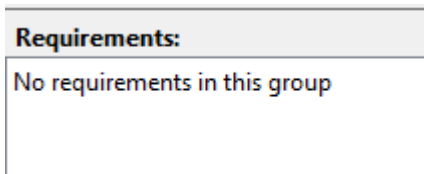
- 1 Open the `sf_car` model.


```
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.

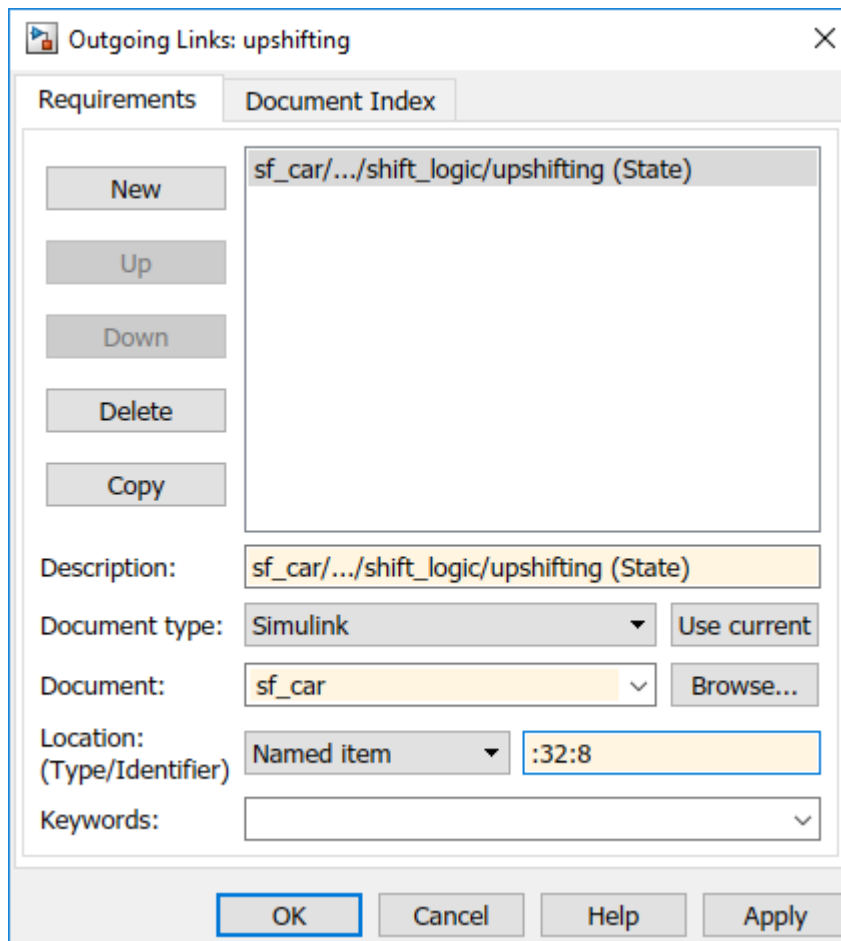
The Signal Builder dialog box opens, displaying four groups of signals. The Passing Maneuver signal group is the current active group. The RMI associates any requirements links that you add to the current active signal group.

- 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. 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

- “Requirement Links” on page 2-32
- “Link Model Objects” on page 8-2
- “Link Test Cases to Requirements Documents” on page 8-3
- “Link Simulink Data Dictionary Entries to Requirements” on page 8-7

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”.

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

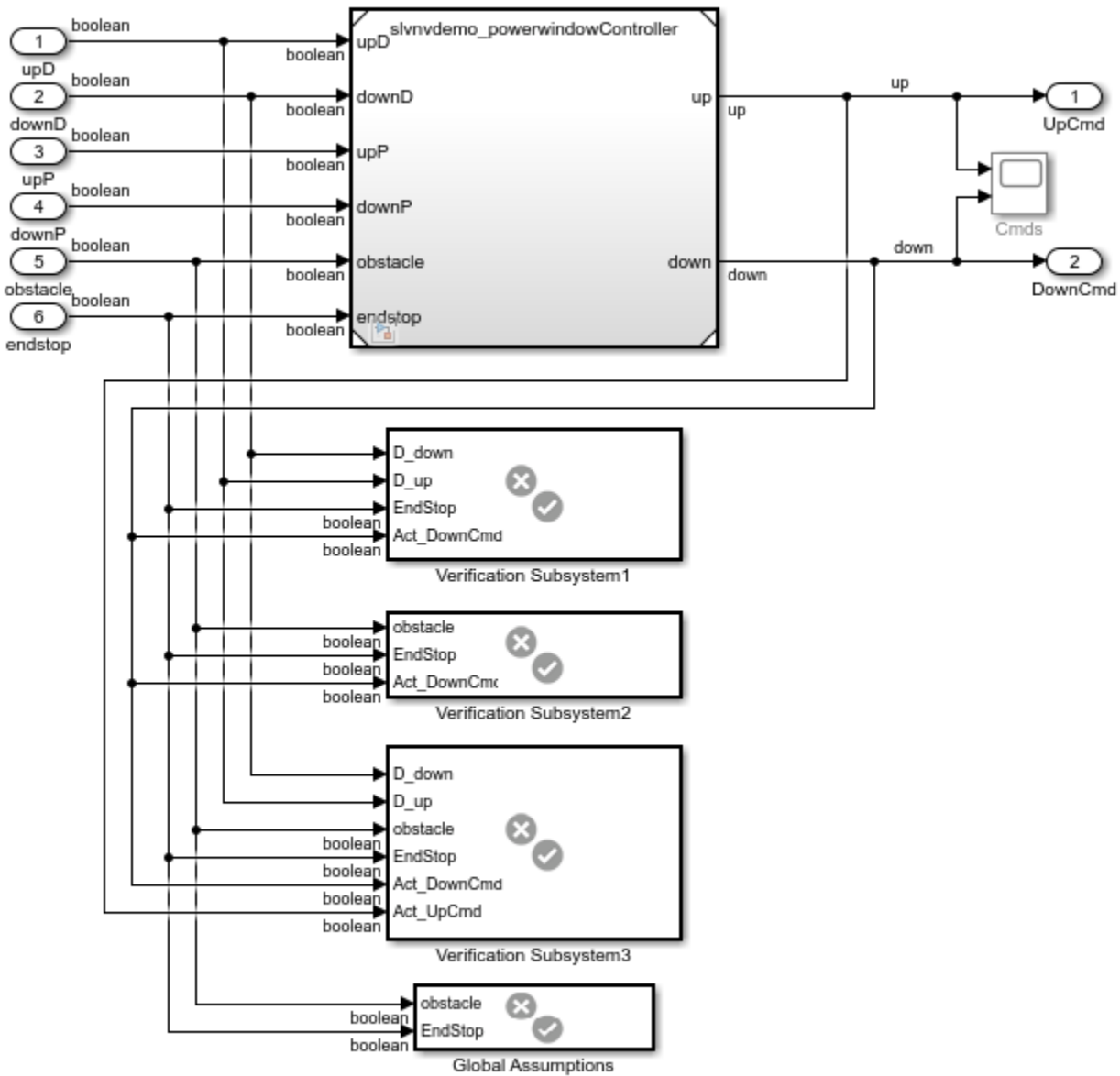
This example shows how to link between verification subsystems and models. You can use verification subsystems to model functional requirements and verify them in simulation. Traceability between the verification and implementation models allow you to summarize analysis and test results in the Requirements 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:

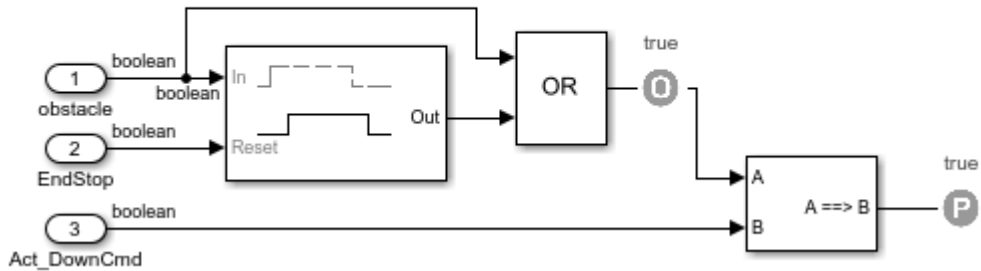
- **Obstacle Response:** When an obstacle is detected, the controller shall give the down command for 1 second.

The requirement is modeled in Verification Subsystem2.

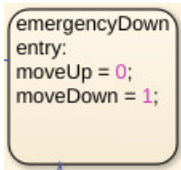
```
open_system('slvndemo_powerwindow_vs/Verification Subsystem2')
```

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:

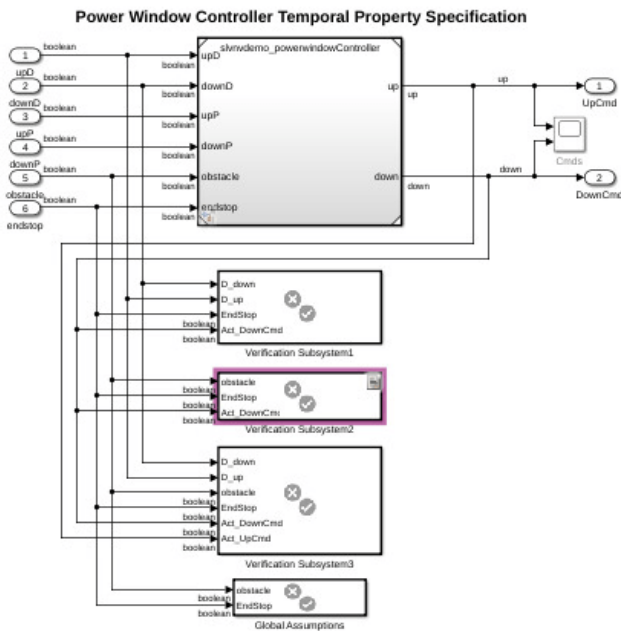


Link from Verification to Design Model

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 Lines” on page 9-2
- “Associate Traceability Information with MATLAB Code Lines in Simulink” on page 9-6

Requirements Traceability for MATLAB Code Lines

Link MATLAB Code Lines to Requirements in a Requirement Set

Create Links by Using Context Menu Shortcuts

To create requirements traceability links from MATLAB code lines to requirements in the Requirements Editor, use the Requirements context menu in the MATLAB Editor.

- 1 Load the requirement set that contains the requirement that you want to link to.
- 2 Navigate to the Requirements Editor and select the requirement.
- 3 In the MATLAB Editor, select the line or lines of code that you want to link.
- 4 Right-click your selection.
- 5 From the context menu, select **Requirements > Link to Selection in Requirements Browser**.

Simulink Requirements creates a traceability link from the MATLAB code lines to the selected requirement in the Requirements Editor. Navigate from your requirements to the MATLAB code lines by clicking **Show Requirements** and using the navigation links available from the **Details** pane, under **Links** in the Requirements Editor.

Create Links Through the Outgoing Links Dialog Box

Create requirements traceability links between MATLAB code lines and requirements in requirement sets by using the Outgoing Links dialog box.

- 1 Load the requirement set that contains the requirement that you want to link to.
- 2 Navigate to the Requirements Editor and select the requirement.
- 3 In the MATLAB Editor, select the line or lines of code that you want to link to the requirement.
- 4 Right-click your selection.
- 5 From the context menu, select **Requirements > Open Outgoing Links dialog**.
- 6 In the Outgoing Links dialog box, click **New**.
- 7 From the **Document type** drop-down list, select **Requirement Set**.
- 8 Populate the **Document** field and the requirement **Description** by clicking **Use current**.
- 9 Click **OK**.

Navigate from your requirements to MATLAB code lines by clicking **Show Requirements** and using the navigation links available from the **Details** pane, under **Links** in the Requirements Editor.

Note When you link code lines from a MATLAB-based Simulink test to a requirement, the code lines that you select determine the type of link and the test to which it is added. See “Test Models Using MATLAB-Based Simulink Tests” (Simulink Test) .

Link MATLAB Code Lines to Requirements Information in External Documents

Create Link by Using Context Menu Shortcuts

To create requirements traceability links from MATLAB code lines to selections in Microsoft Word, Microsoft Excel, or IBM Rational DOORS documents, use shortcuts in the Requirements Traceability context menu.

- 1 In your requirements document, select the target requirement for the traceability link that you want to create.
- 2 In the MATLAB Editor, select the line or lines of code that you want to link to the requirement.
- 3 In the MATLAB Editor, right-click your selection.
- 4 From the context menu, 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**

The software creates a traceability link from the selected MATLAB code range to the selection in the requirements document. If you have bidirectional linking enabled, the software also inserts a navigation object for the selection in the requirements document. The navigation object links to the selected MATLAB code range.

Create and Edit Links Through the Outgoing Links Dialog Box

You can create, edit, and delete traceability links through the Outgoing Links dialog box. To open the Outgoing Links dialog box:

- In the MATLAB Editor, select the line or lines of code that you want to link to requirements.
- Right-click your selection.
- From the context menu, select **Requirements > Open Outgoing Links dialog**.

See “Outgoing Links Editor” on page 10-6.

Enable or Disable Traceability Links Highlighting for MATLAB Code

Review traceability in your MATLAB code by highlighting code lines that have requirements links.

Enable Traceability Highlighting of MATLAB Code

To highlight traceability links in your MATLAB code, do one of the following:

- In the **View** tab, in the **Display** section, select **Highlight Traceability**.
- In the MATLAB Editor, right-click in a line of code with a traceability link. From the context menu, select **Requirements > Enable Traceability Highlighting**.

Disable Traceability Highlighting of MATLAB Code

To turn off highlighting of traceability links in your MATLAB code, do one of the following:

- In the **View** tab, in the **Display** section, clear **Highlight Traceability**.
- In the MATLAB Editor, right-click in a line of code with a traceability link. From the context menu, select **Requirements > Disable Traceability Highlighting**.

Remove Traceability Links from MATLAB Code Lines

Delete Links to Requirements from MATLAB Code Lines

To remove requirements traceability links from a line or lines of MATLAB code:

- 1 In the MATLAB Editor, right-click within a range of code that has requirements traceability links.
- 2 From the context menu, select **Requirements > Delete All Links**.

All links to requirements from this MATLAB code range are deleted. Links to this MATLAB code range from external requirements documents are not deleted.

Delete Link Targets in MATLAB Code Lines

If you have links to MATLAB code ranges from external requirements documents, you can delete the targets for these links from your MATLAB code.

To remove requirements traceability targets from a line or lines of MATLAB code:

- 1 Delete outgoing links as described in “Delete Links to Requirements from MATLAB Code Lines” on page 9-4.
- 2 In the MATLAB Editor, right-click within a previously linked range of code.
- 3 From the context menu, select **Requirements > Discard Named Range**.

When you discard a named range, links to that MATLAB code range from external documents no longer work. Discarding named ranges does not delete navigation objects in external requirements documents.

Traceability for MATLAB Code Lines

Traceability Link Targets

You can create MATLAB code traceability links for:

- Lines of MATLAB code in a standalone file.
- Lines of MATLAB code inside a MATLAB Function block.

You can create links from a line or lines of MATLAB code to:

- Selections in Simulink Requirements.
- Objects in Simulink models.
- Targets in Microsoft Word or Microsoft Excel documents.
- Targets in IBM Rational DOORS databases.
- Targets in text, HTML, or PDF documents.
- HTTP URLs.

Bidirectional linking is supported for targets in MATLAB, Simulink, Microsoft Word, Microsoft Excel, and IBM Rational DOORS. Bidirectional linking creates links to and from the selected link destination. To enable bidirectional linking, in the Requirements Settings dialog box, under the **Selection Linking** tab, select **Modify destination for bidirectional linking**. For more information, see “Selection Linking Tab” on page 5-10.

You can also create links to MATLAB code lines from any external application that supports HTTP navigation.

Traceability Links in Code Generation Reports

Embedded Coder® embeds requirements traceability links for MATLAB files that are saved externally from the Simulink model and referenced from MATLAB Function blocks in Simulink. In the code generation report, click the hyperlink to navigate to the corresponding requirement in the Requirements Editor. See “Generate Code for Models with Requirements Links” on page 11-8.

Storage of Traceability Links

In a standalone MATLAB file, you can create, navigate, and delete traceability links for lines of code without changing the MATLAB file. The Requirements Management Interface (RMI) stores requirements traceability data for a MATLAB file in a `.req` file with the same name and location as the MATLAB file.

If you want to create traceability links for lines of code in a MATLAB Function block, set the parent model to store requirements data externally. For a new model, see “Requirements Link Storage” on page 5-4. For an existing model, see “Move Internally Stored Requirements Links to External Storage” on page 5-5. When you create traceability links for code inside a MATLAB Function block, the RMI stores them in a `.req` file for the parent model. The `.req` file for the model contains requirements traceability data for linked model objects and for linked code in MATLAB Function blocks in the model.

Limitations of MATLAB Code Traceability

The software does not support traceability links for overlapping regions of MATLAB code. If one linked range of code completely overlaps another smaller region of code, the link for the larger range takes precedence over the link for the smaller range. To avoid complications from overlapping linked ranges, when you create traceability links for MATLAB code lines, choose ranges of code that do not overlap.

You can cut or copy a selection of code that has traceability links. When you paste that selection, the software attempts to recreate the corresponding traceability links. Depending on location and code formatting, you might need to recreate the traceability links manually.

If you select code that has traceability links and drag that code to a new location, you might need to recreate traceability links for the code in the new location.

Requirements linked to individual 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 a Web View of a Model” (Simulink Report Generator).

Requirements traceability is not supported for MATLAB Live Editor.

See Also

More About

- “Generate Code for Models with Requirements Links” on page 11-8

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 11-44.

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:

$$\frac{C_m dV}{dt} = I_{\text{total}} - \frac{V - V_0}{R_m}$$

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:

$$V \rightarrow V_0 + \int_0^t \frac{1}{C_m} (I_{\text{total}} - \frac{V - V_0}{R_m}) dt$$

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 `slvndemo_synaptic_transmission` model.

```
sim('slvndemo_synaptic_transmission');

### Starting serial model reference simulation build
### Successfully updated the model reference simulation target for: slvndemo_neuron
```

Build Summary

Simulation targets built:

Model	Action	Rebuild Reason
slvndemo_neuron	Code generated and compiled	slvndemo_neuron_msf.mexw64 does not exist.

1 of 1 models built (0 models already up to date)
Build duration: 0h 1m 0.921s

Manually check the Scope block for results or evaluate the following code.

```
open_system('slvndemo_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','slvndemo_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 `Vsyn_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.

