

# Simulink® Requirements™

User's Guide



# MATLAB® & SIMULINK®

R2020a



## How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://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 2017–2020 by The MathWorks, Inc.

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

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

### **Trademarks**

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

### **Patents**

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

### **Revision History**

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)

## Requirements Definition

<b>Author Requirements in Simulink</b> .....	<b>1-2</b>
Author and Edit Requirements Content by Using Microsoft Word .....	<b>1-4</b>
Customize Requirements Browser View .....	<b>1-4</b>
<b>Requirement Types</b> .....	<b>1-6</b>
<b>Import Requirements from Third-Party Applications</b> .....	<b>1-7</b>
Path Settings .....	<b>1-7</b>
Import Requirements from Microsoft Office Documents .....	<b>1-7</b>
Import Requirements from ReqIF Files .....	<b>1-8</b>
Import Modes .....	<b>1-10</b>
<b>Define Requirements Hierarchy</b> .....	<b>1-12</b>
Requirement Sets .....	<b>1-12</b>
Custom Attributes of Requirement Sets .....	<b>1-12</b>
<b>Create Requirement Set File by Using the Simulink® Requirements™ API</b> .....	<b>1-14</b>
<b>Update Imported Requirements</b> .....	<b>1-17</b>
Considerations for Microsoft Word Documents .....	<b>1-18</b>
<b>Import and Update Requirements from a Microsoft Word Document</b> ..	<b>1-19</b>
Import Requirements .....	<b>1-19</b>
Update Requirements .....	<b>1-20</b>
<b>Export Requirement Sets and Link Sets to Previous Versions of Simulink Requirements</b> .....	<b>1-21</b>
Export Link Sets .....	<b>1-21</b>
Export Requirement Sets .....	<b>1-21</b>
<b>Use Command-line API to Document Simulink Model in Requirements Editor</b> .....	<b>1-22</b>
<b>Round Trip Workflows with ReqIF Files</b> .....	<b>1-40</b>
Import Requirements from ReqIF Files .....	<b>1-40</b>
Edit Imported Content .....	<b>1-41</b>
Export Requirements Content .....	<b>1-42</b>
<b>Best Practices and Guidelines for ReqIF Round Trip Workflows</b> .....	<b>1-43</b>
Managing Requirement Custom IDs .....	<b>1-43</b>
Guidelines for Updating Referenced Requirements Content .....	<b>1-43</b>
Guidelines for Editing Referenced Requirements Content .....	<b>1-43</b>
Guidelines for Adding Details to Imported Requirements .....	<b>1-43</b>

Guidelines for Exporting Requirements to ReqIF Files . . . . .	1-43
<b>Create and Edit Attribute Mappings</b> . . . . .	<b>1-45</b>
Specify Default ReqIF Requirement Type . . . . .	1-45
Specify ReqIF Template . . . . .	1-46

## Requirements Traceability and Consistency

### 2

<b>Link Blocks and Requirements</b> . . . . .	<b>2-2</b>
Work with Simulink Annotations . . . . .	2-4
<b>Track Requirement Links with a Traceability Matrix</b> . . . . .	<b>2-5</b>
Generate a Traceability Matrix . . . . .	2-5
Using the Traceability Matrix . . . . .	2-7
Limitations . . . . .	2-10
<b>Requirement Links</b> . . . . .	<b>2-11</b>
Linkable Items . . . . .	2-11
Link Types . . . . .	2-12
Review Requirement Links . . . . .	2-13
Resolve Links . . . . .	2-13
Load and Unload Link Information . . . . .	2-13
<b>Define Custom Requirement and Link Types</b> . . . . .	<b>2-14</b>
<b>Requirements Consistency Checks</b> . . . . .	<b>2-16</b>
Check Requirements Consistency in Model Advisor . . . . .	2-16
<b>Manage Navigation Backlinks in External Requirements Documents</b> . . . . .	<b>2-20</b>
<b>Use Command-line API to Update or Repair Requirements Links</b> . . . . .	<b>2-21</b>

## Requirements-Based Verification

### 3

<b>Review Requirement Implementation Status Metrics Data</b> . . . . .	<b>3-2</b>
<b>Summarize Requirements Verification Status</b> . . . . .	<b>3-3</b>
Display Verification Status . . . . .	3-3
Update Verification Status by Running Tests or Analyses . . . . .	3-4
Include Verification Status in Report . . . . .	3-5
<b>Justify Requirements</b> . . . . .	<b>3-6</b>
<b>Linking to a Test Script</b> . . . . .	<b>3-8</b>
Linking to a Test Script Using the Outgoing Links Editor . . . . .	3-8
Linking to a Test Script Using the API . . . . .	3-11
Integrating Results from a MATLAB Unit Test Case . . . . .	3-14

<b>Include Results from External Sources in Verification Status</b> .....	<b>3-16</b>
How to Populate Verification Results from External Sources .....	<b>3-16</b>
<b>Linking to a Result File</b> .....	<b>3-19</b>
Linking to a Result File Using the Outgoing Links Editor .....	<b>3-19</b>
Linking to a Result File Using the API .....	<b>3-22</b>
<b>Validate Requirements by Analyzing Model Properties</b> .....	<b>3-26</b>
<b>Integrating results from a custom authored MATLAB script as a test</b> ..	<b>3-33</b>
<b>Integrating Results from an External Result file</b> .....	<b>3-37</b>
<b>Integrating results from a custom authored MUnit script as a test</b> ....	<b>3-41</b>

## Change Tracking and Team-Based Workflows

### 4

<b>Requirements-Based Development in Projects</b> .....	<b>4-2</b>
Organizing Requirements, Models, and Tests .....	<b>4-2</b>
<b>Track Changes to Requirements Links</b> .....	<b>4-3</b>
Enable Change Tracking for Requirements Links .....	<b>4-3</b>
Resolve Change Issues for Requirement Links .....	<b>4-4</b>
Add Comments to Links .....	<b>4-4</b>
Considerations for Using Links Change Tracking .....	<b>4-5</b>
<b>Compare Requirements Sets</b> .....	<b>4-6</b>
Compare Two .slreqx Simulink Requirements Sets .....	<b>4-6</b>
Review Changes in Source-Controlled Files .....	<b>4-6</b>
<b>Compare Link Sets</b> .....	<b>4-7</b>
<b>Report Requirements Information</b> .....	<b>4-8</b>
Report Navigation Links .....	<b>4-10</b>

## Requirements Management Interface Setup

### 5

<b>Configure RMI for Interaction with Microsoft Office and IBM Rational</b>	
<b>DOORS</b> .....	<b>5-2</b>
Configure RMI for Microsoft Office .....	<b>5-2</b>
Configure RMI for IBM Rational DOORS .....	<b>5-2</b>
Configure RMI for IBM Rational DOORS Next Generation .....	<b>5-3</b>
<b>Requirements Link Storage</b> .....	<b>5-4</b>
Save Requirements Links in External Storage .....	<b>5-4</b>
Load Requirements Links from External Storage .....	<b>5-5</b>
Move Internally Stored Requirements Links to External Storage .....	<b>5-5</b>

Move Externally Stored Requirements Links to the Model File .....	5-5
External Storage .....	5-6
Guidelines for External Storage of Requirements Links .....	5-6
<b>Supported Requirements Document Types .....</b>	<b>5-8</b>
<b>Requirements Settings .....</b>	<b>5-10</b>
Selection Linking Tab .....	5-10
Filter Requirements with User Tags .....	5-11

## Microsoft Office Traceability

### 6

<b>Link to Requirements in Microsoft Word Documents .....</b>	<b>6-2</b>
Create Bookmarks in a Microsoft Word Requirements Document .....	6-2
Open the Example Model and Associated Requirements Document .....	6-3
Create a Link from a Model Object to a Microsoft Word Requirements Document .....	6-4
<b>Link to Requirements in Excel Workbooks .....</b>	<b>6-6</b>
Navigate from a Model Object to Requirements in an Excel Workbook .....	6-6
Create Requirements Links to the Workbook .....	6-6
Link Multiple Model Objects to a Microsoft Excel Workbook .....	6-7
Change Requirements Links .....	6-7
<b>Navigate to Requirements in Microsoft Office Documents from Simulink .....</b>	<b>6-9</b>
Enable Linking from Microsoft Office Documents to Simulink Objects .....	6-9
Insert Navigation Objects in Microsoft Office Documents .....	6-9
Customize Microsoft Office Navigation Objects .....	6-10
Navigate Between Microsoft Word Requirement and Model .....	6-11

## Requirements Traceability with IBM Rational DOORS

### 7

<b>Configure Requirements Management Interface for IBM Rational DOORS Software .....</b>	<b>7-2</b>
Before You Begin .....	7-2
Manually Install Additional Files for DOORS Software .....	7-2
Diagnose and Fix DXL Errors .....	7-3
<b>Link with Requirements in DOORS Next Generation Project .....</b>	<b>7-4</b>
<b>Requirements Traceability with IBM Rational DOORS Next Generation .....</b>	<b>7-25</b>
Link to Requirements in IBM Rational DOORS Next Generation .....	7-25
Navigate to Requirements from Simulink .....	7-26
Work with IBM Rational DOORS Next Generation Projects with Configuration Management Enabled .....	7-26

<b>Navigate to Requirements in IBM Rational DOORS Databases from Simulink</b> .....	<b>7-28</b>
Enable Linking from IBM Rational DOORS Databases to Simulink Objects .....	<b>7-28</b>
Insert Navigation Objects into IBM Rational DOORS Requirements .....	<b>7-28</b>
Navigate Between IBM Rational DOORS Requirement and Model Object .....	<b>7-30</b>
Why Add Navigation Objects to IBM Rational DOORS Requirements? .....	<b>7-30</b>
Customize IBM Rational DOORS Navigation Objects .....	<b>7-31</b>
<b>Synchronize Simulink Models with IBM Rational DOORS Databases by using Surrogate Modules</b> .....	<b>7-32</b>
Synchronize a Simulink Model to Create a Surrogate Module .....	<b>7-32</b>
Create Links Between Surrogate Module and Formal Module in an IBM Rational DOORS Database .....	<b>7-33</b>
Resynchronize IBM Rational DOORS Surrogate Module to Reflect Model Changes .....	<b>7-34</b>
Navigate with the Surrogate Module .....	<b>7-35</b>
Customize IBM Rational DOORS Synchronization .....	<b>7-36</b>
Synchronization with IBM Rational DOORS Surrogate Modules .....	<b>7-41</b>
Advantages of Synchronizing Your Model with a Surrogate Module .....	<b>7-42</b>
<b>Working with IBM Rational DOORS 9 Requirements</b> .....	<b>7-43</b>

## Simulink Traceability Between Model Objects

# 8

<b>Link Model Objects</b> .....	<b>8-2</b>
Link Objects in the Same Model .....	<b>8-2</b>
Link Objects in Different Models .....	<b>8-2</b>
<b>Link Test Cases to Requirements Documents</b> .....	<b>8-3</b>
Establish Requirements Traceability for Testing .....	<b>8-3</b>
<b>Link Simulink Data Dictionary Entries to Requirements</b> .....	<b>8-7</b>
<b>Link Signal Builder Blocks to Requirements and Simulink Model Objects</b> .....	<b>8-8</b>
Link Signal Builder Blocks to Requirements Documents .....	<b>8-8</b>
Link Signal Builder Blocks to Model Objects .....	<b>8-9</b>
<b>Requirements Links for Library Blocks and Reference Blocks</b> .....	<b>8-11</b>
Introduction to Library Blocks and Reference Blocks .....	<b>8-11</b>
Library Blocks and Requirements .....	<b>8-11</b>
Copy Library Blocks with Requirements .....	<b>8-11</b>
Manage Requirements on Reference Blocks .....	<b>8-11</b>
Manage Requirements Inside Reference Blocks .....	<b>8-12</b>
Links from Requirements to Library Blocks .....	<b>8-14</b>
<b>Navigate to Requirements from Model</b> .....	<b>8-15</b>
Navigate from Model Object .....	<b>8-15</b>
Navigate from System Requirements Block .....	<b>8-15</b>

**9**

<b>Requirements Traceability for MATLAB Code Lines</b> .....	<b>9-2</b>
Link MATLAB Code Lines to Requirements in a Requirement Set .....	<b>9-2</b>
Link MATLAB Code Lines to Requirements Information in External Documents .....	<b>9-2</b>
Enable or Disable Traceability Links Highlighting for MATLAB Code .....	<b>9-3</b>
Remove Traceability Links from MATLAB Code Lines .....	<b>9-3</b>
Traceability for MATLAB Code Lines .....	<b>9-4</b>

**URL and Custom Traceability**

**10**

<b>Requirement Links and Link Types</b> .....	<b>10-2</b>
Requirements Traceability Links .....	<b>10-2</b>
Supported Model Objects for Requirements Linking .....	<b>10-2</b>
Links and Link Types .....	<b>10-2</b>
Link Type Properties .....	<b>10-3</b>
Outgoing Links Editor .....	<b>10-6</b>
<b>Custom Link Types</b> .....	<b>10-8</b>
Create a Custom Requirements Link Type .....	<b>10-8</b>
Implement Custom Link Types .....	<b>10-13</b>
Why Create a Custom Link Type? .....	<b>10-14</b>
Custom Link Type Functions .....	<b>10-14</b>
Custom Link Type Registration .....	<b>10-14</b>
Custom Link Type Synchronization .....	<b>10-15</b>

**Review and Maintain Requirements Links**

**11**

<b>Highlight Model Objects with Requirements</b> .....	<b>11-2</b>
Highlight Model Objects with Requirements Using Model Editor .....	<b>11-2</b>
Highlight Model Objects with Requirements Using Model Explorer .....	<b>11-3</b>
<b>Navigate to Simulink Objects from External Documents</b> .....	<b>11-4</b>
Provide Unique Object Identifiers .....	<b>11-4</b>
Use the rmiobjnavigate Function .....	<b>11-4</b>
Determine the Navigation Command .....	<b>11-4</b>
Use the ActiveX Navigation Control .....	<b>11-4</b>
Typical Code Sequence for Establishing Navigation Controls .....	<b>11-5</b>
<b>View Requirements Details for a Selected Block</b> .....	<b>11-6</b>
Requirements Details Workflow .....	<b>11-6</b>
Requirements Details Limitations .....	<b>11-6</b>



<b>Generate Code for Models with Requirements Links</b> .....	<b>11-7</b>
How Requirements Information Is Included in Generated Code .....	<b>11-8</b>
<b>Create and Customize Requirements Traceability Reports</b> .....	<b>11-9</b>
Create Requirements Traceability Report for Model .....	<b>11-9</b>
Customize Requirements Traceability Report for Model .....	<b>11-10</b>
<b>Create Requirements Traceability Report for A Project</b> .....	<b>11-24</b>
<b>Validate Requirements Links</b> .....	<b>11-25</b>
Validate Requirements Links in a Model .....	<b>11-25</b>
Validate Requirements Links in a Requirements Document .....	<b>11-29</b>
Validation of Requirements Links .....	<b>11-31</b>
<b>Delete Requirements Links from Simulink Objects</b> .....	<b>11-33</b>
Delete a Single Link from a Simulink Object .....	<b>11-33</b>
Delete All Links from a Simulink Object .....	<b>11-33</b>
Delete All Links from Multiple Simulink Objects .....	<b>11-33</b>
<b>Document Path Storage</b> .....	<b>11-34</b>
Relative (Partial) Path Example .....	<b>11-34</b>
Relative (No) Path Example .....	<b>11-34</b>
Absolute Path Example .....	<b>11-34</b>

## Requirements Management Interface

**12**

## Verification and Validation

**13**

<b>Test Model Against Requirements and Report Results</b> .....	<b>13-2</b>
Requirements - Test Traceability Overview .....	<b>13-2</b>
Display the Requirements .....	<b>13-2</b>
Link Requirements to Tests .....	<b>13-3</b>
Run the Test .....	<b>13-4</b>
Report the Results .....	<b>13-5</b>
<b>Analyze a Model for Standards Compliance and Design Errors</b> .....	<b>13-7</b>
Standards and Analysis Overview .....	<b>13-7</b>
Check Model for Style Guideline Violations and Design Errors .....	<b>13-7</b>
<b>Perform Functional Testing and Analyze Test Coverage</b> .....	<b>13-9</b>
Incrementally Increase Test Coverage Using Test Case Generation .....	<b>13-9</b>
<b>Analyze Code and Test Software-in-the-Loop</b> .....	<b>13-12</b>
Code Analysis and Testing Software-in-the-Loop Overview .....	<b>13-12</b>
Analyze Code for Defects, Metrics, and MISRA C:2012 .....	<b>13-12</b>



# 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
- “Define Requirements Hierarchy” on page 1-12
- “Create Requirement Set File by Using the Simulink® Requirements™ API” on page 1-14
- “Update Imported Requirements” on page 1-17
- “Import and Update Requirements from a Microsoft Word Document” on page 1-19
- “Export Requirement Sets and Link Sets to Previous Versions of Simulink Requirements” on page 1-21
- “Use Command-line API to Document Simulink Model in Requirements Editor” on page 1-22
- “Round Trip Workflows with ReqIF Files” on page 1-40
- “Best Practices and Guidelines for ReqIF Round Trip Workflows” on page 1-43
- “Create and Edit Attribute Mappings” on page 1-45

## 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

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:

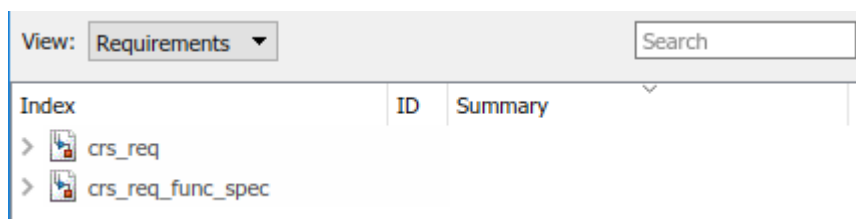
```
slreqCCProjectStart
```



- 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`. You can choose whether to share requirements with other models.
- 6 Add a requirement to your requirement set by selecting the requirement set and clicking **Add Requirement** .
- 7 In the **Properties** pane, enter the details for the requirement. You can copy and paste or drag requirements from another source to the **Properties** pane. 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. Right-click R1 and select **Add Requirement After**. 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 right-clicking R2 and selecting **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 the   or by dragging requirements.

## 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

### View or Hide Columns in the Requirements Editor

Right-click the header row and click **Select Columns** to change the view configuration of the Requirements Editor. Add, remove, and reorder attribute columns through the Column Selector dialog box. 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 using the **Search** field at the top of the Requirements Browser. 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.

---

**Tip** Clear the filtered contents by clicking **X** in the **Search** text box.

---

## Requirement Types

When you create or import requirements in Simulink Requirements, you can specify the requirement type by using the **Type** drop-down list in the **Properties** sidebar of the Requirements Editor or the Requirements Perspective View.

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 and Verification status metrics of the requirement set that they are in.
- **Container:** Group requirements. Container requirements do not contribute to the Implementation and Verification status metrics of the requirement set that they are in. However, all the Functional requirements under a Container requirement contribute to the status metrics.
- **Informational:** Provide supplemental information. Informational requirements and all requirements under them do not contribute to the Implementation and Verification status metrics of the requirement set that they are in.

You can also define custom requirement types. For more information, see “Define Custom Requirement and Link Types” on page 2-14.



## Import Requirements from Third-Party Applications

You can work with third-party requirements management applications by importing requirements. You can import requirements as new requirement sets, or reference requirements in the third-party application. Supported applications include:

- Microsoft Word
- Microsoft Excel®
- IBM® Rational® DOORS®
- Applications that use the Requirements Interchange Format (ReqIF). See “Import Requirements from ReqIF Files” on page 1-8.

### Path Settings

Add requirements documents to the MATLAB path. You can:

- Store the relative path for the currently running instance of MATLAB,
- Add the parent folder of the requirements document to the MATLAB path, or
- Update the Simulink Requirements path preference to always use the relative path. For more information on requirements document path preference, see “Document Path Storage” on page 11-34.

### Import Requirements from Microsoft Office Documents

Importing requirements from Microsoft Office documents is supported on Windows platforms.

In the Requirements Editor:

- 1** Select **File > Import**.
- 2** Select the **Document type**.
- 3** Import the most recently opened document, or browse for another document.

#### 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 content such as graphics and tables.

By default, imported requirements content matches the Microsoft Word document outline of section headings. You can also import requirements selectively by using the following qualifiers from the **Requirement Identification** menu:

- Predefined bookmarks in Microsoft Word to identify items and to serve as custom IDs. It is recommended to use bookmarks as requirement Custom IDs as they are persistently stored in the document and cannot be duplicated.
- Regular expression search patterns to identify items by occurrence. See “Regular Expressions” (MATLAB).
- You can choose to ignore outline numbers in the section headers of your Microsoft Word document. If you import requirements as references, it is recommended to ignore outline numbers to prevent issues with the Update process.

**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.

Use the qualifiers from the **Requirement Identification** menu to select a subset of your spreadsheet to import requirements from.

- 1 Choose individual rows and columns by mapping columns to requirement attributes. Select **Specify rows and columns** and click **Configure columns**. If there are no predefined headers in your spreadsheet, Simulink Requirements prompts you to specify the row that contains headers for attribute names.
- 2 In the Configure columns dialog box, select the range of rows and columns to import. Select how each column in your spreadsheet can be mapped to Properties and Custom Attributes by choosing an option from that drop-down list. When you map columns to Properties and Custom Attributes, consider:

- You can select only one column each for the **Custom ID** and **Summary**. 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 select one or more continuous columns for the **Description** and the **Rationale**. The contents of these columns are concatenated into one field after the import is completed.
- You must select at least one column for the **Summary** or the **Description**.

To omit columns from the import, select the **Ignore** option.

- 3 You can use regular expression search patterns to selectively identify and import items by occurrence. See “Regular Expressions” (MATLAB).

## 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, or as new requirement sets.

Third-party applications that use ReqIF with particular attribute mapping include:

- Polarion™
- PREEvision
- IBM Rational DOORS
- IBM Rational DOORS Next Generation

Other third-party applications can use generic ReqIF mapping.

### Third-party Specific Server Configuration

*Polarion:* When working with Polarion, modify the Polarion server configuration to use the actual server name in `repo` and `base.url` property values. Do not use `localhost`.

- 1 Open the `polarion.properties` file found in the `<polarion_installation>/polarion/configuration/` folder.
- 2 Modify these lines:
  - `repo=http://localhost:80/repo/`
  - `base.url=http://localhost:80/`

by replacing `localhost` with the externally known name for your server.

### ReqIF Import Workflow

To import requirements from a ReqIF file:

- 1 In the Requirements Editor, select **File > Import**.
- 2 For **Document type**, select ReqIF file (`*.reqif` or `*.reqifz`).
- 3 For **Document location**, select the ReqIF file location.
- 4 Simulink Requirements detects the source tool of the ReqIF file. You can also manually select a **Source tool**, or select **Generic** if the source is unknown.
- 5 Select the location for the destination requirement set.
- 6 Select whether to allow updates to the imported requirements:
  - 1 If your requirements are maintained in an external tool, and you want to be able to update the imported requirement set with updated versions of the ReqIF file, select **Allow updates from external source**.
  - 2 To establish the Simulink Requirements set as the primary requirements artifact, do not select **Allow updates from external source**.
- 7 Complete the import process by clicking **Import**.

### Importing Multiple ReqIF Specifications

You can import multiple source specifications from ReqIF files. When you import ReqIF files that contain multiple source specifications, you can choose to:

- 1 Select a single ReqIF source specification to import into a requirement set.
- 2 Combine ReqIF source specifications into one requirement set. Each specification is imported into its own Import node. You can update each Import node independently.
- 3 Import each ReqIF source specification into a separate requirement set. Instead of selecting a destination requirement set, you select a destination folder. The import operation creates multiple requirement set files in the destination folder.

If a ReqIF file contains a single specification, options 2 and 3 above are not available.

For large ReqIF files, import each source specification into a separate requirement set. This can help reduce file conflicts and simplify change tracking and differencing of individual requirement sets.

In ReqIF, a link is represented as a `SpecRelation` between two `SpecObjects`. Select **Import links** to preserve links in the ReqIF file. **Import links** is enabled if the ReqIF file has `SpecRelations` between `SpecObjects`. After import, Simulink Requirements link set files contain links between requirements or external URLs.


If the ReqIF file does not define SpecRelations, the **Import links** option is disabled. Only valid links are imported. The link import operation depends on how you import the source specifications:

- 1 Importing a single specification into a requirement set imports only the SpecRelations within the specification's SpecObjects. As a result, some links can be omitted.
- 2 Combining ReqIF source specifications into one requirement set imports resolved links into one link set.
- 3 Importing each ReqIF source specification into a separate requirement set imports resolved links into separate linksets.

## Customize Attribute Display

ReqIF represents a requirement as a SpecObject with user-defined attributes. You can customize how the Requirements Editor and Requirements Browser displays imported requirements data and properties.

To customize the display of imported requirements data, map the attributes of the SpecObject to either built-in or custom attributes of a requirement. You can save this mapping as an XML file for future use.


To modify the attribute mapping after you import, select the top-level Import node of the requirement set (denoted by ) and expand the **Attribute Mapping** pane. You can also load a previously saved attribute mapping by clicking **Load mapping**.

## Import Modes

Simulink Requirements provides two import modes for importing requirements content. Before you complete the Import process, you must specify if you want to allow updates to your imported requirements from the external requirements document by selecting or clearing **Allow updates from external source**.


### Import Requirements



If you want to permanently migrate your requirements from the external requirements management application, do not allow updates to imported requirements from the external source document.

Requirements are then imported as `slreq.Requirement` objects and are represented by  in the Requirements Spreadsheet. Importing requirements as `slreq.Requirement` objects allows you to freely edit, delete, and rearrange requirements.

### Import Referenced Requirements

If you choose to allow updates, requirements are imported as referenced requirements (`slreq.Reference` objects) that you can unlock and edit within Simulink Requirements.

Referenced requirements retain some dependencies to the source document and are locked for editing by default. Locked requirements are represented by  in the Requirements Spreadsheet. Edit an individual requirement by navigating to it and clicking **Unlock** in the **Properties** pane.

Unlocked requirements are represented by  in the Requirements Spreadsheet. Unlock all referenced requirements by navigating to the top import node (denoted by ) and clicking **Unlock all** in the **Requirement Interchange** pane. You cannot relock requirements after you unlock them,

except by updating them. You cannot delete or change the hierarchy of referenced requirements from within Simulink Requirements.

If your requirements are imported from an external source, other users are likely to change them in the external source document. To make your referenced requirements reflect the latest version of the requirements as in the external source document, obtain an updated file from the external source. Updating requirements from the external document overwrites all the local changes that you made to imported requirements content.

The Update operation preserves local custom attributes you create within Simulink Requirements. If you have attributes with the same name in the requirement set and in the external source document, the Update operation overwrites the local values 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**.

## See Also

slreq.import


## More About

- “Update Imported Requirements” on page 1-17
- “Round Trip Workflows with ReqIF Files” on page 1-40

## 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 the  icons in the Requirements Editor or 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 Requirement Implementation Status Metrics Data” on page 3-2 and “Summarize Requirements Verification Status” on page 3-3.

## 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 Columns**.

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.

## 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. You can distribute the `.slreqx` file across your organization.

You can integrate the requirement set file into a change management system that users can use as a parent document to create their own requirement sets.

### 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);
```

## 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, select **File > 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 your requirements in the requirement set. Select the top node and click **Update** in the **Requirements Interchange** pane. 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 in the **Comments** pane of the top Import node in the requirement set.
- 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.

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.

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, ensure 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 .

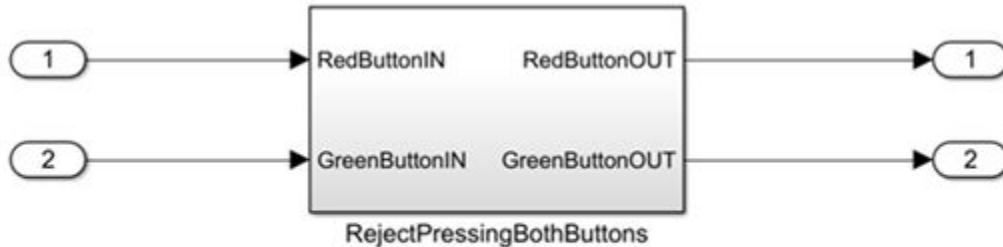
### See Also

### More About

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

## Import and Update Requirements from a Microsoft Word Document

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



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

Open the model. At the MATLAB command prompt, enter:

```
mdl = 'rejectDoublePress';
open_system(fullfile(matlabroot, 'examples', 'slrequirements', mdl))
```

The supporting requirements document is located at `matlab/examples/slrequirements`.

This example uses a Microsoft Word document, `Reject_Double_Button_Press_Model_Requirements.docx`. This document contains a set of functional requirements for the `Reject_Double_Button_Press` 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.

Save the requirements document to a local folder before importing requirements from it.

### Import Requirements

- 1 Open the Requirements Editor. In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Requirements Editor**.
- 2 In the Requirements Editor, select **File > Import**.
- 3 For **Document type**, select **Microsoft Word document**, and select **Reject\_Double\_Button\_Press\_Model\_Requirements.docx** as the **Document location**.
- 4 Import requirements as plain text, read-only references to a destination requirement set. In the **Requirement Identification** option group, select the options to use bookmarks and to ignore outline numbers. For more information on import options, see “Import Options for Microsoft Word Documents” on page 1-7.

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

## Update Requirements

Requirements that you import as read-only references retain their references to the source requirements document. To change your imported requirements, you must make the changes in the source document first and update your requirement set from within Simulink Requirements. For more information on updating requirements, see “Update Imported Requirements” on page 1-17

- 1 In the document, `Reject_Double_Button_Press_Model_Requirements.docx`, add a new requirement:
  - 2.1.5 The Red and Green Button outputs shall be 0 if no buttons are pressed.
- 2 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. Update the requirements by clicking **Update** in the **Properties** side bar on the right.
- 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.

## See Also

`slreq.ReqSet`

## More About

- “Import Requirements from Third-Party Applications” on page 1-7
- “Define Requirements Hierarchy” on page 1-12

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

Export requirement sets and link sets to work with them in previous versions of Simulink Requirements. Starting in R2018b, you can export requirement sets to previous versions of Simulink Requirements (R2017b and beyond).

### Export Link Sets

Export link sets to previous versions of Simulink Requirements by exporting the Simulink models that the link sets are associated with to previous versions of Simulink. See “Export a Model to a Previous Simulink Version” (Simulink).

### Export Requirement Sets

You can export requirement sets to previous versions from within the Requirements Editor. Navigate to the Requirements View and select the requirement set for export. Before you attempt the Export operation, ensure that the requirement set you want to export is not open in a model. In the Requirements Editor window, select **File > Export to Previous**. From the dialog box, select the file name and version you want to export to.

If you export a requirement set with outgoing links to a previous version, Simulink Requirements creates a requirement set and link set files corresponding to that previous version.

## 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(REQSETNAME)` for creating a new Requirement Set
- `ReqSet.add(NAME,VALUE)` for adding entries to a Requirement Set
- `Requirement.add(NAME,VALUE)` for adding child requirements
- `Requirement.Description(TEXT)` for filling-in the Description field
- `slreq.createLink(SRC,DEST)` for creating link from SRC to DEST
- `slreq.find('type',TYPENAME)` for locating Simulink Requirements objects
- `link.setDestination(DEST)` for re-connecting the destination end of an existing link
- `link.setSource(SOURCE)` for moving an existing link to the new source object
- `link.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 (ARGS) 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

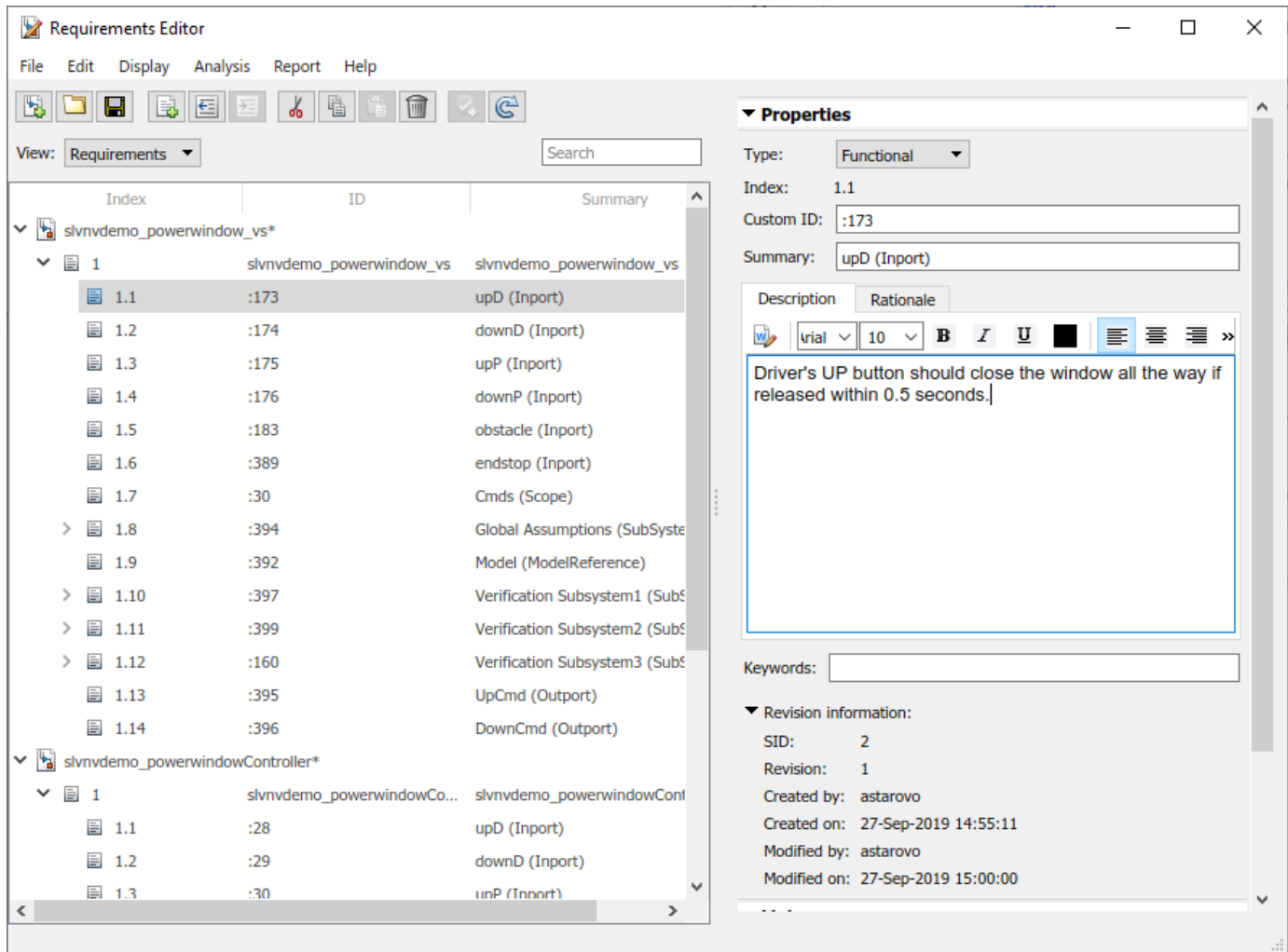
You want to use the Simulink Requirements product to create a detailed description of your Simulink design, and you want to organized your Requirements collection in a hierarchy that matches your Simulink models. You also want an easy way to navigate between the items of this Requirements collection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

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

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 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.



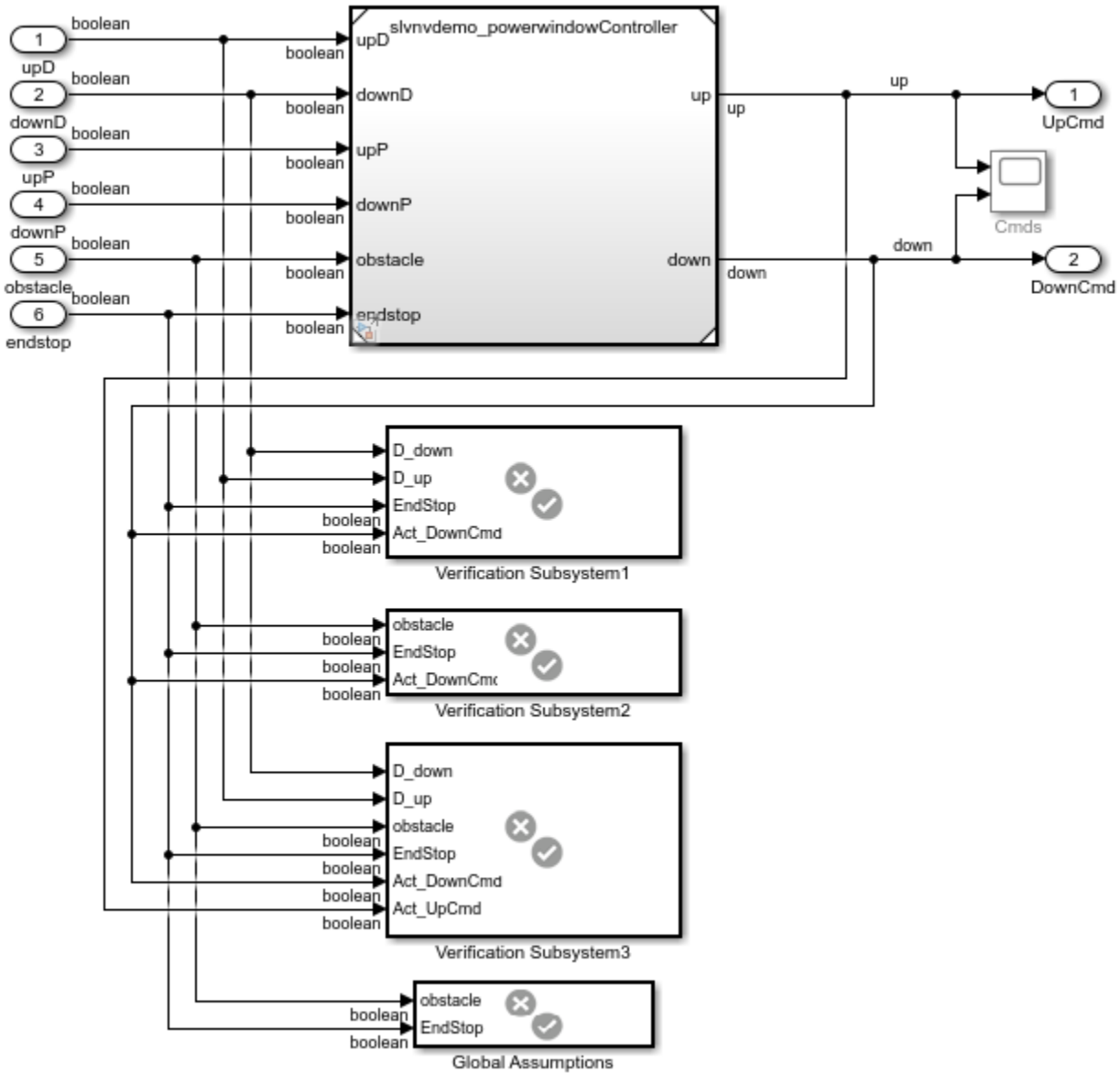


Below is the script that builds one Requirement Set with two model surrogates. The bottom three commands provide an example of how to programmatically fill-in the Description field for a proxy item, but most probably you will do this interactively in the Editor.

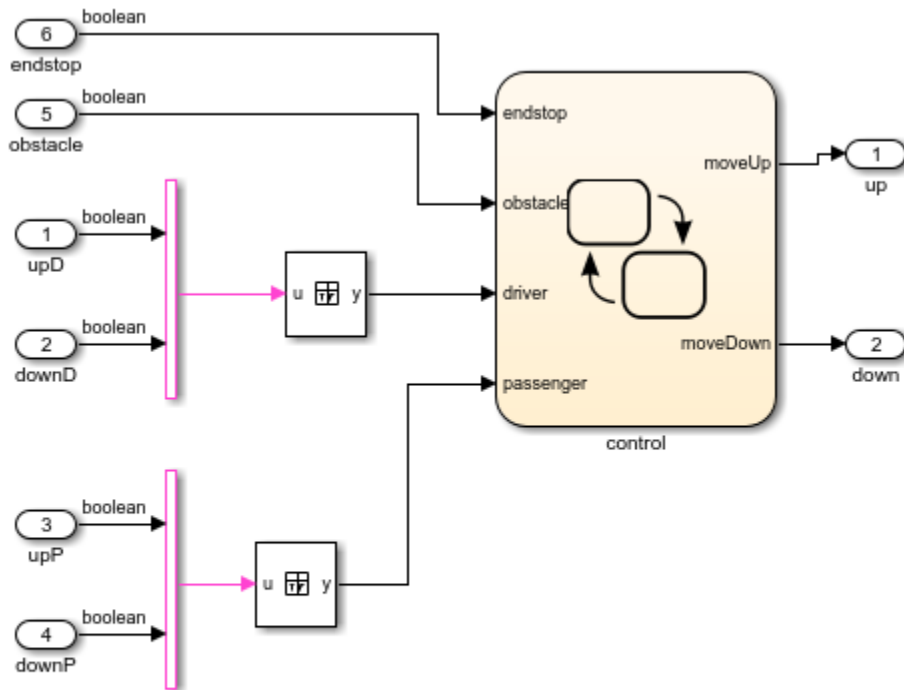
```
models = {'slvndemo_powerwindow_vs', 'slvndemo_powerwindowController'};
workDir = tempname;
disp(['Using ' workDir ' to store generated files.']);
mkdir(workDir);
addpath(workDir);
for modelIdx = 1:length(models)
    modelName = models{modelIdx};
    reqSetFile = fullfile(workDir, [modelName '.slreqx']);
    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 iteself
            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 Descr
roItem.Description = 'Driver's UP button should close the window all the way if released within
Using C:\Users\astarovo\AppData\Local\Temp\tpa1d56261_e0a7_4d8c_94c2_52b7abc0e7ef to store gener
```

### Power Window Controller Temporal Property Specification



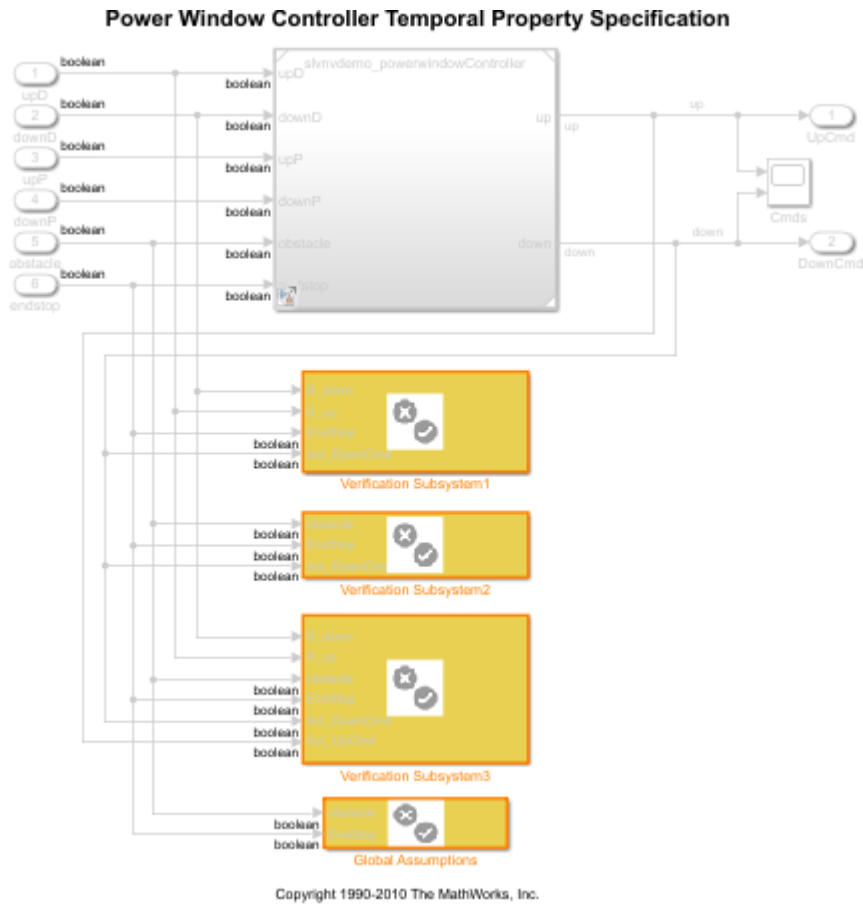
Copyright 1990-2010 The MathWorks, Inc.



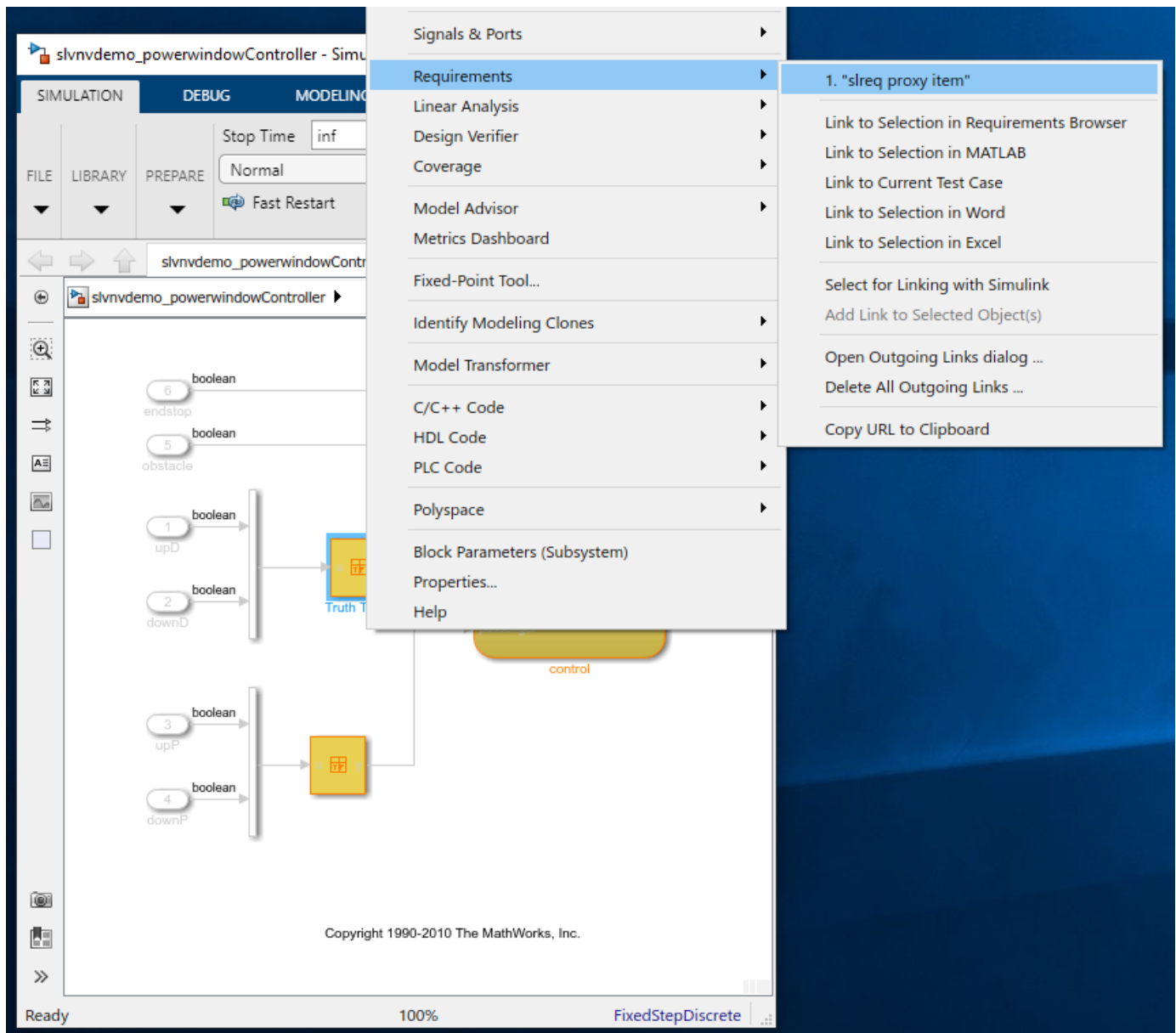
Copyright 1990-2010 The MathWorks, Inc.

### 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(SRC, DEST)` 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 Links panel at the bottom-right.

Requirements Editor

File Edit Display Analysis Report Help

View: Requirements Search

Index	ID	Model (Model)
1.9	:392	Model (Model)
1.10	:397	Verification Si
1.11	:399	Verification Si
1.12	:160	Verification Si
1.13	:395	UpCmd (Outp
1.14	:396	DownCmd (O
slnvdemo_powerwindowController*		
1	slnvdemo_powerwindowController	slnvdemo_pi
1.1	:28	upD (Inport)
1.2	:29	downD (Inpo
1.3	:30	upP (Inport)
1.4	:31	downP (Inpor
1.5	:32	obstacle (Inp
1.6	:33	endstop (Inp
1.7	:15	Mux1 (Mux)
1.8	:16	Mux4 (Mux)
1.9	:39	Truth Table (
1.10	:40	Truth Table1
1.11	:36	control (SubS
1.12	:34	up (Output)
1.13	:35	down (Outpoi

Properties

Type: Functional

Index: 1.9

Custom ID: :39

Summary: Truth Table (SubSystem)

Description Rationale

The passenger and driver input signals are generated by mapping the up and down signals according to the following 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

Keywords:

Revision information:

Links

Implemented by: Truth Table

```

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 slnvdemo_powerwindow_vs:394 ..
linking slnvdemo_powerwindow_vs:394:224 ..
linking slnvdemo_powerwindow_vs:394:272 ..
linking slnvdemo_powerwindow_vs:394:271 ..
linking slnvdemo_powerwindow_vs:394:360 ..
linking slnvdemo_powerwindow_vs:397 ..
linking slnvdemo_powerwindow_vs:397:107 ..
linking slnvdemo_powerwindow_vs:397:300 ..
linking slnvdemo_powerwindow_vs:397:108 ..
linking slnvdemo_powerwindow_vs:397:285 ..
linking slnvdemo_powerwindow_vs:397:307 ..
linking slnvdemo_powerwindow_vs:399 ..
linking slnvdemo_powerwindow_vs:399:650 ..
linking slnvdemo_powerwindow_vs:399:214 ..
linking slnvdemo_powerwindow_vs:399:218 ..
linking slnvdemo_powerwindow_vs:399:273 ..
linking slnvdemo_powerwindow_vs:160 ..
linking slnvdemo_powerwindow_vs:160:643 ..
linking slnvdemo_powerwindow_vs:160:646 ..
linking slnvdemo_powerwindow_vs:160:590 ..
linking slnvdemo_powerwindow_vs:160:591 ..
linking slnvdemo_powerwindow_vs:160:648 ..
linking slnvdemo_powerwindow_vs:160:592 ..
Created 23 links for slnvdemo_powerwindow_vs
  linking slnvdemo_powerwindowController:39 ..
  linking slnvdemo_powerwindowController:40 ..
  linking slnvdemo_powerwindowController:36 ..
Created 3 links for slnvdemo_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.

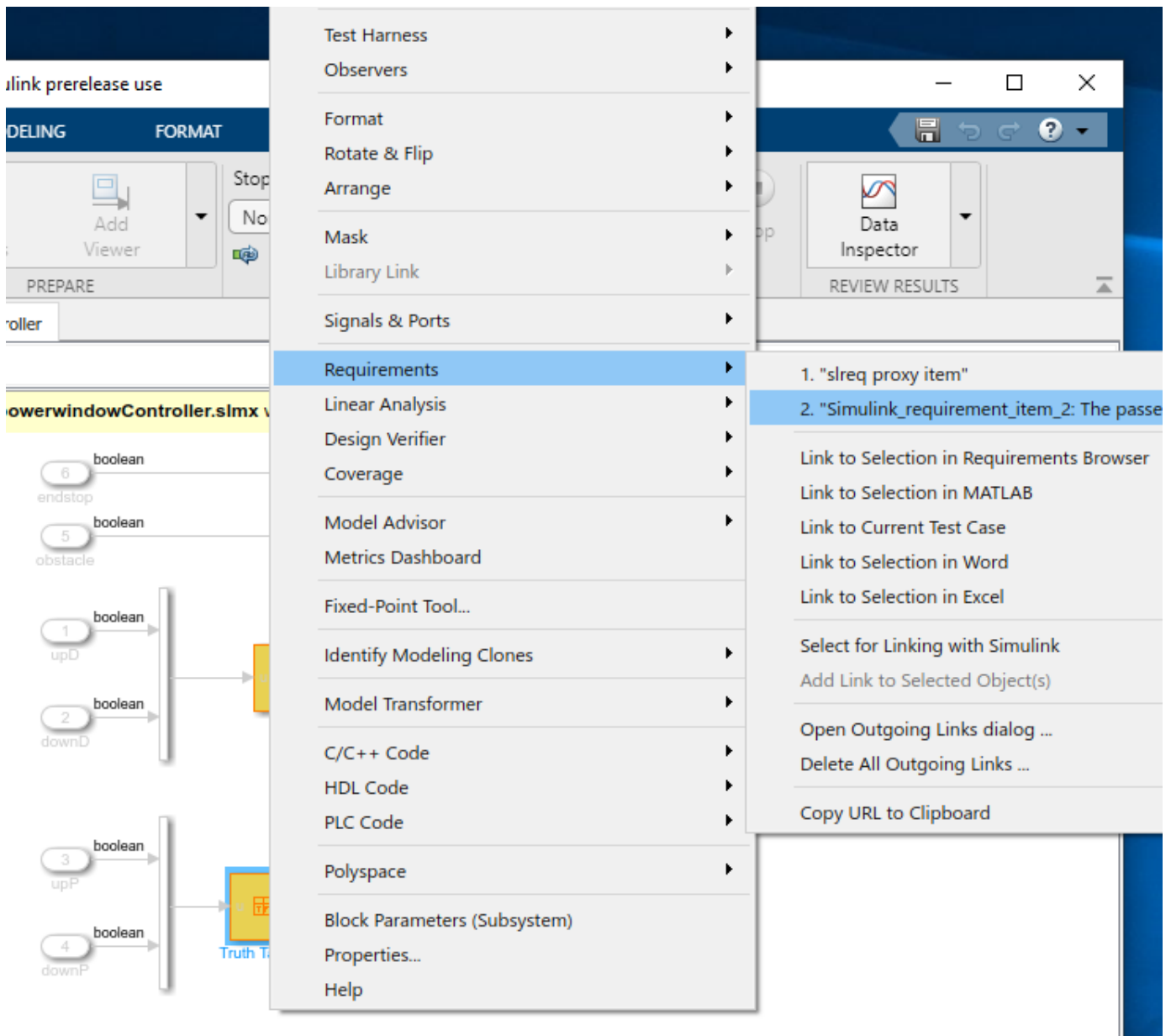


The screenshot shows the Requirements Editor window with the following components:

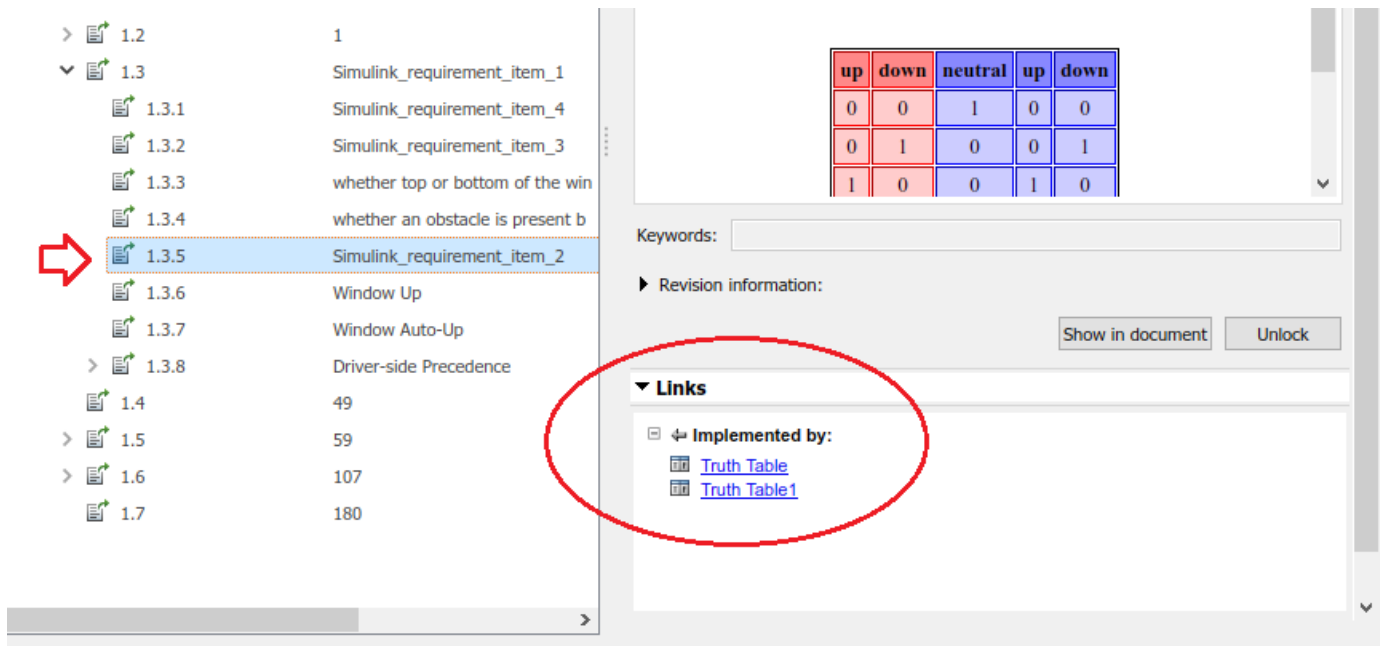
- Tree View (Left):** A hierarchical tree structure under 'PowerWindowSpecification'. The selected item is '1.3.5 Simulink\_requirement\_item\_2'.
- Properties Pane (Right):**
  - Type:** Functional
  - Index:** 1.3.5
  - Custom ID:** Simulink\_requirement\_item\_2
  - Summary:** The passenger and driver input signals are generated by mapping the up and...
  - Description:** The passenger and driver input signals are generated by mapping the up and down signals according to the following table
  - 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
  - Keywords:** (Empty text field)
  - Revision information:** (Collapsible section)
  - Links:** No links
  - Comments:** (Collapsible section)

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.



```

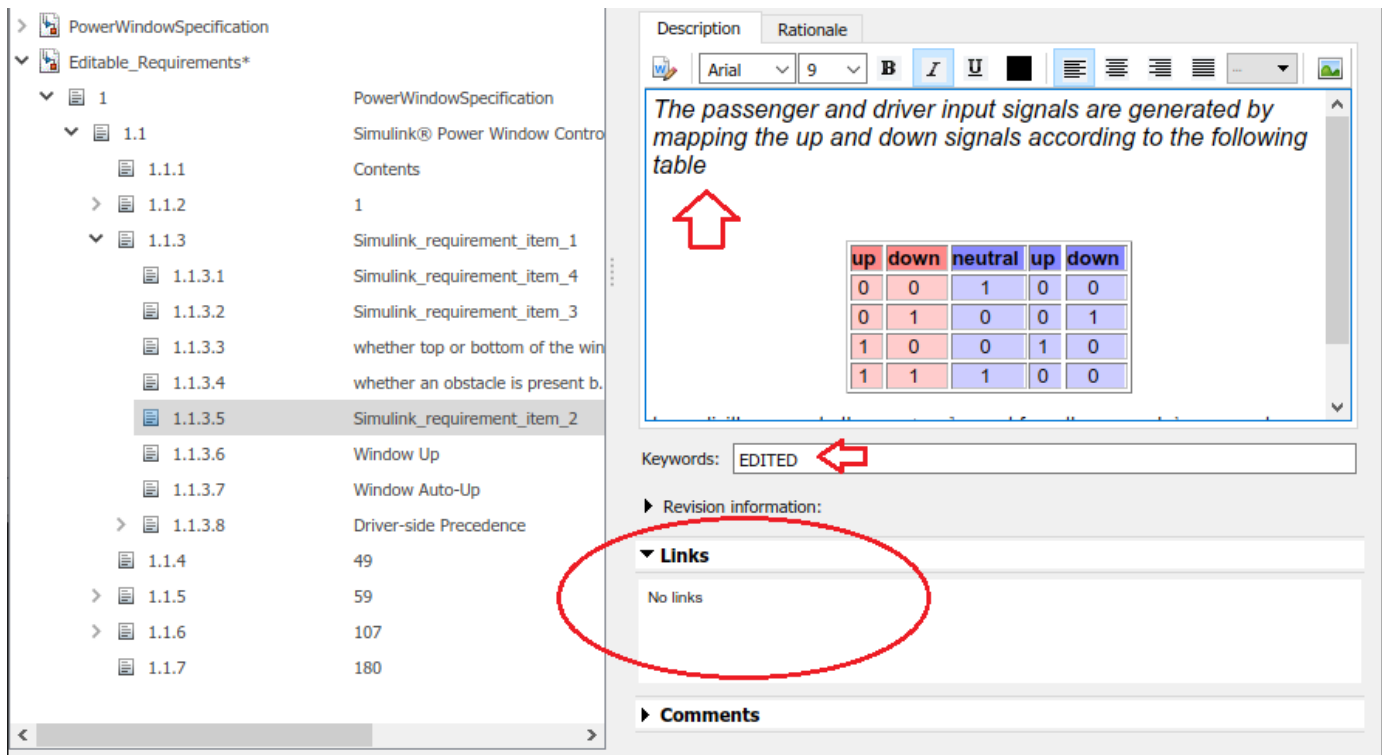
examplesFolder = fullfile(matlabroot, 'toolbox', 'slrequirements', 'slrequirementsdemos');
docsFolder = fullfile(examplesFolder, 'powerwin_reqs');
addpath(docsFolder); % just in case
externalDocName = 'PowerWindowSpecification';
externalDoc = fullfile(docsFolder, [externalDocName '.docx']);
outputFile = fullfile(workDir, 'ReadOnlyImport.slreqx');
[~,~,roReqSet] = slreq.import(externalDoc, 'ReqSet', outputFile); % without extra arguments the
roReqSet.save();
roItem = roReqSet.find('CustomId', 'Simulink_requirement_item_2');
designModel = 'slvndemo_powerwindowController';
link1 = slreq.createLink([designModel '/Truth Table'], roItem); % link to read-only item in imp
link2 = slreq.createLink([designModel '/Truth Table1'], roItem); % link 2nd block to the same re
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 lega

```

### Updating Links for Navigation to Alternative Imported Requirement Set

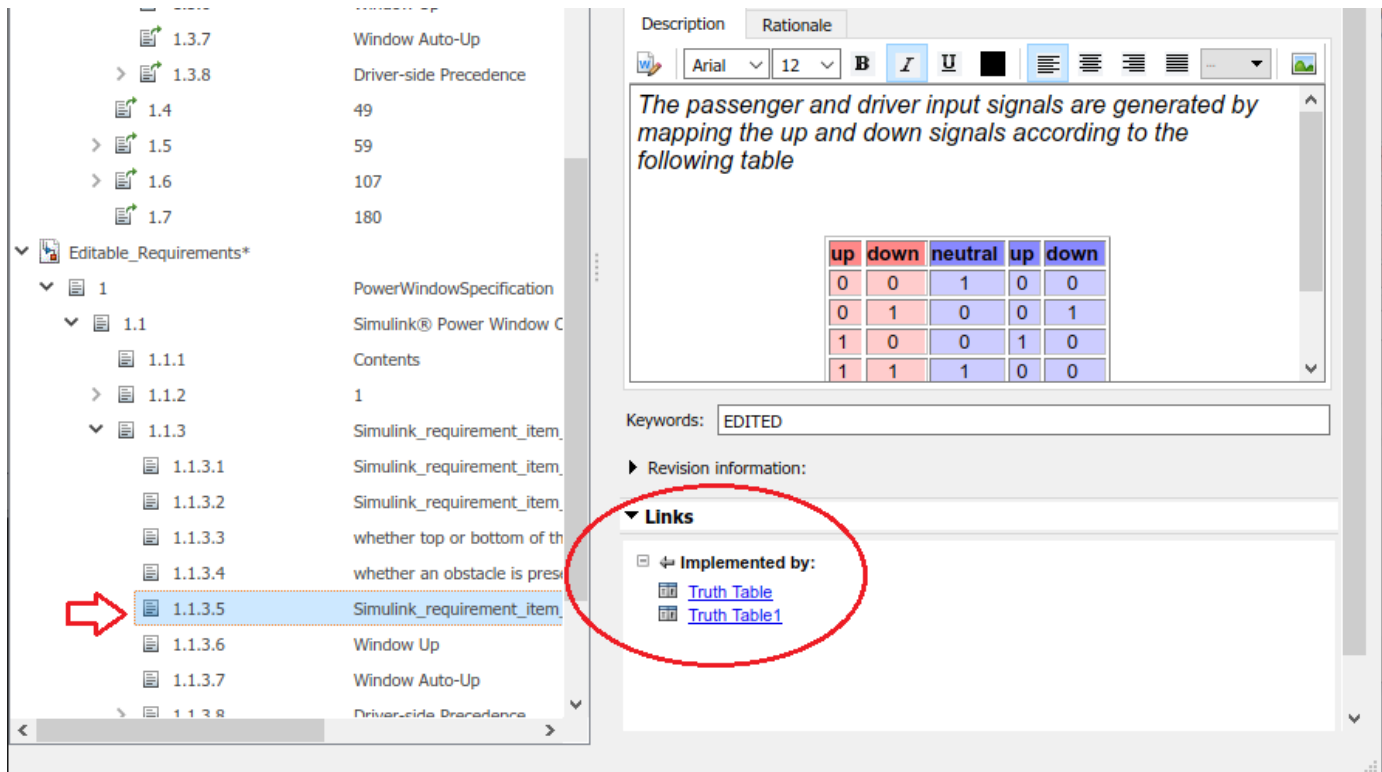
Note that we imported our Word document "as read-only references", which is the default for `slreq.import(DOCFILEPATH)` command when run without optional arguments. This import mode allows to later update the imported items when the newer version of the source document becomes available. Now, supposed 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:



However, there is problem: our previously created links connect from Simulink to the original read-only References, not to the more recentlly 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 `link.setDestination(DEST)` 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');
[~,~,mwReqSet] = slreq.import(externalDoc, 'ReqSet', outputFile, 'AsReference', false); % re-import
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
        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
        link.setDestination(mwItem);
        updateCount = updateCount + 1;
    end
end
disp(['Updated ' num2str(updateCount) ' links from ' designModel]);
slreq.show(link.destination()); % check updated destination of the last link we
rmi('view', [designModel '/Truth Table1'], 2); % navigate again (legacy API), editable item selected
```

Updated 2 links from slvndemo\_powerwindowController

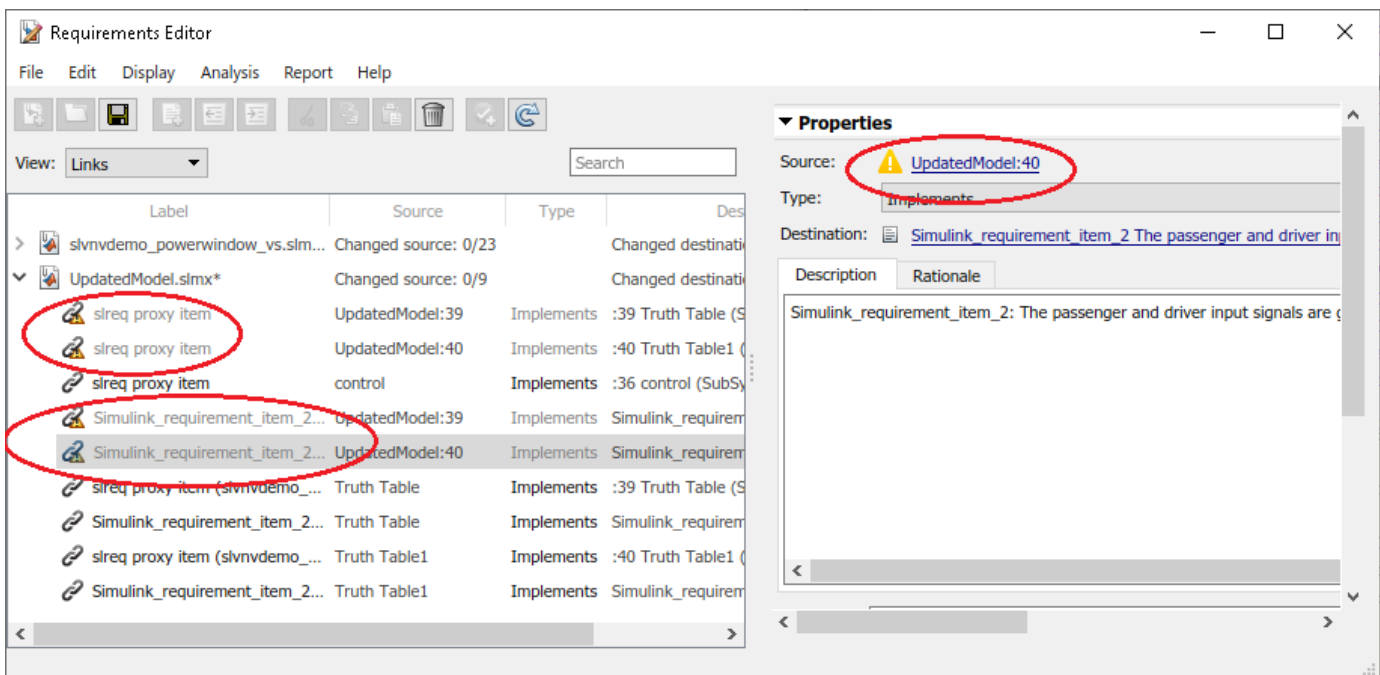
### USE CASE 3: Reuse Existing Outgoing Links After Replacing Source Objects

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 a blocks with same name

exists in the same level of 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 `link.setSource(SRC)` 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.

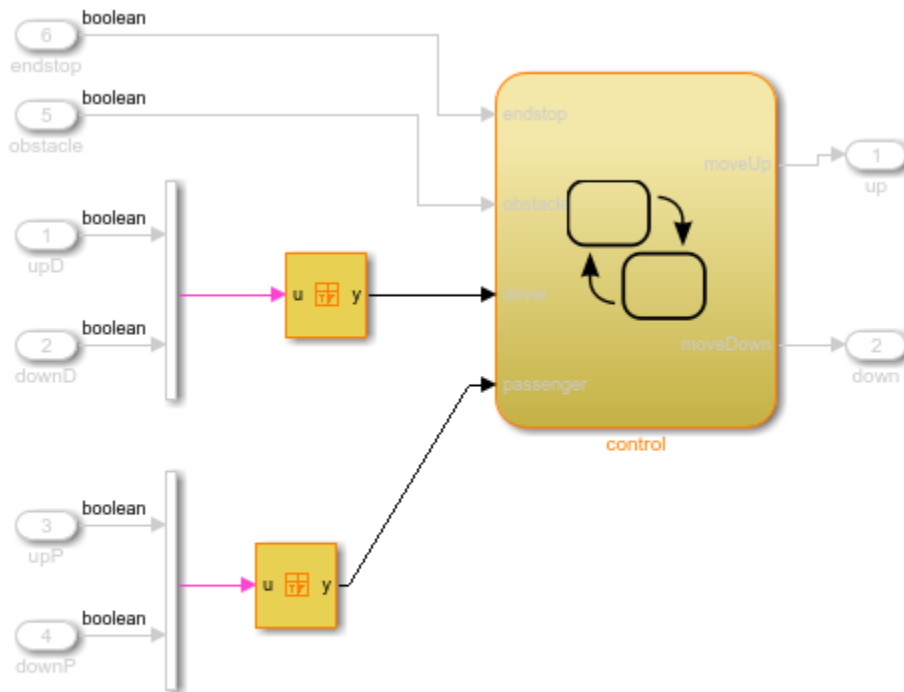
To demonstrate the use of `link.setSource(SRC)` API with our example model, we will simply replace two Truth Table blocks that we linked in the previous section with same exact new blocks. Once this is done, the links become `unresolved`, because new Truth Table copies have new SIDs.

Switch to Links View in Requirements Editor 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`.