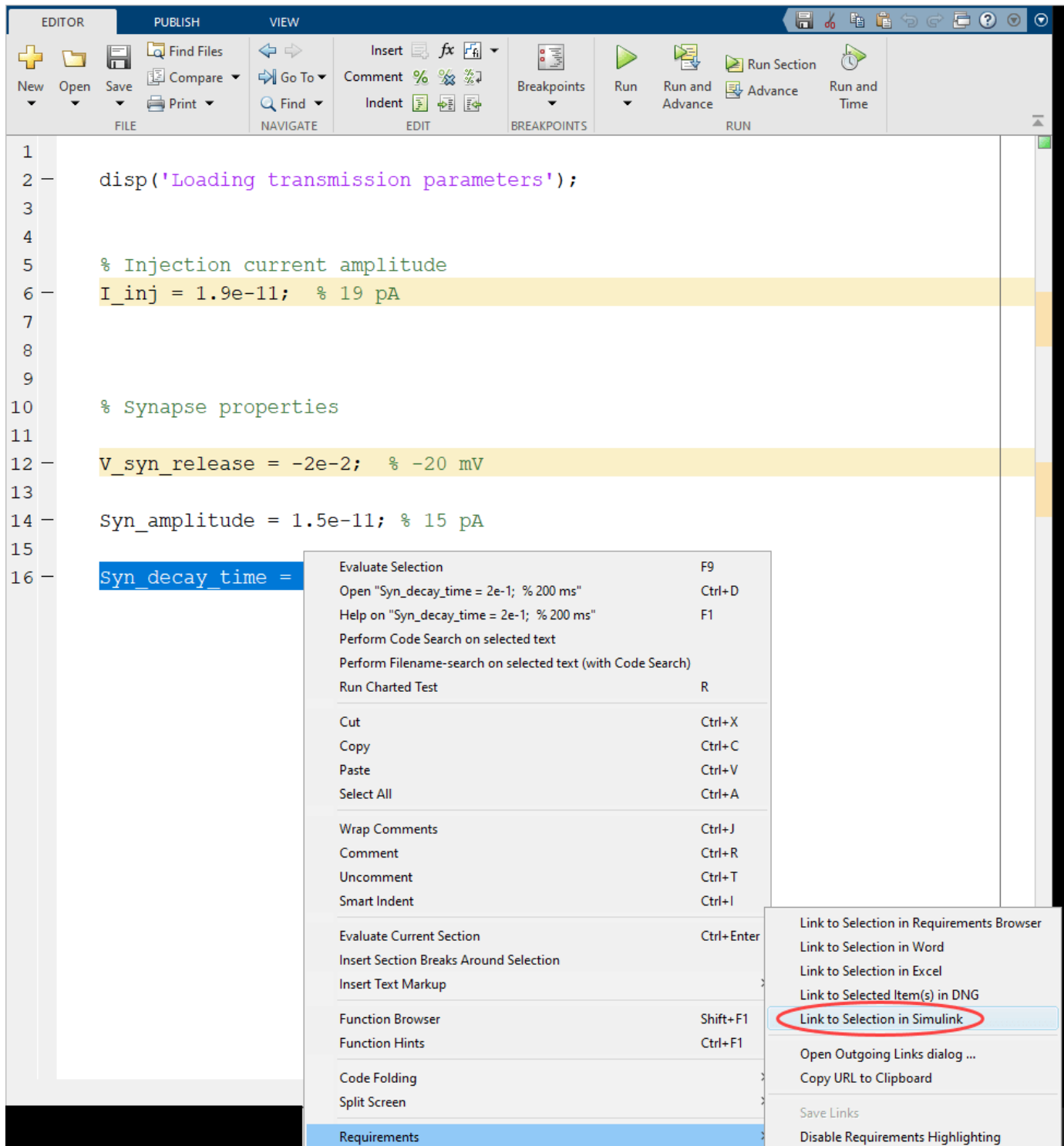
```
rmipref('BiDirectionalLinking',true);
```

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 `slvnvdemo_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.

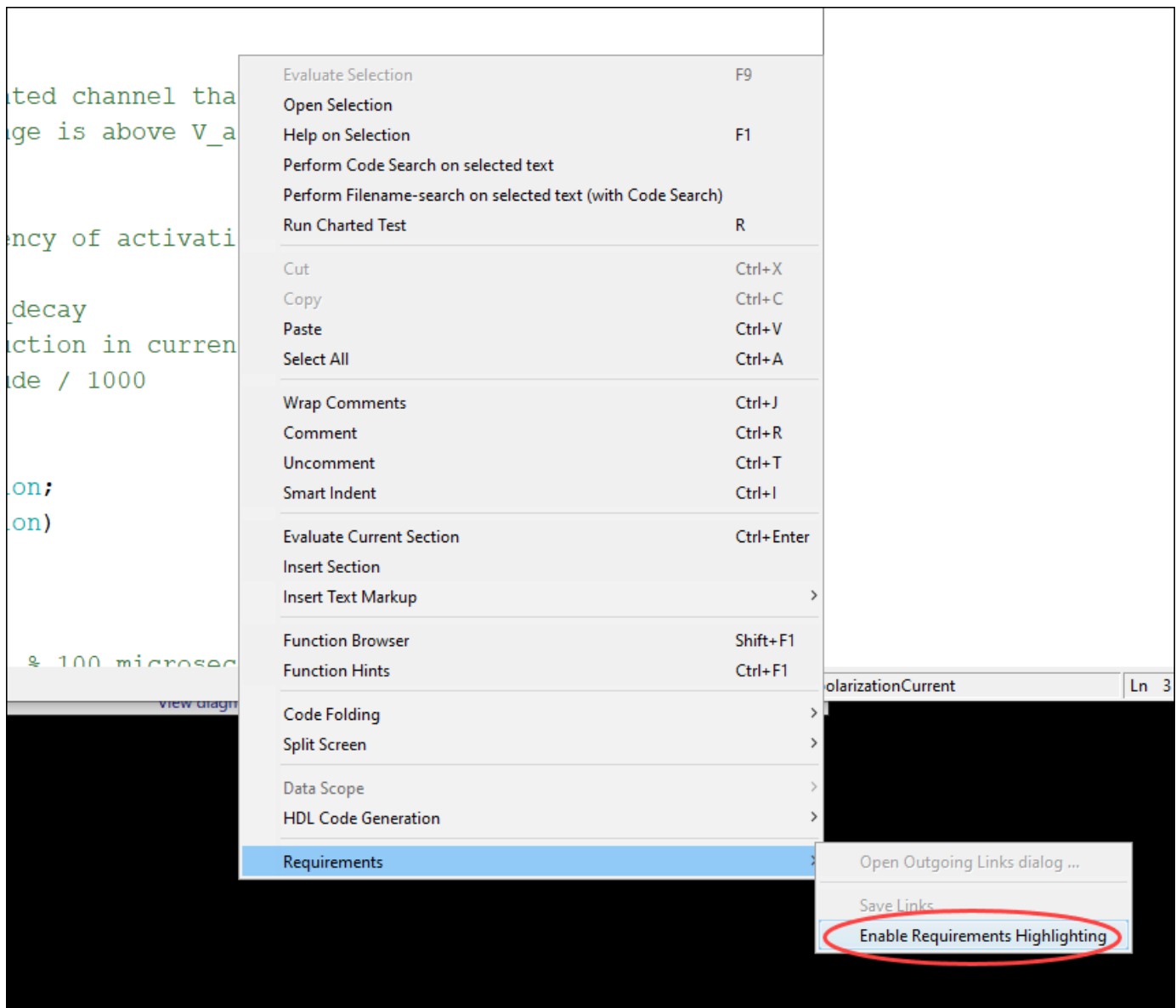
```
rmidemo_callback('emlshow','slvndemo_synaptic_transmission/Synaptic current');
```

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.



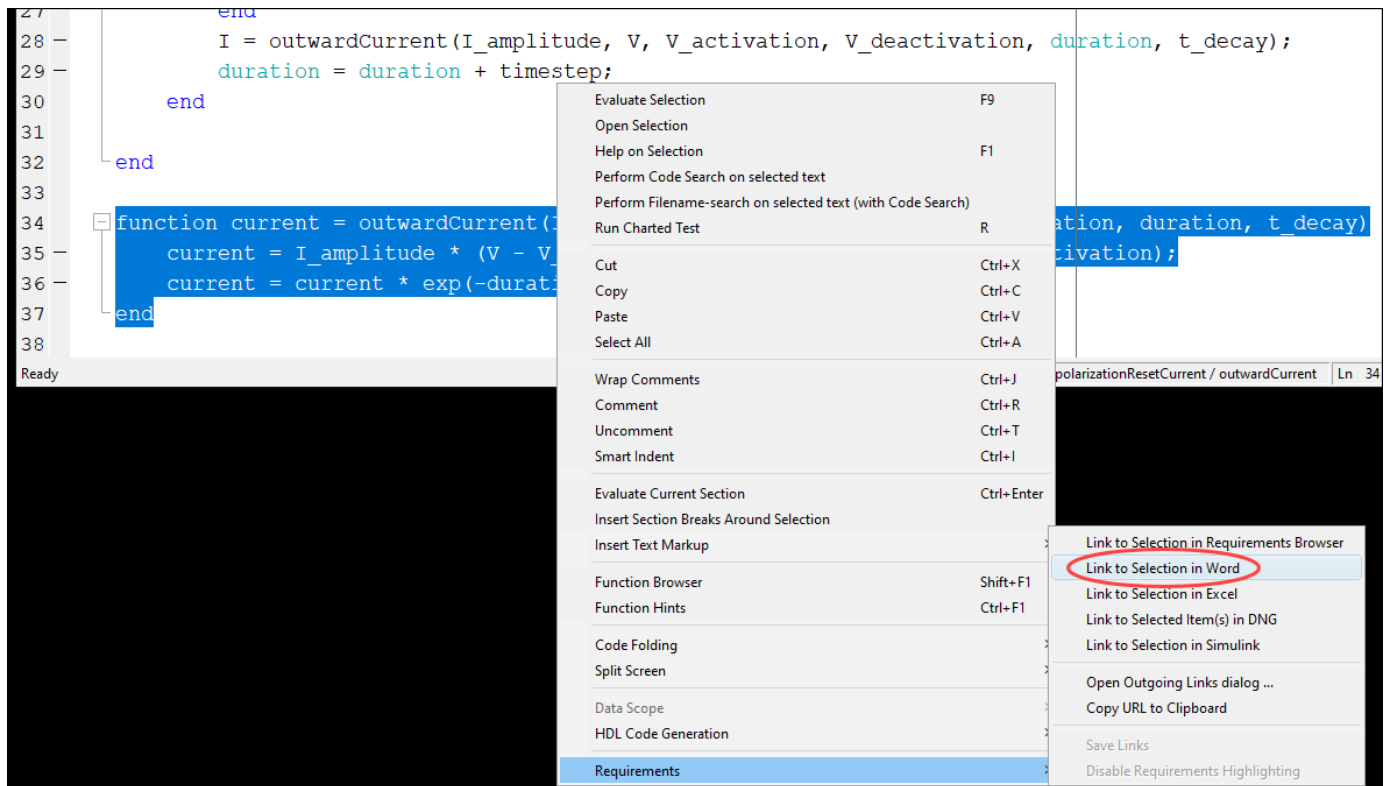
Right-click on a line of code that's been highlighted to indicate it has a requirements link. From the **Requirements** context menu, navigate to a numbered link at the top of the menu. The associated Word document opens to the associated text. You can also open the Word document and visually identify 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')`.

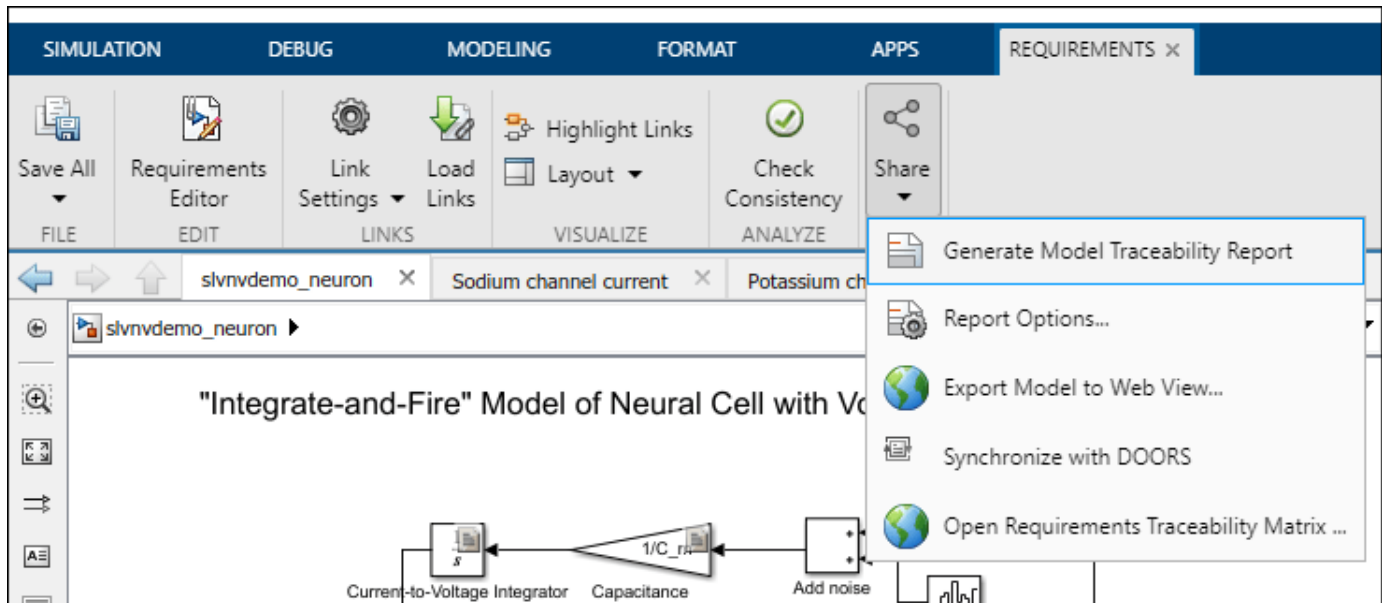
In the MATLAB Function Block Editor, select the implementation for **outwardCurrent** subfunction (lines 34-37). Right-click the selected lines and in the context menu, select **Requirements > Link to Selection in Word**. Minimize the Word document, then navigate from the newly highlighted lines at the 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.

Cleanup

The following commands clean up the workspace by clearing all Simulink Requirements objects and closing all Simulink models.

```
slreq.clear;  
bdclose('all');
```


URL and Custom Traceability

- “Requirement Links and Link Types” on page 10-2
- “Custom Link Types” on page 10-8
- “Implement RMI Extension for Support of Custom Document Type” on page 10-17

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 5-8.)
- A designated location inside the requirements document, such as:
 - Bookmark
 - Anchor
 - ID
 - Page number
 - Line number
 - Cell range
 - Link target
 - Tags 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 10-6, 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 <code>DetailsFcn</code> 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><code>SelectionLinkFcn</code> 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.

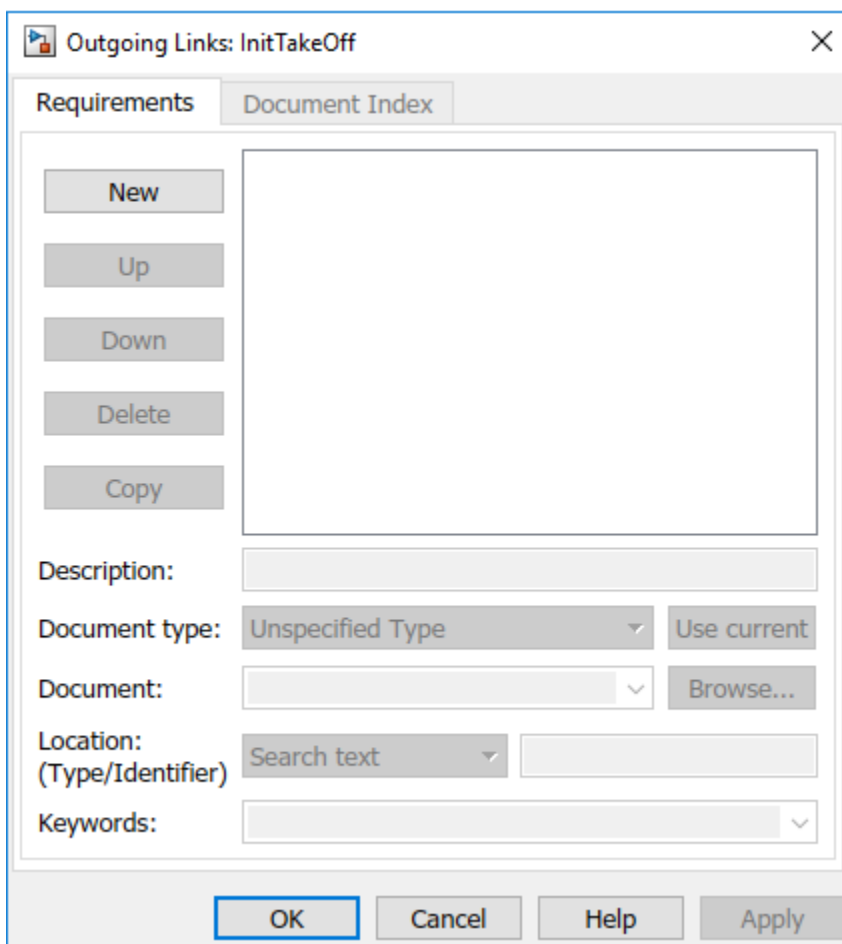
Outgoing Links Editor

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 tags 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 tag 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

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
end

function openFileToLine(fileName, lineNum)
    if lineNum > 0
        if matlab.desktop.editor.isEditorAvailable
            matlab.desktop.editor.openAndGoToLine(fileName, lineNum);
        end
    else
        edit(fileName);
    end
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);
end

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

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

    % Combine and sort the list
    items = [fList1{:}];
    items = sort(items);
    items = strcat('@',items);