```
originalModel = 'slvndemo_powerwindowController';
updatedModel = 'UpdatedModel';
save_system(originalModel, fullfile(workDir, [updatedModel '.slx'])); % this also creates .slmx
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
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
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
```

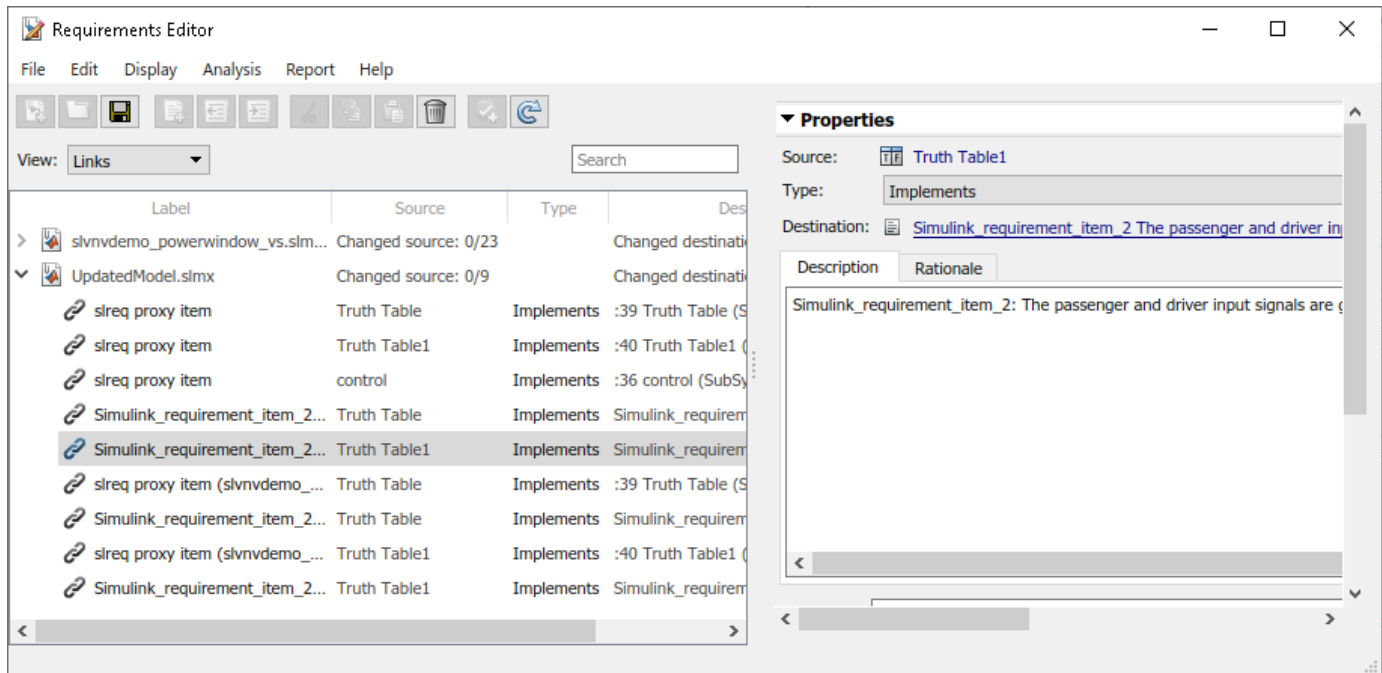


Copyright 1990-2010 The MathWorks, Inc.

### Update Source Ends to Repair Broken Links

Now we need to iterate all the links in the new model, identify the ones with unresolved source using `link.isResolvedSource()` API, then use `link.setSource(SRC)` 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.



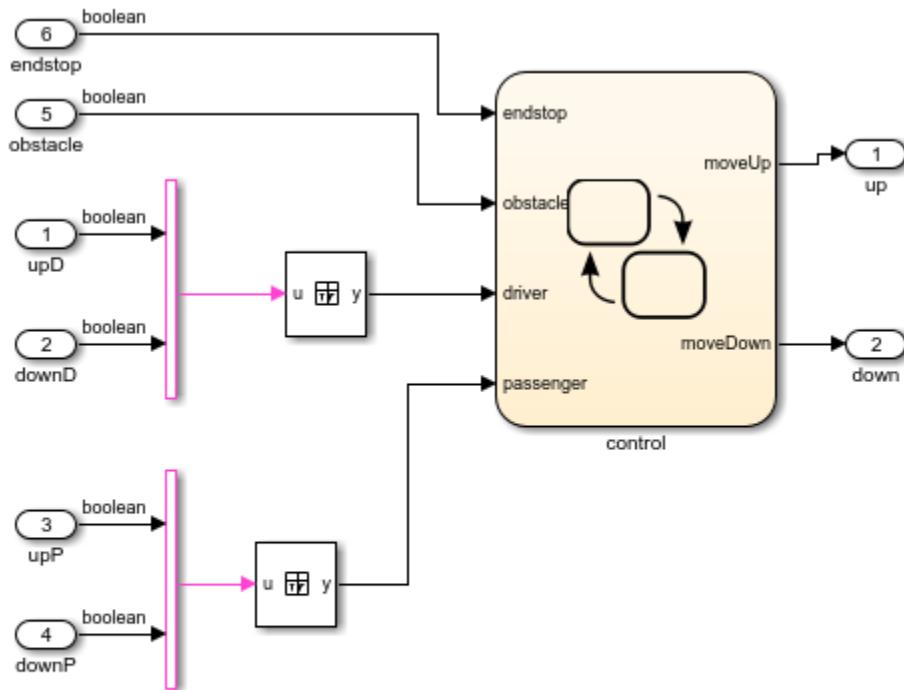
```

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





Copyright 1990-2010 The MathWorks, Inc.

## 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');
slreq.import.docToReqSetMap(externalDoc, 'clear'); % clear stored import location for our document
```

## Round Trip Workflows with ReqIF Files

Simulink Requirements supports round trip workflows with ReqIF files. ReqIF is an open standard XML format developed for lossless exchange of requirements and their associated metadata between requirements management applications. You can import, edit, and export requirements by using ReqIF files.

### 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, or as new requirement sets.

Third-party applications that use ReqIF with particular attribute mapping include:

- Polarion
- PREEvision
- IBM Rational DOORS
- IBM Rational DOORS Next Generation

Other third-party applications can use generic ReqIF mapping.

### Third-party Specific Server Configuration

*Polarion:* When working with Polarion, modify the Polarion server configuration to use the actual server name in `repo` and `base.url` property values. Do not use `localhost`.

- 1 Open the `polarion.properties` file found in the `<polarion_installation>/polarion/configuration/` folder.
- 2 Modify these lines:
  - `repo=http://localhost:80/repo/`
  - `base.url=http://localhost:80/`

by replacing `localhost` with the externally known name for your server.

### ReqIF Import Workflow

To import requirements from a ReqIF file:

- 1 In the Requirements Editor, select **File > Import**.
- 2 For **Document type**, select ReqIF file (`*.reqif` or `*.reqifz`).
- 3 For **Document location**, select the ReqIF file location.
- 4 Simulink Requirements detects the source tool of the ReqIF file. You can also manually select a **Source tool**, or select **Generic** if the source is unknown.
- 5 Select the location for the destination requirement set.
- 6 Select whether to allow updates to the imported requirements:
  - 1 If your requirements are maintained in an external tool, and you want to be able to update the imported requirement set with updated versions of the ReqIF file, select **Allow updates from external source**.
  - 2 To establish the Simulink Requirements set as the primary requirements artifact, do not select **Allow updates from external source**.

- 7 Complete the import process by clicking **Import**.

### Importing Multiple ReqIF Specifications

You can import multiple source specifications from ReqIF files. When you import ReqIF files that contain multiple source specifications, you can choose to:

- 1 Select a single ReqIF source specification to import into a requirement set.
- 2 Combine ReqIF source specifications into one requirement set. Each specification is imported into its own Import node. You can update each Import node independently.
- 3 Import each ReqIF source specification into a separate requirement set. Instead of selecting a destination requirement set, you select a destination folder. The import operation creates multiple requirement set files in the destination folder.

If a ReqIF file contains a single specification, options 2 and 3 above are not available.

For large ReqIF files, import each source specification into a separate requirement set. This can help reduce file conflicts and simplify change tracking and differencing of individual requirement sets.

In ReqIF, a link is represented as a SpecRelation between two SpecObjects.. Select **Import links** to preserve links in the ReqIF file. **Import links** is enabled if the ReqIF file has SpecRelations between SpecObjects. After import, Simulink Requirements link set files contain links between requirements or external URLs.


If the ReqIF file does not define SpecRelations, the **Import links** option is disabled. Only valid links are imported. The link import operation depends on how you import the source specifications:

- 1 Importing a single specification into a requirement set imports only the SpecRelations within the specification's SpecObjects. As a result, some links can be omitted.
- 2 Combining ReqIF source specifications into one requirement set imports resolved links into one link set.
- 3 Importing each ReqIF source specification into a separate requirement set imports resolved links are into separate linksets.

### Customize Attribute Display

ReqIF represents a requirement as a SpecObject with user-defined attributes. You can customize how the Requirements Editor and Requirements Browser displays imported requirements data and properties.

To customize the display of imported requirements data, map the attributes of the SpecObject to either built-in or custom attributes of a requirement. You can save this mapping as an XML file for future use.

To modify the attribute mapping after you import, select the top-level Import node of the requirement set (denoted by ) and expand the **Attribute Mapping** pane. You can also load a previously saved attribute mapping by clicking **Load mapping**.

### Edit Imported Content

Edit imported requirements content by using the editing capabilities of the Requirements Editor. You can unlock and edit a requirement's information such as its **Description** or **Rationale**. You can also

define custom attributes on the requirement set and set values for those custom attributes on selected requirements.

## Unlock and Edit Imported Requirements

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

To add, remove, and edit custom attributes associated with the requirement set, navigate to the top-level node of the requirement set and use the actions available in the **Custom Attribute Registries** pane. Select an individual requirement and unlock it set custom attribute values.

## Update Imported Requirements Content

If you selected **Allow updates from external source** during the Import operation, you can update your imported requirement sets with data from the source ReqIF file. Navigate to the top-level Import node of the requirement set and click **Update**. The Update operation overwrites all local modifications such as edits to unlocked requirements with values from the ReqIF source file. The Update operation preserves comments and local attributes.

## Export Requirements Content

You can export a requirement set or an individual requirement and its child requirements to ReqIF files from Simulink Requirements. Navigate to the node that you want to export and select **File > Export to ReqIF**.

In the **Export to ReqIF** dialog box, you can select the export mapping and the output ReqIF file name and path. If you are exporting requirement artifacts that you previously imported (round trip workflow), it is recommended to use the same import settings for the Export operation.

The Export operation inverts the attribute mapping used by the Import operation. Any local custom attributes that you added to or defined in the Custom Attribute Registry are also included in the export mapping so that they are visible in external requirements management applications.

## See Also

slreq.import

## More About

- “Import Requirements from Third-Party Applications” on page 1-7
- “Update Imported Requirements” on page 1-17
- “Best Practices and Guidelines for ReqIF Round Trip Workflows” on page 1-43

# 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 Workflows with ReqIF Files” on page 1-40
- “Update Imported Requirements” on page 1-17


## Create and Edit Attribute Mappings

The ReqIF format represents a requirement as a `SpecObject`, which has user-defined attributes. You can customize how requirements imported from ReqIF are displayed in Simulink Requirements by mapping `SpecObject` attributes to either built-in or custom attributes of requirement objects. You can also save this mapping for reuse.

Simulink Requirements provides attribute mappings for ReqIF files from:

- Polarion REQUIREMENTS™
- PREEvision
- IBM Rational DOORS
- IBM Rational DOORS Next Generation
- Jama Software

To work with ReqIF files created from different external requirements management tools, use the `Generic` attribute mapping from the **Source tool** drop-down list. If `Generic` does not create the desired attribute mapping, create a custom mapping:

- 1 Open the Requirements Editor and import the ReqIF file by selecting the `Blank` attribute mapping from the **Source tool** drop-down list. See “Import Requirements from ReqIF Files” on page 1-8.
- 2 Navigate to the top Import node  of the imported requirement set and expand the **Attribute Mapping** pane on the right.
- 3 Map each external attribute in the **External Attribute Name** column to a built-in or a custom attribute by using the drop-down list in the **Mapped To** column.
- 4 Click **Save Mapping**. Save the mapping in `matlabroot/toolbox/slrequirements/attribute_maps` as an XML file.
- 5 Restart MATLAB to include the newly created attribute mapping in the **Source tool** drop-down list.

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 which originally created the ReqIF file you are importing:

- 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 REQUIREMENTS 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 REQUIREMENTS 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 in the mapping file the name of a template `.reqif` file produced by the external requirements management tool. During ReqIF export, Simulink Requirements imports the template file and uses it to:

- Create an instance of the ReqIF data model based on the template with the expected data type, attribute, and `SpecObject` type definitions.
- Remap the requirements content to the expected data types and attributes.

Using the template ensures that the exported file can be readily imported into the external requirements management 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 corresponding 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.

## See Also

“Round Trip Workflows with ReqIF Files” on page 1-40 | “Best Practices and Guidelines for ReqIF Round Trip Workflows” on page 1-43



# Requirements Traceability and Consistency

---

- “Link Blocks and Requirements” on page 2-2
- “Track Requirement Links with a Traceability Matrix” on page 2-5
- “Requirement Links” on page 2-11
- “Define Custom Requirement and Link Types” on page 2-14
- “Requirements Consistency Checks” on page 2-16
- “Manage Navigation Backlinks in External Requirements Documents” on page 2-20
- “Use Command-line API to Update or Repair Requirements Links” on page 2-21

## 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.

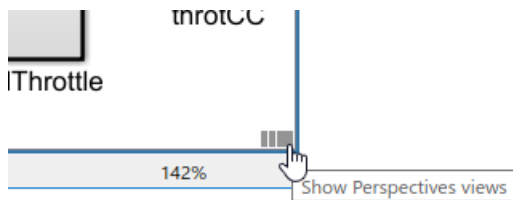
In this tutorial, link requirements to a model by using the model requirements perspective. Visual elements highlight links between requirements and blocks.

- 1 Open the example project by entering

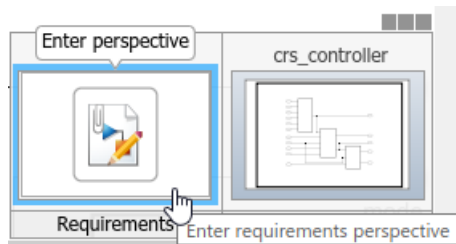
```
slreqCCProjectStart
```

Open `crs_controller` from the `models` folder.

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



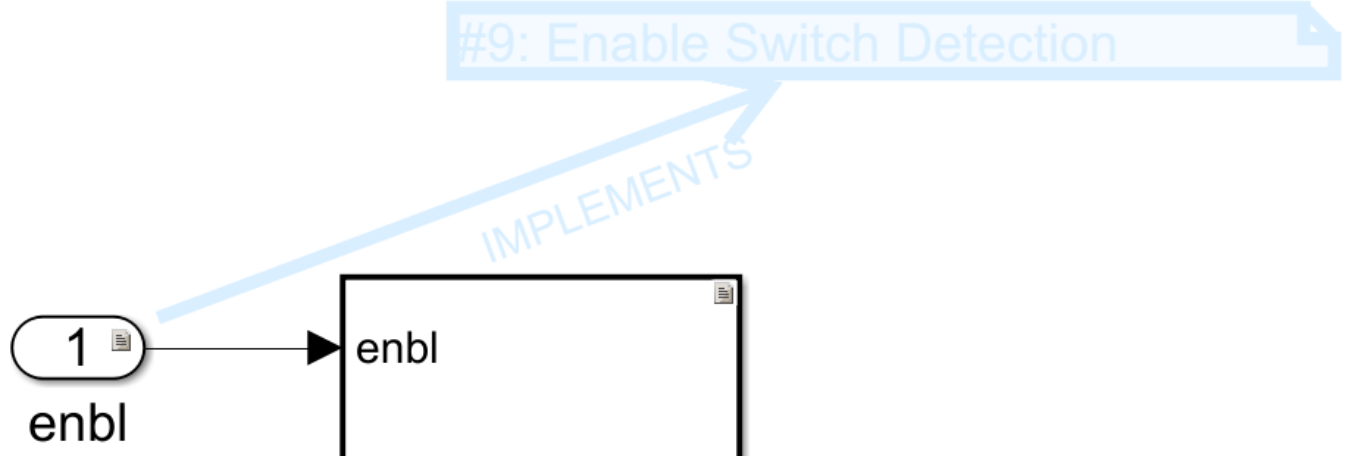
- 3 Open the requirements perspective by clicking the **Requirements** icon.



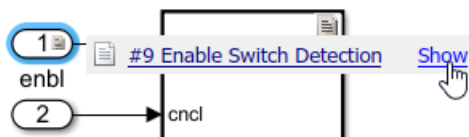
The Requirements Browser appears at the bottom of the model canvas. When you select a requirement, the Property Inspector displays the requirement's properties.

- 4 Link a requirement to a model element:

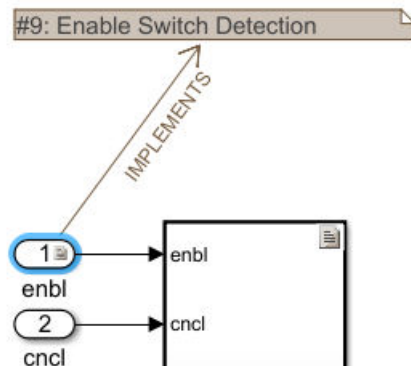
- 1 In the Requirements Browser, search for `Enable Switch Detection`.
- 2 Link to the `enb1 Inport` block by clicking and dragging the requirement to the block. An annotation template appears.
- 3 Place the requirement annotation by clicking on the canvas. Create a link without an annotation by clicking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



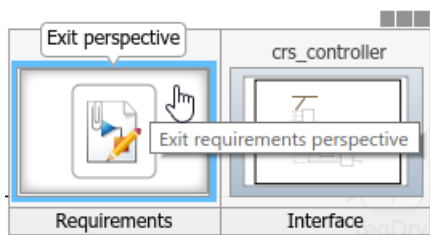
- 5 The block displays a link badge. To display information about the requirement, click the badge and select **Show**.



Clicking **Show** displays the requirement ID, requirement summary, and link type. For information on link types, see "Requirement Links" on page 2-11.



- 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

A traceability matrix allows you to easily view requirements and their links to blocks in your Simulink model in a compact format. A traceability matrix summarizes requirements, links, and model or test entities, and allows you to navigate to link sources or destinations. For example, you can:

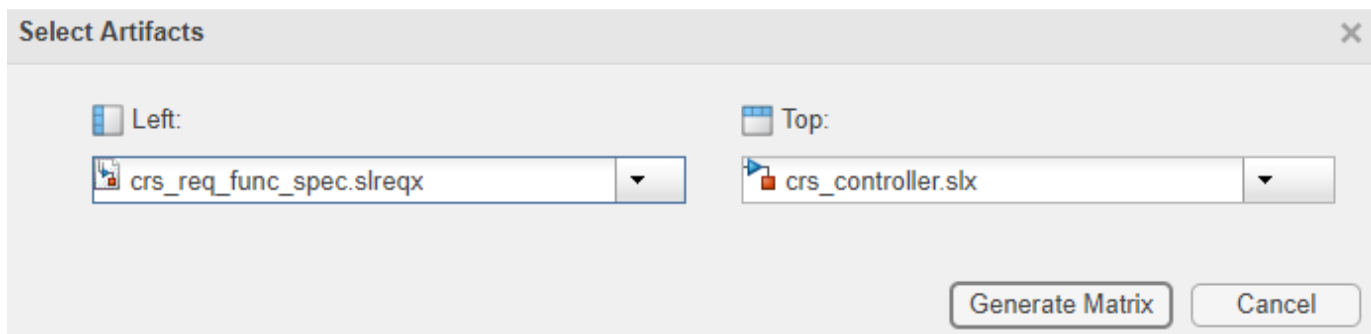
- Identify missing or incorrect links.
- Create missing links.
- Fix incorrect links.
- Display requirements that are linked to model elements.
- Inspect links by navigating to their sources or destinations.
- Filter by requirement type, link type, or model hierarchy.
- Confirm model and requirement completeness.

For large models, you can focus on specific model elements and associated requirements by expanding or collapsing the content in the traceability matrix. Additionally, you can filter out specific parts of the matrix to see only what you need to see.

### Generate a Traceability Matrix

To create a traceability matrix:

- 1 In the **Requirements** tab, select **Share > Open Requirements Traceability Matrix**.
- 2 Click **Add** to create a matrix.



The following artifacts can be used to create a traceability matrix:

- Simulink requirements
- Simulink model
- Simulink test
- Simulink data dictionary
- MATLAB files


For this matrix, the requirements are represented by the rows, while the blocks of the Simulink model are represented by the columns. Once you have selected your artifacts, click **Generate Matrix**. Your traceability matrix appears.

The screenshot displays the 'Simulink Requirements vs Simulink Model' interface. At the top, there is a toolbar with icons for 'Add', 'Update', 'Scope', 'Expand All', 'Collapse All', 'Highlight Missing Links', and 'Create Link'. Below the toolbar is a 'FILTER PANEL' on the left, which includes sections for 'Top', 'Type', 'Link', and 'Cell'. The main area shows a traceability matrix with requirements on the y-axis and model elements on the x-axis.


Requirement	crs_controller	CruiseControlMode	disableCaseDetection	outOfRange	getActStatus	opMode	reqDrv	brakeP	vehSp	key
#1 Driver Switch Request Hand										
#19 Cruise Control Mode		←								
#20 Disable Cruise Control s			←							
#24 Operation mode determi				●	←					
#37 Calculate Target Speed an										
#38 Disabled case										
#39 Enabled case										
#40 Activated case										
#43 Resume mode										
#44 System Interface										
#45 Inputs										

If you update your model or the requirements, click **Update** to refresh your traceability matrix.

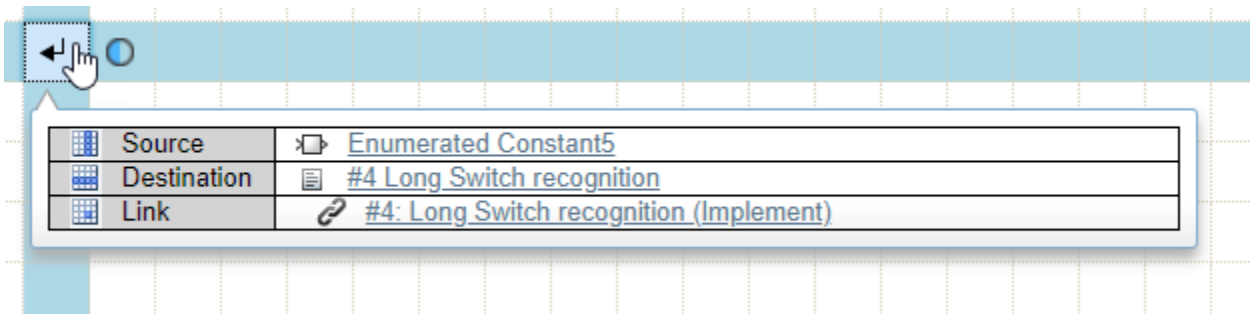
## Using the Traceability Matrix

The traceability matrix is a large grid with icons that show links. The arrow icon  indicates that there is a linked requirement between the requirement row and the model column.

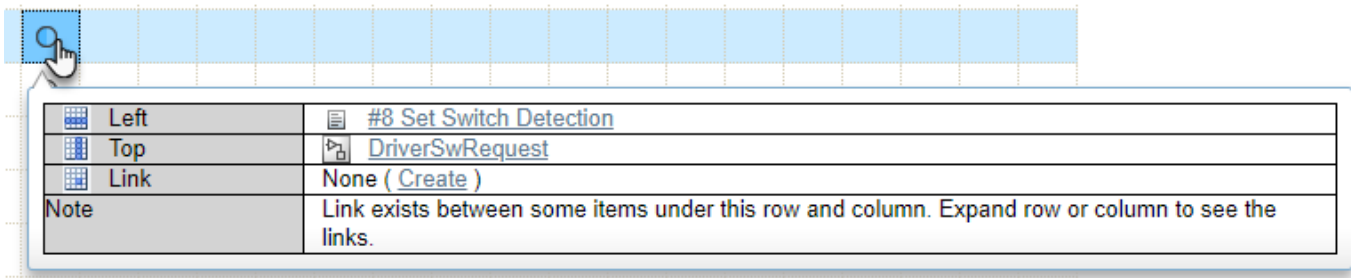
### Expand and Collapse Links

Initially, some rows and columns in your matrix may be compressed. The expand icon  indicates that a link exists in the row and column, but you need to expand the row or column to see the link.

When you click a link icon, you see information about the link.



When you click the expand icon, similar information appears.



When you click on the links in the information box, the corresponding information appears. For example, if you click on the requirement, the Requirements Editor window opens and displays the specified requirement.

### Artifact Hierarchy

You can also choose to show the hierarchy of a specific artifact in your traceability matrix. Right click on the artifact whose hierarchy you want to display. Click **Focus the display**.



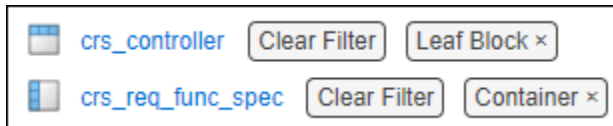
Your traceability chart now shows this level of the requirements or Simulink model. To show the entire hierarchy of the artifact, right click the artifact again and click **Display Entire Hierarchy**.

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**.

### Applying Filters

To the left of the traceability matrix is the Filter Panel. Display specific artifacts by using the Filter Panel. For example, if you click **Has No Links** under Column, the traceability matrix shows columns without links. You can choose to filter your requirements, your Simulink model, or specific links.

When you add a filter to the traceability matrix, it appears at the top of the matrix.



### Highlight Missing Links

To highlight cells in your traceability matrix that are missing links, click **Missing Links**. The artifacts in your traceability matrix without links are highlighted in yellow.

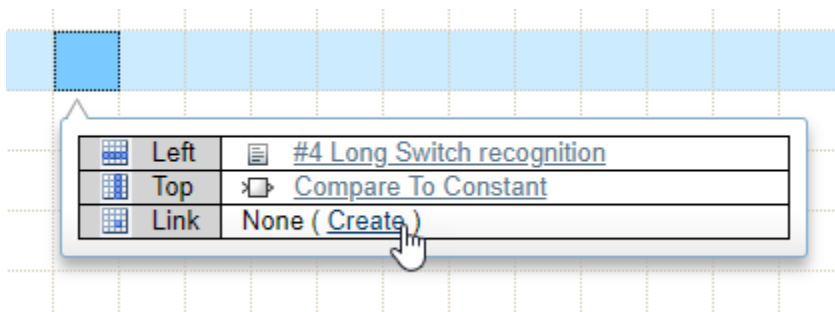


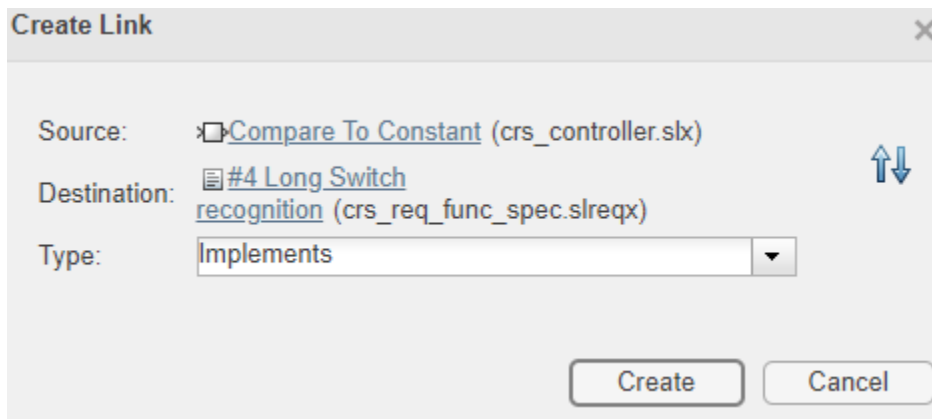
	crs_controller	CruiseControlMode	disableCaseDetection	IsKeyPositionOn	In1	Constant	Relational Operator	Out1	IsShiftDrive	In1
crs_req_func_spec										
#1 Driver Switch Request Hand										
#2 Switch precedence										
#3 Avoid repeating command										
#4 Long Switch recognition										
#7 Cancel Switch Detection										
#8 Set Switch Detection										
#9 Enable Switch Detection										
#10 Resume Switch Detectic										
#11 Increment Switch Detect										


These artifacts appear highlighted even if you display a specific hierarchy that does not include the missing link. View the hierarchy for the entire traceability matrix, to see all missing links. See “Artifact Hierarchy” on page 2-7.

### Add a New Link

Create a link by clicking on a cell, then click **Create** to create a link between the Simulink object and a requirement.





The Create Link window automatically populates the link source and destination. If necessary, you can reverse the link source and destination by clicking the reverse button .

### Limitations

The traceability matrix does not support the following:

- MATLAB Function blocks
- MATLAB scripts

Additionally, invalid links are not displayed in traceability matrices.

### See Also

#### More About

- “Link Blocks and Requirements” on page 2-2
- “Define Custom Requirement and Link Types” on page 2-14
- “Requirement Links” on page 2-11

## 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.

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 are connected to the requirement SID (Session Independent Identifier) and not to its Custom ID.

### Linkable Items

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

- Simulink Requirements objects:
  - `slreq.Requirement` objects
  - `slreq.Reference` objects
  - `slreq.Justification` objects
- Simulink entities
  - Blocks and ports
  - Subsystems
  - Signals
  - Data dictionaries
- Stateflow objects:
  - States
  - Charts and subcharts
  - Transitions
- Simulink Test objects:
  - Test files
  - Test suites
  - Test cases
- MATLAB structures
- MATLAB code lines
- System Composer™ architecture models

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

- 1 Create a link destination structure.

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

myLinkDest =

    struct with fields:
        domain: 'linktype_rmi_url'
        artifact: 'www.mathworks.com'
        id: ''

% Create a link between requirement myReq and myLinkDest
slreq.createLink(myReq, myLinkDest);
```

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

## Link Types

To track how the elements of your design are associated with your requirements, you can specify link types for your requirement links. Link types also describe the nature of requirement-to-requirement links, such as where a requirement is derived from a higher-level requirement.

Simulink Requirements provides these link types.

### Link Types in Simulink Requirements

Type	Description
Related to	General relationship between a requirement and a model element. This link is bidirectional.
Implemented by	Specifies which model elements implement this requirement. These link types contribute to the Implementation Status metric.
Implements	
Verified by	Specifies which verification model elements or test cases verify that this requirement is satisfied. These link types contribute to the Verification Status metric.
Verifies	
Derived from	Specifies which destination artifacts are derived from this source artifact.
Derives	
Refines	Specifies which destination artifacts add additional detail for the functionality specified by the source artifact.
Refined by	
Confirms	Specifies relationship between a requirement set and an external result source. These link types contribute to the Verification Status metric.
Confirmed by	

You can also create custom link types. For more information, see “Define Custom Requirement and Link Types” on page 2-14.

Requirement links have a source artifact and a destination artifact. Most of the link types are defined relative to the link direction. The Related to link type denotes a general relationship between two entities.

The Implements/Implemented by and Verifies/Verified by link types describe requirement-model relationships. Specify the source and the destination artifacts correctly for requirements with these link types because the Implementation Status and Verification Status summary metrics are derived from these link types. For more information on the Implementation Status and Verification Status summary metrics, see “Review Requirement Implementation Status Metrics Data” on page 3-2 and “Summarize Requirements Verification Status” on page 3-3.

## Review Requirement Links

Review your requirement links from the Links View. The Links View is available in the Requirements Editor and the **Requirements Browser** in the Requirements Perspective view. The Links View of the **Requirements Browser** in the Requirements Perspective view displays only the outgoing links from your source artifacts. Toggle between the Requirements and Links Views by using the **View** drop-down list in the toolbar.

When working in the Simulink Editor, you can review requirement links for individual requirements by using the Property Inspector in the Requirements Perspective view. Other links associated with your requirements are available in the Requirements View, in the **Links** pane. 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, Simulink Requirements does not preserve the **CommentedBy**, **CommentedOn**, or SID for the link.

## Resolve Links

A resolved link has an available source and destination. If a link source or destination is not available, the link is unresolved. For example:

- A link becomes unresolved if you delete a linked block from a model.
- A link is unresolved if a source or destination file, such as a Simulink Test test file, is not loaded in memory.

In the Links View, unresolved links are denoted by . Use the `setSource` and `setDestination` methods to resolve links.

## Load and Unload Link Information

All link information related to a requirement set, including link set files, Simulink models, and test files on the MATLAB or Project path are automatically loaded when you load the requirement set into memory. 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. Link information is automatically unloaded when you unload the requirement set from memory.

## See Also

“Define Custom Requirement and Link Types” on page 2-14 | `slreq.refreshLinkDependencies` | `setDestination` | `setSource`

## Define Custom Requirement and Link Types

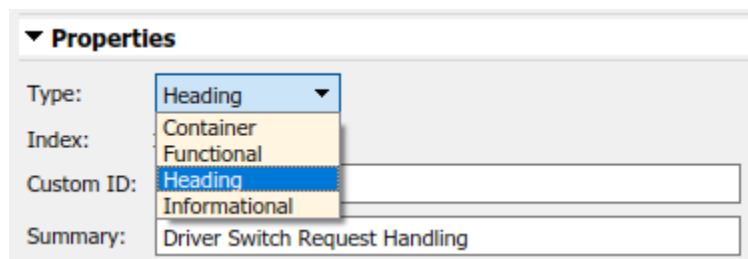
To define custom requirement and link types in addition to the built-in requirement and link types described in “Requirement Types” on page 1-6 and “Link Types” on page 2-12, you customize your Simulink user interface by registering a Simulink customization. For more information, “Registering Customizations” (Simulink).

In this example, you define custom requirement and link types by creating an `sl_customization.m` file in the current working folder. The following `sl_customization.m` file creates a custom requirement type called **Heading** and two custom link types called **Satisfy** and **Solve**. You can define custom requirement and link types to exclude requirements from contributing to the Implementation and Verification status metrics as shown in this code example.

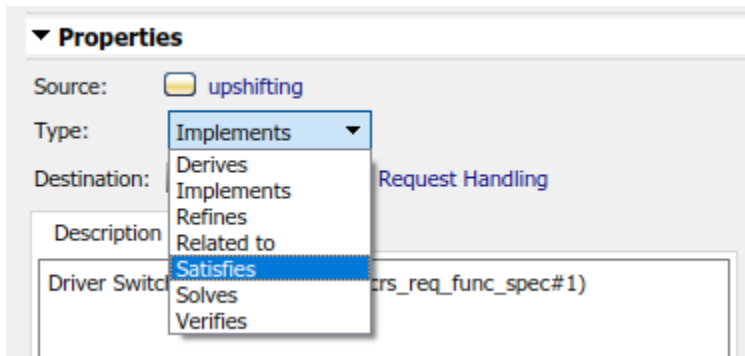
```
function sl_customization(cm)
    cObj = cm.SimulinkRequirementsCustomizer;
    cObj.addCustomRequirementType('Heading', slreq.custom.RequirementType.Container, ...
    'Headings of functional requirements')
    cObj.addCustomLinkType('Satisfy', slreq.custom.LinkType.Verify, 'Satisfies', ...
    'Satisfied by', 'Links to Verification objects')
    cObj.addCustomLinkType('Solve', slreq.custom.LinkType.Implement, 'Solves', ...
    'Solved by', 'Description')
end
```

- The **Heading** custom requirement type is defined as a subtype of the built-in **Container** requirement type. **Heading** requirements do not contribute to the Implementation status metric. All **Functional** requirements that are grouped under them do.
- The **Satisfy** custom link type comprises a source and destination artifact: **Satisfies** and **Satisfied by**. It is defined as a subtype of the **Verifies/Verified by** built-in link type. All **Satisfies/Satisfied by** requirement links contribute to the Verification status metric.
- The **Solve** custom link type comprises a source and destination artifact: **Solves** and **Solved by**. It is defined as a subtype of the **Implements/Implemented by** built-in link type. All **Solves/Solved by** requirement links contribute to the Implementation status metric.

You can select the custom requirement or link type from the Requirements Editor or the Requirements Perspective view. To select the custom requirement type, navigate to the **Requirements** view and select a requirement. Select the custom requirement type from the **Type** drop-down list in the **Properties** pane.



To select the custom link type, navigate to the **Links** view and select a link. Select the custom link type from the **Type** drop-down list in the **Properties** pane.



## Requirements Consistency Checks

### Check Requirements Consistency in Model Advisor

- “Identify requirement links with missing documents” on page 2-16
- “Identify requirement links that specify invalid locations within documents” on page 2-16
- “Identify selection-based links having description fields that do not match their requirements document text” on page 2-17
- “Identify requirement links with path type inconsistent with preferences” on page 2-18
- “Identify IBM Rational DOORS objects linked from Simulink that do not link to Simulink” on page 2-19

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**

<b>Condition</b>	<b>Recommended Action</b>
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 <b>Update</b> in the Model Advisor results to replace the current description in the selection-based link with the text from the requirements document (the external description). Alternatively, you can right-click the object in the model window, select <b>Edit/Add Links</b> from the <b>Requirements</b> menu, and use the Requirements dialog box that appears to synchronize the text.

**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 <b>Selection Linking</b> tab.	Change the preferred document file reference type or the specified paths by doing one of the following: <ul style="list-style-type: none"> <li>• Click <b>Fix</b> to change the current path to the valid path.</li> <li>• In the <b>Apps</b> tab, click <b>Requirements Viewer</b>. In the <b>Requirements Viewer</b> tab, click <b>Link Settings</b>.</li> </ul> Select the <b>Selection Linking</b> tab, and change the value for the <b>Document file reference</b> option.

### 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

Simulink Requirements enables you to insert navigation backlinks in external requirements documents to match requirement links in Simulink. The software also enables you to remove unmatched backlinks from external requirements documents when there is not a matching link from Simulink to a requirement in the document.

You can insert navigation backlinks in:

- Microsoft Word documents,
- Microsoft Excel spreadsheets, and
- IBM Rational DOORS modules.

When you insert a navigation backlink in an external requirements document, Simulink Requirements creates two link artifacts - a link (`slreq.Link` object) in Simulink Requirements and a navigation hyperlink (backlink) icon in the external requirements document. Backlink management in Simulink Requirements helps you synchronize both these link artifacts.

To insert or remove backlinks for your requirements, navigate to the **Links** view in the Requirements Editor or Requirements Perspective View. Right-click the link set file corresponding to the currently open model and click **Update Backlinks**. The **Backlinks checked** dialog box displays the total count of added and removed backlinks.

## See Also

### More About

- “Navigate to Requirements in Microsoft Office Documents from Simulink” on page 6-9
- “Link to Requirements in Microsoft Word Documents” on page 6-2
- “Link to Requirements in Excel Workbooks” on page 6-6

## 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 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
- `ReqSet.find('type', TYPENAME)` to located the wanted entry in a given ReqSet
- `LinkSet.getLinks()` to query all outgoing Links in LinkSet
- `Link.source()` brief information about link source
- `Link.destination()` brief information about link destination
- `Link.getReferenceInfo()` for "as stored" target info, which is different from "as resolved" `Link.destination()`
- `LinkSet.updateDocUri(ORIG_DOC, NEW_DOC)` to update link destinations when target document moved
- `ReqSet.updateSrcFileLocation()` to update previously imported set when source document moved
- `Reference.updateFromDocumet()` to update previously imported References from updated document
- `linkSet.redirectLinksToImportedReqs(reqSet)` 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 (ARGS) APIs that are inherited from Requirements Management Interface (RMI) part of the retired SLVnV Product.

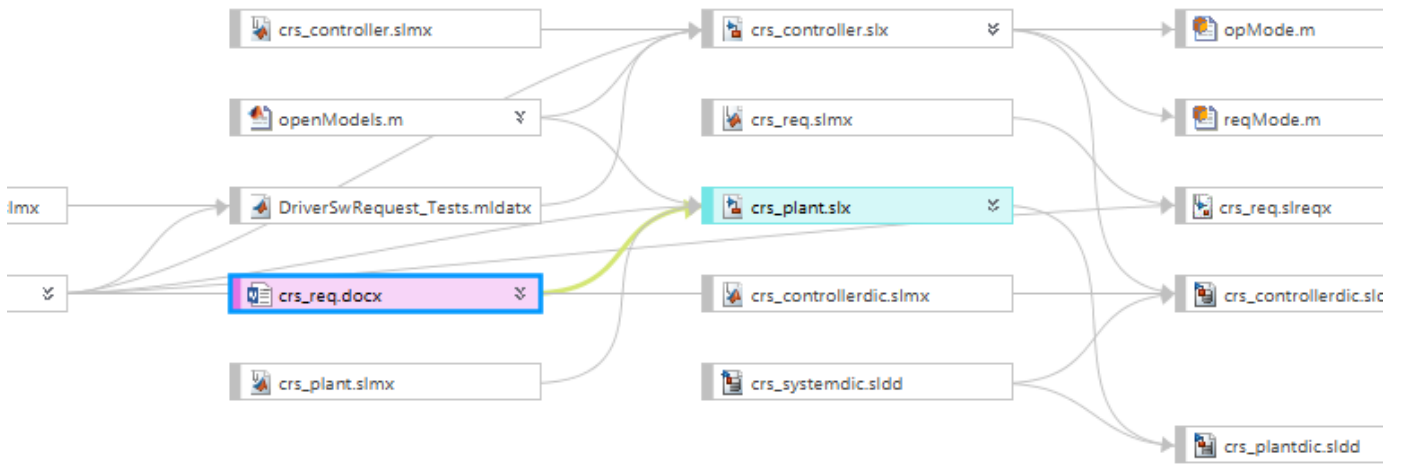
### Example Project Files

Before you begin, ensure a clean initial state by running `slreq.clear` command. Then type `slreqProjectStart` 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.

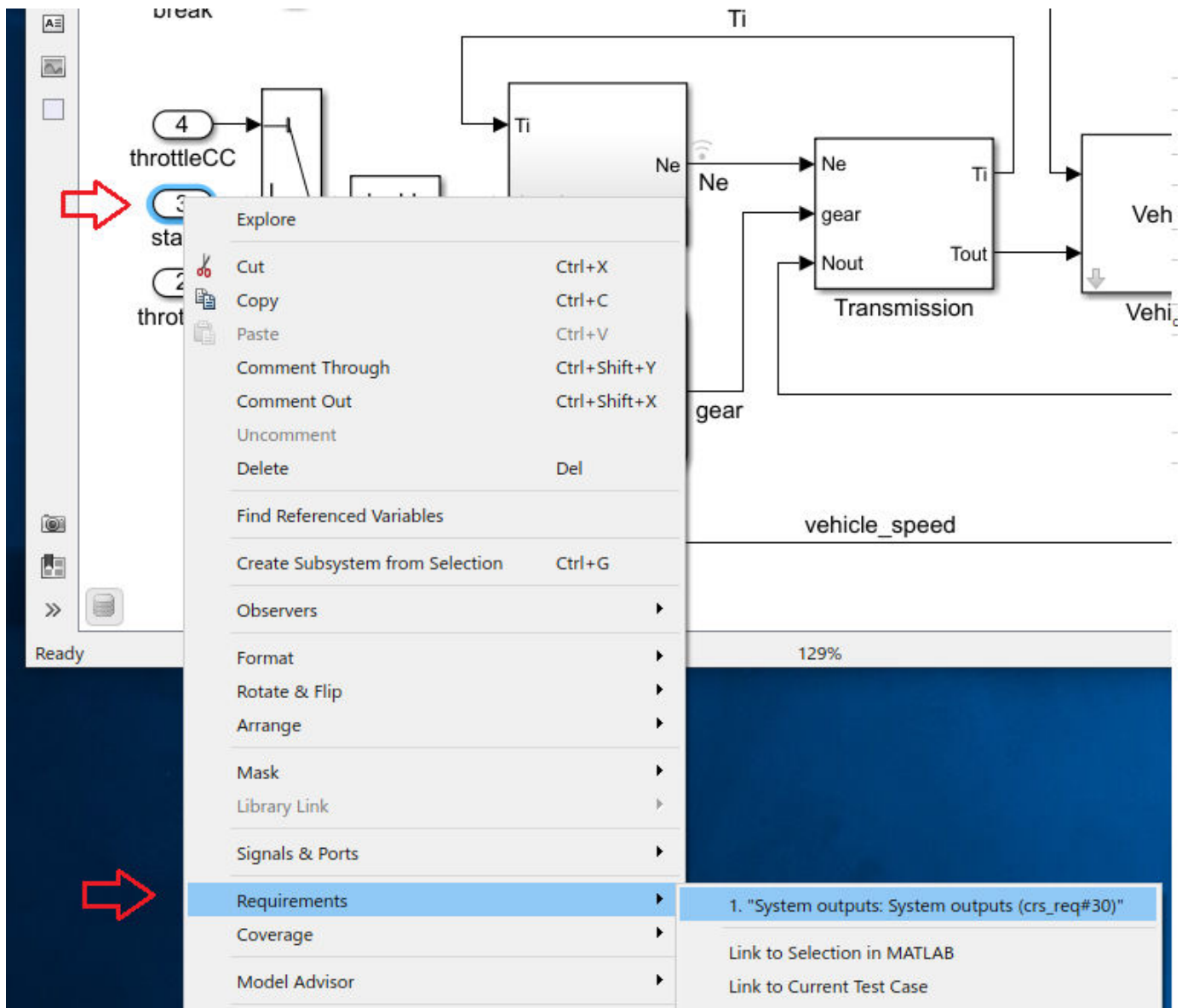
```
slreq.clear();
slreqCCProjectStart();
```

### Simulink Model Linked to Requirements

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



Navigate one of the links to open the linked document.



Word document opens to the correct section:

crs\_req.docx - Saved to this PC

do

AAE 1.1 AA 1.1.1 Aa 1.1.1.1 AaB AaBbCcD AaBbCcD AaBbCcD AABBCCD AaBb

ding 1 Heading 2 Heading 3 Heading 4 Title Subtitle Subtle Em... Emphasis Intense E... Stro

Styles

driver throttle	single	%	Throttle from ac
-----------------	--------	---	------------------

4.2 SYSTEM OUTPUTS

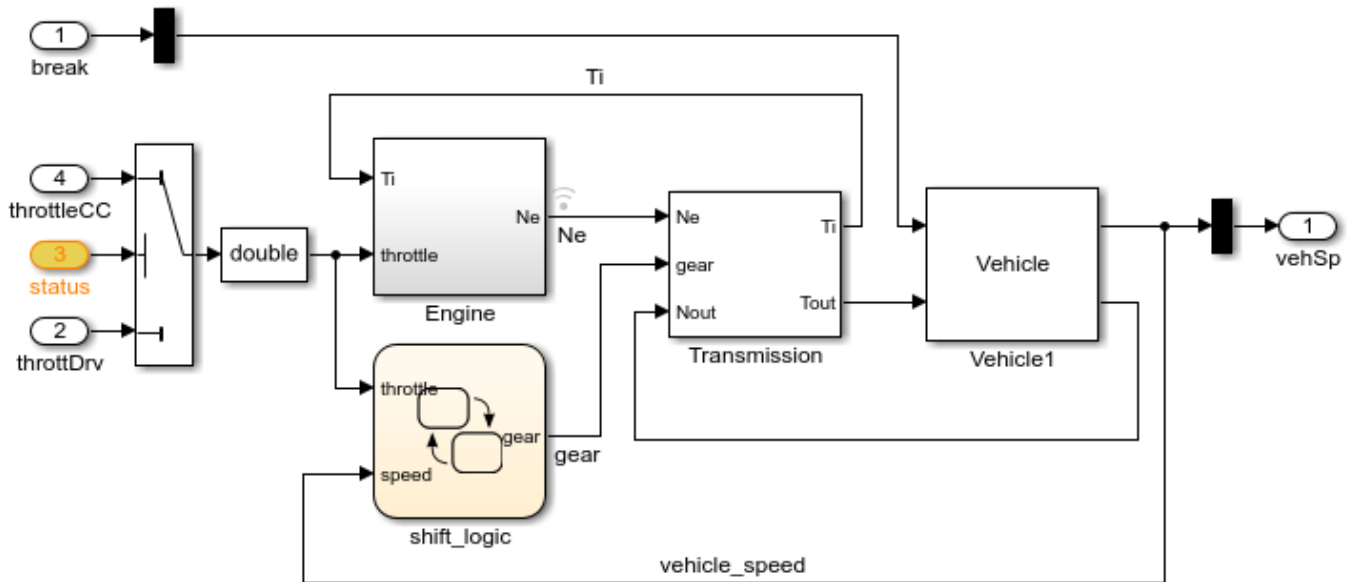
Name	Data type	Units	Description
reqDrv	Enum: reqMode		Driver's request
status	boolean		Cruise control sy

Here is how to use command-line APIs and check for links from crs\_plant.slx to crs\_reqs.docx.

```
open_system('crs_plant');
rmi('view', 'crs_plant/status', 1);
linkSet = slreq.find('type', 'LinkSet', 'Name', 'crs_plant');
links = linkSet.getLinks();
disp('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([bdroot source.id]),newline,'') ' to crs_
    end
end
```

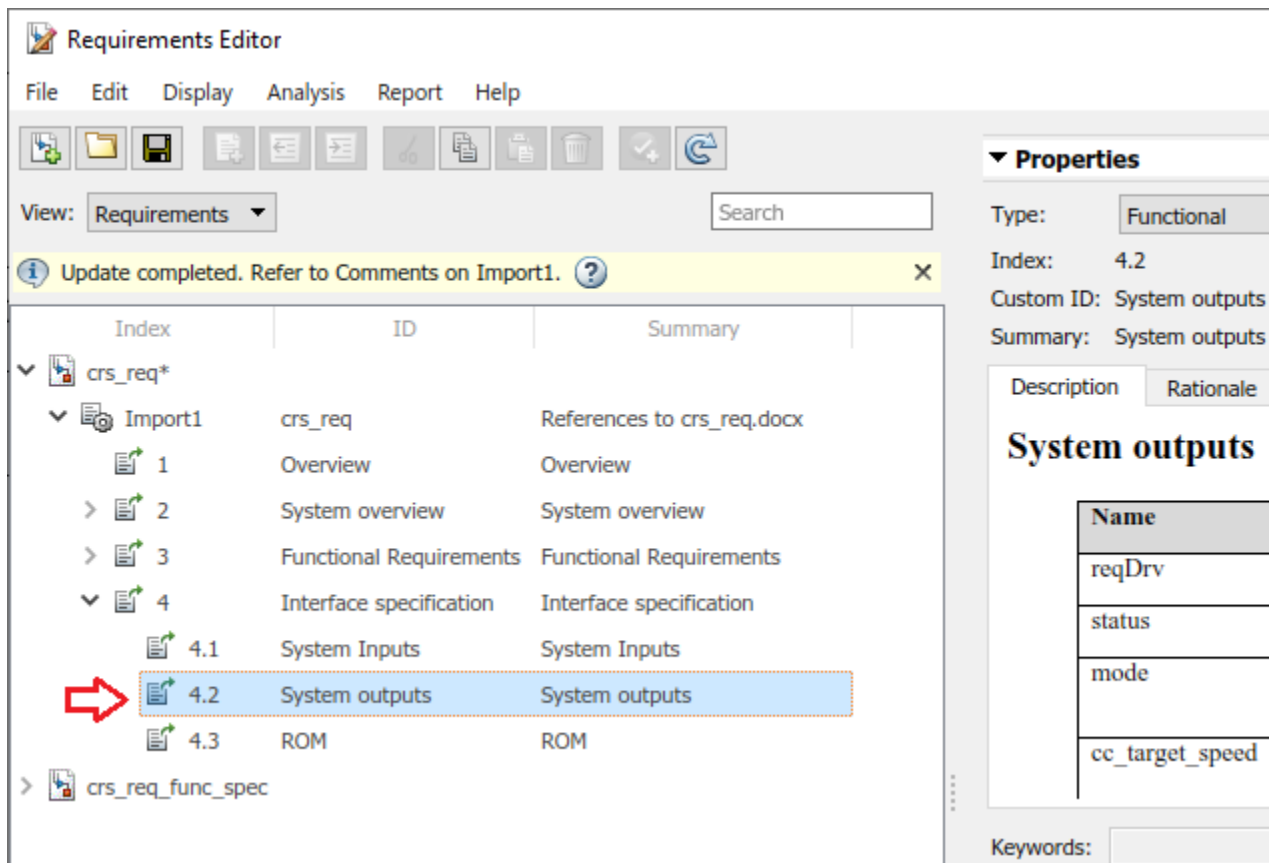
```
Original Links to Word document:
found link from crs_plant/Vehicle1/vehiclespeed to crs_req.docx
found link from crs_plant/throttDrv to crs_req.docx
found link from crs_plant/status to crs_req.docx
found link from crs_plant/throttleCC to crs_req.docx
```





### Navigation of Direct Links in the Presence of Imported References.

Open Simulink Requirements Editor. You will see two Requirement Sets loaded: `crs_req.slreqx` and `crs_req_funct_spec.slreqx`. The first Requirement Set is a collection of references imported from `crs_req.docx`, and the 2nd was manually created in 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 `LinkSet.updateDocUri(ORIG_DOC, NEW_DOC)` 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 Correct Update of Matching Links

Now we can navigate the same link and confirm that the correct version of the external document opens. IF we iterate all links as before, this confirms that all 4 links updated correctly:

```
rmi('view', 'crs_plant/status', 1); % updated document opens
links = linkSet.getLinks();
disp('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([bdrroot source.id]),newline,' ') ' to crs_req_v2.docx'])
    end
end
end

```

Links to Word document after update:

```

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

```

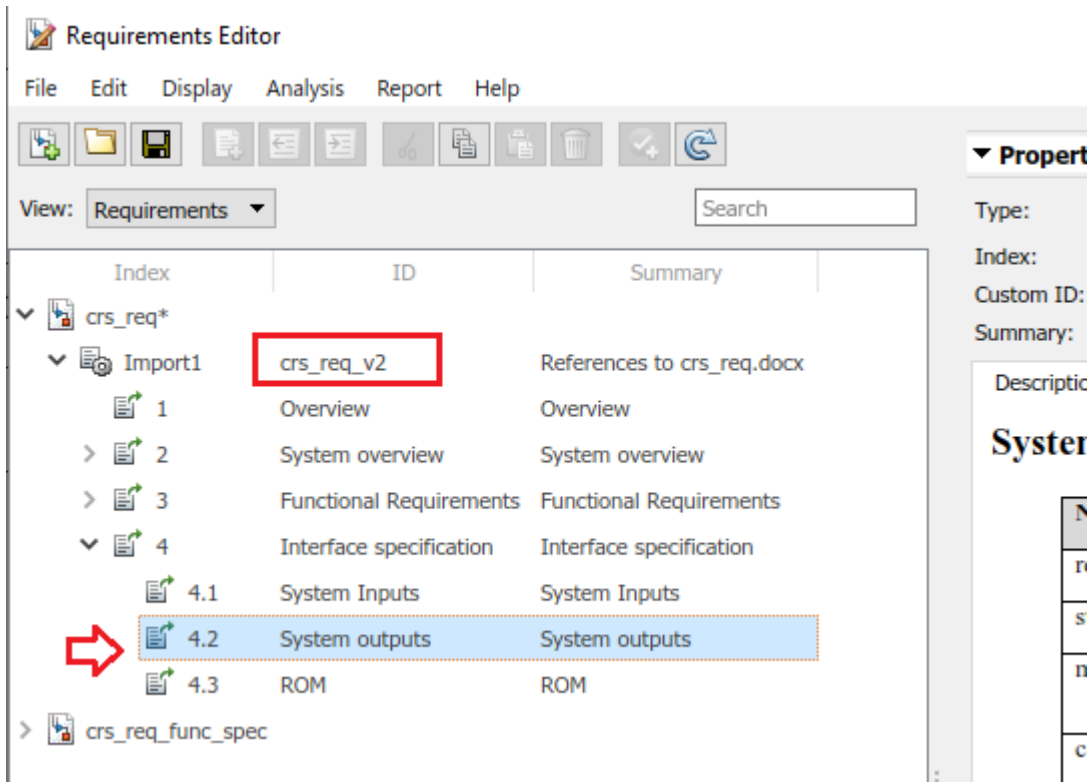
### Navigate to Imported References After Updating Links

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 three rows: "reqDrv" with data type "Enum: reqMode", "status" with data type "boolean", and "driver\_throttle" with data type "single" and units "%". A red arrow points to the "do" button in the top left corner of the Word window.

Name	Data type	Units	Description
reqDrv	Enum: reqMode		Driver's request
status	boolean		Cruise control sy
driver_throttle	single	%	Throttle from ac

To avoid this inconsistency we need to update the previously imported references for association with the updated document name. We use the `ReqSet.updateSrcFileLocation()` API to accomplish this task. Additionally, because the updated document may have modified Requirements, we must use `importNode.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 correct matching imported reference.



```
reqSet = slreq.find('type', 'ReqSet', 'Name', 'crs_req'); % ReqSet with imported References
reqSet.updateSrcFileLocation('crs_req.docx', 'crs_req_v2.docx');
importNode = reqSet.find('CustomId', 'crs_req_v2'); % top-level Import node
importNode.updateFromDocument();
rmdotnet.MSWord.application('kill'); % close Word application
rmi('view', 'crs_plant/status', 1); % navigate to highlight the updated reference in Requirements
```

### Cleanup After USE CASE 1

```
slreq.clear(); % discard link data changes to avoid prompts on Project Close
prj = simulinkproject(); prj.close(); % close the Simulink Project (also cleans-up MATLAB path)
rmdotnet.MSWord.application('kill'); % close MS Word application
```

### 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 we restart from the same initial point by reopening the Cruise Control Project example in a new subfolder. We then use `linkSet.redirectLinksToImportedReqs(reqSet)` API to update all the direct links in `crs_plant.slmx`. After updating the LinkSet in this way, we loop over all the links to confirm the absence of "direct" links to `crs_req.docx` file.

```
slreqCCProjectStart();
copyfile(fullfile(pwd, 'documents/crs_req.docx'), fullfile(pwd, 'documents/crs_req_v2.docx'));
open_system('crs_plant'); % open the Simulink model
```

```

linkSet = slreq.find('type', 'LinkSet', 'Name', 'crs_plant'); % LinkSet for crs_plant.slx
reqSet = slreq.find('type', 'ReqSet', 'Name', 'crs_req'); % ReqSet with imported References
linkSet.redirectLinksToImportedReqs(reqSet); % Convert all direct links to reference links
links = linkSet.getLinks();
disp('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']); % should not happen
        counter = counter + 1;
    end
end
disp([' Total ' num2str(counter) ' links to external document']); % should be 0 direct links
rmi('view', 'crs_plant/status', 1); % navigate from Simulink model to updated Reference

```

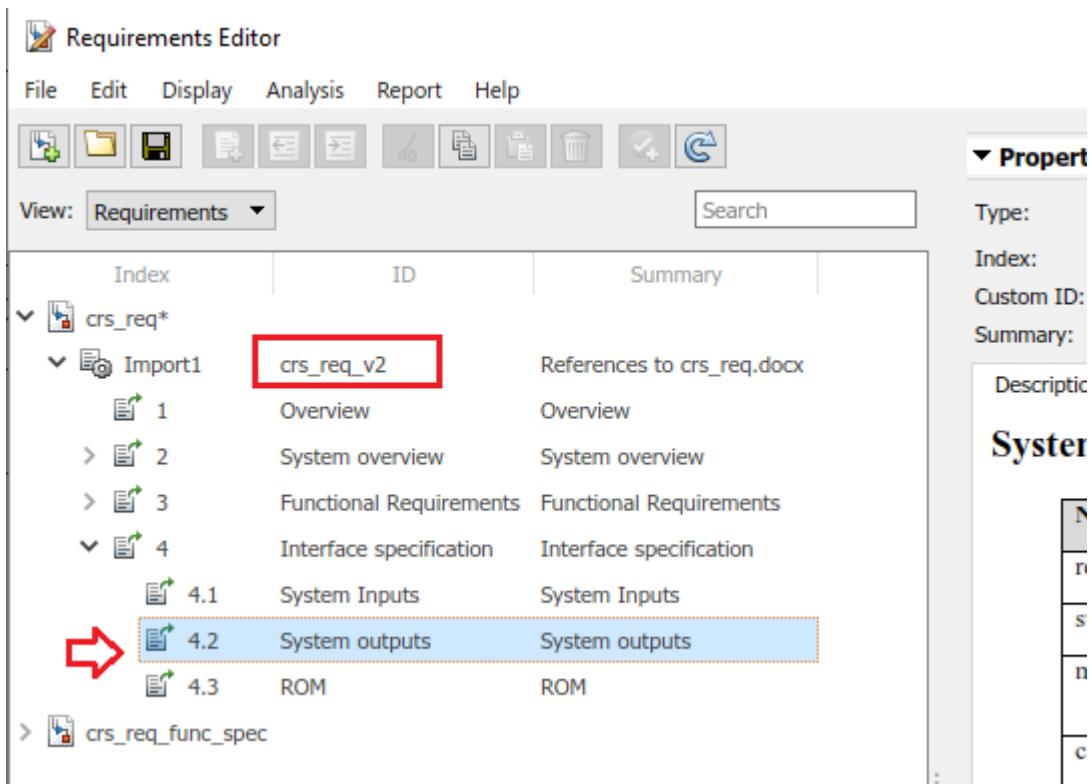
```

Check for links to original external document:
    Total 0 links to external document

```

### 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 correctly 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.

```
reqSet = slreq.find('type', 'ReqSet', 'Name', 'crs_req'); % ReqSet with imported References
reqSet.updateSrcFileLocation('crs_req.docx', 'crs_req_v2.docx');
importNode = reqSet.find('CustomId', 'crs_req_v2'); % top-level Import node
importNode.updateFromDocument();
rmi('view', 'crs_plant/status', 1); % navigates to a Reference in updated Requirement Set
```

### Cleanup After USE CASE 2

```
slreq.clear(); % discard link data changes to avoid prompts on Project C
prj = simulinkproject(); prj.close(); % close the Simulink Project (also cleans-up MATLAB path
rmdir('net.MSWord.application('kill'); % close MS Word application
```

### 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 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.

The screenshot shows the Requirements Editor interface. The left pane displays a tree view of requirements, with '4.2 System outputs' selected. The right pane shows the 'Properties' for this requirement, including a table of system outputs and revision information. A red arrow points to the 'crs\_req' folder in the tree view, and another red arrow points to the 'Original model not loaded' text next to the links in the 'Links' section.