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

    depths = [];
end

```

- 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 10-6 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\slvnc\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:

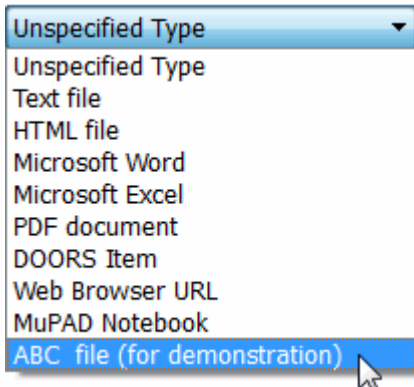
```
openExample('simulink_aerospace/DevelopingTheApolloLunarModuleDigitalAutopilotExample')
```

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.



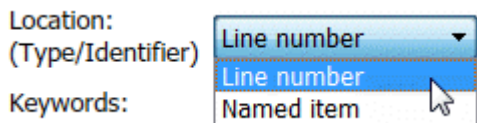
- 8 Set **Document type** to ABC file (for demonstration) and browse to the `matlabroot\toolbox\slvnx\rmidemos\demo_req_1.abc` file. The browser shows only the files with the `.abc` extension.
- 9 To define a particular location in the requirements document, use the **Location** field.

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).



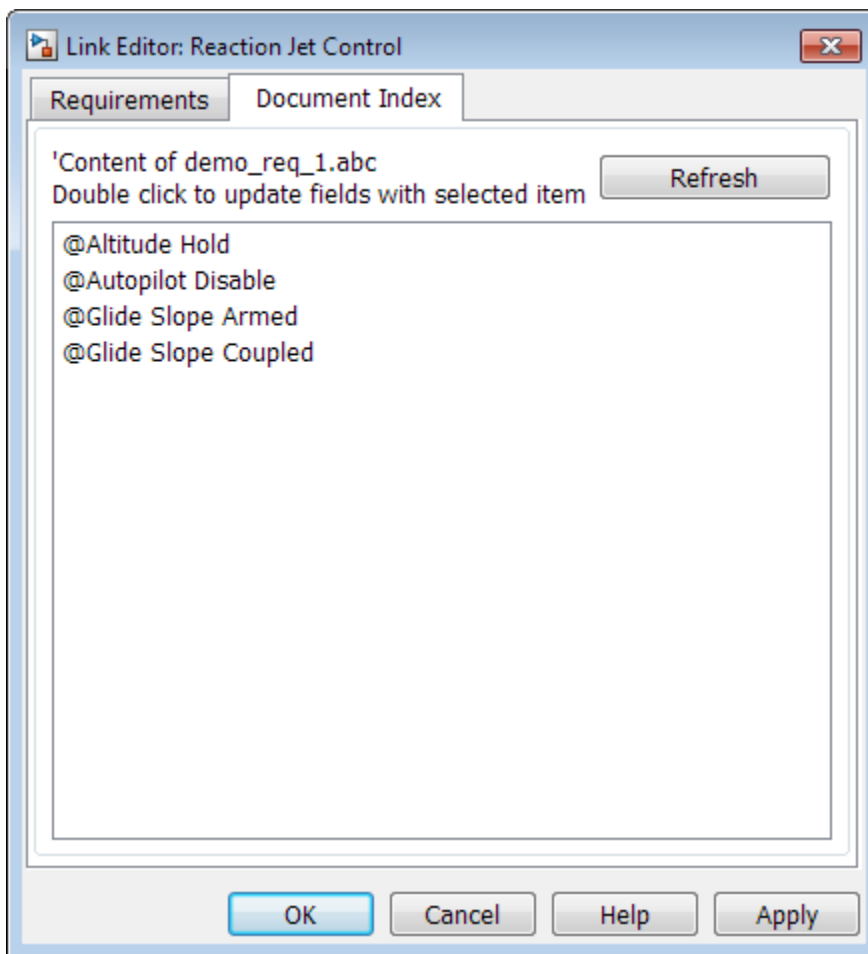
- 10 Select **Line number**. Enter the number 26, which corresponds with the line number for the Altitude Hold requirement in `demo_req_1.abc`.
- 11 In the **Description** field, enter Altitude Hold, to identify the requirement by name.
- 12 Click **Apply**.
- 13 Verify that the Altitude Hold requirement 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 10-14.
- 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 10-3.

- 3 Register the custom link type using the `rmi` command 'register' option, as described in "Custom Link Type Registration" on page 10-15.

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 10-6 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 Simulink Requirements 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 Simulink Requirements 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 Simulink Requirements 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 `slvndemo_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

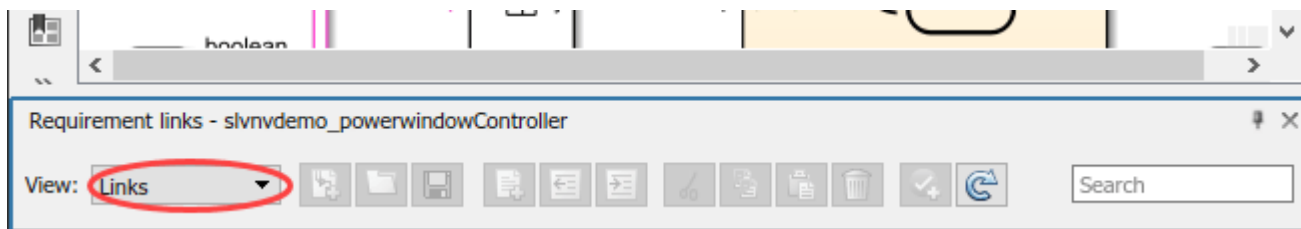
For the purposes of this example tutorial, you will link objects in the `slvndemo_powerwindowController.slx` model with slides in the `powerwindowController.pptx` PowerPoint presentation. Open the Simulink model `slvndemo_powerwindowController.slx`.

```
open_system('slvndemo_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 `slvndemo_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

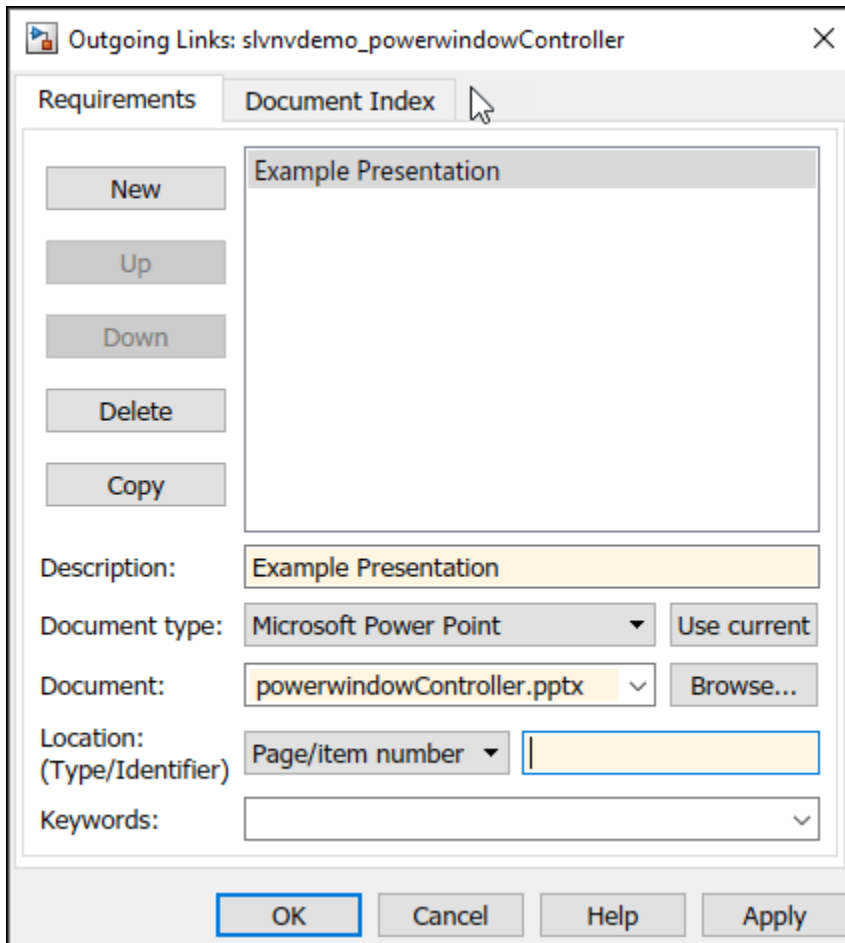
The file `rmidemo_pp_linktype.m` in the current working directory contains link type information for the RMI to work with Microsoft PowerPoint files. Register the link type with the RMI by evaluating the following.

```
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.

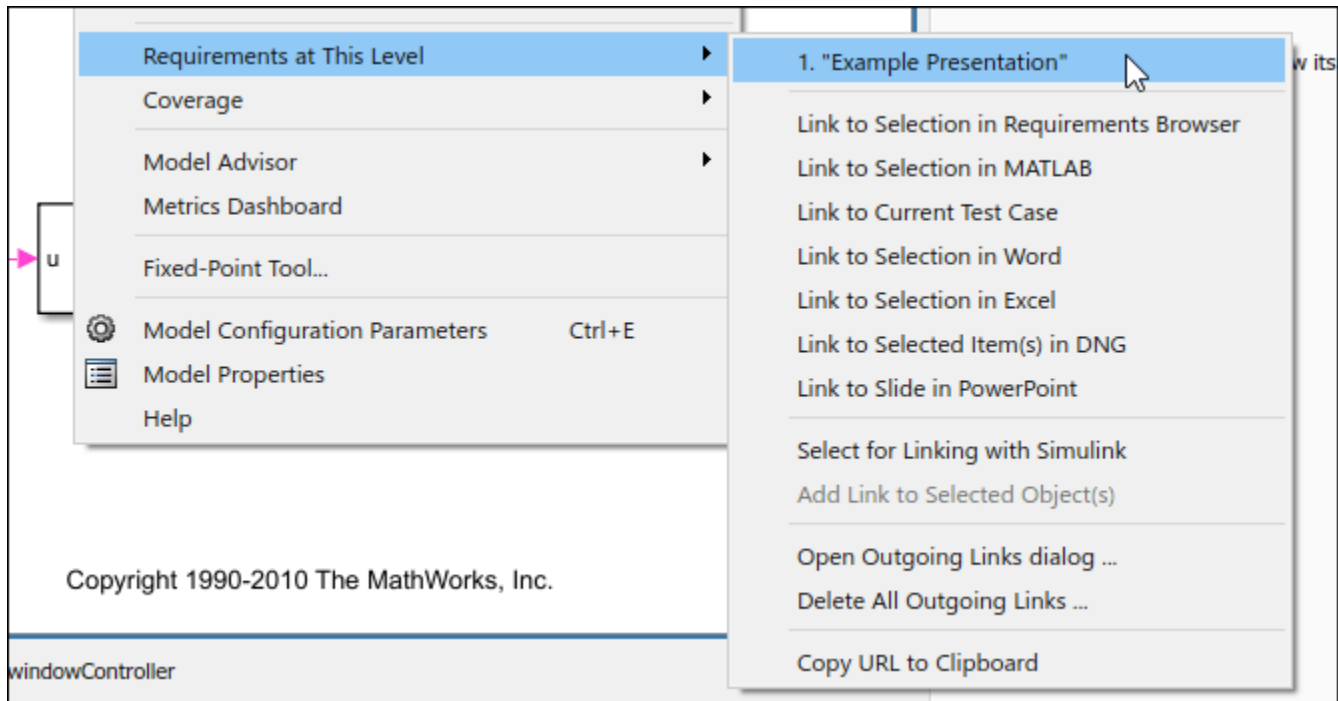
```
firstReq = rmi('createempty');
firstReq.reqsys = 'rmdemo_pp_linktype';
firstReq.doc = 'powerwindowController.pptx';
firstReq.description = 'Example presentation';
rmi('set', 'slnvdemo_powerwindowController', firstReq);
```

Navigation to the Document

Navigation to the PowerPoint document is provided with functions in the `rmdemo_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 `rmdemo_pp_linktype` methods. See Microsoft's Developer Reference pages for information on which PowerPoint Objects and Methods are available via COM API.

The `rmdemo_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:

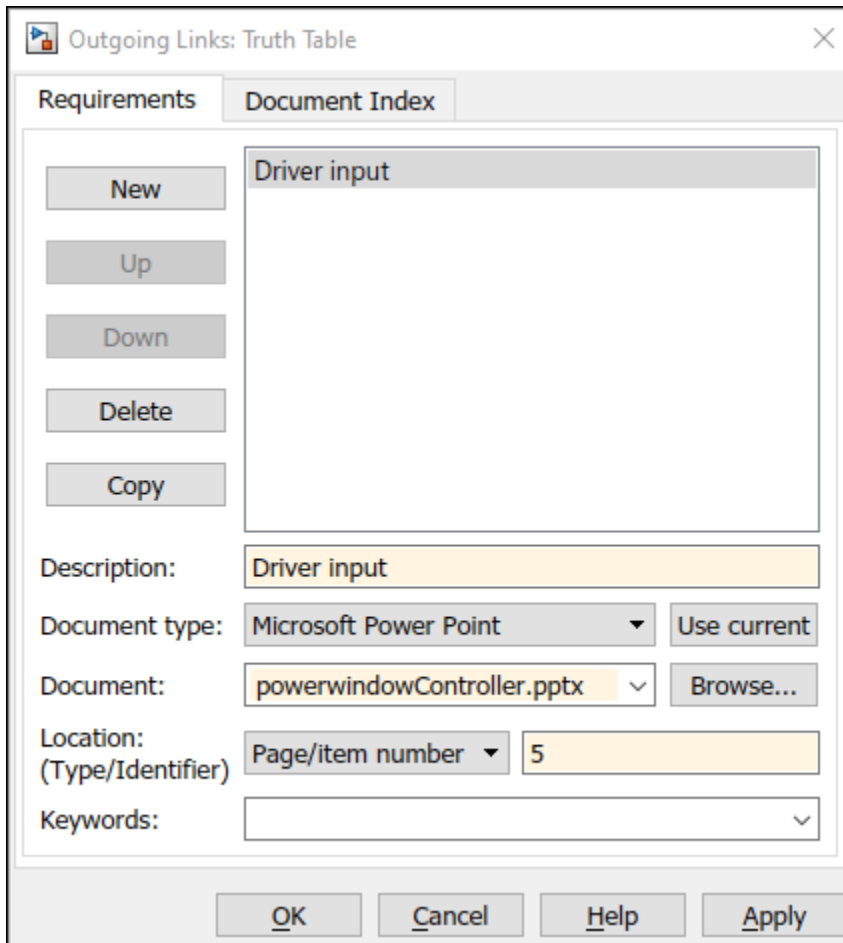
```
rmi('view','slvndemo_powerwindowController', 1)
```

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')
```

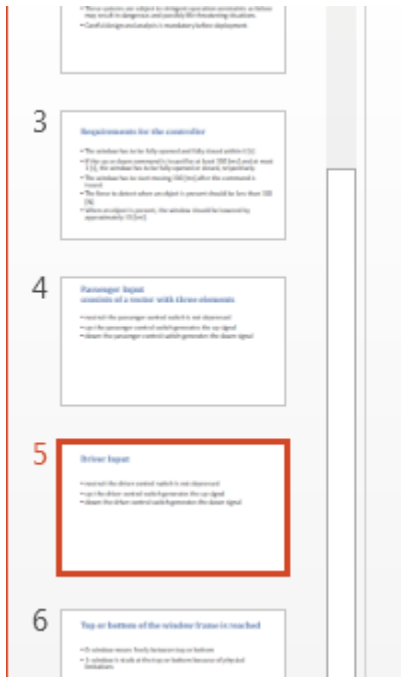
- Right-click the block, select **Requirements > Open Outgoing Links Dialog...** to bring up the Outgoing Links dialog box.
- Click **New** to create a new link.
- Specify the document type and filename as before.
- Enter *Driver input* into the **Description** field.
- Enter 5 into the **Location/Identifier** 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.

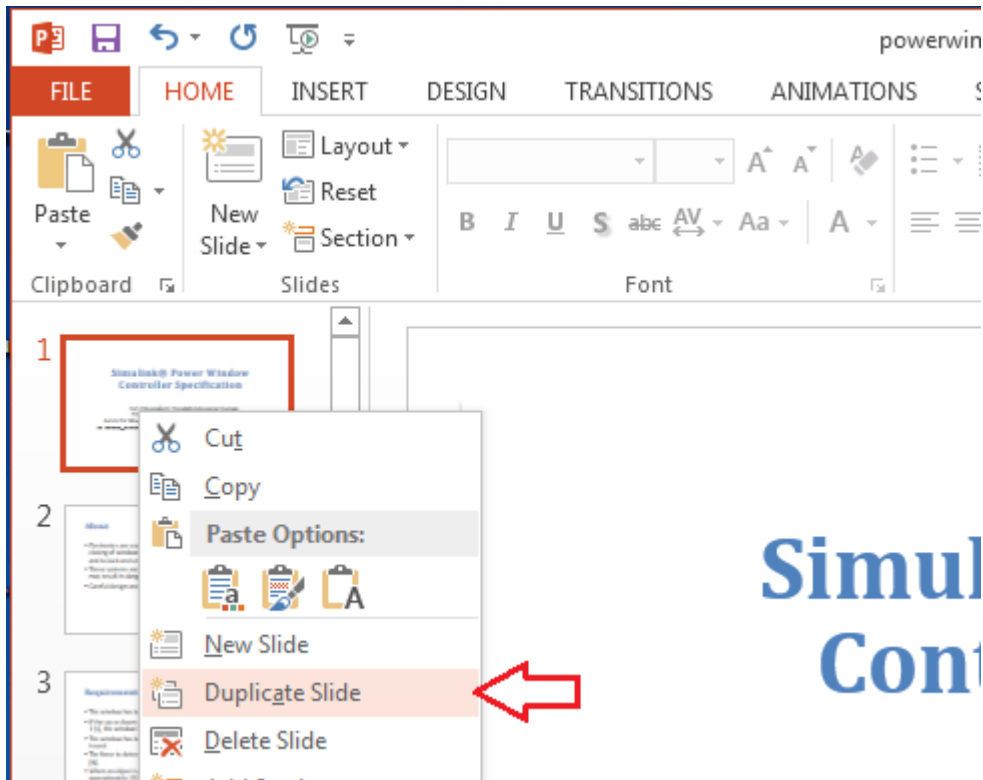
```
secondReq = rmi('createempty');
secondReq.reqsys = 'rmidemo_pp_linktype';
secondReq.doc = 'powerwindowController.pptx';
secondReq.description = 'Driver input';
secondReq.id = '#5';
rmi('set', 'slvndemo_powerwindowController/Truth Table', secondReq);
```

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.

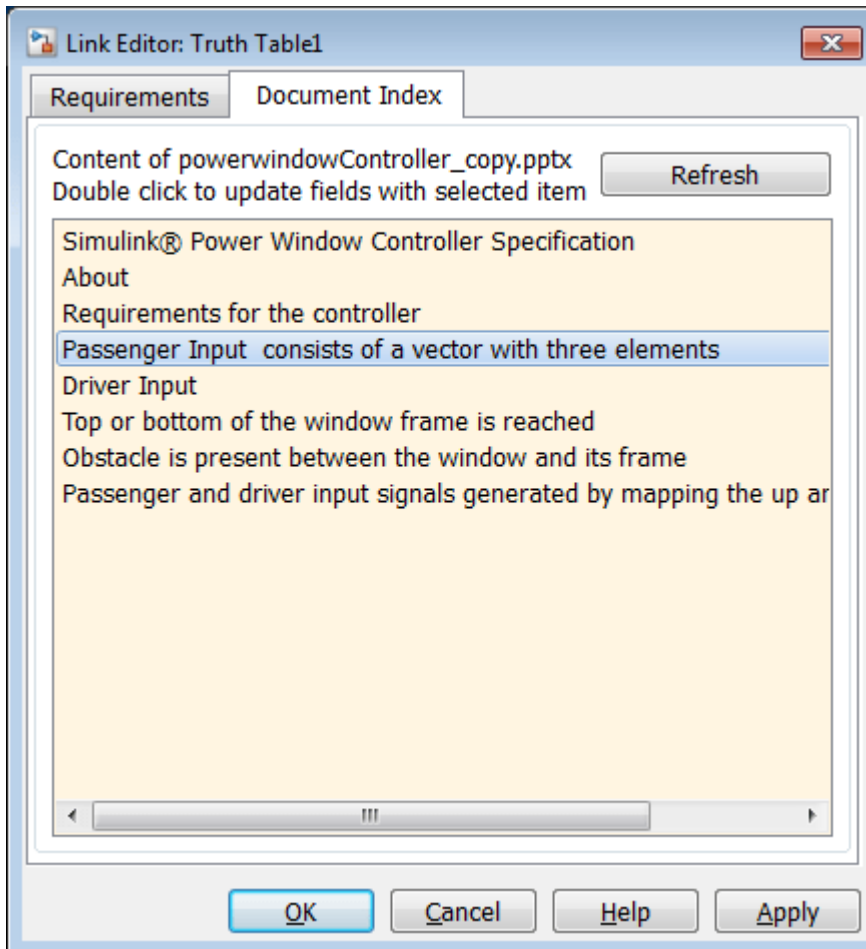
Use the **Document Index** tab in the **Outgoing Link Editor** to create a link.

- 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', 'slvndemo_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

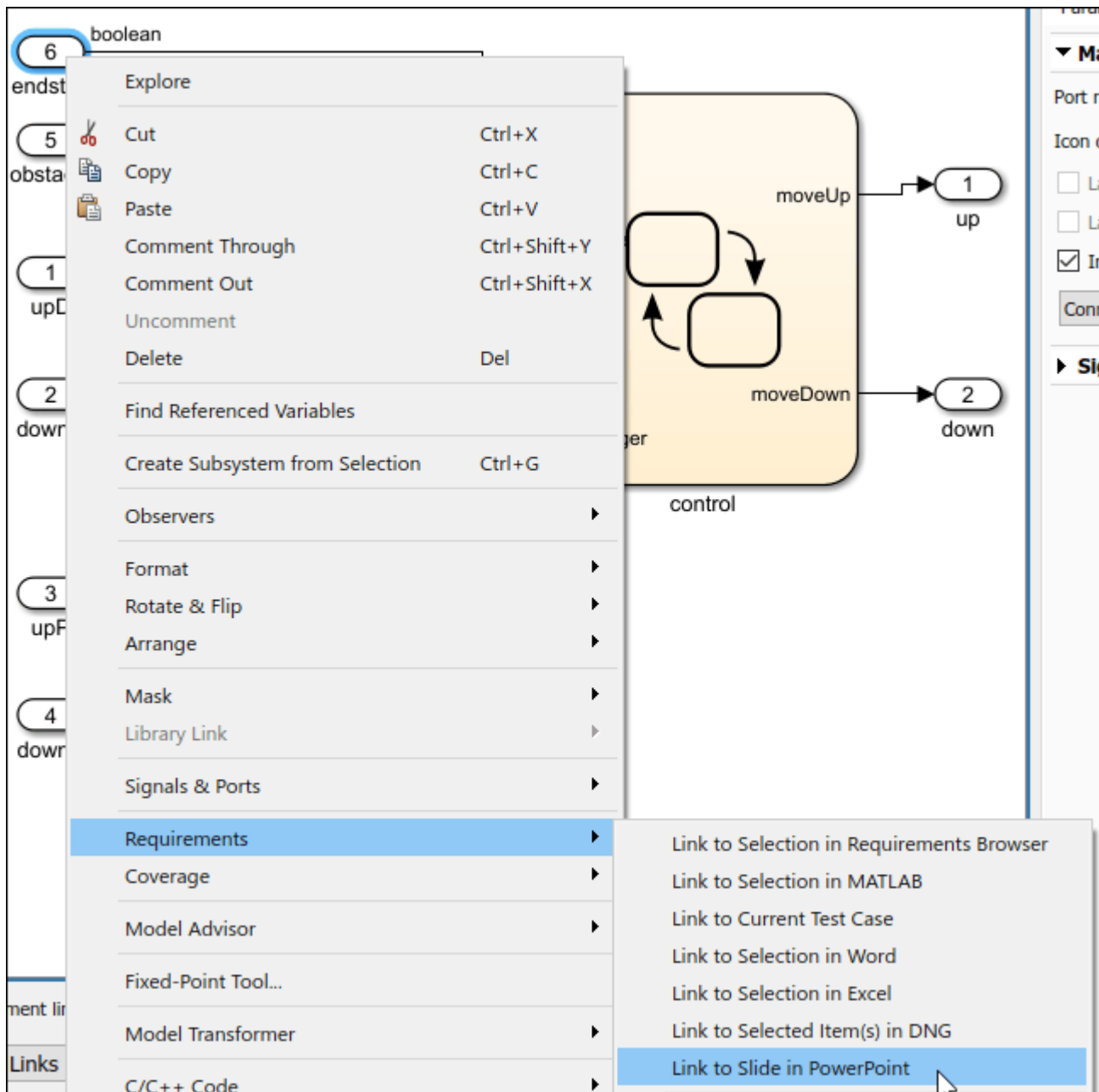
An even better way to support robust persistent links is via Selection Linking Shortcuts. The RMI API allows you to define the `SelectionLinkFcn()` function for linking with the current object in the current document. In the next step of our tutorial, you will automate linking to the current slide in 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.

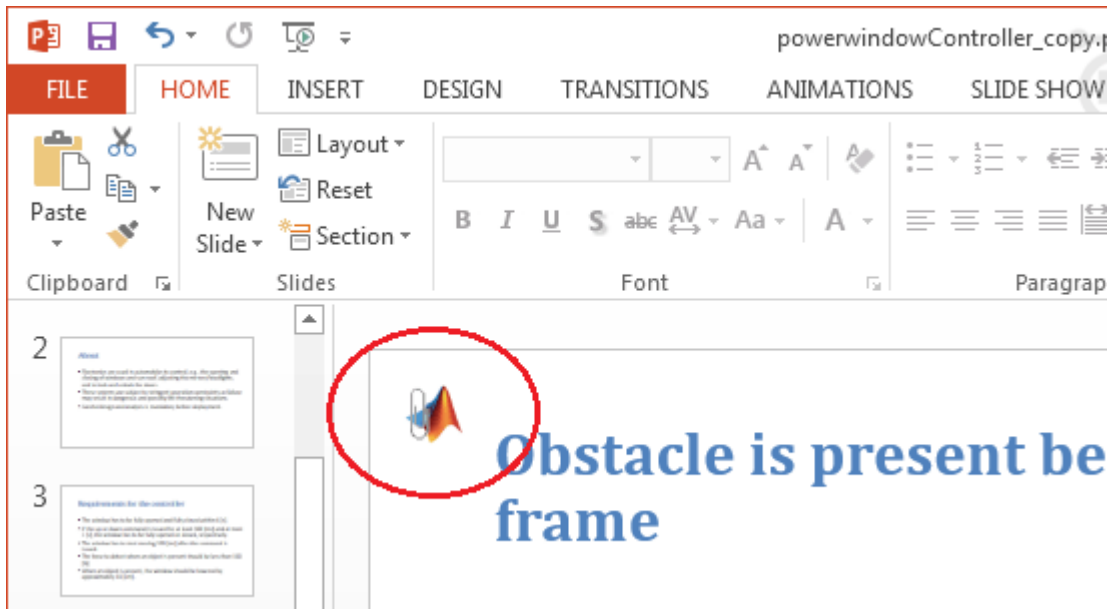
To try this out, repeat the selection linking procedure for the `obstacle` signal input block, to associate it with the corresponding slide in the example presentation.

- 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.



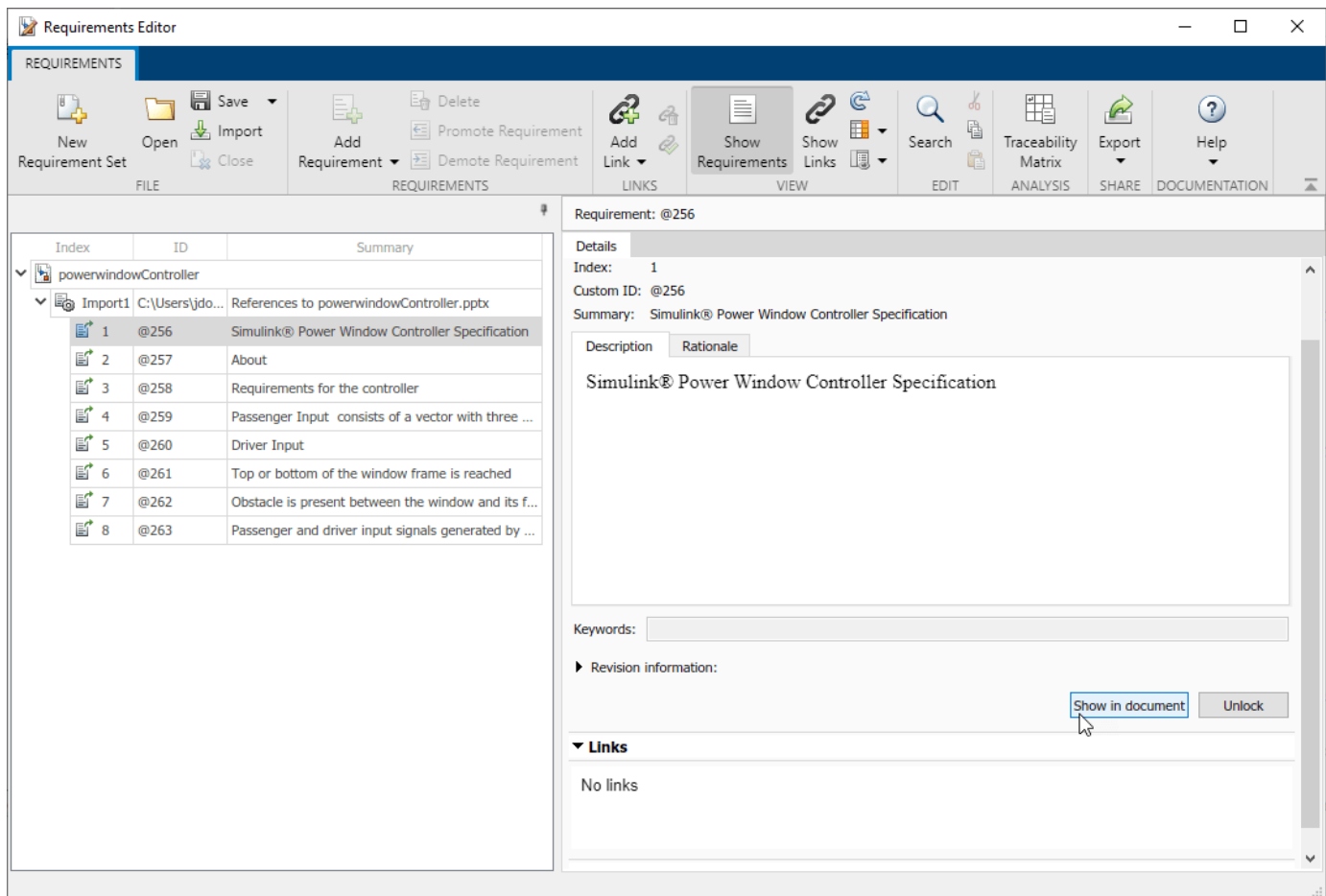
Follow Microsoft PowerPoint's instructions to follow the link, and it should highlight the corresponding block in the `slvnvdemo_powerwindowController` model.

Importing Items from PowerPoint Document into Simulink Requirements

Simulink Requirements product 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 Simulink Requirements Editor UI. 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 `rmdemo_pp_linktype`, clears open requirement sets without saving changes, and closes open models without saving changes.

```
rmi('unregister', 'rmdemo_pp_linktype');
slreq.clear();
bdclose all;
```


Review and Maintain Requirements Links

- “Highlight Model Objects with Requirements” on page 11-2
- “Navigate to Simulink Objects from External Documents” on page 11-4
- “View Requirements Details for a Selected Block” on page 11-6
- “Generate Code for Models with Requirements Links” on page 11-8
- “Create and Customize Requirements Traceability Reports” on page 11-10
- “Create Requirements Traceability Report for A Project” on page 11-25
- “Validate Requirements Links” on page 11-26
- “Delete Requirements Links from Simulink Objects” on page 11-34
- “Document Path Storage” on page 11-35
- “How to Include Linked Requirements Details in Generated Report” on page 11-37
- “Managing Requirements Without Modifying Simulink Model Files” on page 11-44

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:

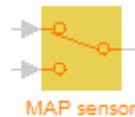
- 1 Open the example model:

```
slvndemo_fuelsys_officereq
```

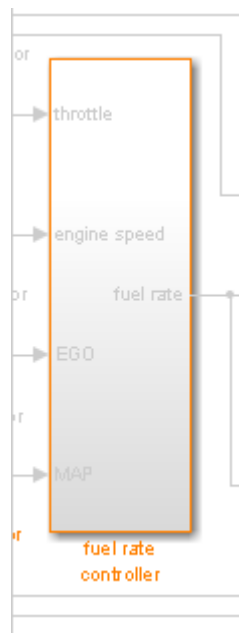
- 2 Select **Coverage Highlighting** from the **Coverage** app.

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 Simulink Requirements, 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			<div style="width: 75%; background-color: blue;"></div>	<div style="width: 25%; background-color: yellow;"></div>
1	#1	Driver Switch Request Handling	<div style="width: 75%; background-color: blue;"></div>	<div style="width: 25%; background-color: yellow;"></div>
1.1	#2	Switch precedence	<div style="width: 0%; background-color: blue;"></div>	<div style="width: 0%; background-color: yellow;"></div>
1.2	#3	Avoid repeating commands	<div style="width: 100%; background-color: blue;"></div>	<div style="width: 0%; background-color: yellow;"></div>
1.3	#4	Long Switch recognition	<div style="width: 50%; background-color: blue;"></div>	<div style="width: 0%; background-color: yellow;"></div>
1.4	#7	Cancel Switch Detection	<div style="width: 100%; background-color: blue;"></div>	<div style="width: 100%; background-color: yellow;"></div>

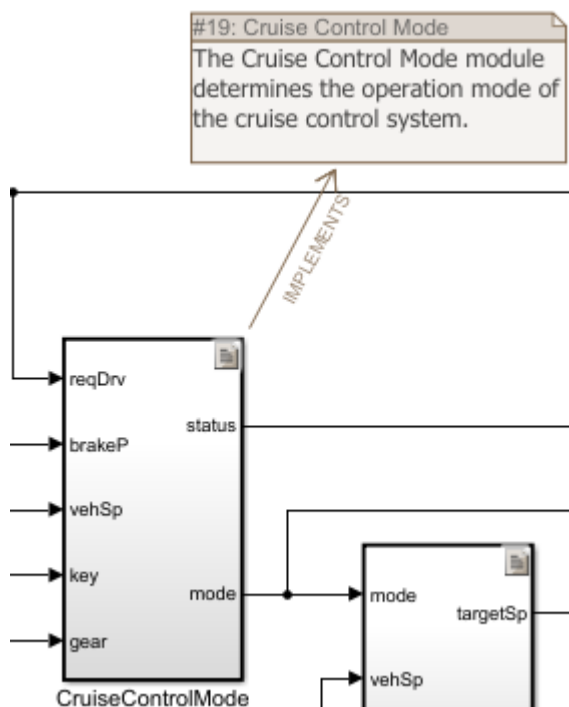
Select the highlighted requirement in the **Requirements** pane. The **Property Inspector** displays the requirement details.

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

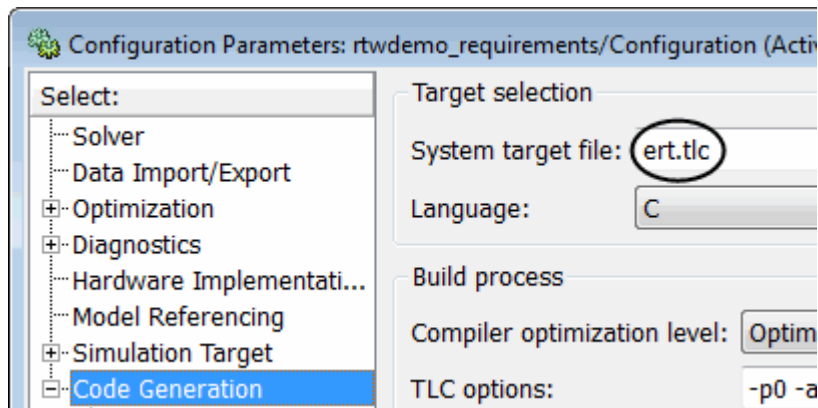
“Navigate to Requirements from Model” on page 8-16

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 5-4.
-

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><model>.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><model>.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)

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	
<hr/>	
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 11-11.

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 11-19.

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 11-18.

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 11-18.

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.

<h2>Requirements Report for slvnvdemo_fuelsys_officereq</h2> <p>username</p> <p>17-Jun-2010 10:57:04</p> <hr/> <h3>Table of Contents</h3> <ul style="list-style-type: none">1. Model Information for "slvnvdemo_fuelsys_officereq"2. Document Summary for "slvnvdemo_fuelsys_officereq"3. System - slvnvdemo_fuelsys_officereq4. System - engine gas dynamics5. System - fuel rate controller6. System - Mixing & Combustion7. System - Airflow calculation8. System - Sensor correction and Fault Redundancy9. System - MAP Estimate10. 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 16:11:43 1998	<i>Creator</i>	The MathWorks Inc.
<i>LastModifiedDate</i>	Sat Jun 12 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 slvnvdemo_FuelSys_RequirementsSpecification.docx	unknown	1
DOC2	fuelsys_req_docs\slvnvdemo_FuelSys_DesignDescription.docx	29-Oct-2009 10:56:01	8
DOC3	fuelsys_req_docs\slvnvdemo_FuelSys_RequirementsSpecification.docx	29-Oct-2009 10:56:02	6
DOC4	fuelsys_req_docs\slvnvdemo_FuelSys_TestScenarios.xlsx	29-Oct-2009 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 **User 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.