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 system
cc target speed	single	km/h	Target speed

Revision information:

- SID: 30
- Revision: 1
- Updated on: 02-Feb-2018 13:23:13
- Created by:
- Created on: N/A
- Modified by:
- Modified on: 03-Aug-2017 17:34:56

Links:

- Related to:
  - crs\_plant:263
  - crs\_plant:75

Original model not loaded

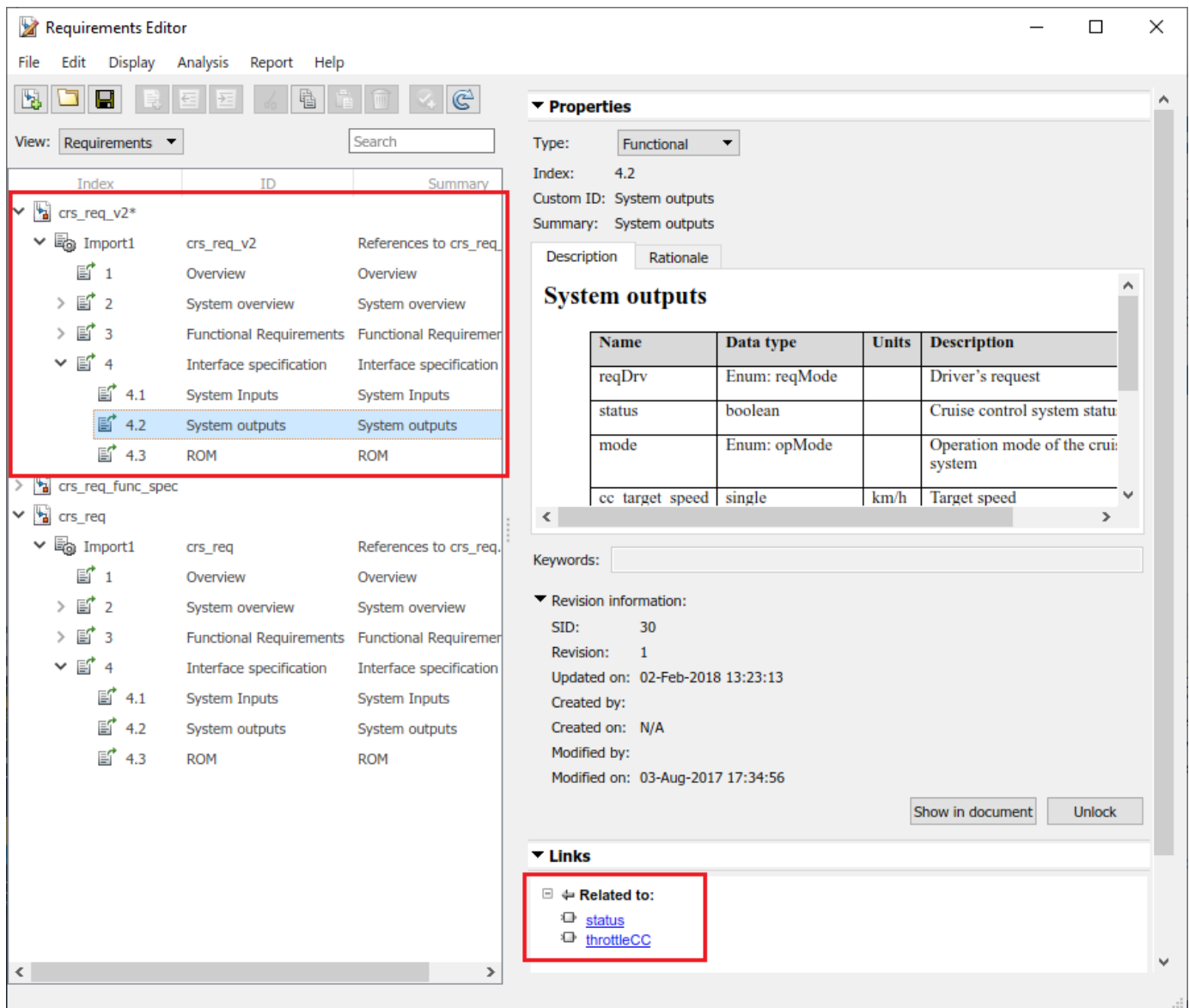
```

slreqCCProjectStart();
open_system('models/crs_plant.slx');
linkSet = slreq.find('type', 'LinkSet', 'Name', 'crs_plant'); % LinkSet for crs_plant.slx
reqSet = slreq.find('type', 'ReqSet', 'Name', 'crs_req'); % ReqSet with imported References
linkSet.redirectLinksToImportedReqs(reqSet); % Convert all direct links to reference links
mkdir(fullfile(pwd, 'copied'));
save_system('crs_plant', fullfile(pwd, 'copied/crs_plant_v2.slx')); % this also creates crs_plant_v2.slx
reqSet.save(fullfile(pwd, 'copied/crs_req_v2.slreq')); % new ReqSet copy to use with v2 model
copyfile('documents/crs_req.docx', 'copied/crs_req_v2.docx'); % new document copy to use with v2 model
reqSet.updateSrcFileLocation('crs_req.docx', fullfile(pwd, 'copied/crs_req_v2.docx')); % associate v2 model with v2 document
importNode = reqSet.find('CustomId', 'crs_req_v2'); % top-level Import node
importNode.updateFromDocument(); % ensure contents in renamed ReqSet match v2 model
rmi('view', 'crs_plant_v2/status', 1); % navigation from renamed Simulink model: the old item in

```

### Update Links in Renamed Source to Use the Renamed Destination as the Target

Similarly to USE CASE 1, we can use `LinkSet.updateDocUri(OLD, NEW)` API to update links in `crs_plant_v2.slmx` to use the renamed Requirement Set `crs_req_v2.slreqx` as the link target, instead of the original `crs_req.slreqx`. Once this is done, navigate again from the block in the renamed model. The correct requirement in the renamed Requirement Set is selected, and the links in the **Links** pane at bottom-right are resolved.



```
linkSet = slreq.find('type', 'LinkSet', 'Name', 'crs_plant_v2'); % LinkSet for crs_plant_v2.slx
linkSet.updateDocUri('crs_req.slreqx', 'crs_req_v2.slreqx'); % crs_plant_v2.slx should link v
rmi('view', 'crs_plant_v2/status', 1); % navigation from renamed Simulink model: correct item in
```

### Cleanup After USE CASE 3

```
slreq.clear(); % discard link data changes to avoid prompts on Project c
close_system('crs_plant_v2'); % this Model is not in Project, hence need to close separa
```



```
prj = simulinkproject(); prj.close(); % close the Simulink Project (also cleans-up MATLAB path  
rmdir(net.MSWord.application('kill')); % close MS Word application
```



# Requirements-Based Verification

---

- “Review Requirement Implementation Status Metrics Data” on page 3-2
- “Summarize Requirements Verification Status” on page 3-3
- “Justify Requirements” on page 3-6
- “Linking to a Test Script” on page 3-8
- “Include Results from External Sources in Verification Status” on page 3-16
- “Linking to a Result File” on page 3-19
- “Validate Requirements by Analyzing Model Properties” on page 3-26
- “Integrating results from a custom authored MATLAB script as a test” on page 3-33
- “Integrating Results from an External Result file” on page 3-37
- “Integrating results from a custom authored MUnit script as a test” on page 3-41

## Review Requirement Implementation Status Metrics Data

Simulink Requirements provides you with Implementation Status summaries for your requirement sets. You can use these status summaries to identify gaps in requirement implementation in your design. Requirements that have the link type set to **Implemented by** contribute to the Implementation Status summary metric.

The Implementation Status metrics for a requirement set are cumulatively aggregated over all the requirements in the set. Each child requirement belonging to a parent requirement must be implemented for the parent requirement to be considered as implemented.

You can view the Implementation Status metric for your requirement sets from both the Requirements Editor and the Requirements Browser in the Requirements Perspective View. To toggle the metric display, select **Display > Implementation Status** from the Requirements Editor menu. Hover your mouse over the Implemented column in the Requirements Editor or **Requirements Browser** for each requirement or requirement set to view the Implementation Status metric associated with it.

View: Requirements				
Index	ID	Summary	Implemented	Verified
▼	crs_req_func_spec			
▼	1	#1 Driver Switch Request Handling		
	1.1	#2 Switch precedence		
	1.2	#3 Avoid repeating commands		
>	1.3	#4 Long Switch recognition		
	1.4	#7 Cancel Switch Detection		
	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		
>	2.1	#20 Disable Cruise Control system		
>	2.2	#24 Operation mode determination		
▼	3	#37 Calculate Target Speed and Throttle ...		
	3.1	#38 Disabled case		
	3.2	#39 Enabled case		

### See Also

“Summarize Requirements Verification Status” on page 3-3 | “Justify Requirements” on page 3-6

## Summarize Requirements Verification Status

### In this section...

“Display Verification Status” on page 3-3

“Update Verification Status by Running Tests or Analyses” on page 3-4

“Include Verification Status in Report” on page 3-5

You can view the verification status of your requirements in the Requirements Browser and Requirements Editor. Verification status reflects results from simulation testing using Simulink Test or property proving using Simulink Design Verifier™. Use **Verified by** links from requirements to simulation assessments or proof objectives.

- **Simulation testing:** Verification status reflects the result of Simulink Test test files, test suites, and test cases linked to requirements. Pass, fail, or untested results derive from test assessments, including:
  - Model Verification library blocks in the system under test.
  - `verify` (Simulink Test) statements.
  - Baseline or equivalence data comparison.

Run tests from the 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).

- **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 2-11.

### Display Verification Status

Verification status is summarized in the **Verified** column of the Requirements Browser and Requirements Editor. To display the column,

- In the Requirements Editor, select **Display > Verification Status**, or
- 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:

View: Requirements

Index	ID	Summary	Implemented	Verified
▼	crs_req_func_spec			
▼	1	#1 Driver Switch Request Handling		
	1.1	#2 Switch precedence		
	1.2	#3 Avoid repeating commands		
>	1.3	#4 Long Switch recognition		
	1.4	#7 Cancel Switch Detection		
	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		
>	2.1	#20 Disable Cruise Control system		
>	2.2	#24 Operation mode determination		
▼	3	#37 Calculate Target Speed and Throttle ...		
	3.1	#38 Disabled case		
	3.2	#39 Enabled case		

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** (green): The requirement is excluded from the summary with a justification. For more information, see “Justify Requirements” on page 3-6.
- **Unexecuted**: (yellow):
  - The linked test(s) have not run, or the linked objective(s) have not executed, or
  - A linked test or objective has been updated more recently than the most recent result.
- **None** (colorless): The requirement does not have **Verified by** link(s).

### Update Verification Status by Running Tests or Analyses

You can update 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 4-8

## See Also

“Review Requirement Implementation Status Metrics Data” on page 3-2 | “Link to Test Cases from Requirements”


## Justify Requirements

Use requirement justification to exclude requirements from the Implementation and Verification Status metrics calculation for your requirement sets. You may have non-functional requirements in your model design specification that cannot be implemented in your design. You may 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 metrics and appear light blue in the Implemented and Verified columns of the Requirements Editor.

View: Requirements			Implemented	Verified
Index	ID	Summary		
▼	crs_req_func_spec			
▼	1	#1 Driver Switch Request Handling		
	1.1	#2 Switch precedence		
	1.2	#3 Avoid repeating commands		
>	1.3	#4 Long Switch recognition		
	1.4	#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		
>	2.1	#20 Disable Cruise Control system		
>	2.2	#24 Operation mode determination		
▼	3	#37 Calculate Target Speed and Throttle ...		
	3.1	#38 Disabled case		
	3.2	#39 Enabled case		

There are two workflows for justifying requirements in Simulink Requirements. You can either create a justification object and link requirements to it or use an existing justification object and link requirements to it.

- 1 Create a justification object by clicking the  icon in the Requirements Editor or **Requirements Browser** toolbar.
- 2 Right-click the requirement you want to link with the justification object and select **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 **Property Inspector**.



---

**Note** You cannot link justification objects to objects that are not requirements.

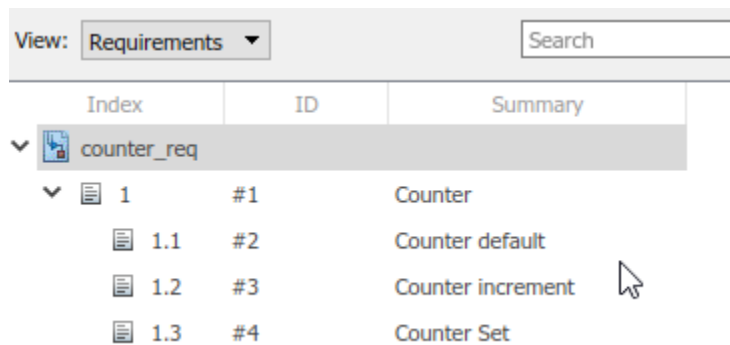
---

## Linking to a Test Script

In this example, 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 examples follow the workflow for including external test results in the requirement verification summary. For more information, see “Include Results from External Sources in Verification Status” on page 3-16.

### 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”.



The screenshot shows the Requirements Editor interface. At the top, there is a 'View: Requirements' dropdown and a search box. Below this is a table with columns 'Index', 'ID', and 'Summary'. The table is expanded to show a requirement set 'counter\_req' with four child requirements:

Index	ID	Summary
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 3-8
- “Create the Link” on page 3-9
- “View the Verification Status” on page 3-10

#### 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
```

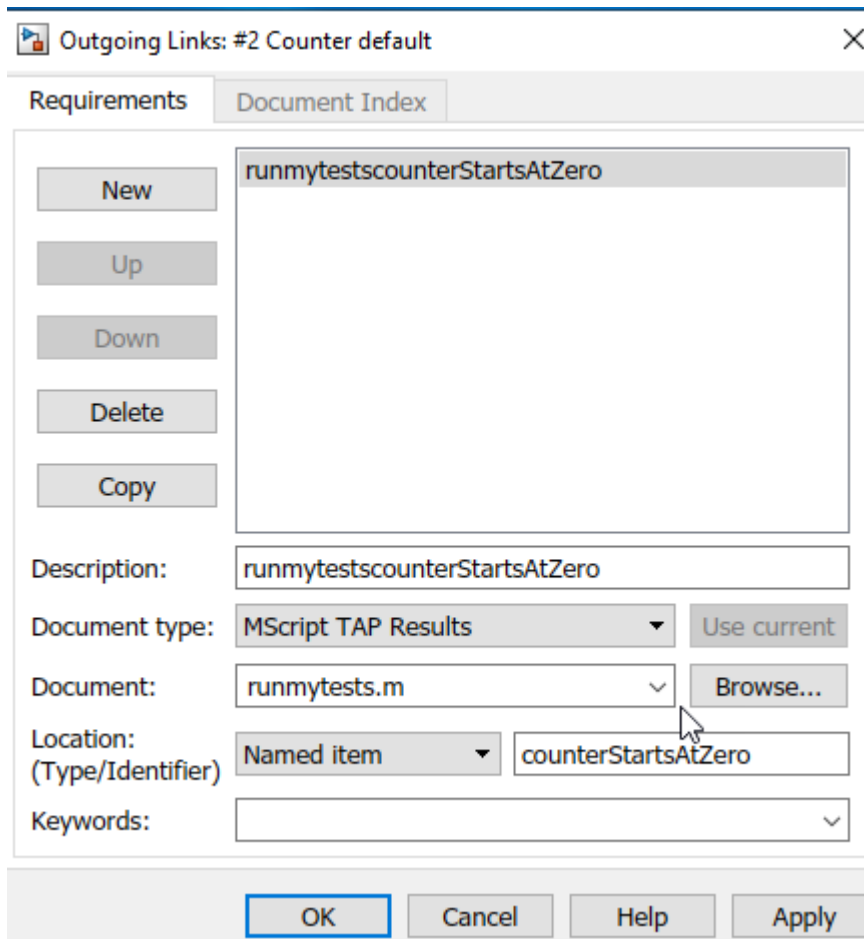
---

## 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 on the child requirement 1.1 and select **Open Outgoing Links Editor**.
- 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 on **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 screenshot displays the Requirements Editor interface. On the left, a tree view shows a requirement set 'counter\_req' with three child requirements: 1 '#1 Counter', 1.1 '#2 Counter default', 1.2 '#3 Counter increment', and 1.3 '#4 Counter Set'. The 'Verified' column shows progress bars for each. Requirement 1.1 is selected, and a tooltip indicates its verification status: 'Passed: 1, Justified: 0, Failed: 0, Unexecuted: 0, None 2, Total: 3'. The right pane shows the details for requirement 1.1, including a text editor with the content 'Counter default should be 0' and a 'Links' section with a link to 'counterStartsAtZero' marked as confirmed.

The requirements for counterStartsAtZero are fully verified. Here, the verification status shows that out of three tests, one test passed.

## Linking to a Test Script Using the API

Create a requirement set called counter\_req.s1reqx in the Requirements Editor and save it in a writable location. This requirement set has child requirements that have requirement IDs and descriptions. For more details on how to create requirement sets, see “Work with Requirements in the Simulink Editor”.

Index	ID	Summary
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 3-12
- “Create the Link” on page 3-13
- “View the Verification Status” on page 3-13

### 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

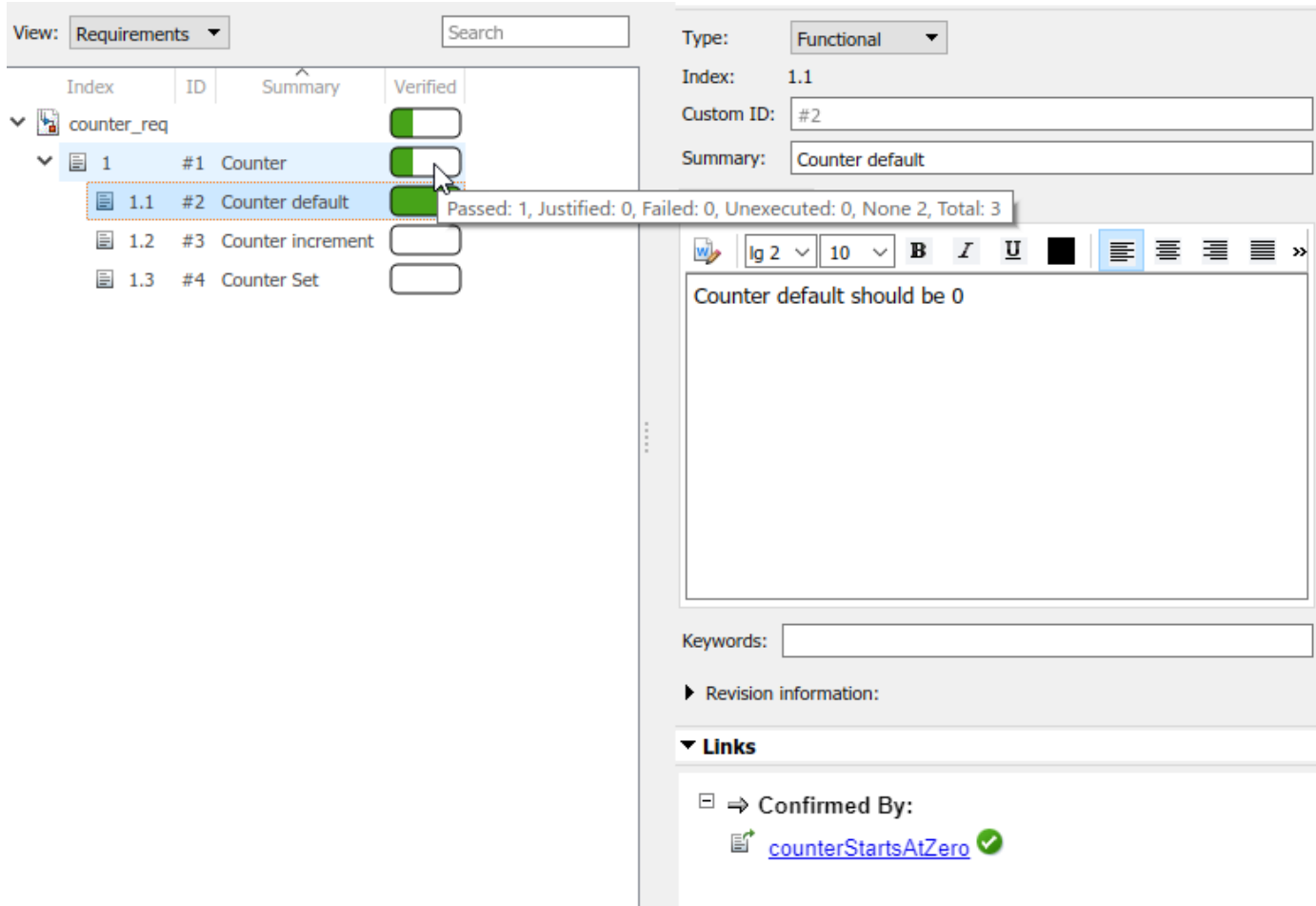
Update the verification status. At the MATLAB command prompt, type:

```
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.



Index	ID	Summary	Verified
1	#1	Counter	<input checked="" type="checkbox"/>
1.1	#2	Counter default	<input checked="" type="checkbox"/>
1.2	#3	Counter increment	<input type="checkbox"/>
1.3	#4	Counter Set	<input type="checkbox"/>

Passed: 1, Justified: 0, Failed: 0, Unexecuted: 0, None 2, Total: 3

Counter default should be 0

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.



View: Requirements

Index	ID	Summary	Verified
counter_req			<input checked="" type="checkbox"/>
1	#1	Counter	<input checked="" type="checkbox"/>
1.1	#2	Counter default	<input checked="" type="checkbox"/>
1.2	#3	Counter increment	<input type="checkbox"/>
1.3	#4	Counter Set	<input type="checkbox"/>

Type: Functional

Index: 1.1

Custom ID: #2

Summary: Counter default

Description: Counter default should be 0

Keywords:

Revision information:

Links

Confirmed By: [testCounterStartsAtZero](#)  Passed

### See Also

“Include Results from External Sources in Verification Status” on page 3-16

## 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 “Summarize Requirements Verification Status” on page 3-3.

You can also include test results from:

- Continuous integration (CI) servers such as Jenkins
- Custom results updated manually or with test scripts

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. You can also 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.

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:

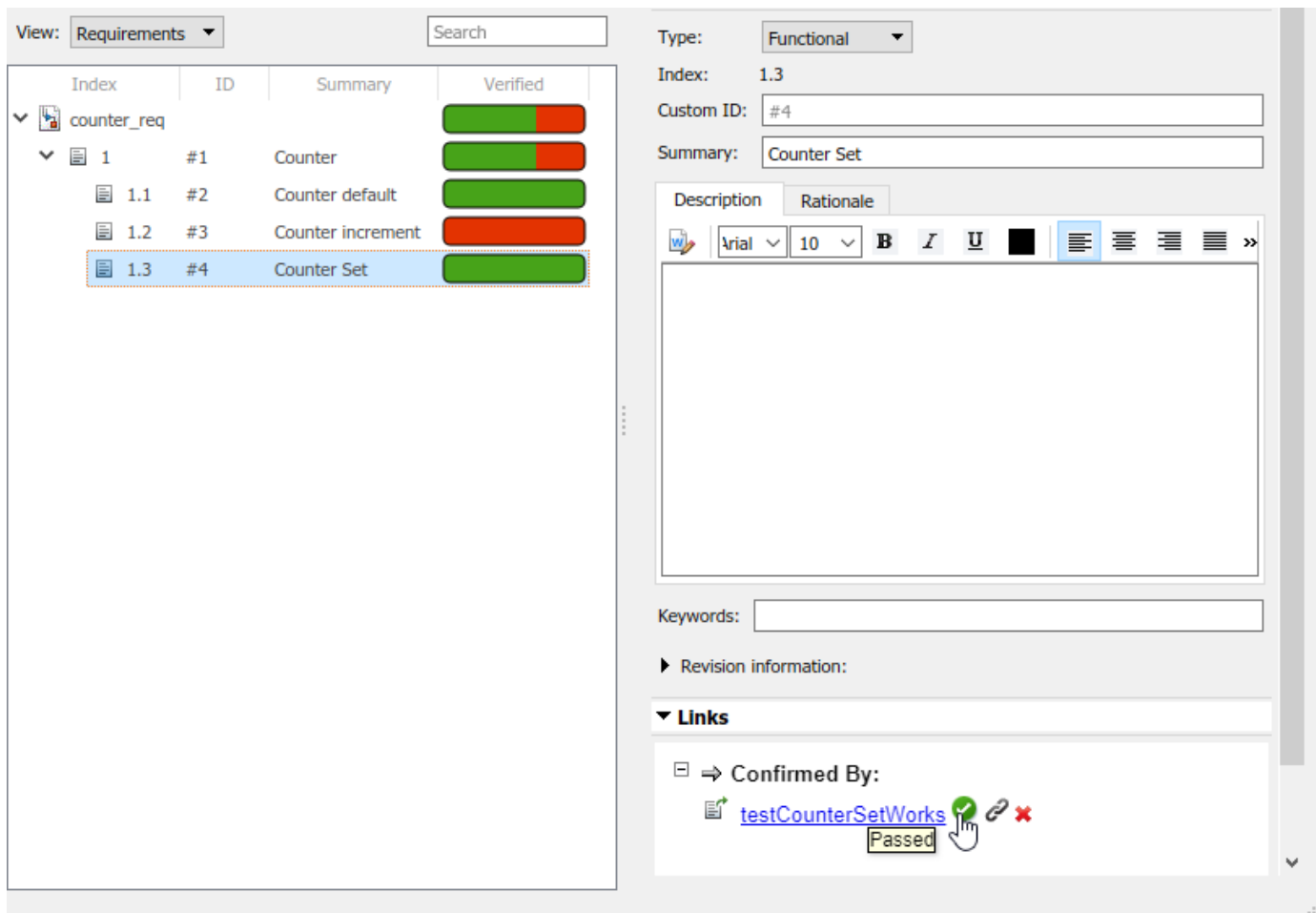
- Linking directly to a result file. The external result artifact is used as the link destination and the requirements are used as the links source. To create custom link type, you must know:
  - 1 The file location
  - 2 The file format (for example, JUnit or Excel)

For details, see “Linking to a Result File” on page 3-19.

- 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 3-8.

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 10-14.
- 2 **Register the custom link type:** See “Custom Link Type Registration” on page 10-14. 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 “Summarize Requirements Verification Status” on page 3-3.
  - 5 **Refresh the requirements view:** After the tests run, refresh the verification status by clicking the **Refresh**  button.



The screenshot displays the Requirements Management Interface (RMI) with a table of requirements and a detailed view of a specific requirement.


Index	ID	Summary	Verified
counter_req			
1	#1	Counter	
1.1	#2	Counter default	
1.2	#3	Counter increment	
1.3	#4	Counter Set	

The detailed view for requirement 1.3 shows the following information:

- Type:** Functional
- Index:** 1.3
- Custom ID:** #4
- Summary:** Counter Set
- Description:** (Empty text area)
- Rationale:** (Empty text area)
- Keywords:** (Empty text area)
- Revision information:** (Collapsible section)
- Links:** Confirmed By: [testCounterSetWorks](#)

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

“Linking to a Test Script” on page 3-8 | “Linking to a Result File” on page 3-19

### Related Examples

- “Integrating results from a custom authored MATLAB script as a test” on page 3-33
- “Integrating Results from an External Result file” on page 3-37
- “Integrating results from a custom authored MUnit script as a test” on page 3-41

## Linking to a Result File

In this example, you link a requirement to a test result file that is in 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 examples follow the workflow for including external test results in the requirement verification summary. For more information, see “Include Results from External Sources in Verification Status” on page 3-16.

### Linking to a Result File 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”.

View:	Requirements	Search
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 results file is 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

Suppose you want to update the verification information for the `counterSetsValue` test case based on the Excel status log. Follow these steps to create and verify links to the result file:

- “Create and Register the Link Type” on page 3-20
- “Create the Link” on page 3-21
- “View the Verification Status” on page 3-21

### 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_myexcelresults.m`.
- 3 In `linktype_myexcelresults.m`:
  - a Replace the function name `linktype_TEMPLATE` with `linktype_myexcelresults`.
  - 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_myexcelresults` and enter:

```
linktype.GetResultFcn = @GetResultFcn;
.....
function result = GetResultFcn(link)
    testID = link.destination.id;
    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_myexcelresults.m`.
- 5 Register the link type. At the command line, enter:

```
rmi register linktype_myexcelresults
```

---

**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_myexcelresults
```

---


## Create the Link

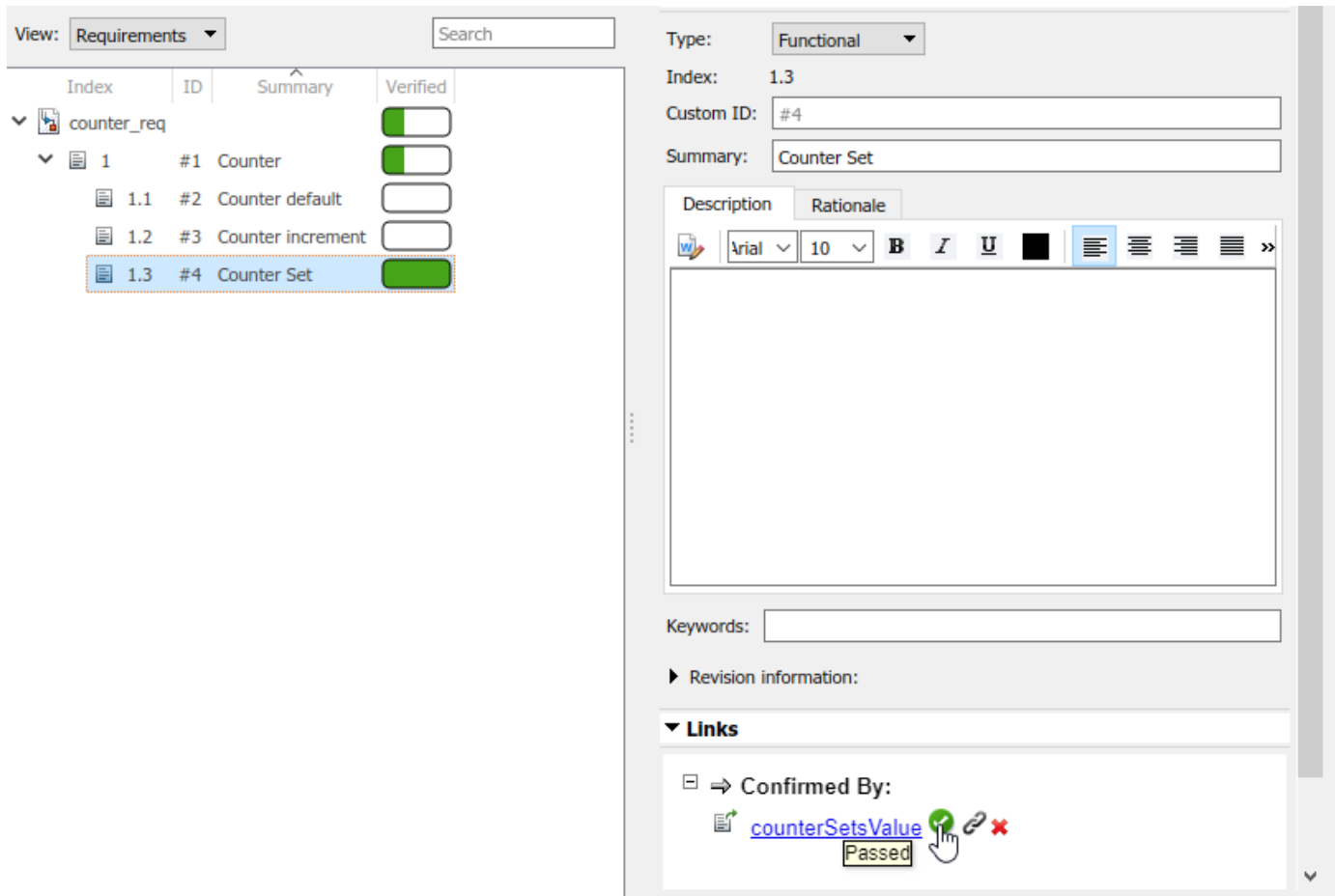
Follow these steps to add the link manually in the Outgoing Links Editor:

- 1 Open the Requirements Editor and, in the counter\_req.slreqx requirement set, right-click on the child requirement 1.3 and select **Open Outgoing Links Editor**.
- 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.

The screenshot shows the 'Outgoing Links Editor' dialog box. The title bar is 'Outgoing Links: #4 Counter Set'. There are two tabs: 'Requirements' and 'Document Index'. The 'Requirements' tab is active. On the left side, there are five buttons: 'New', 'Up', 'Down', 'Delete', and 'Copy'. The 'New' button is highlighted. In the center, there is a list box containing 'resultcounterSetsValue'. Below the list box, there are several input fields: 'Description:' with 'resultcounterSetsValue', 'Document type:' with a dropdown set to 'Excel Results' and a 'Use current' button, 'Document:' with a dropdown set to 'results.xlsx' and a 'Browse...' button, 'Location: (Type/Identifier)' with a dropdown set to 'Named item' and a text field containing 'counterSetsValue', and 'Keywords:' with an empty dropdown. At the bottom, there are four buttons: 'OK', 'Cancel', 'Help', and 'Apply'.

## View the Verification Status

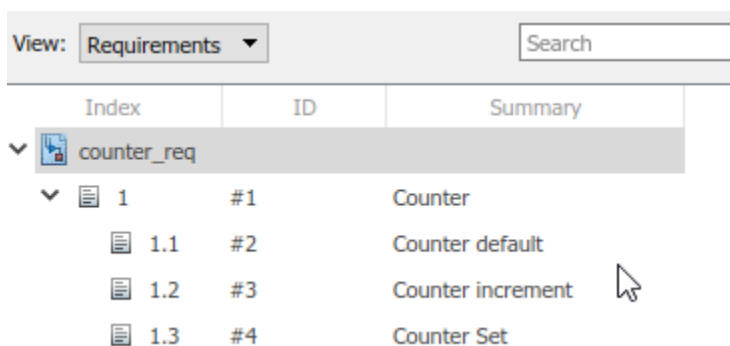
Update the verification status in the Requirements Editor. Click **Refresh**  button to see the verification status for the requirements in the Requirements Editor. This shows the verification status for entire requirement set that passed or failed.



The requirements for counterSetsValue are fully verified. Here, the verification status shows that out of three tests, one test passed.

### Linking to a Result File 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”.





The external results file is 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

Suppose you want to update the verification information for the `counterSetsValue` test case based on the Excel status log. Follow these steps to create and verify links to the result file:

- “Create and Register the Link Type” on page 3-23
- “Create the Link” on page 3-24
- “View the Verification Status” on page 3-25

### Create and Register the Link Type

Open the template file at `matlabroot/toolbox/slrequirements/linktype_examples/linktype_TEMPLATE.m`. Follow these steps:

- 1 Create a new MATLAB file.
- 2 Copy the contents of `linktype_TEMPLATE` into the new file. Save the file as `linktype_myexcelresults.m`.
- 3 In `linktype_myexcelresults.m`:
  - a Replace the function name `linktype_TEMPLATE` with `linktype_myexcelresults`.
  - 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_myexcelresults` and enter:

```
linktype.GetResultFcn = @GetResultFcn;
.....
function result = GetResultFcn(link)
    testID = link.destination.id;
    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\_myexcelresults.m.
- 5 Register the link type. At the command line, enter:

```
rmi register linktype_myexcelresults
```

---

**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_myexcelresults
```

---

#### Create the Link

Follow these steps to create the link:

- 1 From the MATLAB command prompt, enter:  

```
externalSource.id = 'counterSetsValue';
externalSource.artifact = 'results.xlsx';
externalSource.domain = 'linktype_myexcelresults';
```
- 2 Find the requirement related to the link by typing:  

```
requirement = reqSet.find('Type', 'Requirement', 'SID', 4);
```
- 3 Create the link by entering:

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

This creates the link as test case counterSetsValue for the requirement SID. In Requirements Editor, the link appears in the **Links > Confirmed By** section.



## View the Verification Status

Update the verification status for the requirement set. At the MATLAB command prompt, type:

```
reqSet.updateVerificationStatus
```

Fetch the verification status for the requirement by entering:

```
status = reqSet.getVerificationStatus
```

This shows the verification status for entire requirement set that passed or failed. Click on **Refresh**



to see the verification status for the requirements in the Requirements Editor.

The screenshot displays the Requirements Editor interface. On the left, a tree view shows a hierarchy of requirements under 'counter\_req'. The table below shows the verification status for each requirement:

Index	ID	Summary	Verified
1	#1	Counter	<input checked="" type="checkbox"/>
1.1	#2	Counter default	<input type="checkbox"/>
1.2	#3	Counter increment	<input type="checkbox"/>
1.3	#4	Counter Set	<input checked="" type="checkbox"/>

The right pane shows the details for requirement 1.3, 'Counter Set'. It includes fields for Type (Functional), Index (1.3), Custom ID (#4), and Summary (Counter Set). Below these are tabs for Description and Rationale, a rich text editor, and a Keywords field. A 'Revision information' section is collapsed. The 'Links' section shows a link to 'counterSetsValue' with a 'Passed' status and a green checkmark icon.

The requirements for counterSetsValue are fully verified. Here, the verification status shows that out of three tests, one test passed.

## See Also

“Include Results from External Sources in Verification Status” on page 3-16

## Validate Requirements by Analyzing Model Properties

Help validate a set of requirements by analyzing properties that model individual requirements. Falsified properties indicate incompleteness in the requirements set.

### Overview

In this example, you analyze verification logic modeled on four safety requirements of an engine thrust reverser system. Falsified results from the safety requirements analysis suggest that the system *design* requirements are incomplete. That is, the system allows behavior that violates the safety requirements:

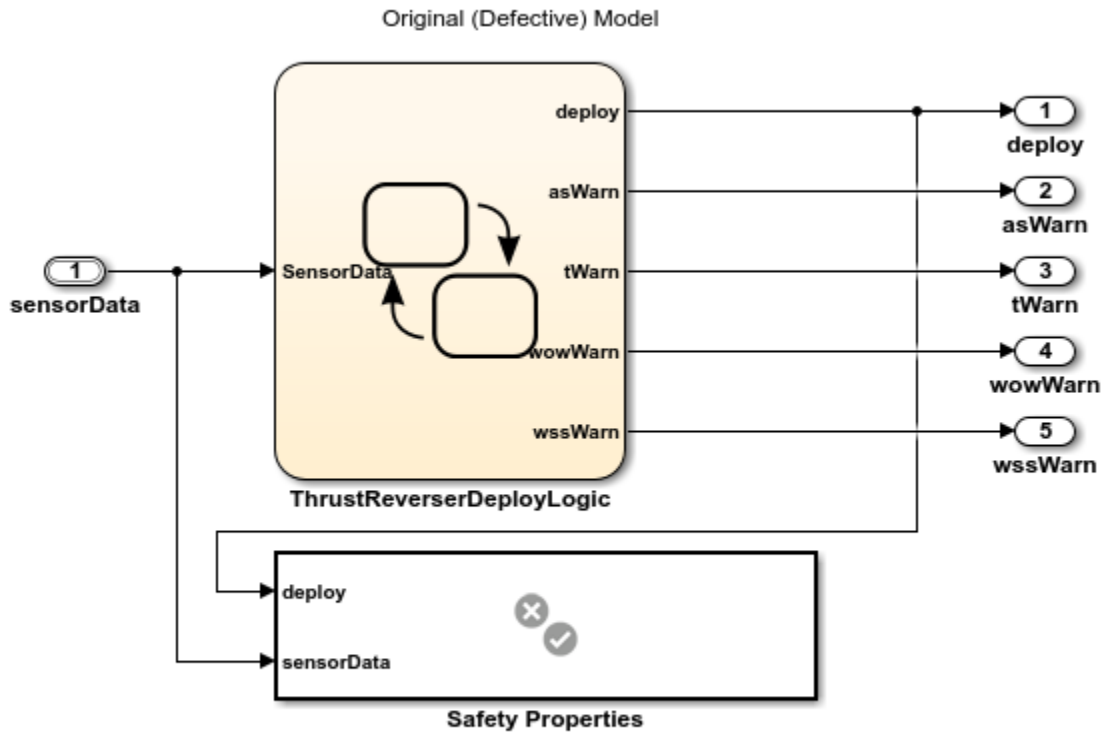
- 1 Thrust reverser shall not deploy if the airspeed is greater than 150 knots.
- 2 Thrust reverser shall not deploy if the aircraft is in the air, as indicated by the value of the weight on wheels sensors. If the aircraft is in the air, the signal value for each of two weight on wheels (WOW) sensors is `false`.
- 3 Thrust reverser shall not deploy if the value of either thrust sensor is positive.
- 4 Thrust reverser shall not deploy if the rotational speed of the landing gear wheels is less than a threshold value.

Further analysis shows that the system lacks necessary control logic. A revised control system passes analysis of the properties modeled on the safety requirements.

### Analyze the Safety Properties

1. Open the `reqs_validation_property_proving_original_model` model from the `matlab/examples/slrequirements` folder.

## Requirements Validation Using Property Analysis Example



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.

2. View the requirements. In the model, click the icon at the lower right and select the **Requirements** tile. The Requirements Browser appears below the model. Expand thrust\_reverser\_safety\_requirements.

The safety requirements link to the Assertion blocks in the Safety Properties subsystem, and 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 appears and shows Unexecuted status for 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	

4. Analyze the model properties. Navigate to a writable folder. In the **Design Verifier** toolstrip, click the **Prove Properties** button.

```

Checking compatibility for property proving: model 'reqs_validation_property_proving_original_model'
Compiling model...done
Building model representation...done

'reqs_validation_property_proving_original_model' is compatible for property proving with Simulink

Proving properties using model representation from 29-Feb-2020 11:18:43...
....

Completed normally.

Generating output files:

Results generation completed.

Data file:
C:\TEMP\Bdoc20a_1326390_10420\ibC22023\2\tp852d6cf0\ex56983666\sldv_output\reqs_validation_p
    
```

When the property analysis completes, click the **Refresh** button to update the status in the **Verified** column. The results show that 3 out of 4 objectives are falsified. For 3 of the 4 objectives, analysis found a signal condition which violates the properties, and therefore the safety 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	

For example, for the airspeed requirement, the analysis finds a condition in which the thrust reverser can deploy when the average airspeed is greater than 150 knots.

## Safety Properties/airspeed property/airspeed valid

### Summary

Model Item: [Safety Properties/airspeed property/airspeed valid](#)

Property: Assert

Status: Falsified

### Counterexample

Time	0	0.1
Step	1	2
sensorData.wowSensors	[ 1 1 1 1 ]	[ 1 1 1 1 ]
sensorData.wheelSpeedSensors	[ 10 10 ]	[ 10 10 ]
sensorData.airSpeedSensors	[ 149.75 150.25 ]	[ 150.25 150.75 ]
sensorData.throttlePositions	[ -0.5 0 -6 -5 ]	[ -6 -5.5 -6 -5.5 ]

The falsified result suggests that the control system algorithm is inadequately designed to avoid a condition in which the thrust reverser can deploy at speed.

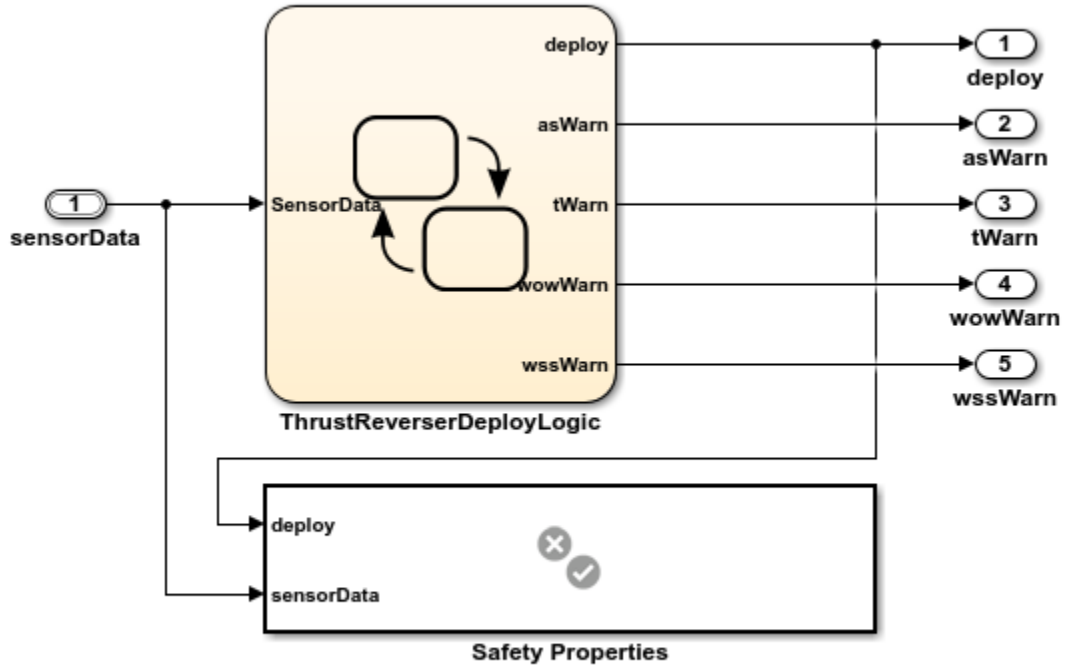
Next, investigate the design model by analyzing the dependencies of the property analysis.

### Analyze the Redesigned System

The redesigned system requires an intermediate deployment mode which delays the deployment of the thrust reverser. Open the `reqs_validation_property_proving_redesigned_model` model. Open the `thrustReversers` chart.

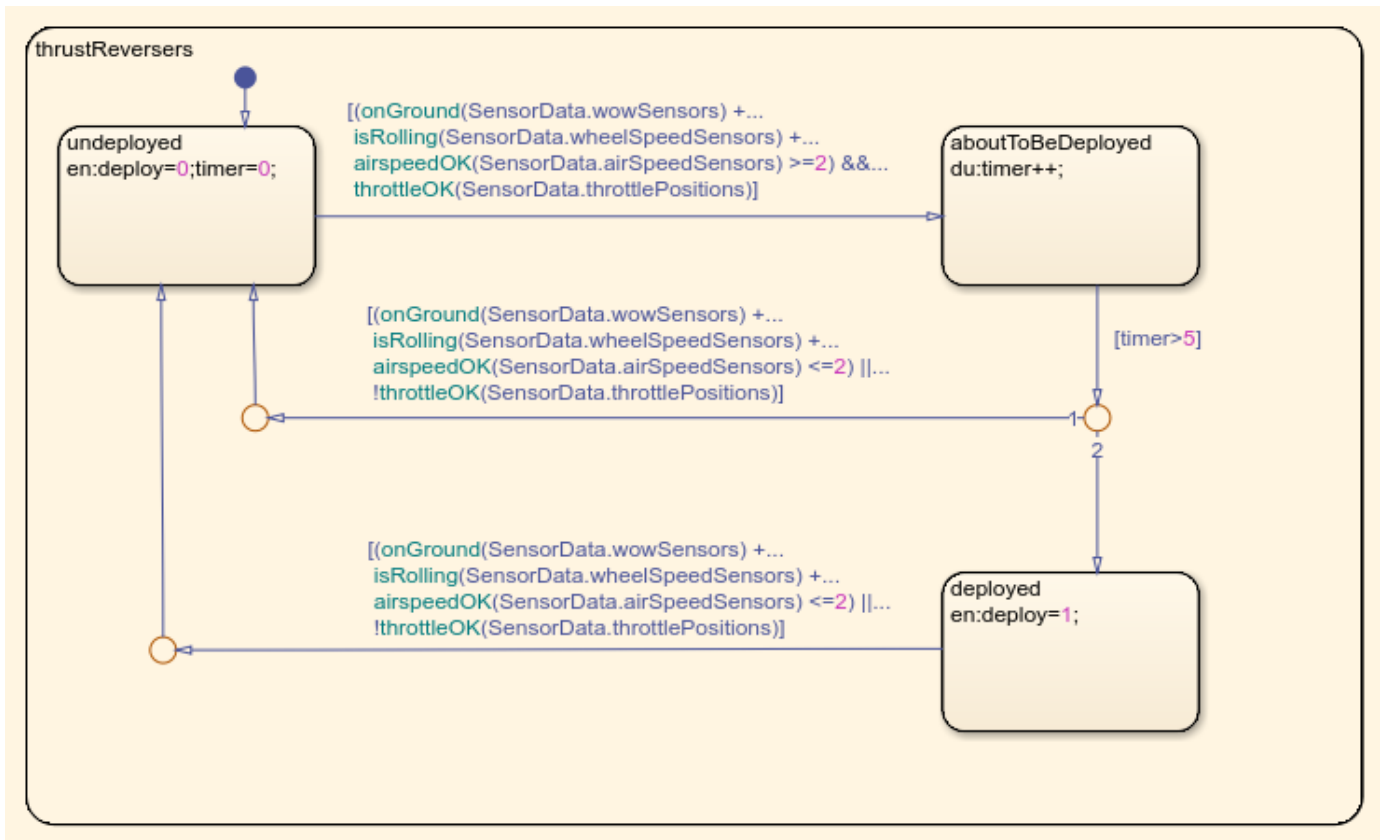
### Requirements Validation Using Property Analysis Example

Redesigned (Fixed) Model



Copyright 2006-2019 The MathWorks, Inc.





The additional design requirement states that the thrust reverser shall deploy after a 5 second pause. The `aboutToBeDeployed` state implements the requirement.

The screenshot displays a state machine diagram for the 'thrustReversers' component. It features three states: 'undeployed', 'aboutToBeDeployed', and 'deployed'. Transitions are labeled with guards and actions, such as 'en.deploy=1; timer=0;', 'aboutToBeDeployed du.timer++;', and 'en.deploy=1;'. Guards include conditions like '[[onGround(SensorData.wowSensors) +... isRolling(SensorData.wheelSpeedSensors) +... airspeedOK(SensorData.airSpeedSensors) >=2] &&... throttleOK(SensorData.throttlePositions)]'.

Below the diagram is a 'Requirements' editor window showing a table of requirements:

Index	ID	Summary
1	R1	Thrust Reverser Deployment Conditions
2	R14	Thrust Reverser Deployment Intermediate State
3	R2	Thrust Reverser Undeployment Conditions
4	R3	Aircraft "on ground" Conditions
5	R4	WOW Warning

To the right of the requirements editor is a 'Requirement: R14' details panel. It shows the requirement is 'Functional', has an 'Index' of 2, and a 'Summary' of 'Reverser Deployment Intermediate State'. The description states: 'The thrust reverser shall deploy after a 5 second pause.'

Before running a property proving analysis on the redesigned model, create links to the safety properties that belong to the redesigned model. Delete the requirement links from `thrust_reverser_safety_requirements.slreqx` that link to the safety properties in the original model and recreate them for the redesigned model.

Run the property proving analysis by double-clicking the **Run** button. View the results in the Requirements Editor.

Index	ID	Summary	Verified
1	R1.1	Airspeed Condition	
2	R1.2	WOW Condition	
3	R1.3	Throttle Condition	
4	R1.4	Wheelspeed Condition	

The results show that all the properties are valid.

## 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.

### Workflow

You have a MATLAB script called `runmytests.m` which 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:

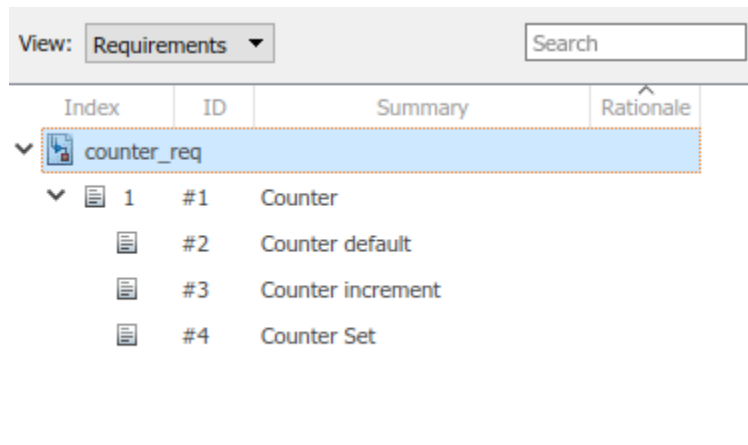
- 1 You start with opening the Requirements set `counter_req.slreqx`.
- 2 You create and register the Linktype using the API.
- 3 You create the link.
- 4 You view the Verification Status.

### Section 1 : Open the Requirements Set

Open the requirements file `counter_req.slreqx` in the Requirements Editor.

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

This will open the requirements set '`counter_req.slreqx`'.



### Section 2 : Create and Register Custom Linktype

The domain registration needed for this example is written in '`linktype_mymscripttap.m`'. The template file for domain registrations is available at: `matlabroot/toolbox/slrequirements/linktype_examples/linktype_TEMPLATE.m`. Take a look at the implementation of `GetResultFcn` in the domain registration file:

```
edit linktype_mymscripttap;
```

Register the custom linktype:

```
rmi register linktype_mymscripttap;
```

If the command returns any warning, then you must unregister the file and follow the command again. Unregister the file by entering: `rmi unregister linktype_mymscripttap`

#### Section 3: Create the Link

Make the struct containing properties of the external test. Follow these steps to create the link:

```
externalSource.id = 'counterStartsAtZero';  
externalSource.artifact = 'runmytests.m';  
externalSource.domain = 'linktype_mymscripttap';
```

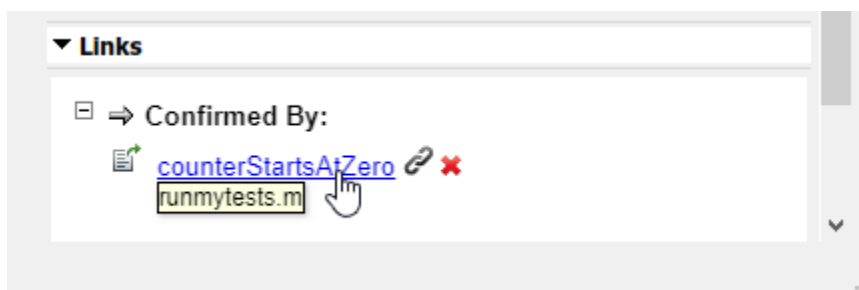
Find the requirement related to the link by typing:

```
requirement = reqSet.find('Type', 'Requirement', 'SID', 2);
```

Create the link by entering:

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

This creates the link as test case *counterStartsAtZero* to the requirement 'SID'. In Requirements Editor, the link appears in the Links Confirmed by section.



#### Section 4: View the Verification Status

To update the verification status for the requirements set, type:

```
reqSet.updateVerificationStatus;
```

Fetch the verification status for the requirement:

```
status = reqSet.getVerificationStatus;
```

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

The screenshot displays the Requirements Editor interface. On the left, a table lists requirements under the 'counter\_req' category:

Index	ID	Summary	Verified
1	#1	Counter	<input checked="" type="checkbox"/>
1.1	#2	Counter default	<input checked="" type="checkbox"/>
1.2	#3	Counter increment	<input type="checkbox"/>
1.3	#4	Counter Set	<input type="checkbox"/>

The right-hand pane shows the details for requirement 1.1:

- Type: Functional
- Index: 1.1
- Custom ID: #2
- Summary: Counter default
- Description: Counter default should be 0
- Keywords: (empty)
- Revision information: (empty)
- Links: Confirmed By: [counterStartsAtZero](#) (with a green checkmark icon) and a red X icon. Below the link is a tooltip that reads "Passed - counterStartsAtZero".

The verification status for the requirements for the counterStartsAtZero is fully verified. The Requirements Editor shows the overall verification statuses for all the other requirements links associated with counter\_req.slreqx.

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

Click on the Refresh button if you are unable to see the verification status for the requirements in the Requirements Editor. The verification status shows that out of three tests, one test passed.

The screenshot displays a requirements management interface. On the left, a tree view shows a hierarchy of requirements under 'counter\_req'. A table lists the following requirements:

Index	ID	Summary	Verified
1	#1	Counter	<input checked="" type="checkbox"/>
1.1	#2	Counter default	<input checked="" type="checkbox"/>
1.2	#3	Counter increment	<input type="checkbox"/>
1.3	#4	Counter Set	<input type="checkbox"/>

The 'Counter default' requirement (ID #2) is selected, and its details are shown in the right-hand pane. The details include:

- Type: Functional
- Index: 1.1
- Custom ID: #2
- Summary: Counter default
- Statistics: Passed: 1, Justified: 0, Failed: 0, Unexecuted: 0, None 2, Total: 3
- Text content: Counter default should be 0
- Keywords: (empty field)
- Revision information: (collapsed)
- Links: Confirmed By: [counterStartsAtZero](#) ✓

Clear the requirements once the simulation is completed.

## Integrating Results from an External Result file

In this example, you link a requirement to a test result file that is in Excel Format. The verification status in the Simulink Requirements Editor reflects the test results. The external results file is 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

### Workflow:

Suppose you want to update the verification information for the `counterSetsValue` test case based on the Excel status log. Follow these steps to create and verify links to the result file:

- 1 You start with opening the Requirements set `counter_req.slreqx'`.
- 2 You create and register the Linktype using the API.
- 3 You create the link.
- 4 You view the Verification Status.

### Section 1: Open the Requirements Set

Before creating links, open the requirements file containing the requirements.

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

This will open the requirements set `'counter_req.slreqx'`.

View:	Requirements	Search	
Index	ID	Summary	Rationale
▼	counter_req		
▼	1	#1 Counter	
		#2 Counter default	
		#3 Counter increment	
		#4 Counter Set	

## Section 2: Create and Register Custom Linktype

The domain registration needed for this example is written in 'linktype\_myexcelresults.m'. The template file for domain registrations is available at: matlabroot/toolbox/slrequirements/linktype\_examples/linktype\_TEMPLATE.m. Take a look at the implementation of GetResultFcn in the domain registration file:

```
edit linktype_myexcelresults;
```

Register the custom linktype:

```
rmi register linktype_myexcelresults;
```

Warning: Link type "'linktype\_myexcelresults'" is already registered

If the command returns any warning, then you must unregister the file and follow the command again. Unregister the file by entering: rmi unregister linktype\_myexcelresults

## Section 3: Create the Link

Make the struct containing properties of the external test. Follow these steps to create the link:

```
externalSource.id = 'counterSetsValue';
externalSource.artifact = 'results.xlsx';
externalSource.domain = 'linktype_myexcelresults';
```

Find the requirement related to the link by typing:

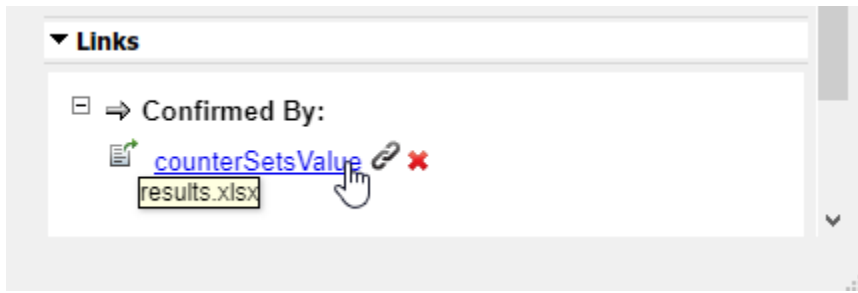
```
requirement = reqSet.find('Type', 'Requirement', 'SID', 2);
```

Create the link by entering:

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

This creates the link as test case counterSetsValue to the requirement 'SID'. In Requirements Editor, the link appears in the Links Confirmed by section.





#### Section 4: View the Verification Status

To update the verification status for the requirements set, type:

```
reqSet.updateVerificationStatus;
```

Fetch the verification status for the requirement:

```
status = reqSet.getVerificationStatus;
```

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

Index	ID	Summary	Verified
counter_req			<input checked="" type="checkbox"/>
1	#1	Counter	<input checked="" type="checkbox"/>
1.1	#2	Counter default	<input type="checkbox"/>
1.2	#3	Counter increment	<input type="checkbox"/>
1.3	#4	Counter Set	<input checked="" type="checkbox"/>

View: Requirements Search

Type: Functional

Index: 1.3

Custom ID: #4

Summary: Counter Set

Description Rationale

Keywords:

Revision information:

Links

Confirmed By:

counterSetsValue

Passed

The verification status for the requirements for the `counterSetsValue` is fully verified. The Requirements Editor shows the overall verification statuses for all the other requirements links associated with `counter_req.slreqx`.

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

If unable to see the verification status, click on the Refresh button to see the verification status for the requirements in the Requirements Editor. The verification status shows that out of three tests, one test passed.

The screenshot displays the Requirements Editor interface. On the left, a tree view shows a hierarchy starting with 'counter\_req', which is expanded to show a table of requirements. The table has columns for Index, ID, Summary, Rationale, and Verified. Requirement 1.1 is selected, and its details are shown in a right-hand pane. The 'Verified' column for requirement 1.1 shows a green progress bar and a tooltip indicating 'Passed: 1, Justified: 0, Failed: 0, Unexecuted: 0, None 2, Total: 3'. The right-hand pane shows the requirement's details, including its index (1.1), custom ID (#2), and summary (Counter default). Below the details, there is a text area containing the requirement text: 'Counter default should be 0'. At the bottom of the pane, under the 'Links' section, there is a 'Confirmed By:' field with a link to 'counterSetsValue' and a green checkmark icon.

Index	ID	Summary	Rationale	Verified
counter_req				
1	#1	Counter		
1.1	#2	Counter default		Passed: 1, Justified: 0, Failed: 0, Unexecuted: 0, None 2, Total: 3
1.2	#3	Counter increment		
1.3	#4	Counter Set		

Clear the requirements once the simulation is completed.

## Integrating results from a custom authored MUnit script as a test

You can integrate the results from a MATLAB xml Unit test by linking to a test script. In this example, you link a requirement to a MATLAB Unit test case script. The verification status in the Simulink Requirements Editor reflects the test results.

### Workflow

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`. 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 `testCounterStartsAtZero` in `CounterTests.m`:

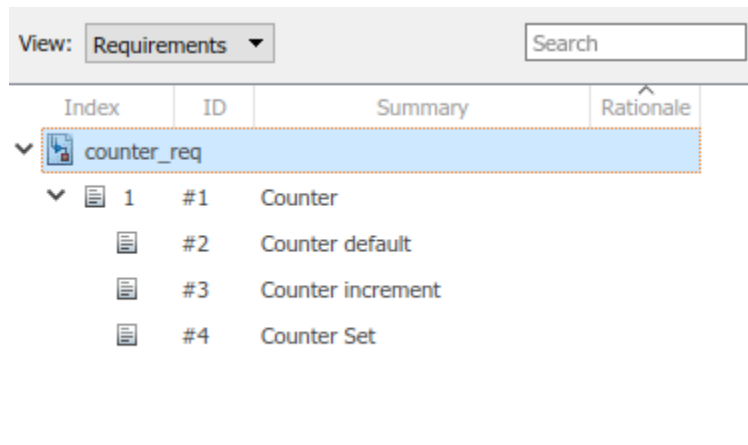
- 1 You start with opening the Requirements set `counter_req.slreqx`.
- 2 You create and register the Linktype using the API.
- 3 You create the link.
- 4 You view the Verification Status.

### Section 1: Open the Requirements Set

Open the requirements file `counter_req.slreqx` in the Requirements Editor.

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

This will open the requirements set '`counter_req.slreqx`'.



### Section 2: Create and Register Custom Linktype

The domain registration needed for this example is written in '`linktype_mymljunitresults.m`'. The template file for domain registrations is available at: `matlabroot/toolbox/slrequirements/linktype_examples/linktype_TEMPLATE.m`. Take a look at the implementation of `GetResultFcn` in the domain registration file:

```
edit linktype_mymljunitresults;
```

Register the custom linktype:

```
rmi register linktype_mymljunitresults;
```

If the command returns any warning, then you must unregister the file and follow the command again. Unregister the file by entering: `rmi unregister linktype_mymljunitresults`

#### Section 3: Create the Link

Make the struct containing properties of the external test. Follow these steps to create the link:

```
externalSource.id = 'testCounterStartsAtZero';  
externalSource.artifact = 'counterTests.m';  
externalSource.domain = 'linktype_mymljunitresults';
```

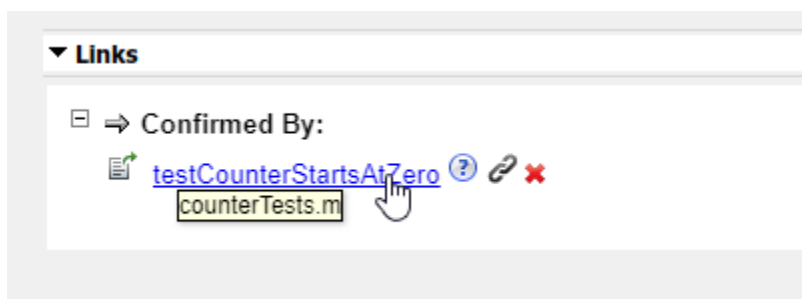
Find the requirement related to the link by typing:

```
requirement = reqSet.find('Type', 'Requirement', 'SID', 2);
```

Create the link by entering:

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

This creates the link as test case `testCounterStartsAtZero` to the requirement 'SID'. In Requirements Editor, the link appears in the `Links Confirmed by` section.



#### Section 4: View the Verification Status

To update the verification status for the requirements set, type:

```
reqSet.updateVerificationStatus;
```

Fetch the verification status for the requirement:

```
status = reqSet.getVerificationStatus;
```

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

The screenshot displays the Requirements Editor interface. On the left, a table lists requirements under the 'counter\_req' category:

Index	ID	Summary	Verified
1	#1	Counter	<input type="checkbox"/>
1.1	#2	Counter default	<input checked="" type="checkbox"/>
1.2	#3	Counter increment	<input type="checkbox"/>
1.3	#4	Counter Set	<input type="checkbox"/>

The right-hand pane shows details for requirement 1.1:

- Type: Functional
- Index: 1.1
- Custom ID: #2
- Summary: Counter default
- Description: Counter default should be 0
- Keywords: (empty)
- Revision information: (empty)
- Links: Confirmed By: [testCounterStartsAtZero](#)  Passed

The verification status for the requirements for the `testCounterStartsAtZero` is fully verified. The Requirements Editor shows the overall verification statuses for all the other requirements links associated with `counter_req.slreqx`.

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

If you are unable to see the verification status, click on the Refresh button to see the verification status for the requirements in the Requirements Editor. The verification status shows that out of three tests, one test passed.

The screenshot displays a requirements management interface. On the left, a table lists requirements under the 'counter\_req' category:

Index	ID	Summary	Verified
1	#1	Counter	<input checked="" type="checkbox"/>
1.1	#2	Counter default	<input checked="" type="checkbox"/>
1.2	#3	Counter increment	<input type="checkbox"/>
1.3	#4	Counter Set	<input type="checkbox"/>

The right-hand side of the interface provides details for the selected requirement (Index 1.1, ID #2, Summary: Counter default). It includes a 'Type' dropdown set to 'Functional', an 'Index' field with '1.1', a 'Custom ID' field with '#2', and a 'Summary' field with 'Counter default'. A status bar indicates 'Passed: 1, Justified: 0, Failed: 0, Unexecuted: 0, None 2, Total: 3'. Below this is a rich text editor with the text 'Counter default should be 0'. Further down, there is a 'Keywords' field, a 'Revision information' section, and a 'Links' section containing a 'Confirmed By:' field with a link to 'testCounterStartsAtZero' and a green checkmark icon.

Clear the requirements once the simulation is completed.

# Change Tracking and Team-Based Workflows

---

- “Requirements-Based Development in Projects” on page 4-2
- “Track Changes to Requirements Links” on page 4-3
- “Compare Requirements Sets” on page 4-6
- “Compare Link Sets” on page 4-7
- “Report Requirements Information” on page 4-8

## 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?” (Simulink).

### 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	●	✓	
requirements	·	✓	
functional_reqs.slreqx	●	✓	
tests	·	✓	
baseline_test.mldatx	●	✓	Test
baseline_test.slmx	●	✓	
baseline_test_data.xlsx	●	✓	Test
baseline_test_result_criteria.xlsx	●	✓	
simulation_tests.mldatx	●	✓	Test
simulation_tests.slmx	●	✓	
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 Requirements Links

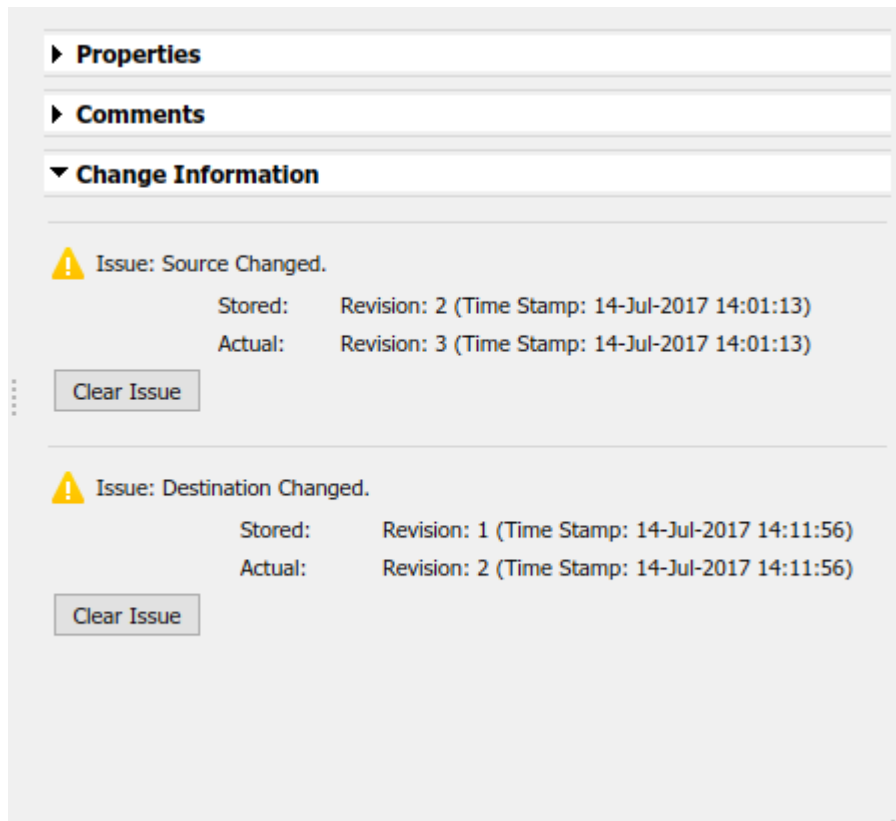
After you “Author Requirements in Simulink” on page 1-2 and create links between design elements and your requirements, Simulink Requirements tracks the requirements links and detects when link artifacts (source or destination) or requirements change. In the Requirements Editor, changes to requirements links are highlighted in red. You can then resolve link issues or clear links changes that have no impact on the requirement status. Track change information from the **Links** view of the Requirements Editor.

### Enable Change Tracking for Requirements Links

To view the change information for your requirements links:

- 1 Open the Requirements Editor. In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Requirements Editor**.
- 2 Open a requirements set.
- 3 From the **View** drop-down list, select **Links**.
- 4 To enable change tracking from the Requirements Editor, select **Display > Change Information**.

The **Links** view of the Requirements Editor highlights any links with revision mismatch issues in red.



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**.

The **Change Information** section for each link details any requirement link change issues. Simulink Requirements reports any source or destination requirements that have more recent changes than when you established the requirement link. Simulink Requirements considers the revision information and timestamp for a requirement item when the link was created and the current revision information and timestamp for the requirement item.

### Review Requirements Change Issues from the Requirements View

You can view change issues associated with a particular requirement item from the Requirements view. After you “Enable Change Tracking for Requirements Links” on page 4-3, select a requirement item from the **Requirements** view. Any requirements items with requirement link change issues are highlighted in red. The relevant requirement link with a change issue is also highlighted in red in the Links section of the Requirements Editor.

The screenshot shows the Simulink Requirements Editor interface. The 'Requirements' view is active, displaying a table of requirements. The requirement with ID #12, 'Increment Short Switch Detection', is highlighted in red, indicating a change issue. The 'Links' section on the right shows a list of links, with the link '3.5 TargetSpeedIncrement Target Speed Increment' highlighted in red. A tooltip 'Show in Links View' is visible over the link.