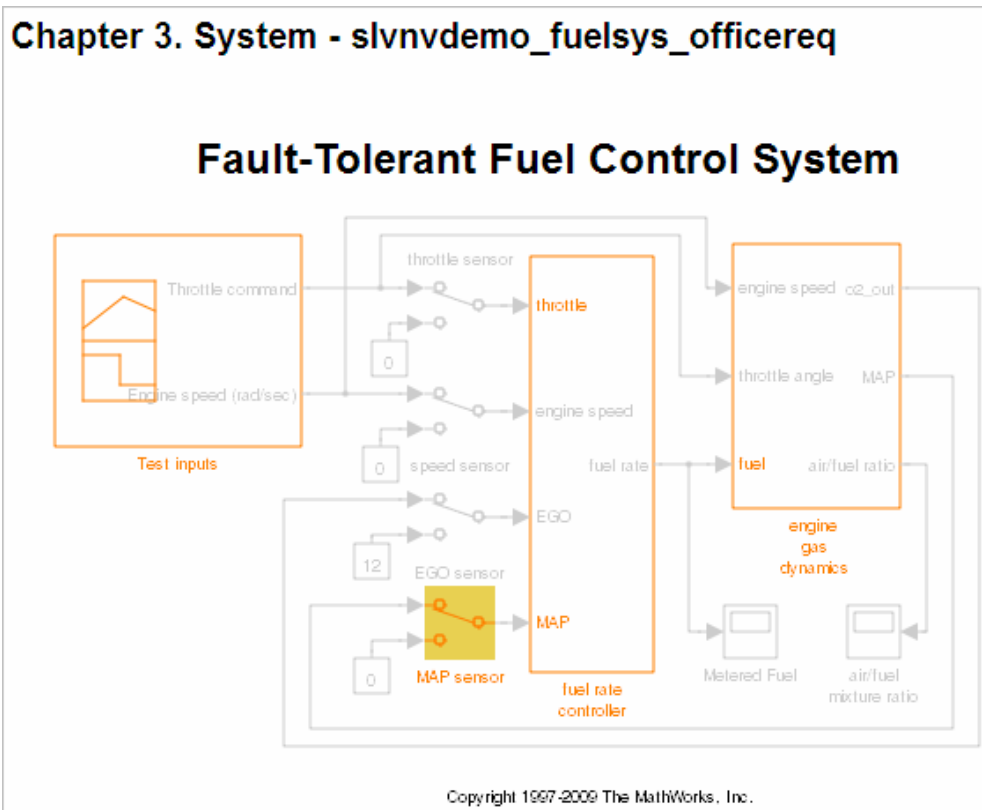
When your requirements documents have long path names that can clutter the report, select the **User document IDs in requirements tables** option. This option is disabled by default, as you can see in the examples in this section.

- **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 Microsoft Word Document	slvndemo FuelSys DesignDescription.docx

- 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 data-bbox="383 359 1162 533">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 data-bbox="456 543 1162 594">Target: fuelsys_req_docs\slvndemo_FuelSys_TestScenarios.xlsx at 'Simulink requirement item 2'</p>

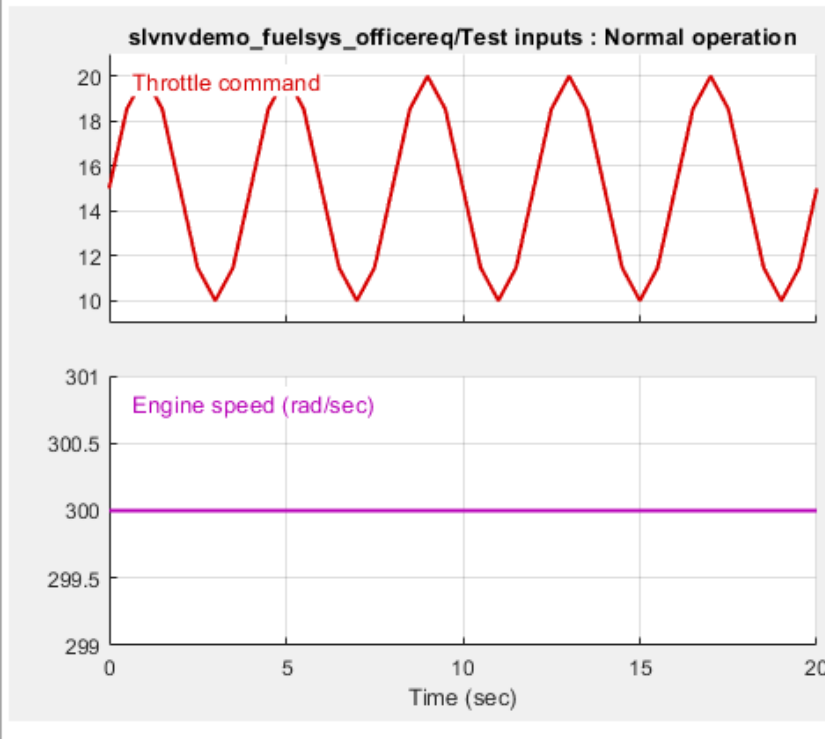
The preceding table does not include these blocks in the top-level model because:

- 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"	fuelsys_req_docs/slvndemo_FuelSys_TestScenarios.xlsx, at "Simulink_requirement_item_3"

[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

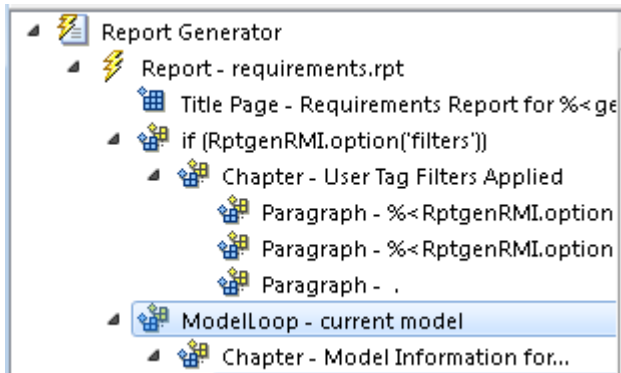
Name	Requirements
warmup	Link#1 label: "During a calibratable warm up period the oxygen sensor correction shall be disabled." Target: fuelsys_req_docs\slvndemo_FuelSys_RequirementsSpecification.docx, at 'Simulink requirement item 3'
[speed==0 & press < zero_thresh]/...	Link#1 label: "Speed sensor failure detection" Target: fuelsys_req_docs\slvndemo_FuelSys_DesignDescription.docx, at 'Simulink requirement item 6'
Rich_Mixture	Link#1 label: "Enriched mixture usage" Target: fuelsys_req_docs\slvndemo_FuelSys_DesignDescription.docx, at 'Simulink requirement item 4'
Warmup	Link#1 label: "During a calibratable warm up period the oxygen sensor correction shall be disabled." Target: fuelsys_req_docs\slvndemo_FuelSys_RequirementsSpecification.docx, at 'Simulink requirement item 3'

Report for Requirements in Model Blocks

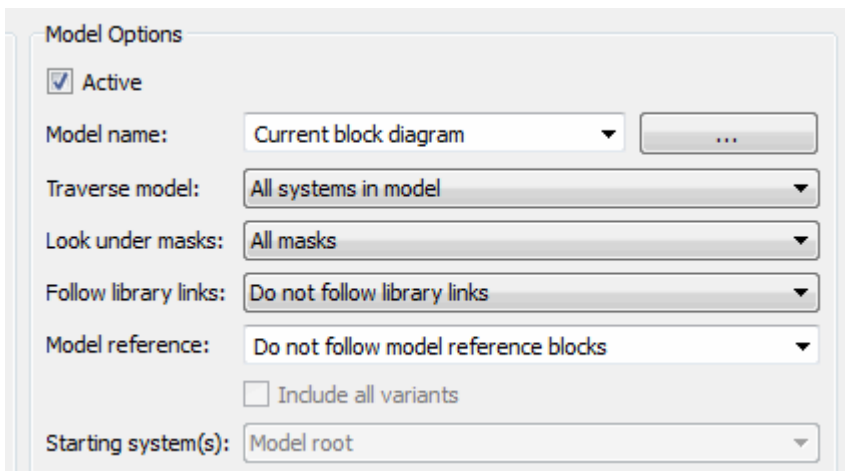
If your model contains Model blocks that reference external models, the default report does not include information about requirements in the referenced models. To generate a report that includes requirements information for referenced models, you must have a license for the Simulink Report Generator 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 .

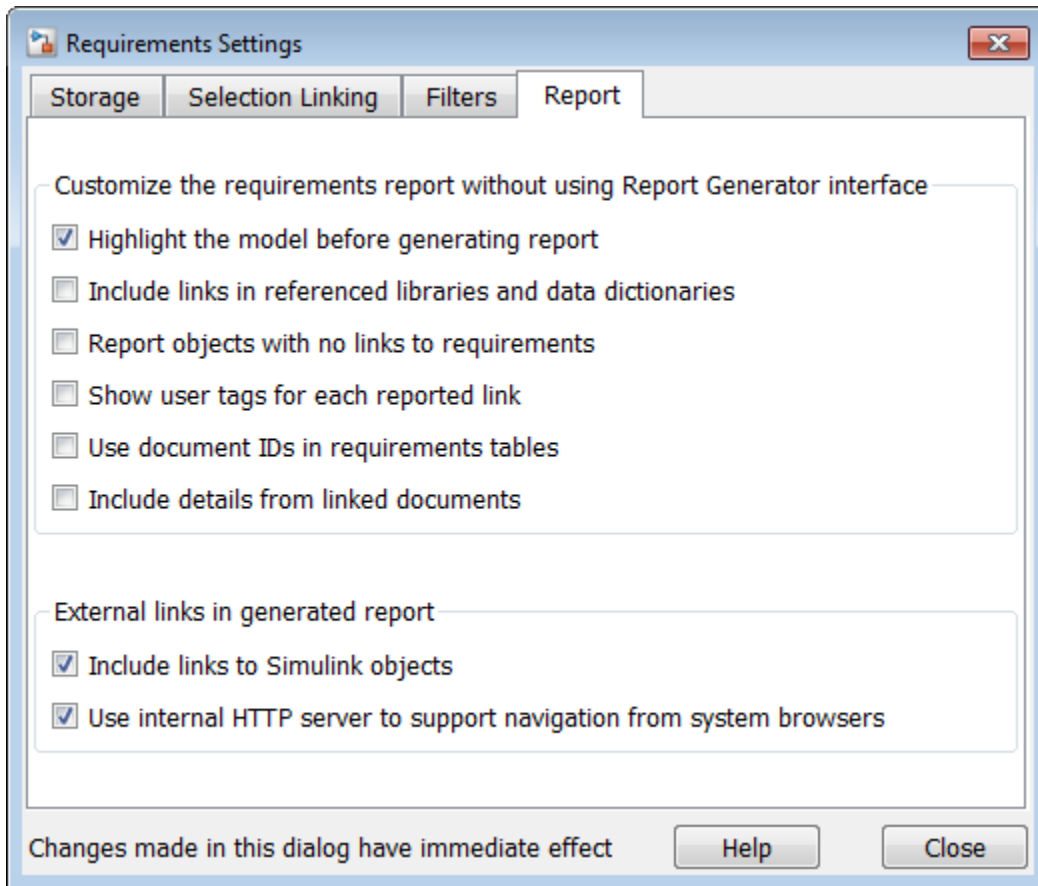
Customize Requirements Report

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 11-19
- “Customize Requirements Report Using Simulink Report Generator” on page 11-22

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
Highlight the model before generating report	Enables highlighting of Simulink objects with requirements in the report graphics.
Include links in referenced libraries and data dictionaries	Includes requirements links in referenced libraries in the generated report.
Report objects with no links to requirements	Includes lists of model objects that have no requirements.
Show user tags for each reported link	Lists the user tags, if any, for each reported link.
Use document IDs in requirements tables	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
Include details from linked documents	Includes additional content from linked requirements. The following requirements documents are supported: <ul style="list-style-type: none"> • Microsoft Word • Microsoft Excel • IBM Rational DOORS
Include links to Simulink objects	Includes links from the report to objects in Simulink.
Use internal HTTP server to support navigation from system browsers	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 tags for each reported link** — The report lists the user tags (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"> • DOORS Object Heading • DOORS Object Text • All other attributes except Created Thru, attributes with empty string values, and system attributes that are false. <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 tags 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.

Simulink Report Generator Component	Report Information
Missing Requirements Block Loop (Simulink Report Generator)	Applies all child components to blocks that have no requirements
Missing Requirements System Loop (Simulink Report Generator)	Applies all child components to systems that have no requirements
Requirements Block Loop (Simulink Report Generator)	Applies all child components to blocks that have requirements
Requirements Documents Table (Simulink Report Generator)	Inserts a table that lists requirements documents
Requirements Signal Loop (Simulink Report Generator)	Applies all child components to signal groups with requirements
Requirements Summary Table (Simulink Report Generator)	Inserts a property table that lists requirements information for blocks with associated requirements
Requirements System Loop (Simulink Report Generator)	Applies all child components to systems with requirements
Requirements Table (Simulink Report Generator)	Inserts a table that lists system and subsystem requirements
Data Dictionary Traceability Table (Simulink Report Generator)	Inserts a table that links data dictionary information to requirements
MATLAB Code Traceability Table (Simulink Report Generator)	Inserts a table that links MATLAB code to requirements
Simulink Test Suite Traceability Table (Simulink Report Generator)	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 11-11.

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 11-19.

See Also

More About

- “Report Requirements Information” on page 4-10
- “Requirement Links” on page 2-32

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

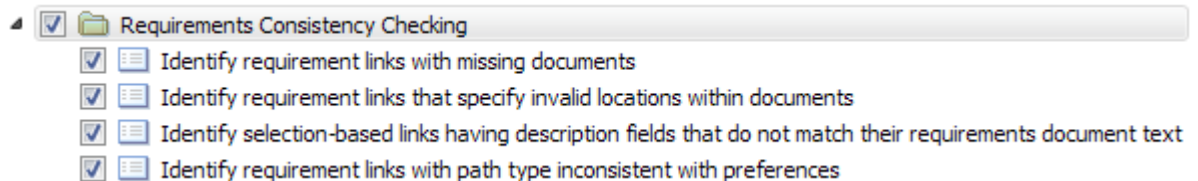
Validate Requirements Links in a Model

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
Identify requirement links with missing documents	The Model Advisor cannot find the requirements document. This might indicate a problem with the path to the requirements document.
Identify requirement links that specify invalid locations within documents	The Model Advisor cannot find the designated location for the requirement. This check is implemented for: <ul style="list-style-type: none"> • Microsoft Word documents • Microsoft Excel documents • IBM Rational DOORS documents • Simulink objects

Consistency Check	Problem Identified
Identify selection-based links having description fields that do not match their requirements document text	<p>The Description 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 Description field. This check is implemented for:</p> <ul style="list-style-type: none"> • Microsoft Word documents • Microsoft Excel documents • IBM Rational DOORS documents • Simulink objects
Identify requirement links with path type inconsistent with preferences	<p>The path to the requirements document does not match the Document file reference field in the Requirements Settings dialog box Selection Linking tab. This might indicate a problem with the path to the requirements document.</p> <p>On Linux systems, this check is named Identify requirement links with absolute path type. The check reports a warning for each requirements links that uses an absolute path.</p> <hr/> <p>Note For information about how the RMI resolves the path to the requirements document, see “Document Path Storage” on page 11-35.</p>

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

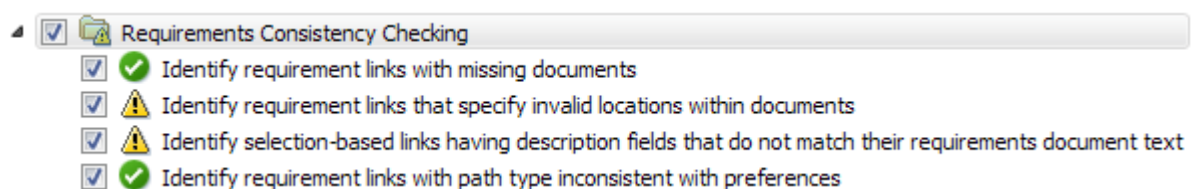
- If your model has links to Microsoft Word or Microsoft Excel documents, the consistency check requires that you close all instances of those applications. If you have one of these applications open, it displays a warning and does not continue the checks. The consistency checks must verify up-to-date 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 7-59 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 11-26, 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.

Block	Requirements
slvndemo_fuelsys_officereq/fuel_rate_controller/Sensor_correction_and Fault Redundancy/Terminator1	This section will be deleted

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 10-6 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	
slvndemo_fuelsys_officereq/Test inputs	Normal mode of operation	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.	Update
slvndemo_fuelsys_officereq/fuel rate controller/Sensor correction and Fault Redundancy/MAP Estimate	Manifold pressure failure	Manifold pressure failure mode	Update

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 10-6 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:

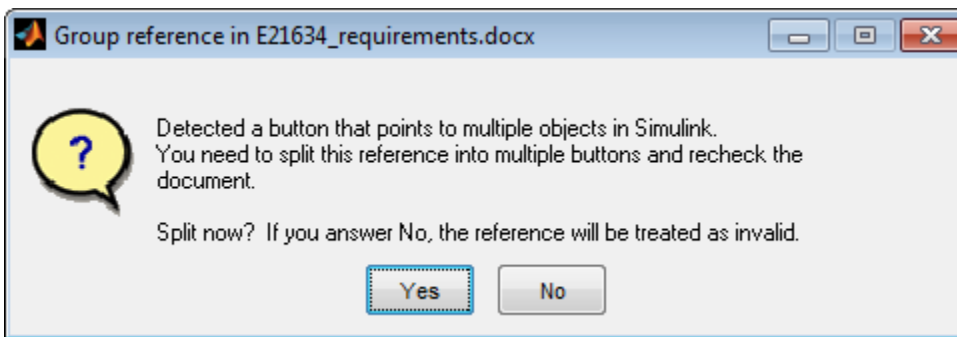
If you want to...	Do the following...
Fix the invalid links	Follow the instructions in “Fix Invalid Links in a Requirements Document” on page 11-31.
Keep the changes to the navigation controls without fixing the invalid links	Save the requirements document.

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
Transmission Requirements	sf_car_doors.mdl
Engine Torque Requirements	

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.	Fix all
Reset the navigation controls for these invalid links to their original state, the state before you checked the requirements document.	Reset all
Make no changes to the requirements document. Any modifications rmi made to the navigation controls remain in the requirements document.	Cancel

- 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 Simulink Requirements 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 11-32
- “How the rmi Function Checks a Requirements Document” on page 11-33

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 11-26.

Note For more information about inserting navigation controls in requirements documents, see:

- “Insert Navigation Objects in Microsoft Office Documents” on page 6-11
 - “Insert Navigation Objects into IBM Rational DOORS Requirements” on page 7-35
-

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 C:\work\models\reqs\pid.html

Relative (No) Path Example

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

Absolute Path Example

Current MATLAB folder	C:\work\scratch
-----------------------	-----------------

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

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 11-44 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);
rmimap.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.

```
rmidemo_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.

```
rmidemo_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.

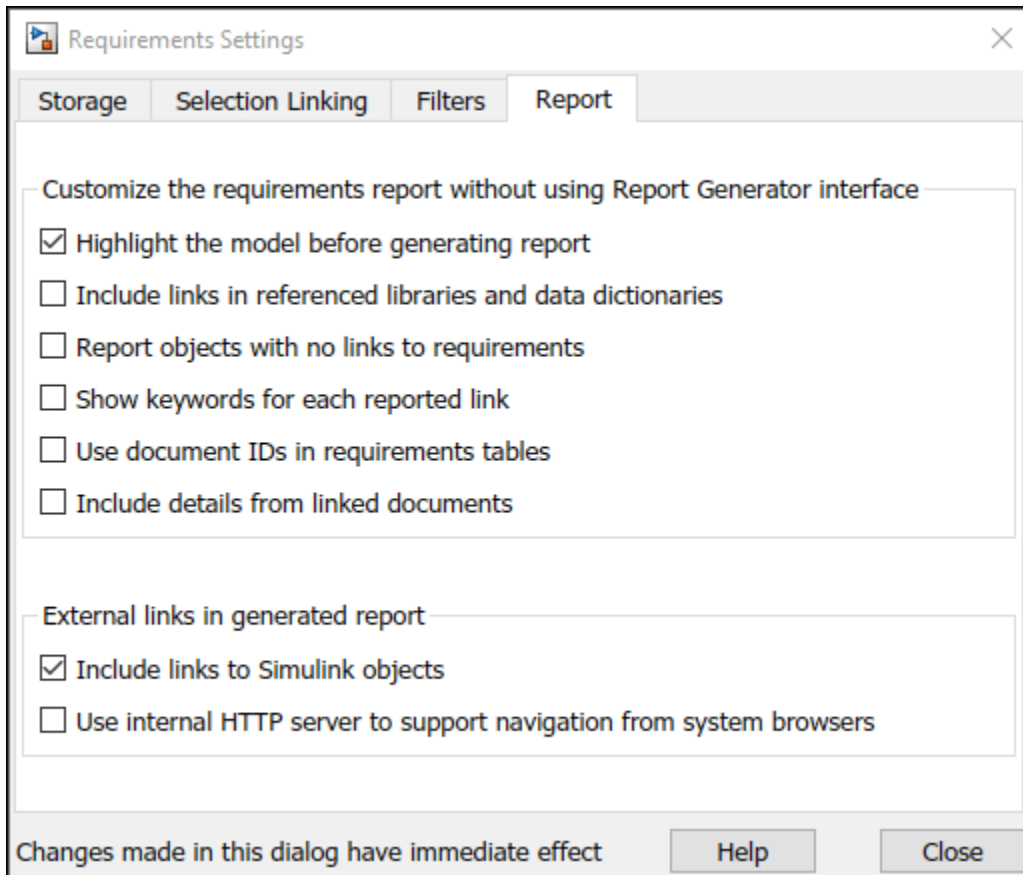
```
rmidemo_callback('locate', 'slvndemo_powerwindowController/Mux1');
```

- Passenger-side Mux4 links to just a subheader. Evaluate the code to navigate to the Mux4 block.

```
rmidemo_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
control	1. "High Level Discrete Event Control Specification - header" PowerWindowSpecificationWord.sreqx, at "6"
Mux1	1. "driver input - includel bullet items" PowerWindowSpecificationWord.sreqx, at "8"
Mux4	1. "passenger input - sub-header" PowerWindowSpecificationWord.sreqx, at "7"
Truth Table	1. "Signal mapping table with title" PowerWindowSpecificationWord.sreqx, at "11"
Truth Table1	1. "Signal mapping table with title" PowerWindowSpecificationWord.sreqx, at "11"

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.

	up signal * down: the passenger control switch generates the down signal																									
Truth Table	Link#1 label: "Signal mapping table with title" Target: \powerwin_reqs\powerwindowSpecification.docx, at 'Simulink requirement item 2' Details: The passenger and driver input signals are generated by mapping the up and down signals according to the following table <table border="1"> <thead> <tr> <th>up</th> <th>down</th> <th>neutral</th> <th>up</th> <th>down</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> </tr> </tbody> </table>	up	down	neutral	up	down	0	0	1	0	0	0	1	0	0	1	1	0	0	1	0	1	1	1	0	0
up	down	neutral	up	down																						
0	0	1	0	0																						
0	1	0	0	1																						
1	0	0	1	0																						
1	1	1	0	0																						

- 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	<p>Link#1 label: "driver input - includel bullet items"</p> <p>Target: .powerwin_reqs\powerwindowsspecification.docx, at 'Simulink_requirement_item_3'</p> <p>Details: driver input</p> <ul style="list-style-type: none"> * neutral: the driver control switch is not depressed * up: the driver control switch generates the up signal * down: the driver control switch generates the down signal
Mux4	<p>Link#1 label: "passenger input - sub-header"</p> <p>Target: .powerwin_reqs\powerwindowsspecification.docx, at 'Simulink_requirement_item_4'</p> <p>Details: passenger input consists of a vector with three elements</p> <ul style="list-style-type: none"> * neutral: the passenger control switch is not depressed * up: the passenger control switch generates the up signal * 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]	<p>Link#1 label: "ENDSTOP - one cell"</p> <p>Target: .powerwin_reqs\powerwindowsspecification.xlsx, at 'Simulink_requirement_item_2'</p> <p>Details: ENDSTOP CONTROL DISCRETE BOOLEAN TRUE,'FALSE'</p>												
[endstop]	<p>Link#1 label: "AD1.3 - subtable"</p> <p>Target: .powerwin_reqs\powerwindowsspecification.xlsx, at 'Simulink_requirement_item_1'</p> <p>Details: 4x6 range in C:\Work\Aslrtw\matlab\toolbox\slvn\rmidemos\powerwin_reqs\powerwindowsspecification.xlsx</p> <table border="1"> <tr> <td>AD1.3</td> <td>DETECT_OBSTACLE_ENDSTOP</td> <td></td> </tr> <tr> <td>ENDSTOP_MIN</td> <td>DATA</td> <td>CONSTANT REAL VALUE: 0.0 [m]</td> </tr> <tr> <td>ENDSTOP_MAX</td> <td>DATA</td> <td>CONSTANT REAL VALUE: 0.4 [m]</td> </tr> <tr> <td>OBSTACLE_MAX</td> <td>DATA</td> <td>CONSTANT REAL VALUE: 0.3 [m]</td> </tr> </table>	AD1.3	DETECT_OBSTACLE_ENDSTOP		ENDSTOP_MIN	DATA	CONSTANT REAL VALUE: 0.0 [m]	ENDSTOP_MAX	DATA	CONSTANT REAL VALUE: 0.4 [m]	OBSTACLE_MAX	DATA	CONSTANT REAL VALUE: 0.3 [m]
AD1.3	DETECT_OBSTACLE_ENDSTOP												
ENDSTOP_MIN	DATA	CONSTANT REAL VALUE: 0.0 [m]											
ENDSTOP_MAX	DATA	CONSTANT REAL VALUE: 0.4 [m]											
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: \powerwin_reqs\powerwindowsspecification.docx, at 'Simulink_requirement_item_5'</p> <p>Details: Links to the following paragraph under 'Driver-side Precedence': * 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: \powerwin_reqs\powerwindowsspecification.xlsx, at 'Simulink_requirement_item_2'</p> <p>Details: ENDSTOP CONTROL DISCRETE BOOLEAN 'TRUE','FALSE'</p>
	<p>Link#1 label: "AD1.3 - subtable"</p> <p>Target: \powerwin_reqs\powerwindowsspecification.xlsx, at 'Simulink_requirement_item_1'</p> <p>Details: 4x6 range in 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 Details: Scope Description</p> <ul style="list-style-type: none"> * TPS2000 Digital Storage Oscilloscope Series * Powerful productivity from bench to field. * 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>Created By: astarovo Created On: 21 January 2011 Created Thru: Manual Input Last Modified By: astarovo Last Modified On: 16 March 2011</p>	DOORS_m Project/Scr
	<p>Label: 5.1 Pulse Generator connects to Scope Details: Pulse Generator connects to Scope</p> <ul style="list-style-type: none"> * 50MHz sine wave frequency * 25MHz pulse frequency * Arbitrary waveform generator with 256k-point, 14-bit resolution * Built-in function generator capability includes: sine, square, triangle, noise, DC, etc. * Precision pulses and square waves with fast (5ns) rise/fall times * Built-in 10MHz external time base for multiple unit synchronization * Built-in AM, FM, PM, FSK, PWM modulation * Frequency sweep and burst capability 	

- 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` is a function.

Cleanup

Cleanup commands. Clears open requirement sets without saving changes, and closes open models without saving changes.

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

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('slvndemo_powerwindowController', 'clear');
```

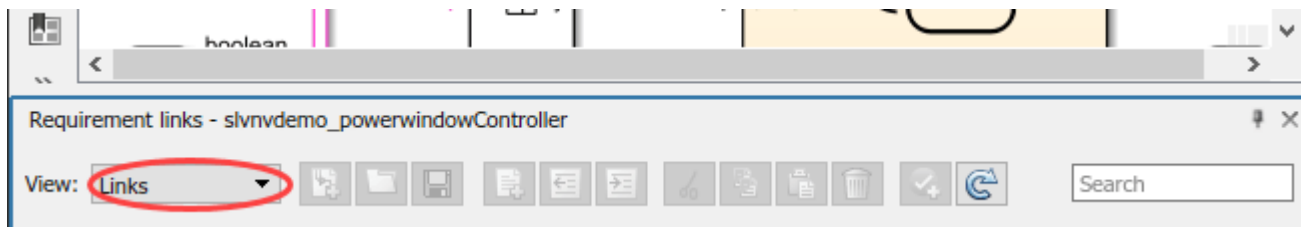
```
Nothing to clear for ...\slrequirements-ex04732634\slvndemo_powerwindowController.slx
```

```
open_system('slvndemo_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**.

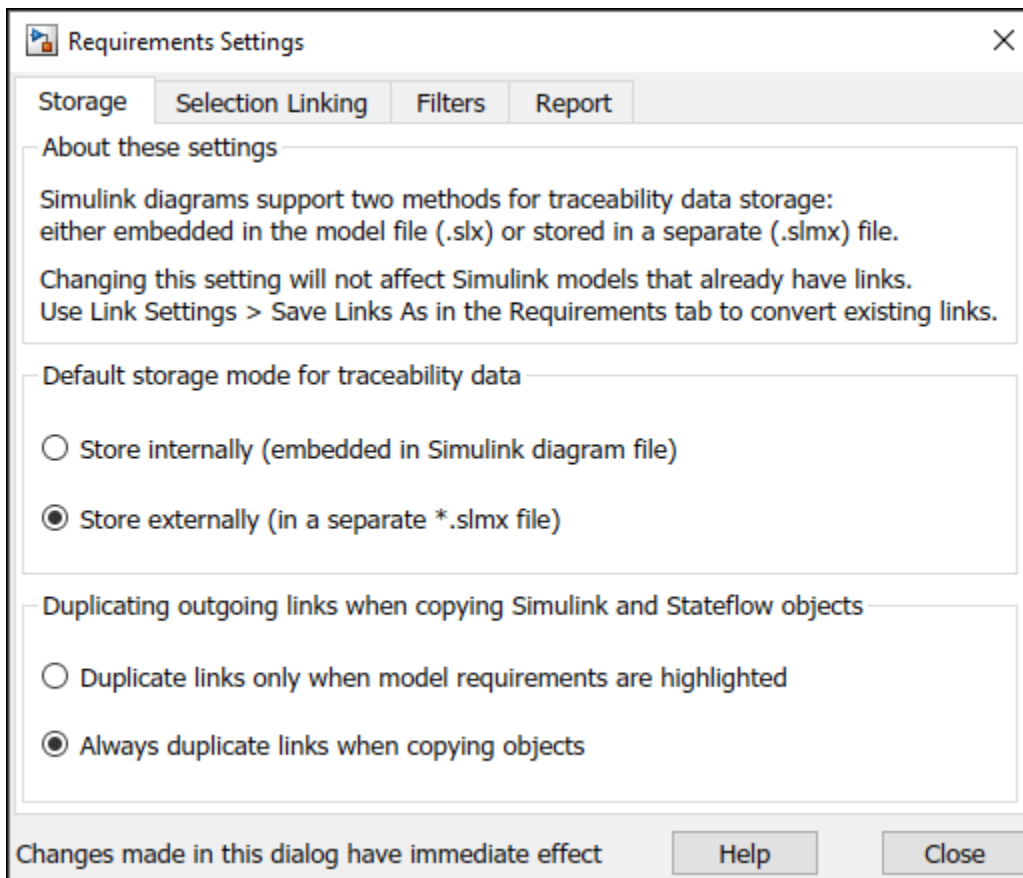
In this example, you will work exclusively in the **Requirements** tab and any references to toolbar buttons are in this tab.



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`.

```
rmdemo_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 `slvndemo_powerwindowController.slmx`.

Close the model.

```
close_system('slvndemo_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 `slvndemo_powerwindowRequirements.slmx` in the working directory, or evaluate the following code.

```
open_system('slvndemo_powerwindowController');  
otherReqFile = 'slvndemo_powerwindowRequirements.slmx';  
rmimap.map('slvndemo_powerwindowController', otherReqFile);
```

Mapping `...\slrequirements-ex04732634\slvndemo_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','slvndemo_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('slvndemo_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.slx`

```
rmidata.saveAs('slvndemo_fuelsys_officereq','slvndemo_fuelsys_officereq.slmx');
```

Points to keep in mind before you move internally stored links to an external file:

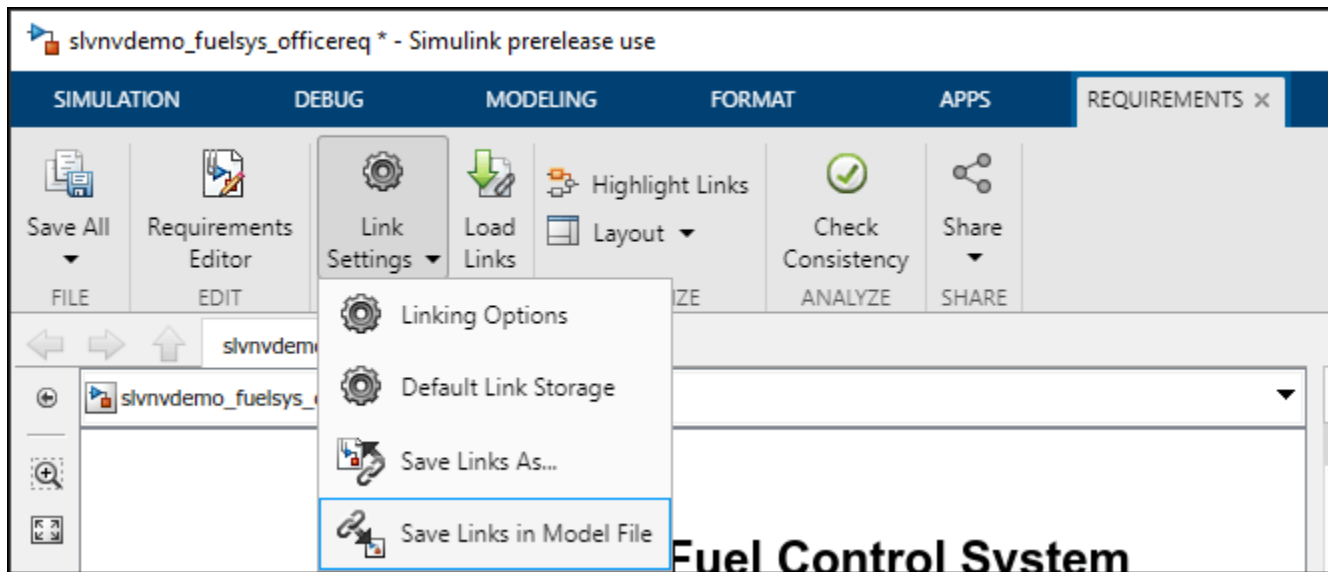
- You will need to carry an extra `.slmx` file along with the model file.
- Non-default file name and location associations are stored in user preferences. If you move or rename the `.slmx` file outside MATLAB, you will have to manually point RMI to the new location when the model is reopened.
- When 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';
```

Requirements Management Interface

Verification and Validation

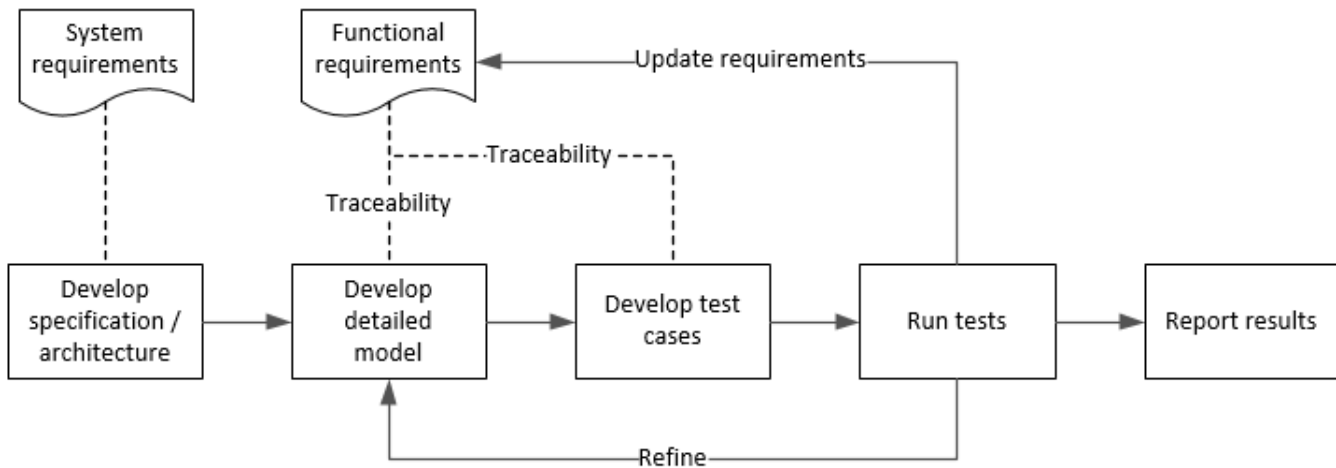
- “Test Model Against Requirements and Report Results” on page 13-2
- “Analyze a Model for Standards Compliance and Design Errors” on page 13-7
- “Perform Functional Testing and Analyze Test Coverage” on page 13-9
- “Analyze Code and Test Software-in-the-Loop” on page 13-12

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 Create a copy of the project in a working folder. The project contains data, documents, models, and tests. Enter:

```

path = fullfile(matlabroot, 'toolbox', 'shared', 'examples', ...
'verification', 'src', 'cruise')
run(fullfile(path, 'slVerificationCruiseStart'))
  
```

- 2 In the project `models` folder, open the `simulinkCruiseAddReqExample.slx` 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 Expand the requirements information to include verification and implementation status. Right-click a requirement and select **Verification Status** and **Implementation Status**.