Index	ID	Summary
1.8	#11	Increment Switch Detection
1.8.3	#14	Intermediate state
1.8.1	#12	Increment Short Switch Detection
1.8.2	#13	Increment Long Switch Detection
1.6	#9	Enable Switch Detection
1.9	#15	Decrement Switch Detection
1.4	#7	Cancel Switch Detection
1.2	#3	Avoid reprecading commands
2	#19	Cruise Control Mode
3	#37	Calculate Target Speed and Throttle ...

To view the link change issue, and then “Resolve Change Issues for Requirement Links” on page 4-4, click the link icon to the right of requirement link.

### Resolve Change Issues for Requirement Links

If you enable change tracking for requirements links, changes to your requirements that cause link issues are highlighted in red in the **Links** view of the Requirements Editor. If a change has no impact, you can clear the issue to update to the latest revision number for the requirement. In the Links view of the Requirements Editor, select the link with a link change issue. In the **Change Information** section, click **Clear Issue**. To clear all change issues for an entire link set, right-click the link set and select **Clear All Change Issues**.

If the link issue affects the status of your requirements, you can change the model, the requirements, the test cases, or the links themselves to resolve the discrepancy between the requirements and your model and testing.

### Add Comments to Links

Whenever you resolve link issues, it is good practice to add a comment to the link describing the action that you took. Each link has a **Comments** section. To add a comment:

- 1 In the **Links** view of the Requirements Editor, select the link.
- 2 In the Comments section, select **Add Comment**.

## Considerations for Using Links Change Tracking

Change tracking information is updated only when you enable change tracking or when you manually refresh the change tracking information. To refresh the change tracking information manually:

- From the Requirements Editor, select **Analysis > Refresh Change Information** .
- From the Requirements Perspective, or from the Requirements Editor, click the **Refresh** button



Simulink Requirements provides change tracking information only for link sources that are resolved when you view the change information.

You can resolve some links by using the Requirements Editor.

- 1 In the **Links** view of the Requirements Editor, select the link with an unresolved link item. The Change Information section shows **Unresolved Link Item(s)**.
- 2 To open the link source, click the **Source** link in the **Properties** section.

When the link model opens, the link source is loaded and the Change Information section displays a resolved link. For more information, see “Review Requirement Links” on page 2-13.

## See Also

### More About

- “Create a Project from a Model” (Simulink)

## Compare Requirements Sets

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

### Compare Two .slreqx Simulink Requirements Sets

If you have two versions of a .slreqx Simulink requirements set file, use the Simulink “Compare Revisions” (Simulink) 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” (Simulink).

## Compare Link Sets

If you have two versions of a .slmx Simulink link set file, use the Simulink “Compare Revisions” (Simulink) 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  Change Information

Empty Sections

Group Links By:

Artifact  Link Type

Generate Report Cancel Help

To create a report by using the Report Generation Options dialog box:

- 1 Right-click a requirement set in the Requirements Editor or Requirements Browser, and select **Generate Report**.

To create a report with multiple requirements sets, select **Report > Generate Report** in the Requirements Editor menu.

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. **Ctrl**+click a link to open the linked item.

#### ↔Implemented by

 Switch2

 Enumerated Constant2

#### ↔Derived from

 3.3\_Activatingcruisecon Activating cruise control

 3.4\_Deactivatingcruise Deactivating cruise control

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`



# Requirements Management Interface Setup

---

- “Configure RMI 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

## Configure RMI for Interaction with Microsoft Office and IBM Rational DOORS

The Requirements Management Interface (RMI) communicates with external tools such as Microsoft Office and IBM Rational DOORS so that you can establish links between requirements and Simulink model elements.

Configure the RMI to:

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

### Configure RMI 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 RMI 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
```
- 3 Press N to skip the ActiveX Automation Server setup.  
  
The interface setup utility opens.
- 4 Verify the path to your IBM Rational DOORS installation. Press 1 to configure the software to communicate with MATLAB.
- 5 If the DOORS installation was not detected in the previous step, press 2 to enter the installation folder.

---

**Note** You can directly access the IBM Rational DOORS - MATLAB Interface setup utility. At the command prompt, enter:

```
rmi setup doors
```

---

## Configure RMI for IBM Rational DOORS Next Generation

- 1 At the command prompt, enter:  

```
rmi setup
```
- 2 You do not need to set up the ActiveX Automation Server because it is not required for requirements linking with IBM Rational DOORS Next Generation. Skip this step by pressing N.
- 3 If you have IBM Rational DOORS installed, you are prompted to select the installation path to configure for communication with MATLAB. If you are using RMI with IBM Rational DOORS Next Generation only, you do not need to configure IBM Rational DOORS. Skip this step by pressing 3.
- 4 Configure RMI by pressing Y.
- 5 Complete the setup by entering the IBM Rational DOORS Next Generation **Server Address** and the **Port Number** in the next prompt.

### Install the Simulink Requirements Widget in IBM Rational DOORS Next Generation

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

- 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 Rational DOORS Next Generation installation. The location of this folder depends on your server version.
- 3 After copying the `dngsllink` folder to your server, add the **Simulink Requirements** widget to the **Mini Dashboard** in DOORSNext Generation.
- 4 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 version 6.0.6 installed, the `extensions` subfolder is located in: `JazzTeamServer_6.0.06\server\liberty\servers\clm\dropins\war\extensions`. The corresponding URL for adding the widget is: `https://JAZZSERVERNAME:9443/extensions/dngsllink_config/dngsllink_config.xml`.

## Requirements Link Storage

The Requirements Management Interface (RMI) stores the requirements links associated with your Simulink models in two modes - internal and external. When you create links from a model to requirements, by default, the Requirements Management Interface (RMI) stores the link information in an external `.slmx` file in the same folder as the model. External storage does not modify your model when creating or modifying 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 requirements links 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 <b>Apps</b> tab, click <b>Requirements Manager</b> . In the <b>Requirements</b> tab, click <b>Save All</b> .	Save the requirements links in an external file using a file name that you specify. The model itself is not saved.
In the <b>Apps</b> tab, click <b>Requirements Manager</b> . In the <b>Requirements</b> tab, click <b>Save Links Only</b> .	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.

Select...	To...
In the <b>Simulation</b> tab, click <b>Save</b> .	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 <b>Simulation</b> tab, <b>Save &gt; Save As</b>	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 deleted the internally stored links 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, click **Requirements Manager**. In the **Requirements** tab, click **Save All**.

The Select a file to store RMI data dialog box prompts you to save the file with the default name *model\_name.slmx*.

- 3 Accept the default name, or enter a different file name if required.
- 4 Click **Save**.

---

**Note** Use the default name for externally stored requirements. For more information about this recommendation, see “Guidelines for External Storage of Requirements Links” on page 5-6.

---

## 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 only externally stored requirements links.

- 2 Make sure the right set of requirements links are loaded from the external file.
- 3 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.

- 4 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.

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 resave the model with a new name or in a different location, in the **Simulation** tab, click **Save**. Selecting this option causes the RMI to resave 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.

## 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"> <li>• <b>Named item</b> — 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.</li> <li>• <b>Search text</b> — A search string. The RMI links to the first occurrence of that string in the document. This search is not case sensitive.</li> <li>• <b>Page/item number</b> — A page number. The RMI links to the top of the specified page.</li> </ul>
Excel 2003 or later	<ul style="list-style-type: none"> <li>• <b>Named item</b> — 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.</li> <li>• <b>Search text</b> — A search string. The RMI links to the first occurrence of that string in the workbook. This search is not case sensitive.</li> <li>• <b>Sheet range</b> — A cell location in a workbook: <ul style="list-style-type: none"> <li>• Cell number (A1, C13)</li> <li>• Range of cells (C5:D7)</li> <li>• Range of cells on another worksheet (Sheet1!A1:B4)</li> </ul> </li> </ul> <p>The RMI links to that cell or cells.</p>
IBM Rational DOORS	<b>Page/item number</b> — The unique numeric ID of the target DOORS object. The RMI links to that object.
Text	<ul style="list-style-type: none"> <li>• <b>Search text</b> — A search string. The RMI links to the first occurrence of that string within the document. This search is not case sensitive.</li> <li>• <b>Line number</b> — A line number. The RMI links to the beginning of that line.</li> </ul>
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>&lt;A NAME=valve_timing&gt; ...contents... &lt;/A&gt;</pre> <p>in the <b>Location</b> field, enter <code>valve_timing</code> or, from the document index, choose the anchor name.</p> <p>Select the <b>Document Index</b> tab in the “Outgoing Links Editor” on page 10-6 to see available anchors in an HTML file.</p>



<b>Requirements Document Type</b>	<b>Location Options</b>
Web browser URL	<p>The RMI can link to a URL location. In the <b>Document</b> field, type the URL string. When you click the link, the document opens in a Web browser:</p> <ul style="list-style-type: none"><li>• <b>Named item</b> — An anchor name. The RMI links to that location on the Web page at that URL.</li></ul>
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 11-18.

### 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:	
<b>Enabled applications</b>	Enable selection-based linking shortcuts to Microsoft Word, Excel, or DOORS applications.
<b>Document file reference</b>	Select type of file reference. For information on what settings to use, see “Document Path Storage” on page 11-34.
<b>Apply this keyword to new links</b>	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:	
<b>Modify destination for bidirectional linking</b>	Creates links both to and from selected link destination.
<b>Store absolute path to model file</b>	Select to store the absolute path to the Simulink model file.
<b>Use custom bitmap for navigation controls in documents</b>	Select and browse for your bitmap. You can use your own bitmap file to control the appearance of navigation links in your document.
<b>Use ActiveX buttons in Word and Excel (backward compatibility)</b>	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:

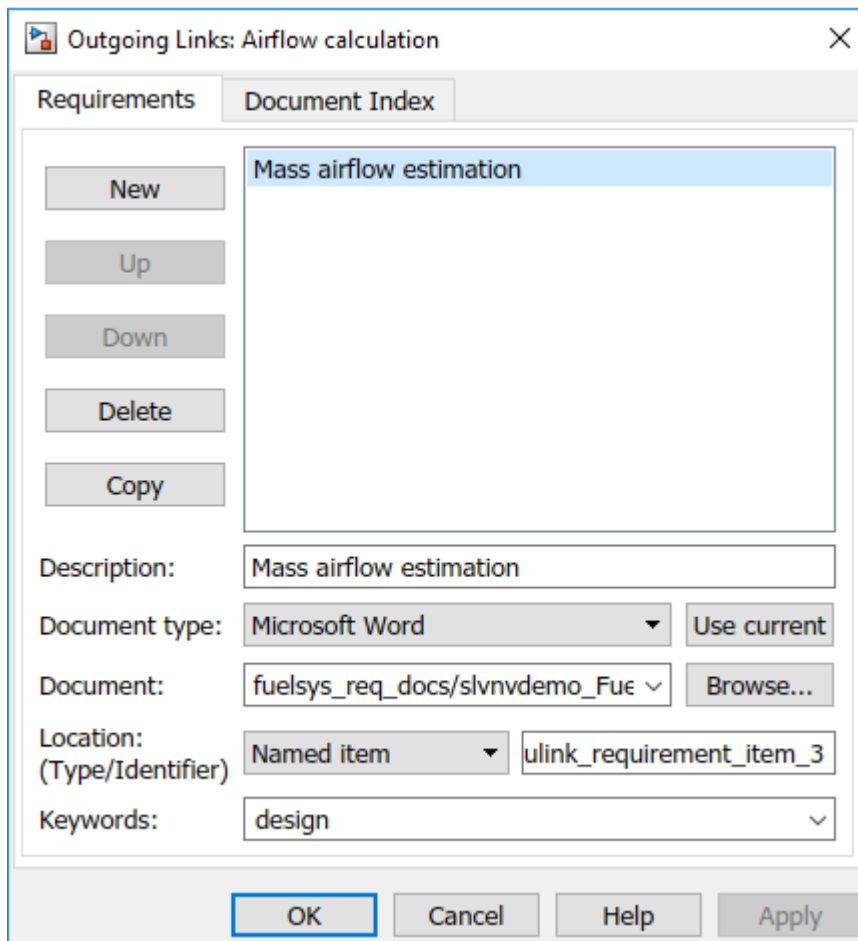
- Highlight or report only those requirements that have a specific user tag.
- Highlight or report only those requirements that have one of several user tags.
- Do not highlight and report requirements that have a specific user tag.

### 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:

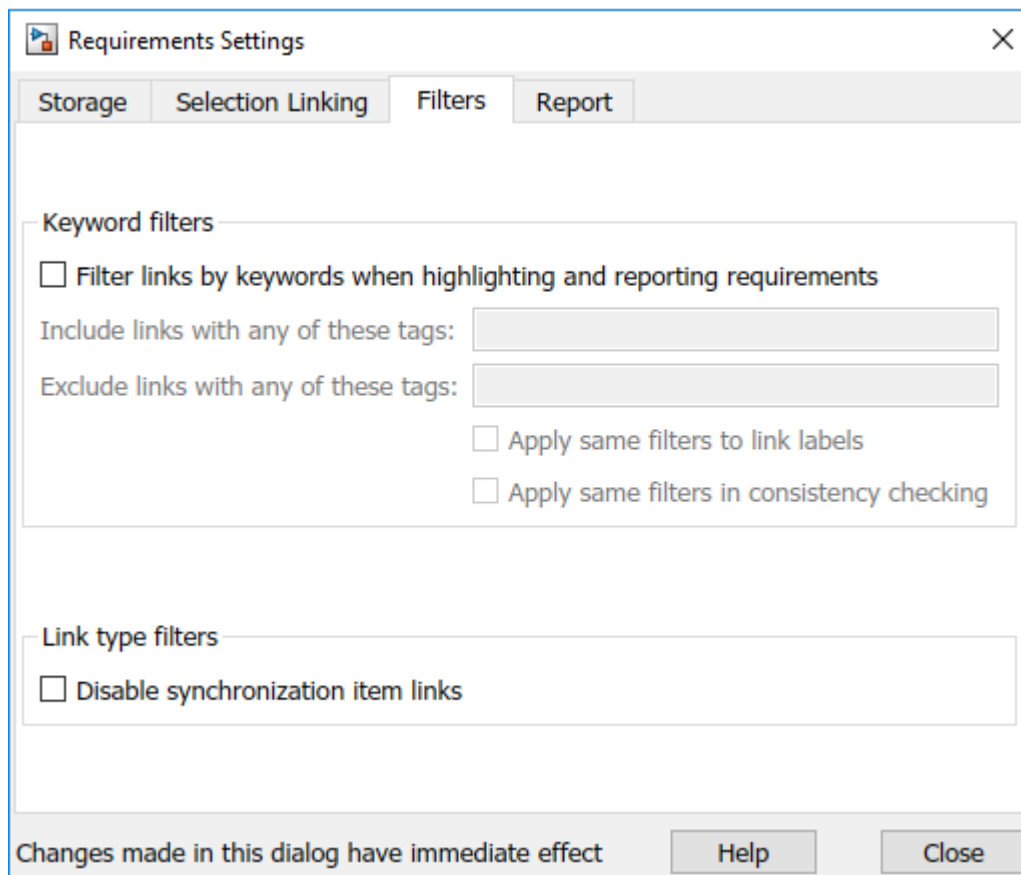
- Are not case sensitive.
- Can consist of multiple words. For example, if you enter `design requirement`, the entire phrase constitutes the user tag. Separate user tags with commas.

- 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**.

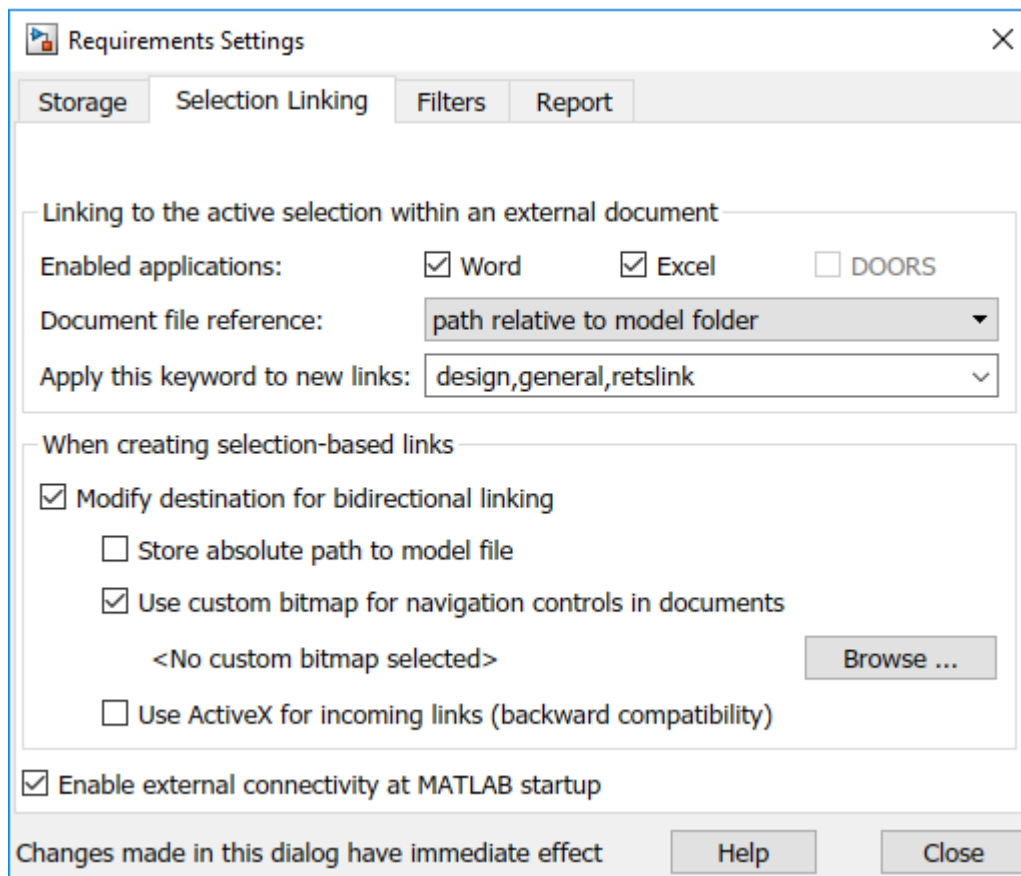
The report does not include information about objects whose requirements have the `test` user tag.

## 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.

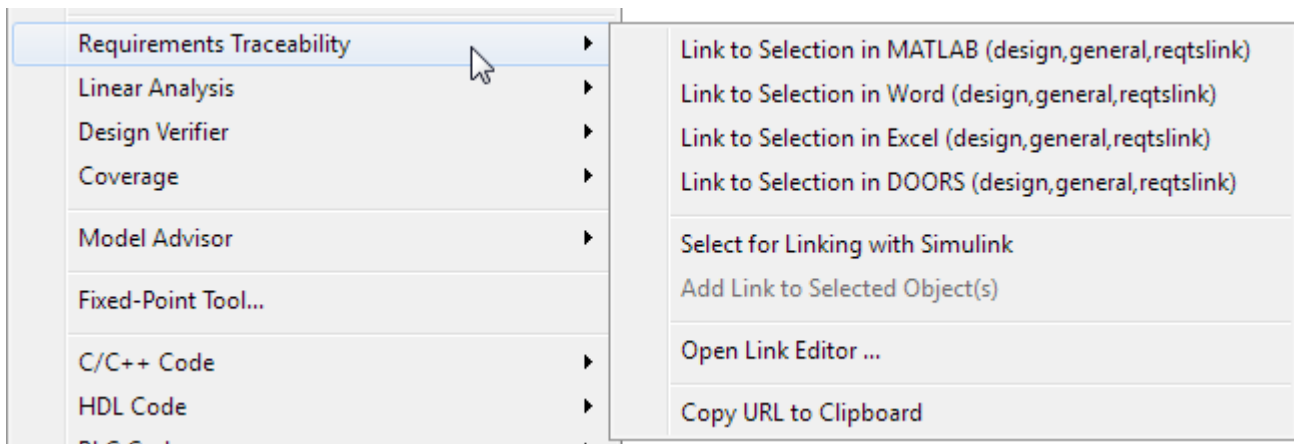


- 3 In the **Apply this keyword to new links** field, enter one or more user tags, separated by commas.

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
<b>Filter links by keyword when highlighting and reporting requirements</b>	Enables filtering for highlighting and reporting, based on specified user tags.
<b>Include links with any of these tags</b>	Includes information about requirements that have the specified user tags. Separate multiple user tags with commas.
<b>Exclude links with any of these tags</b>	Excludes information about requirements that have the specified user tags. Separate multiple user tags with commas or spaces.
<b>Apply same filters to link labels</b>	Disables link labels in context menus if one of the specified filters are satisfied, for example, if a requirement has a designated user tag.
<b>Apply same filters in consistency checking</b>	Includes or excludes requirements with specified user tags when running a consistency check between a model and its associated requirements documents.
<b>Under Link type filters, Disable synchronization item links in context menus</b>	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.





# Microsoft Office Traceability

---

- “Link to Requirements in Microsoft Word Documents” on page 6-2
- “Link to Requirements in Excel Workbooks” on page 6-6
- “Navigate to Requirements in Microsoft Office Documents from Simulink” on page 6-9

## Link to Requirements in Microsoft Word Documents

### Create Bookmarks in a Microsoft Word Requirements Document

You can identify requirements for linking by bookmarking your Word requirements documents. You use an existing bookmark when you create a link from a requirement.

Compared to creating bookmarks when you link, including bookmarks in your Word document before you link allows you to:

- Give a bookmark a meaningful name that represents the requirement content.
- Link to the requirements document using RMI, without modifying the requirements document.

---

**Note** When you link to an existing bookmark, navigating the link highlights the entire range of the existing bookmark. Therefore, when you create a bookmark for requirement linking, make sure to select only the document information relevant to your requirement.

---

If you have a requirements document containing bookmarks, follow these steps to create requirements links from your Simulink model to the bookmarks:

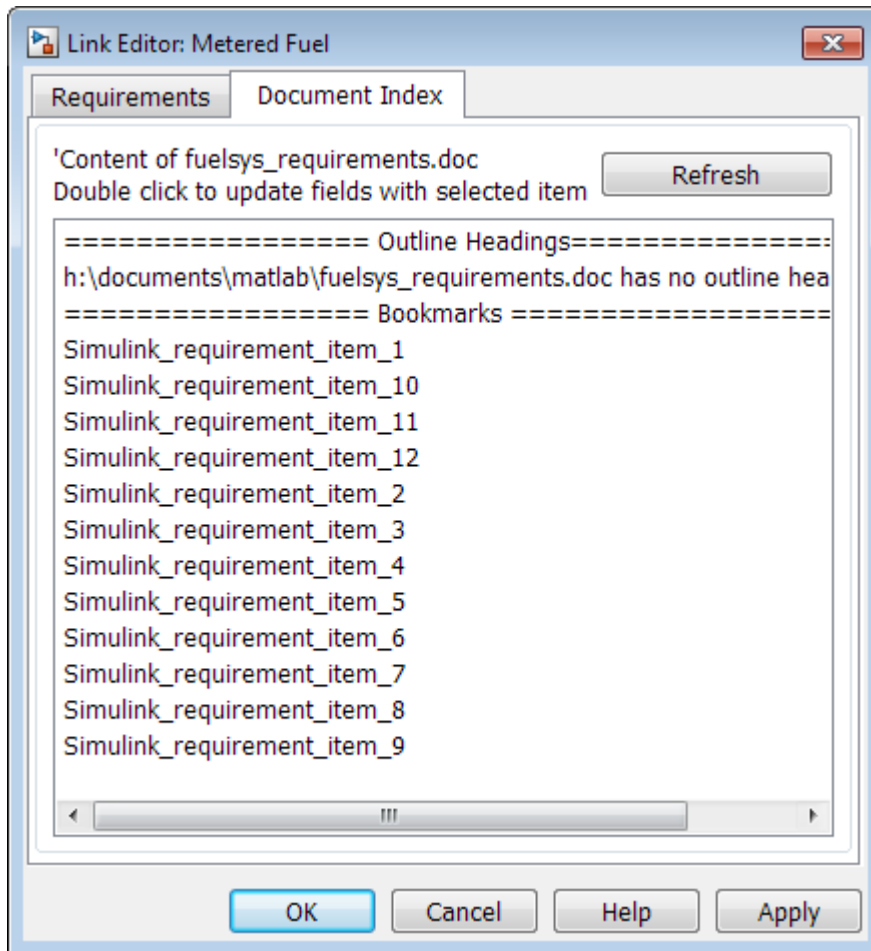
- 1 Open your model.
- 2 Open your Microsoft Word requirements document that contains bookmarks that identify requirements.
- 3 Right-click a block in the model that you want to link to a requirement and select **Requirements > Open Link Editor**

The Requirements Traceability Link Editor opens.

- 4 Click **New**.
- 5 Click **Browse** and navigate to the bookmarked document.
- 6 Open the document. RMI populates the **Document** and **Document type** fields.
- 7 Click the **Document Index** tab of the Link Editor.

The **Document Index** tab lists bookmarks in the requirements document and section headings (text that you have styled as **Heading 1**, **Heading 2**, etc.).

The document index lists the bookmarks in alphabetical order.



- 8 Select the bookmark that you want to link the block to and click **OK**.

RMI links from the block to the bookmark in the requirements document without modifying the document.

## Open the Example Model and Associated Requirements Document

This example describes how to create links from objects in a Simulink model to selected requirements text in a Word document.

Navigate from the model to the requirements document:

- 1 Open the example model:

```
slvndemo_fuelsys_officereq
```

- 2 Open a requirements document associated with that model:

```
rmi('view','slvndemo_fuelsys_officereq',1);
```

Keep the example model and the requirements document open.

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

Create a link from the Airflow calculation subsystem in the `slvnvdemo_fuelsys_officereq` model to selected text in the requirements document:

- 1 In `slvnvdemo_FuelSys_DesignDescription.docx`, find the section titled **2.2 Determination of pumping efficiency**.
- 2 Select the header text.
- 3 Open the example model:  
`slvnvdemo_fuelsys_officereq`
- 4 In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, select **Link Settings > Linking Options**. The Requirements Settings dialog box opens.
- 5 On the **Selection Linking** tab of the Requirements Settings dialog box:
  - Set the **Document file reference** option to `path relative to model folder`.
  - Enable **Modify destination for bidirectional linking**.

When you select this option, every time you create a selection-based link from a model object to a requirement, the RMI inserts navigation objects at the designated location.

For more information about the settings, see “Requirements Settings” on page 5-10.

- 6 Double-click the fuel rate controller subsystem to open it.
- 7 Open the Airflow calculation subsystem.
- 8 Right-click the Pumping Constant block and select **Requirements > Link to Selection in Word**.

The RMI inserts a bookmark at that location in the requirements document and assigns it a generic name, in this case, `Simulink_requirement_item_7`.

---

**Note** Run both MATLAB and Microsoft Word with the same privilege level. Running either as an administrator can prevent link creation.

---

- 9 To verify that the link was created, in the **Requirements** tab, click **Highlight Links**.  
The Pumping Constant block, and other blocks with requirements links, are highlighted.
- 10 To navigate to the link, right-click the Pumping Constant block and select **Requirements > 1. “Determination of pumping efficiency”**.

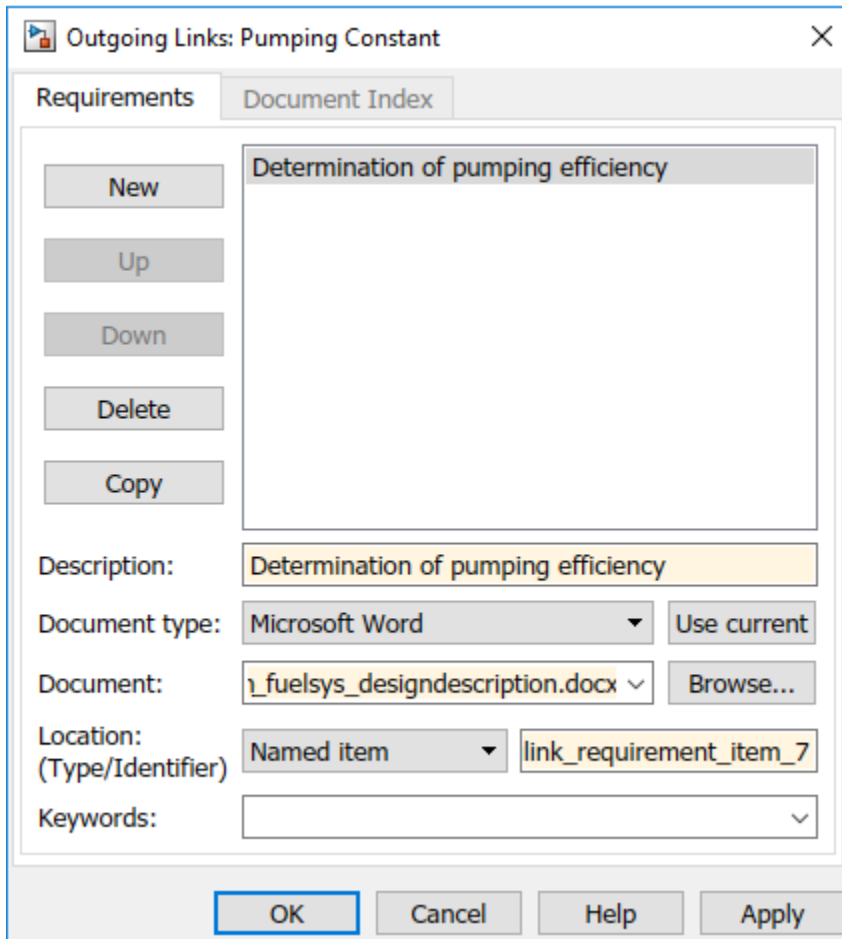
The section **2.2 Determination of pumping efficiency** is displayed, selected in the requirements document.

Keep the example model and the requirements document open.

### View Link Details

To view the details of the link that you just created, right-click the Pumping Constant block and select **Requirements > Open Outgoing Links dialog**.

The link editor dialog box opens.



The dialog box contains the following information for the new link:

- Link description: Determination of pumping efficiency, which matches the text of the requirements document.
- Document name: slvnvdemo\_FuelSys\_DesignDescription.docx.
- Document type: Microsoft Word.
- The type and identifier of the location in the requirements document. RMI bookmarks the Word document with a bookmark named Simulink\_requirement\_item\_7.

To avoid RMI modifying Word document when it creates links, bookmark the Word document before linking, as described in “Create Bookmarks in a Microsoft Word Requirements Document” on page 6-2.

- Keyword, a user-defined keyword. This link does not have a keyword. For more information about keywords, see “Filter Requirements with User Tags” on page 5-11.

## Link to Requirements in Excel Workbooks

### 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 8-8.

### 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 11-34.

- 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.

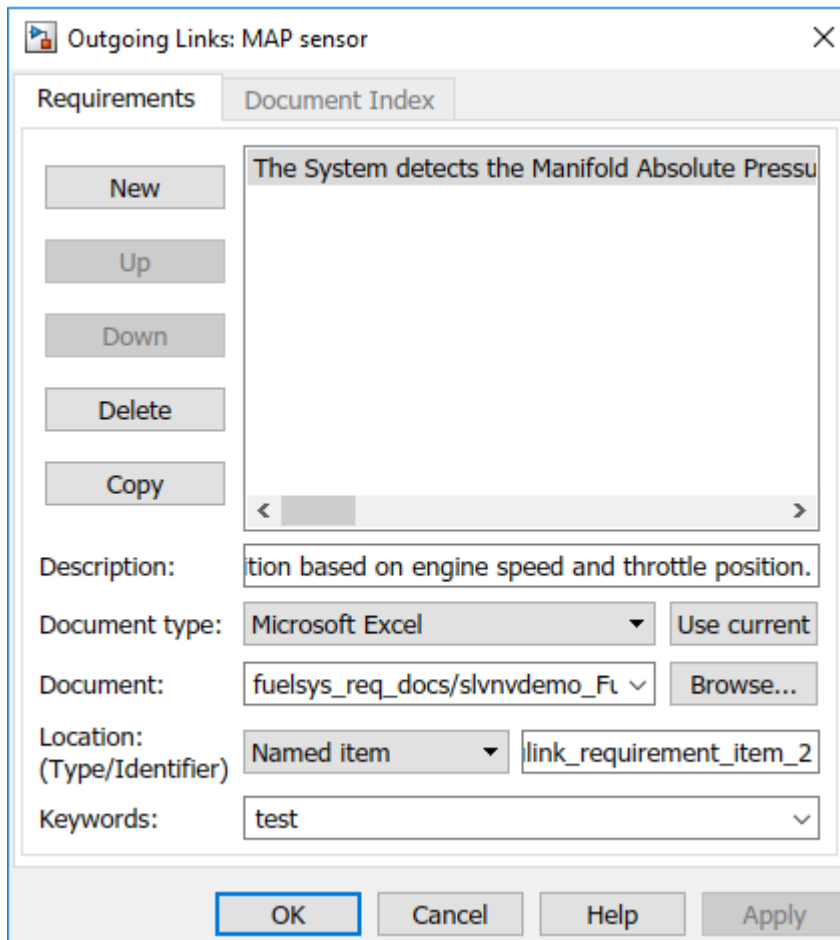
For information about which setting to use for your working environment, see “Document Path Storage” on page 11-34.

- 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

## Enable Linking from Microsoft Office Documents to Simulink Objects

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 also configure the RMI to insert a navigation object in a Microsoft Office 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. If you want to insert a navigation object into the requirements document when you create a link from a model object to a requirement, you must change the RMI's settings. The following tutorial uses the `slvndemo_fuelsys_officereq` example model to illustrate how to do this.

To enable linking from a Word or Excel document to the example model:

- 1 Open the model:

```
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.

---

- 2 In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, select **Link Settings > Linking Options**.

The Requirements Settings dialog box opens.

- 3 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.

- 4 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 6-9, the RMI can insert a navigation object into the requirements document.

- 1 Open the Word document:

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

- 2 Select the **Throttle Sensor** header.
- 3 In the slvnvdemo\_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 6-9.
- 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.**

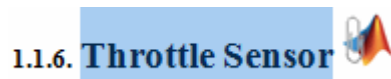
The next time you insert a navigation object into a requirements document, the RMI uses the file you selected.

## Navigate Between Microsoft Word Requirement and Model

In “Insert Navigation Objects in Microsoft Office Documents” on page 6-9, you created a link between a Microsoft Word 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.