The screenshot shows the Simulink Requirements Manager interface. On the left, a Simulink model is displayed with several input blocks: CruiseOnOff (boolean), Brake (boolean), Speed (single), CoastSetSw (boolean), and AccelResSw (boolean). These are connected to a central block labeled 'Compute target speed'. On the right, the Property Inspector shows details for Requirement A 1.2, including its type (Functional), index (1.2), and summary (Set Speed / Decelerate Button). Below the model, the Requirements table is visible, showing a list of requirements with columns for Index, ID, Summary, Verified, and Implemented.

Index	ID	Summary	Verified	Implemented
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.

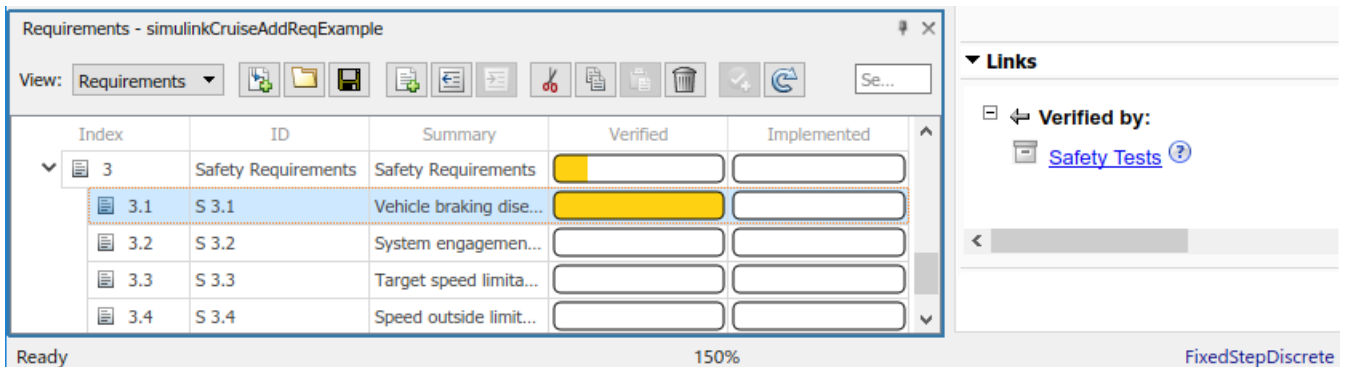
Link Requirements to Tests

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:

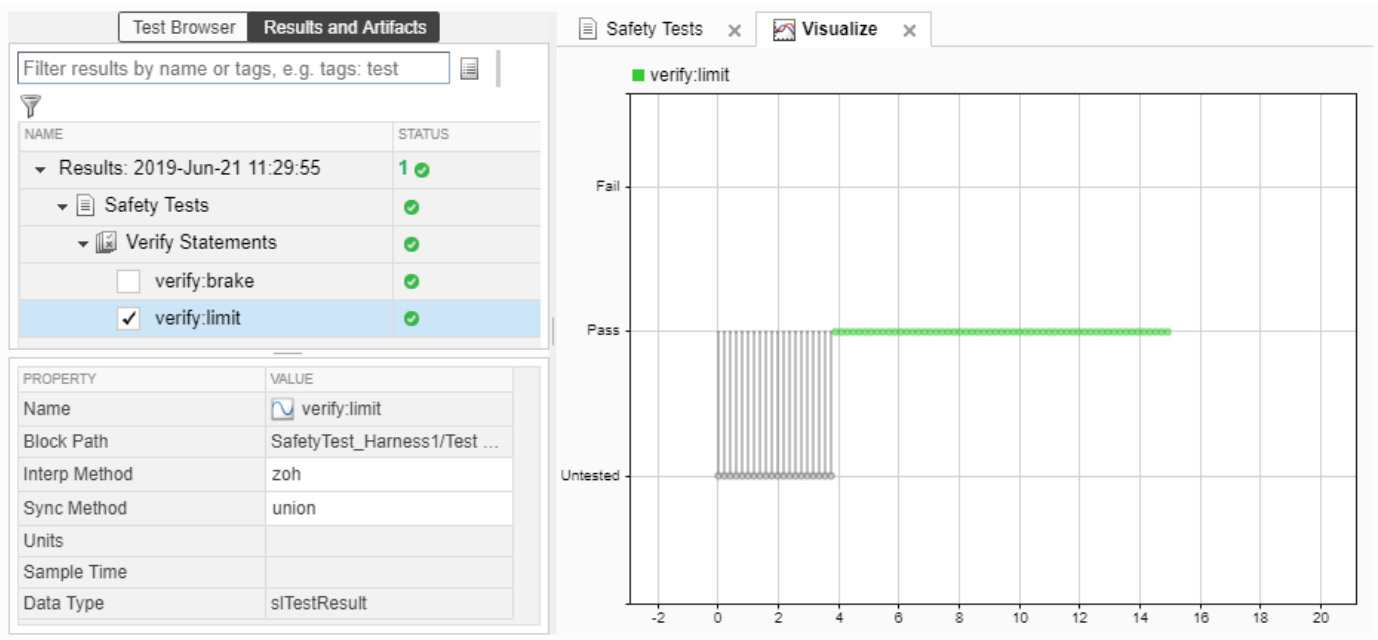
- The `BrakeTest` sequence engages the cruise control, then applies the brake. It includes the `verify` statement

```
verify(engaged == false,...
      'verify:brake',...
      'system must disengage when brake applied')
```

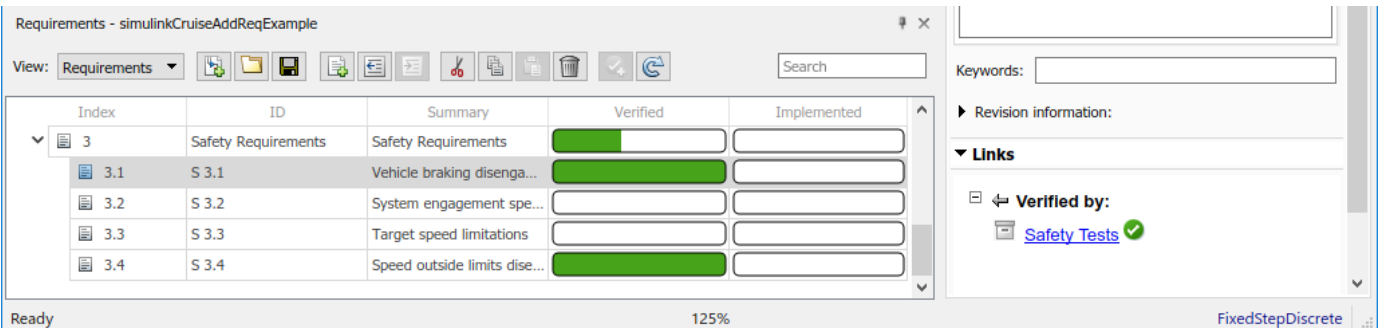
- The `LimitTest` sequence engages the cruise control, then ramps up the vehicle speed until it exceeds the upper limit. It includes the `verify` statement.

```
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 11-26
- “Customize Requirements Traceability Report for Model” on page 11-11

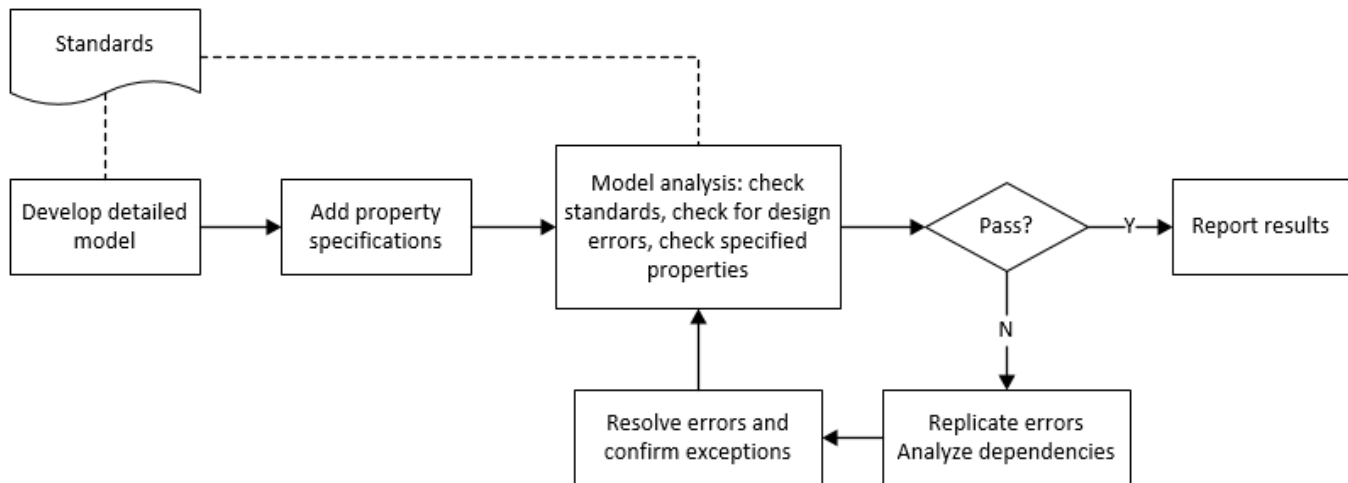
External Websites

- Requirements-Based Testing Workflow

Analyze a Model 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

In Model Advisor, you can check that your model complies with MAB modeling guidelines.

- 1 Create a copy of the project in a working folder. On the command line, enter

```
path = fullfile(matlabroot, 'toolbox', 'shared', 'examples', ...
'verification', 'src', 'cruise')
run(fullfile(path, 'slVerificationCruiseStart'))
```

- 2 Open the model. On the command line, enter

```
open_system simulinkCruiseErrorAndStandardsExample
```

- 3 In the **Modeling** tab, select **Model Advisor**.
- 4 Click OK to choose `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 Selected Checks**.
- c To review the configuration parameter settings that violate MAB style guidelines, click on the **Check model diagnostic parameters** check. The analysis results appear in the right pane and include the recommended action.
- d Click the parameter hyperlinks, which opens the Configuration Parameters dialog box, and update the model diagnostic parameters. Save the model.
- e To verify that your model passes, rerun the check. Repeat steps c and d, if necessary, to reach compliance.
- f To generate a results report of the Simulink Check checks, select the **MAB Checks** node, and then, in the right pane click **Generate 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 In the right pane, select **Show report after run** and **Run Selected Checks**.
- 4 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 **Highlight analysis results on model**. Click the **Compute target speed** subsystem, outlined in red. 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

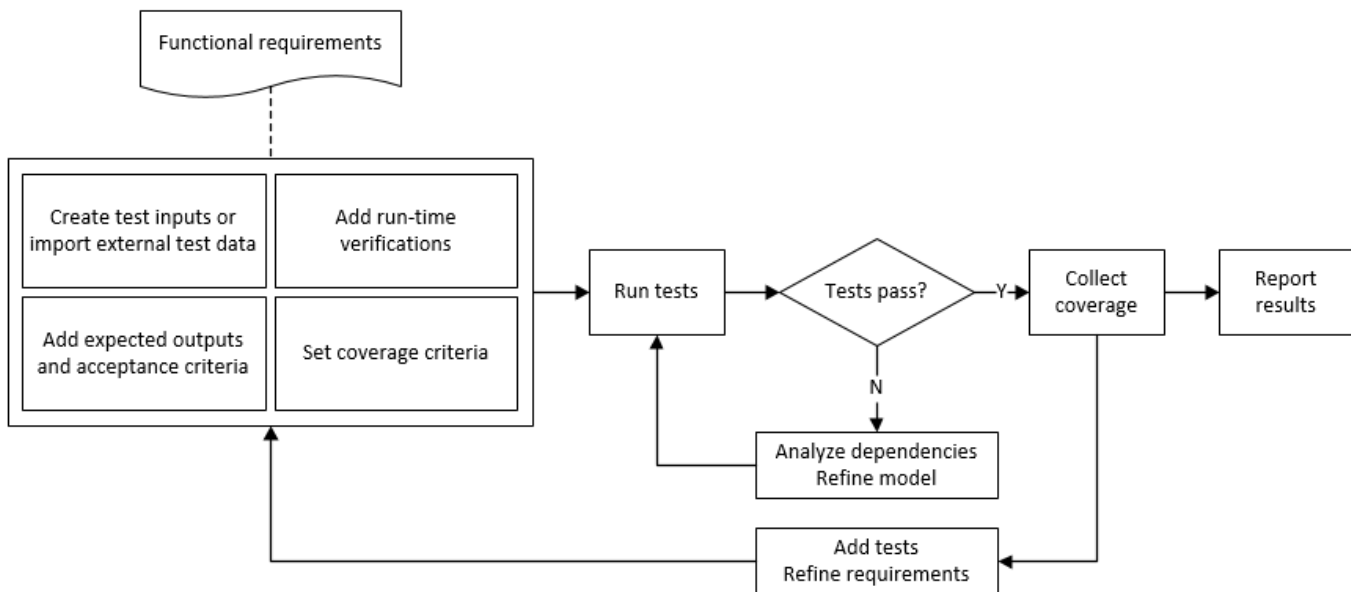
Related Examples

- “Check Model Compliance by Using the Model Advisor” (Simulink Check)
- “Collect Model Metrics Using the Model Advisor” (Simulink Check)
- “Run a Design Error Detection Analysis” (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 a functional testing-based testing workflow for a cruise control model. You start with a model that has tests linked to an external requirements document, analyze the model for coverage in Simulink Coverage, incrementally increase coverage with Simulink Design Verifier, and report the results.

Explore the Test Harness and the Model

- 1 Create a copy of the project in a working folder. At the command line, enter:

```
path = fullfile(matlabroot,'toolbox','shared','examples',...
'verification','src','cruise')
run(fullfile(path,'slVerificationCruiseStart'))
```

- 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. At the command line, enter:

```
sltest.testmanager.load('slReqTests.mldatx')
sltest.testmanager.view
```

- 4 Open the test sequence block. The sequence tests that the system disengages when the:
 - Brake pedal is pressed
 - Speed exceeds a limit

Some test sequence steps are linked to requirements document `simulinkCruiseChartReqs.docx`.

Measure Model Coverage

- 1 In the Simulink Test Manager, click the `slReqTests` test file.
- 2 To enable coverage collection for the test file, in the right page under **Coverage Settings**:
 - Select **Record coverage for referenced models**
 - Use **Coverage filter filename** to specify a coverage filter to use for the coverage analysis. The default setting honors the model configuration parameter settings. Leaving the field empty attaches no coverage filter.
 - Select **Decision**, **Condition**, and **MCDC**.
- 3 To run the tests, on the Test Manager toolstrip, click **Run**.
- 4 When the test finishes select the Results in the Test Manager. The aggregated coverage results show that the example model 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.

See Also

Related Examples

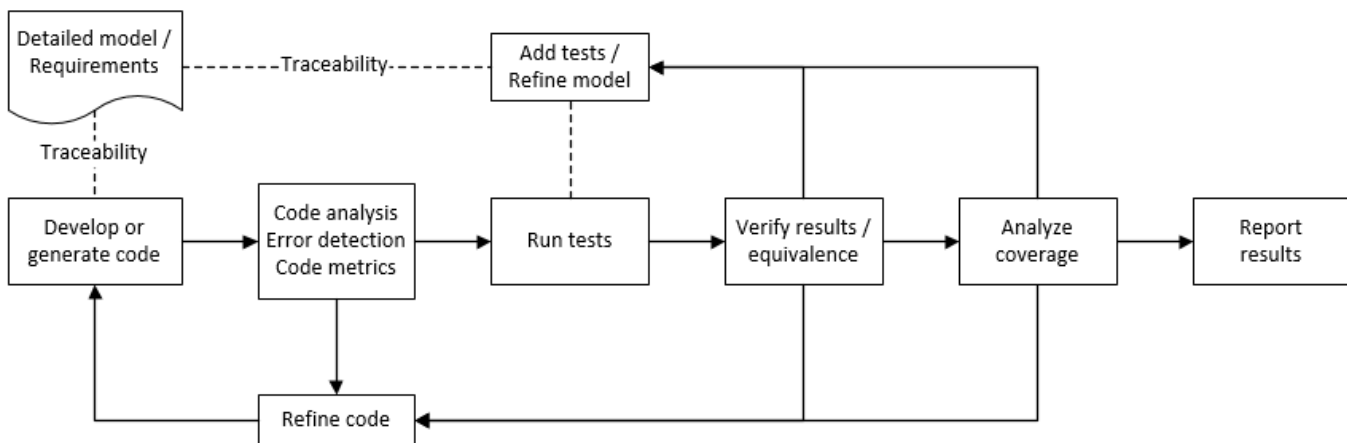
- “Link to Requirements” (Simulink Test)
- “Assess Model Simulation Using verify Statements” (Simulink Test)
- “Compare Model Output to Baseline Data” (Simulink Test)
- “Generate Test Cases for Model Decision Coverage” (Simulink Design Verifier)
- “Increase Test Coverage for a Model” (Simulink Test)

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 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

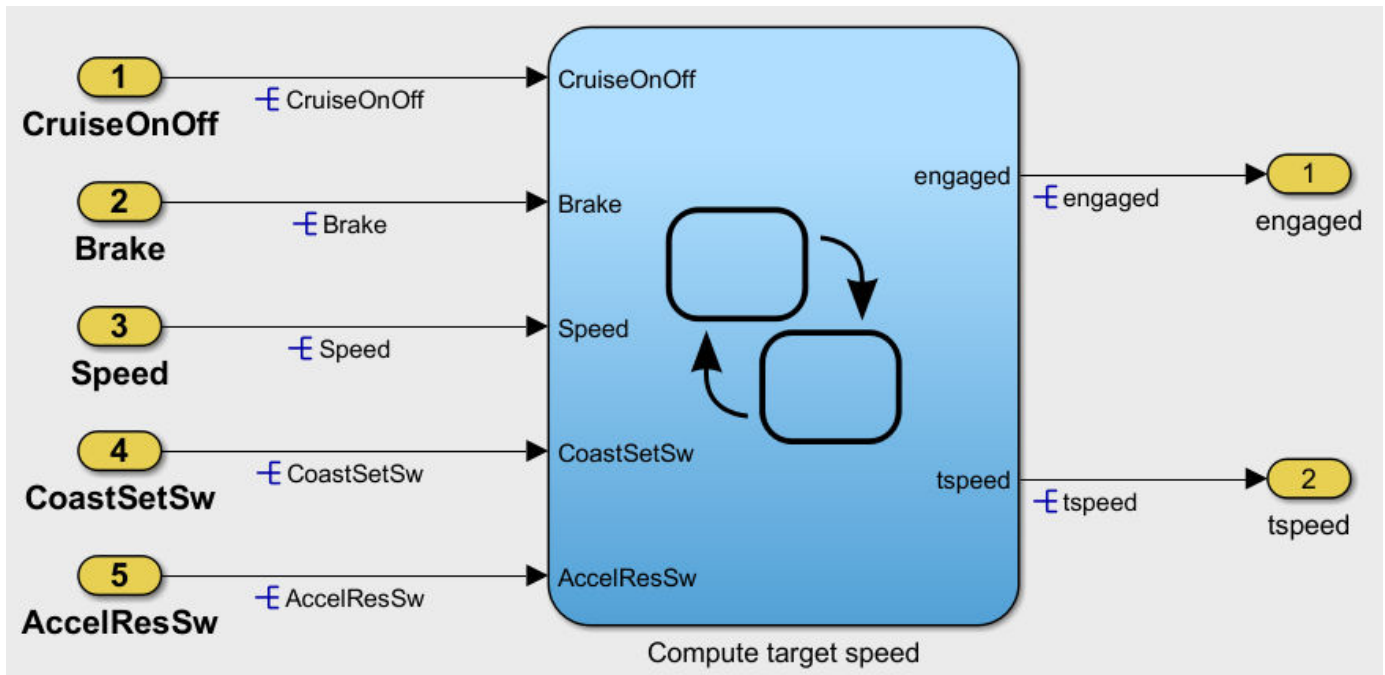
This workflow describes how to check if your model produces MISRA® C:2012 compliant code and how to check your generated code for code metrics and defects. To produce more MISRA compliant code from your model, you use the code generation and Model Advisor. To check whether the code is MISRA compliant, 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.

```

path = fullfile(matlabroot,'toolbox','shared','examples',...
'verification','src','cruise')
run(fullfile(path,'slVerificationCruiseStart'))
  
```

- 2 From the project, open the model `simulinkCruiseErrorAndStandardsExample`.

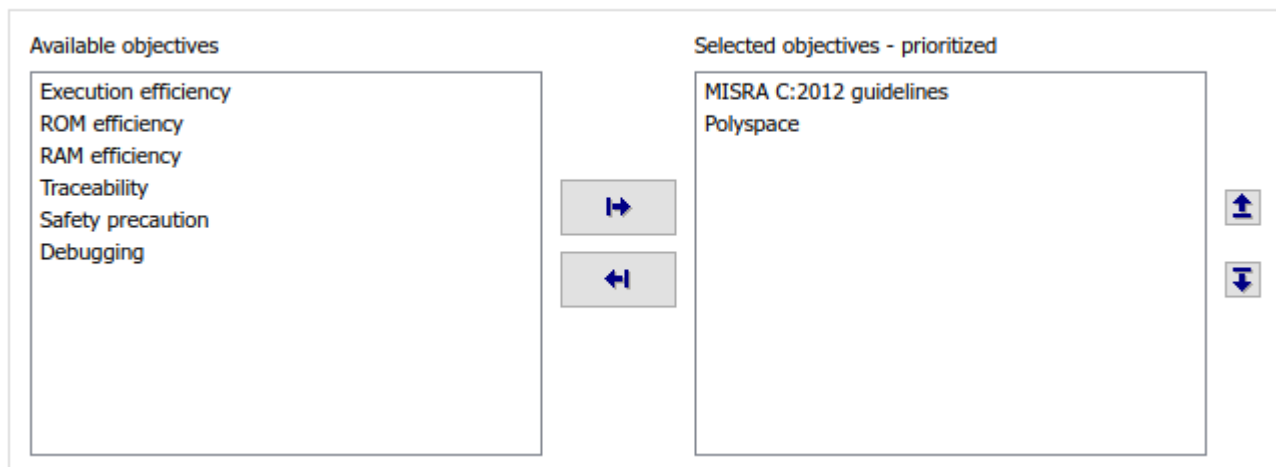


Run Code Generator Checks

Before you generate code from your model, use the Code Generation Advisor to check your model so that it generates 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.

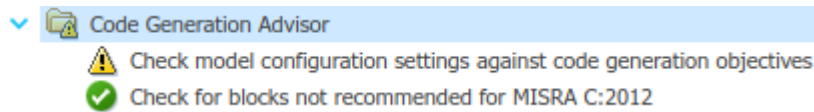
Code Generation Objectives (System target file: ert.tlc)



- 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

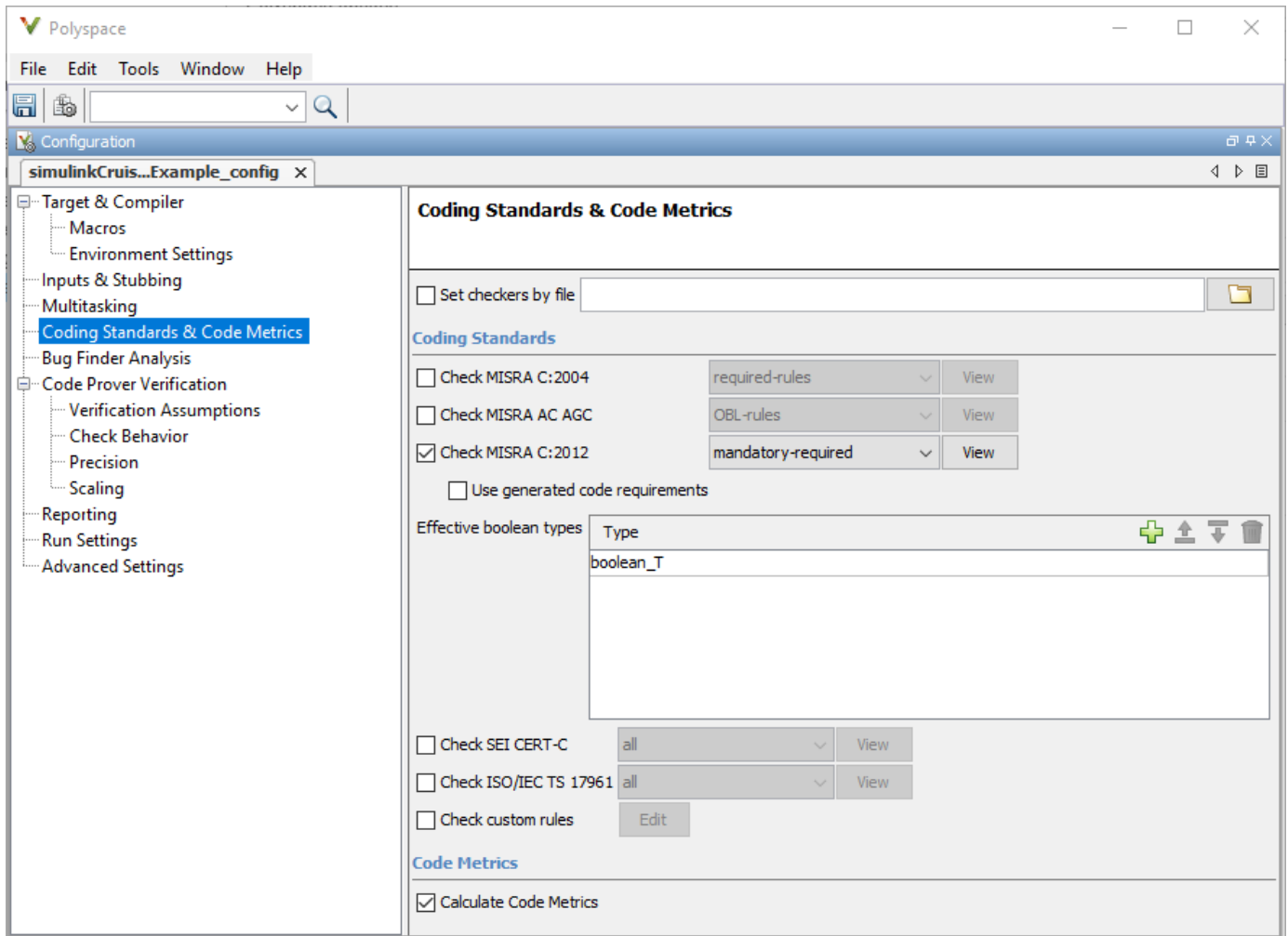
Before you generate code from your model, use the Model Advisor to check your model for MISRA C and Polyspace compliance. This example shows you how to use the Model Advisor to check your model before generating code.

- 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 Selected 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**.

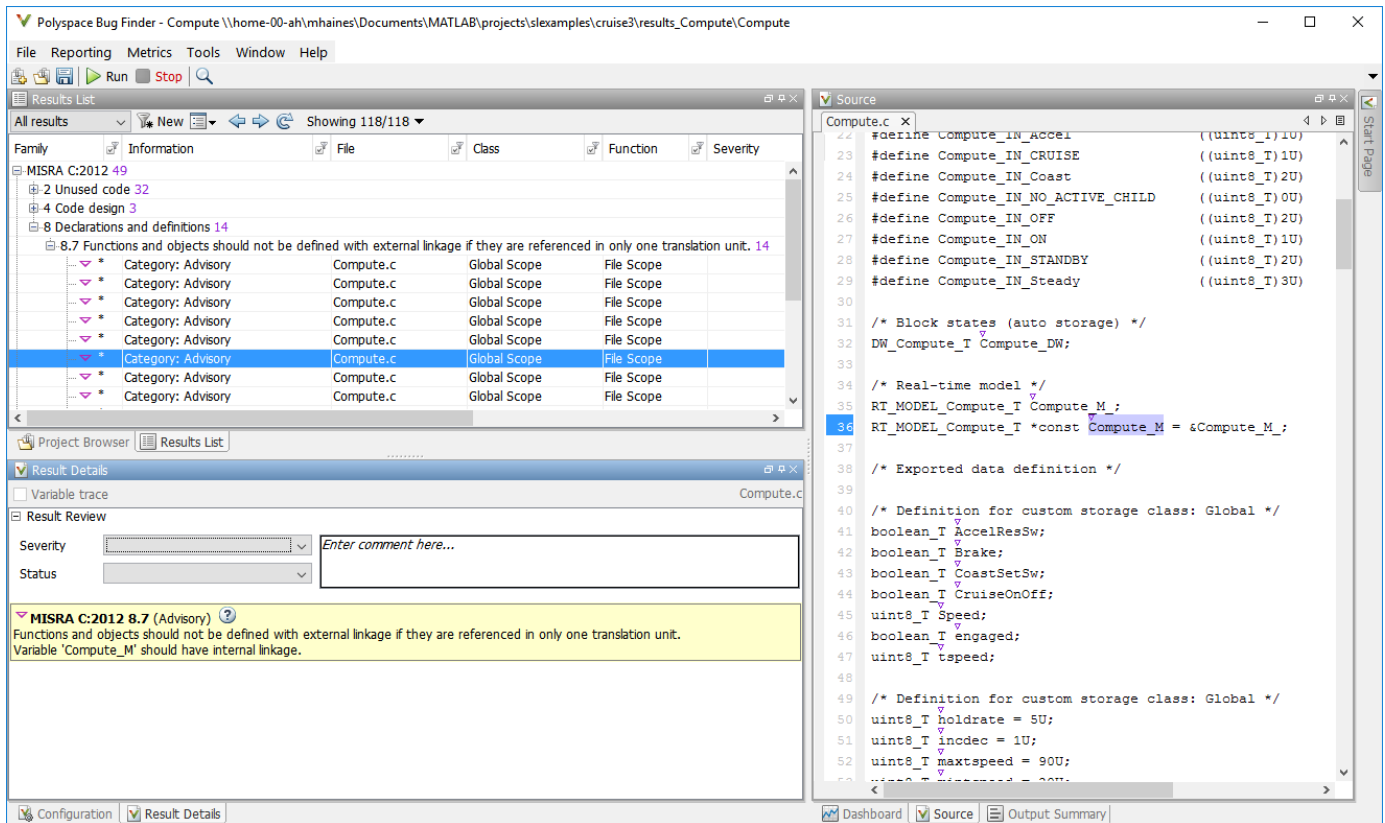
Polyspace Bug Finder analyzes the generated code for a subset of MISRA checks. You can see the progress of the analysis in the MATLAB Command Window. After the analysis finishes, the Polyspace environment opens.

Review Results

The Polyspace environment shows you the results of the static code analysis.

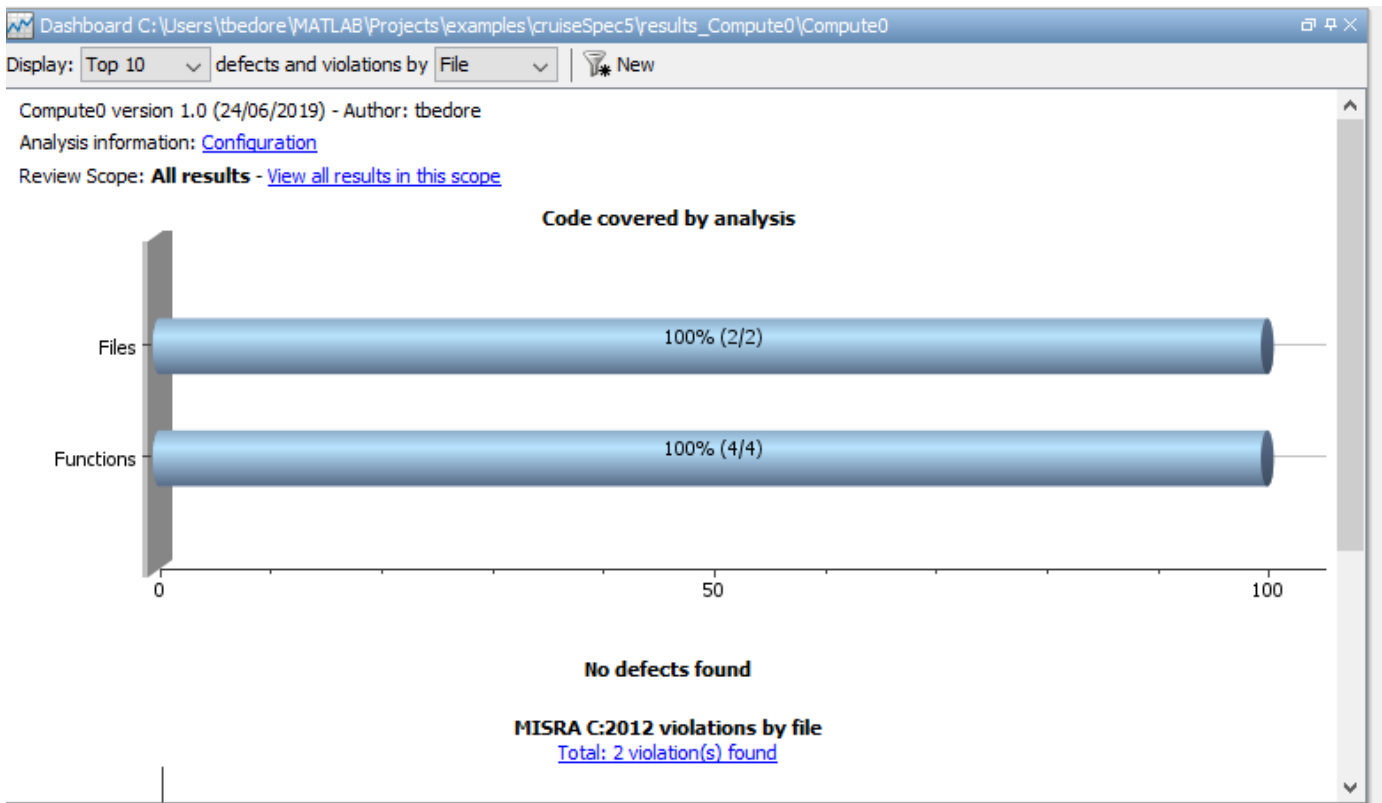
- 1 Expand the tree for rule 8.7 and click through the different results.

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.

The screenshot shows the Test Manager interface. The top toolbar includes icons for New, Open, Save, Cut, Copy, Paste, Delete, Test Spec Report, Run, Run with Stepper, Stop, Parallel, Report, Visualize, Highlight in Model, Export, Import, Testing Dashboard, Preferences, and Help. Below the toolbar, the Test Browser pane shows a list of test cases under the heading 'Results: 2021-Feb-04 10:35:44'. The 'SIL Equivalence Test Case' is selected. The Results and Artifacts pane shows the following details:

- SUMMARY**
- LOGS**
- DESCRIPTION**: Double-click to edit
- COVERAGE RESULTS**

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%	--	--	--

At the bottom of the Results and Artifacts pane, there are buttons for '+ Add Tests for Missing Coverage' and 'Export'.

The Test Browser pane also shows a table of properties for the selected test case:

PROPERTY	VALUE
Name	SIL Equivalence Te...
Status	1
Start Time	02/04/2021 10:36:10
End Time	02/04/2021 10:38:36
Type	Equivalence Test

- 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 “Processor-in-the-Loop Simulation” (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)