**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 6-9.

---

**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.

# Requirements Traceability with IBM Rational DOORS

---

- “Configure Requirements Management Interface for IBM Rational DOORS Software” on page 7-2
- “Link with Requirements in DOORS Next Generation Project” on page 7-4
- “Requirements Traceability with IBM Rational DOORS Next Generation” on page 7-25
- “Navigate to Requirements in IBM Rational DOORS Databases from Simulink” on page 7-28
- “Synchronize Simulink Models with IBM Rational DOORS Databases by using Surrogate Modules” on page 7-32
- “Working with IBM Rational DOORS 9 Requirements” on page 7-43

## Configure Requirements Management Interface for IBM Rational DOORS Software

### Before You Begin

If you plan to use DOORS software with the RMI, make sure to install additional files to establish communication between the DOORS application and the Simulink software. Follow the instructions in “Configure RMI for Interaction with Microsoft Office and IBM Rational DOORS” on page 5-2.

### 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
```

## **Diagnose and Fix DXL Errors**

If you try to synchronize your Simulink model to a DOORS project, 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 RMI for Interaction with Microsoft Office and IBM Rational DOORS” on page 5-2, and restart the DOORS software.

## Link with Requirements in DOORS Next Generation Project

IBM® Rational® DOORS® Next Generation (DNG) is a requirements management tool for IBM Rational Jazz platform. Linking file-based MBD artifacts (Simulink blocks, Test Cases, Data Dictionary entries) with items managed by shared server and accessed via web client requires certain configuration steps before you can create new links. This example provides a step-by-step demonstration of DNG integration feature in Simulink Requirements.

### The Hard Way: Direct Links between Simulink and DNG

You can link *directly* with DNG artifacts, using the **Link to Selected Item(s) in DNG** context menu shortcut in **Requirements** menu. This requires considerable overhead, because you access DNG using the web browser client, and links require notification about which one is the "selected item". The following several sections of this example show you how to set up your environment for *direct* linking with DNG.

### Server-side configuration

Copy the Jazz server `dngsllink_config` subfolder in `MATLAB_INSTALL_DIR/toolbox/slrequirements/slrequirements/resources/` into the DNG 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 version 6.0.6, your extensions folder may be here:

```
C:\Program Files\IBM\JazzTeamServer_6.0.6\server\liberty\servers\clm\dropins\war\extensions
```

You may also need to "enable dropins" in DNG server configuration. Please see the DNG instructions for details:

<https://jazz.net/wiki/bin/view/Main/RMExtensionsHostingGuide605>

You need to edit `server.xml` file in the `C:\[JAZZ_INSTALL_DIR]\server\liberty\servers\clm` folder.

1. Open this file in a text editor, and locate this line:

```
<applicationMonitor dropinsEnabled='false' pollingRate='10s' updateTrigger='mbean' />
```

2. change `dropinsEnabled` to `'true'`.

3. Restart the server. Below is the screenshot from IBM's instruction:

- Before you start the server, you must edit a configuration file. From the root installation directory navigate to `c:\Programs\DOORSNG\server\liberty\servers\clm` and locate the `server.xml` file.
- Open this file in a text editor and locate this line

```
<applicationMonitor dropinsEnabled='false' pollingRate='10s' updateTrigger='mbean' />
```

Change `dropinsEnabled='false'` to `dropinsEnabled='true'` and save the file.

- Navigate to the `c:\Programs\DOORSNG\server\liberty\servers\clm` directory.
- Create a directory called `dropins`.
- In the `dropins` directory create another directory called `war`.
- In the `war` directory another directory called `extensions`. This should give you a directory structure like this:

```
c:\Programs\DOORSNG\server\liberty\servers\clm\dropins\war\extensions
```

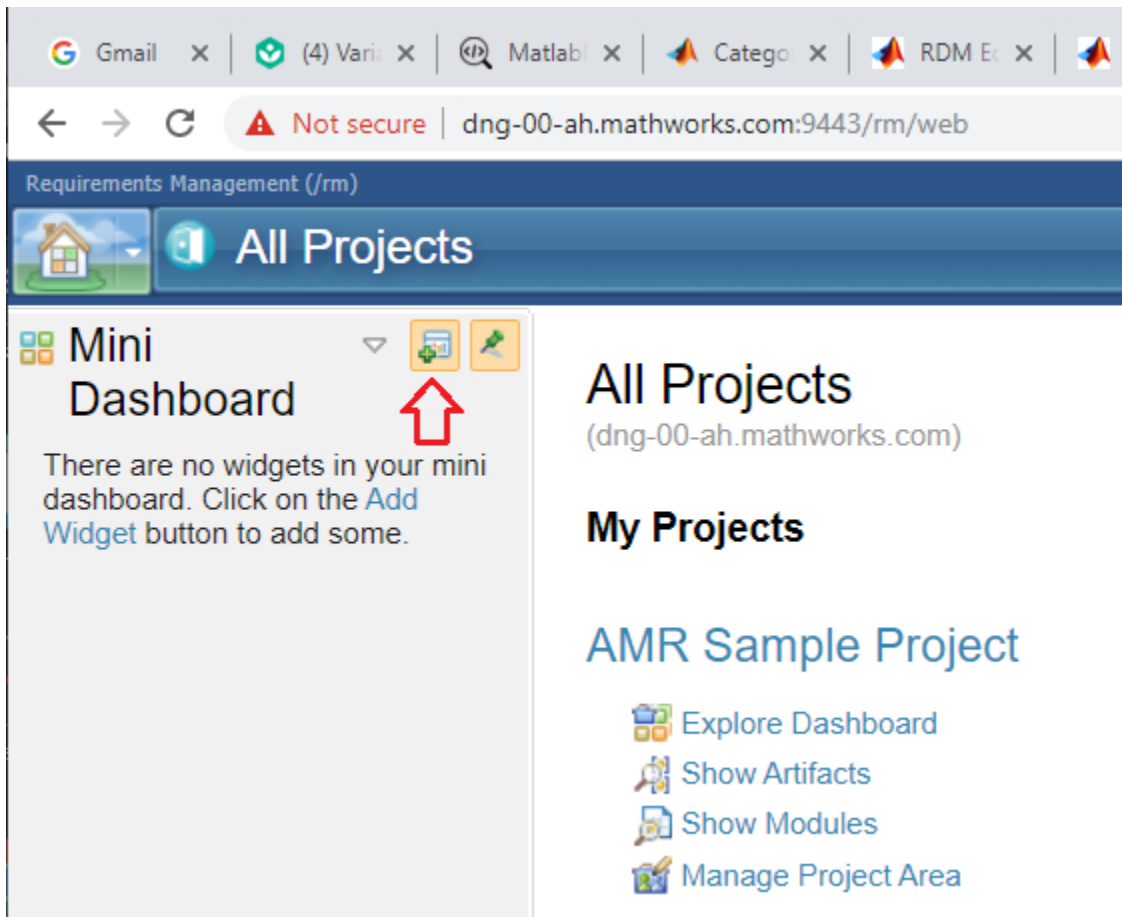
- Place the `.xml` file that represents the extension in the `extensions` directory. For information about a suitable extension to start with, see [the Hello World example](#). For example, the extension might be called `HelloWorld.xml`.
- Start the RM server.



## Client Browser Configuration

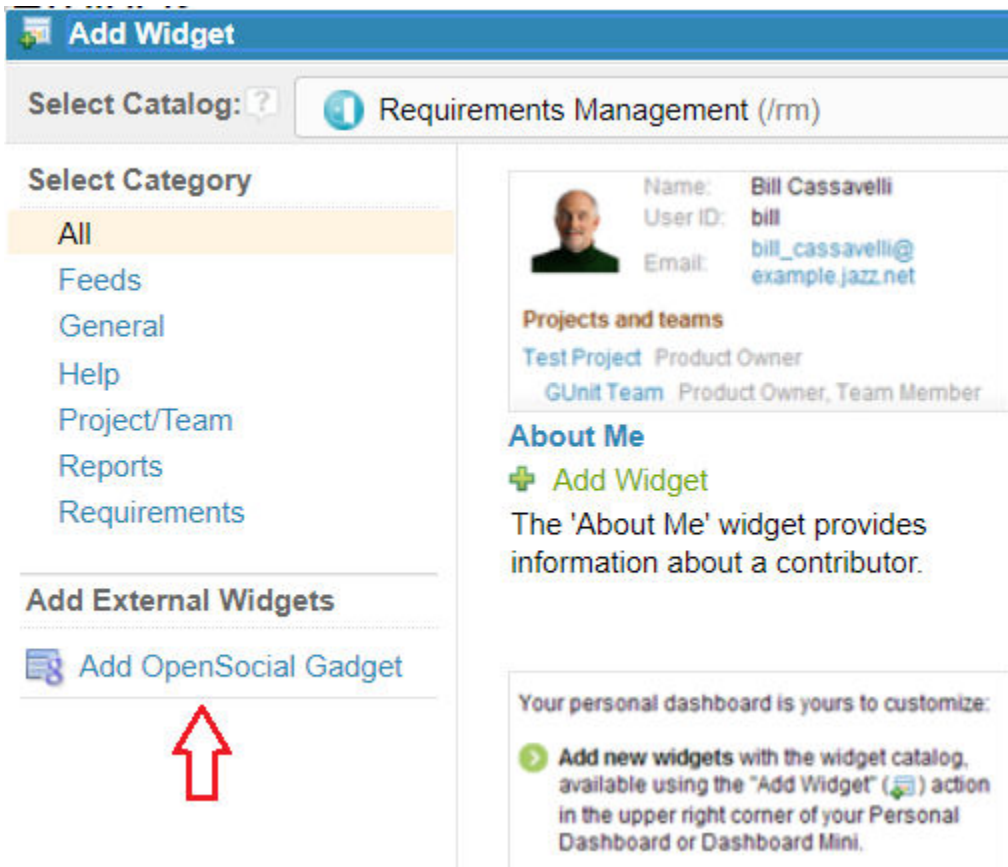
Once `dngsllink_config` custom extension is available on your DNG server, each user interested in linking with DNG needs to add this *custom widget* to the **Mini Dashboard** in DNG client browser. After logging into DNG:

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



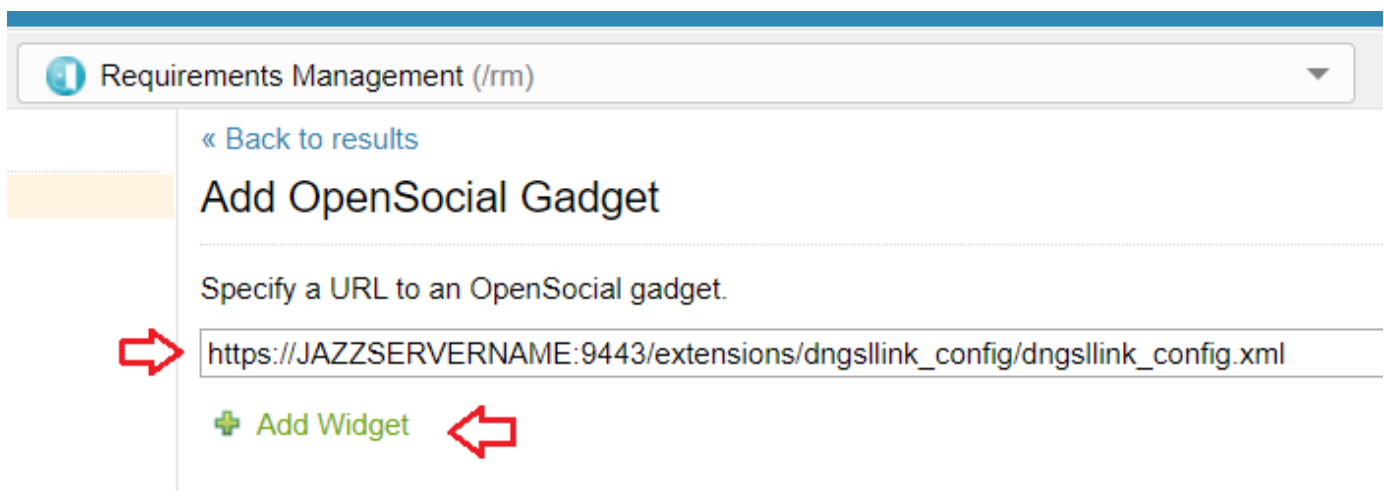
Custom gadgets menu will open

2. Click **Add OpenSocial Gadget**:

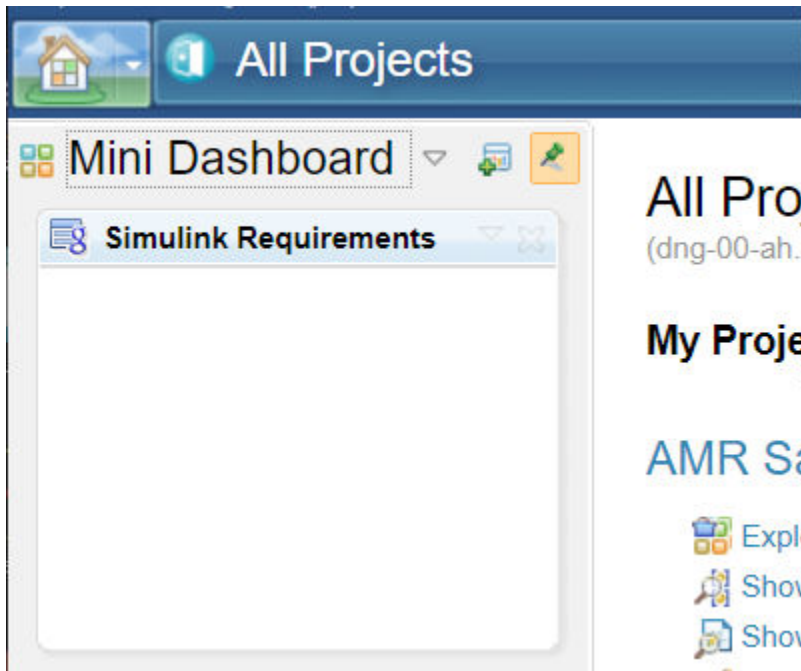


3. Specify the URL that matches the location of Simulink Requirements widget code on your server. For example:

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



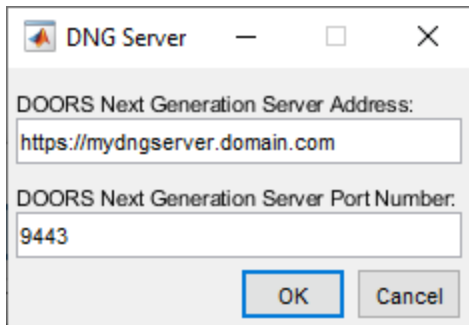
4. Click **Add Widget**. Your **Mini Dashboard** displays the **Simulink Requirements** widget:



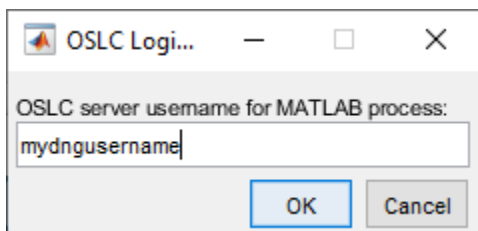
### Configuring MATLAB session

Use the `slreq.dngConfigure` command to prepare your MATLAB session for linking with DNG. 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 DNG password each time.

1. When prompted, enter your DNG 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 DNG pages, enter the default value of "443".

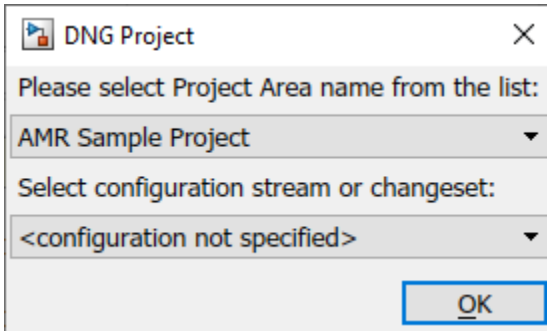


2. Enter your DNG user name, which may be different from your computer login user name:

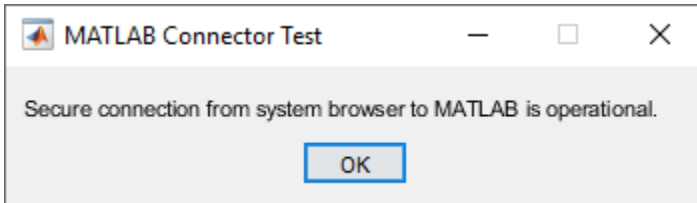


3. When prompted, enter your DNG password and press **Enter**. After connection to the server, the list of available DNG projects is retrieved.

4. Select the project and configuration stream. It is normal to see some warnings in MATLAB's command window when establishing connection with DNG. The feature will operate, unless there are errors.



5. A browser-to-localhost connection test runs automatically. This communication channel is required for your MATLAB® session to receive messages when you interact with DNG in system browser. A popup indicates that you are ready for linking:



6. If you do not see the confirmation message as shown above, it is possible that your system browser is blocking HTTPS connections to localhost. To resolve this, allow the connection. The exact steps depend on your 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 \(unsafe\)](#)



When you get a confirmation popup, it confirms that your browser instance can communicate with HTTPS listeners on your machine. It is best to reuse this same browser window for your DNG session, when linking with Simulink Requirements.

7. To test your browser without rerunning `slreq.dngConfigure` procedure, paste the following URL into your browser's address bar: `https://localhost:31515/matlab/oslc/inboundTest`

### One-way Links from MATLAB/Simulink to DNG

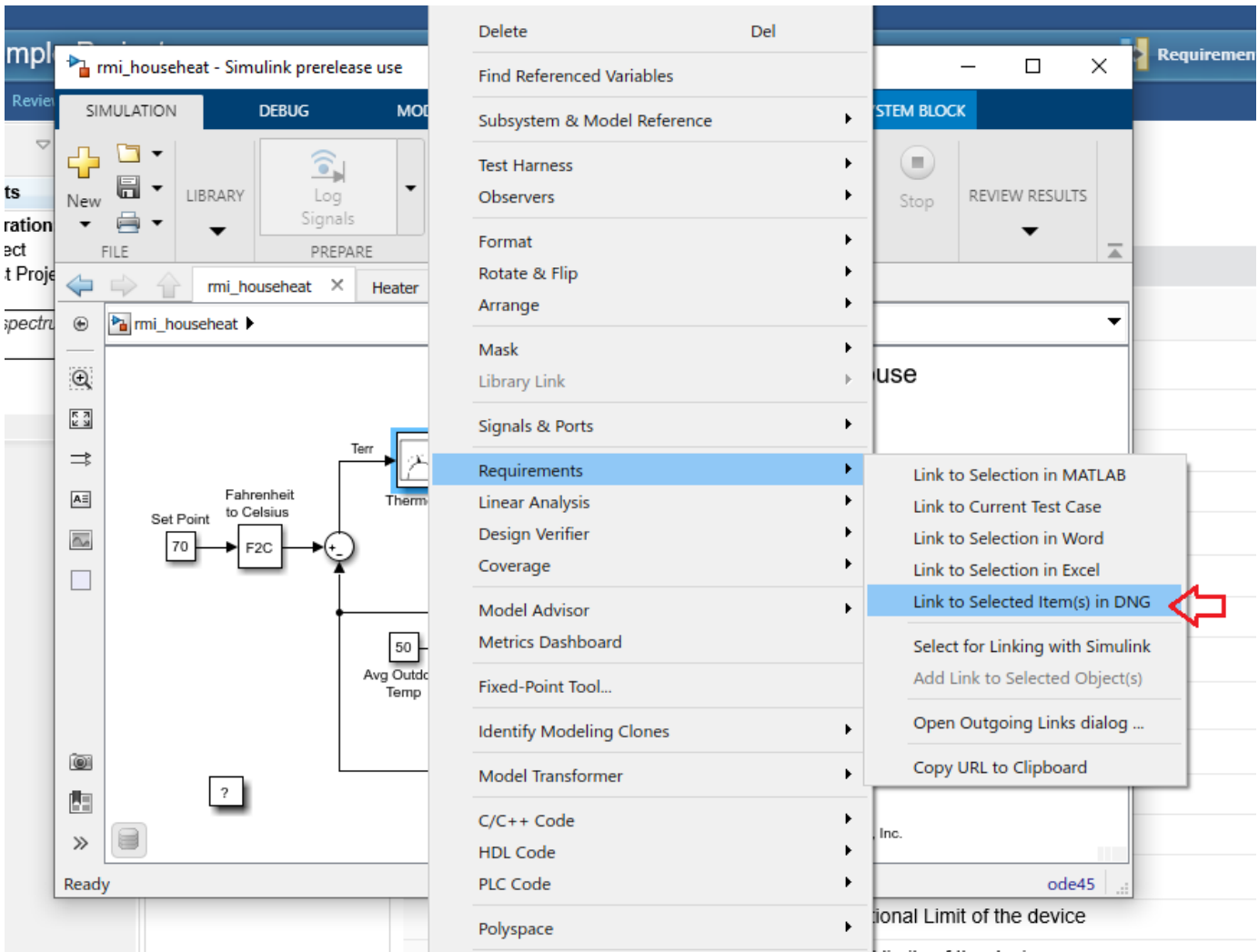
In DNG, open **Show artifacts** view for the requirements collection of interest, and select the checkbox for the item you want to link with. You will notice that the **Simulink Requirements** widget is updated to confirm the ID and label of the selected item. This information is sent to MATLAB when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

The screenshot displays the IBM Rational DOORS DNG interface. The main window shows a list of requirements artifacts under the heading '01 Requirements > 3 AMR Hazards and Risks'. The artifacts are organized into a tree structure:

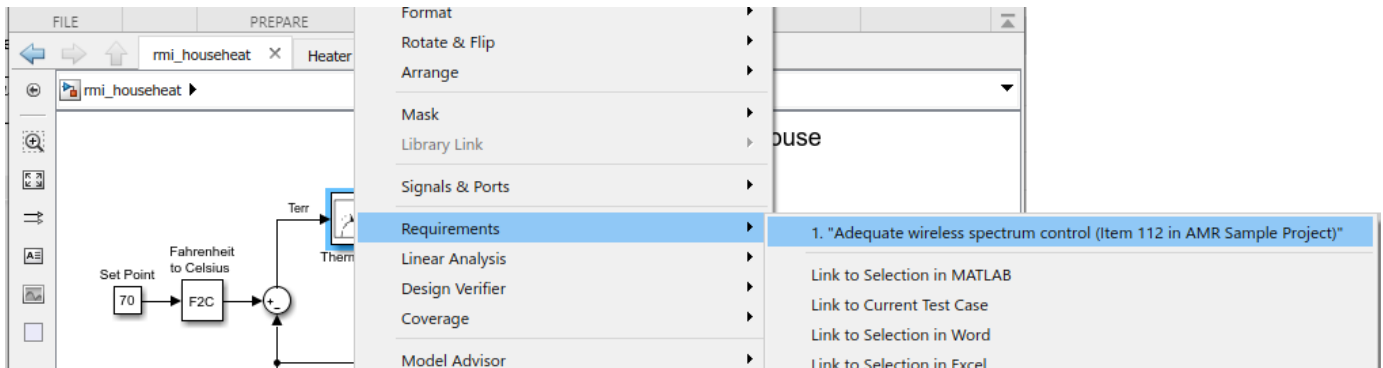
- 336 -1 System Hazards
  - 263 -1.1 Physical Hazards
    - 43 Sharp Edges
    - 297 Pinch Areas
    - 340 Electrocutation
    - 152 -1.1.1 Size Limitations
      - 280 Future AMR handheld size growth
      - 347 Future AMR handheld mass growth
    - 268 1.1.2 Fire/Explosion
  - 55 -1.2 Environmental Hazards
    - 13 -1.2.1 Animals
      - 89 Leashed Pets
      - 30 Stray Animals
    - 70 -1.2.2 External Weather
      - 245 Temperature Operational Limit of the device (Selected)
      - 69 Humidity operational limits of the device
  - 212 -1.3 External Communications

The right-hand pane shows the details for the selected artifact '3: AMR Hazards and Risks', including its description, project information, and creation/modification dates. The Simulink Requirements widget on the left is updated to show the selected artifact's ID (245) and label (Temperature Operational Limit of the device).

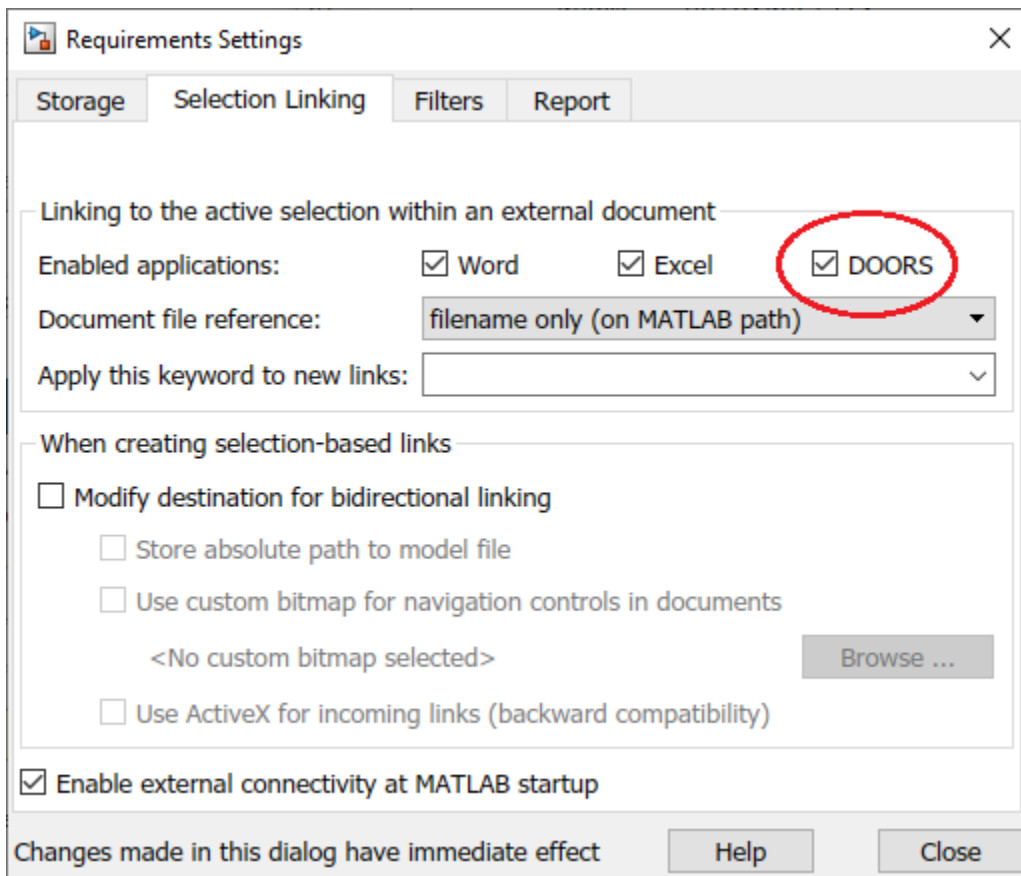
In Simulink, right-click a block you want to link from, then select **Link to Selected Item(s) in DNG** under **Requirements** context menu. It takes a few seconds for MATLAB to retrieve additional data from DNG and create the link.



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 DNG:



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 DNG Side

DNG integration feature in Simulink Requirements allows you to query MATLAB/Simulink links from DNG context. When you select an item from the artifacts list in DNG 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 DNG item.



The screenshot shows the DOORS Next Generation web interface. The browser address bar displays the URL: `https://dng-00-ah.mathworks.com:9443/rm/web#action=com.ibm.rdm.web.pages.showArtifact&artifactURI:`. The page title is "1: AMR System Requirements". The navigation bar includes "Project Dashboard", "Artifacts", "Reviews", and "Reports".

The "Mini Dashboard" on the left shows "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 content area shows a list of requirements. The selected requirement (ID 239) is highlighted in orange. The table below shows the list of requirements:

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

The screenshot shows a web browser window with the URL: `https://localhost:31515/?id=239 - Links from MATLAB/Simulink id=239 - Internet Explorer`. The page title is "Links from MATLAB/Simulink id=239".

The page displays a table of linked artifacts:

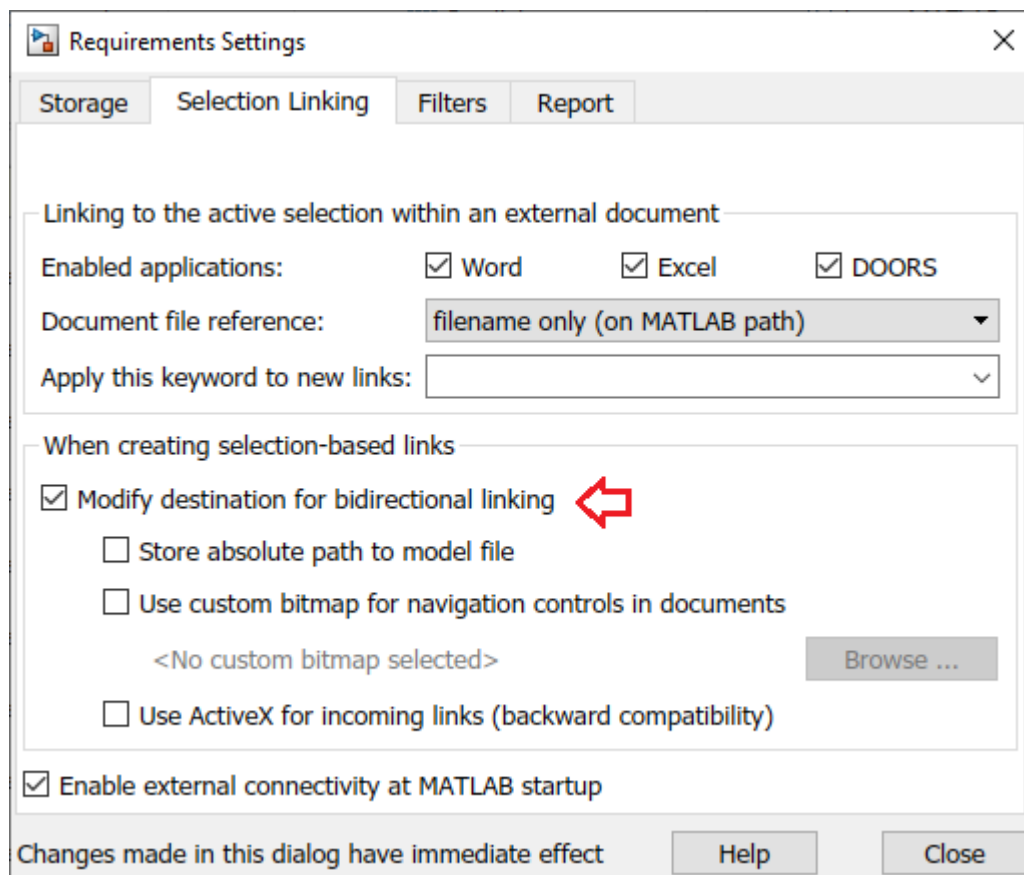
Source Artifact	Source Object	Linked Document	Version	Item
rmi_househeat.slx	Thermostat (SubSystem)	AMR Sample Project	Requirements Test Project Initial Stream	239

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, it is important to ensure that all related linked artifacts on MATLAB/Simulink side are loaded in the current MATLAB session, and browser-to-MATLAB communication is allowed by the web browser. You can review the list of loaded Link Sets by switching the **Requirements Editor** into the **Links View** (more on this below).

### Store Links in DNG for Two-Way Traceability

If you prefer to always find your MATLAB/Simulink links in DNG 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 DNG, which you can see in the **Links** panel for the linked DNG item. You can use the hyperlinks in the **Links** pane to navigate from DNG item to linked objects in MATLAB/Simulink.

The screenshot displays a requirement artifact in a software development tool. At the top right, there are icons for lock, refresh, menu, and an 'Edit' button. The main content area is titled 'Overview' and shows a requirement ID '112: <Adequate wireless spectrum control>'. Below this, it indicates 'No Tags Defined' and provides a 'Description:' section with the following details: 'Project: Requirements Test Project', 'Team Ownership: AMR Sample Project', 'Created On: May 7, 2018, 8:11:58 AM', 'Created By: Kot Matroskin', and 'Modified On: May 7, 2018, 8:11:58 AM'. A 'Comments' section is visible but empty. The 'Links (2)' section is highlighted with a red oval and contains two entries: '[speed < up\_th] (Transition)' and 'rmi\_househeat/Thermostat (SubSystem)'. At the bottom right, there is a signature 'jazz' and a dropdown arrow.

When enabling **Modify destination for bi-directional linking** option in **Requirements Settings**:

1. Every DNG user will see these links when working with same version of this DNG project, even if they do not use Simulink or do not have access to linked MBD artifacts.
2. Navigation from DNG will fail, unless MATLAB is running, and linked artifact is either already loaded or can be found on MATLAB path.
3. Links inserted into DNG by Simulink Requirements do not synchronize automatically. If you delete a link on Simulink side, links in DNG still exists. You need to remove these manually.
4. These links behave as *shared*. For example, if Simulink user A linked a DNG requirement to a block in his Simulink model, and user B linked the same DNG requirement with a different block in same or some other Simulink model, both users will see both links, and both links will navigate to the

corresponding linked block, as long as the MATLAB/Simulink end of the link exists in both user's sessions.

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

### Improved Integration with DOORS Next Generation

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

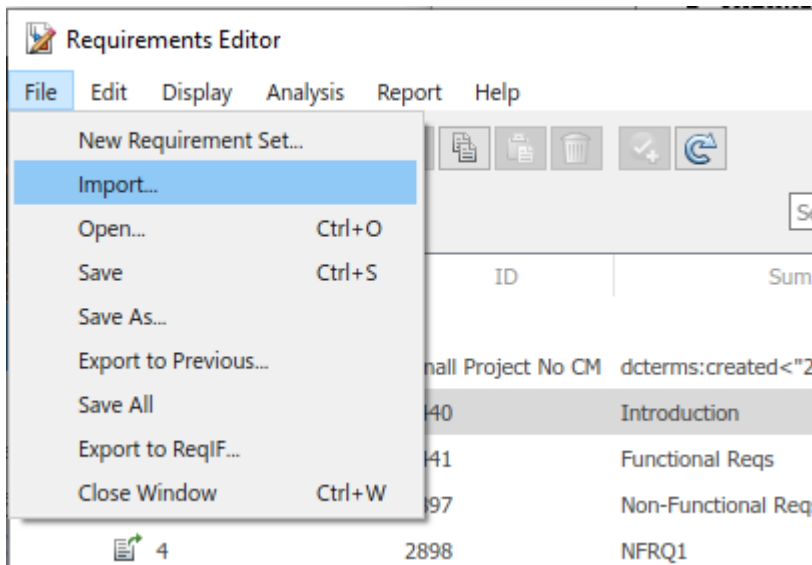
- *direct linking* depends on your ability to modify the DNG server configuration and install the custom Simulink Requirements *gadget*,
- *direct linking* requires that you allow HTTPS communication between your system browser and the local MATLAB process, which could be considered a security risk when using same browser for external webpages,
- 1-way links are difficult to discover from DNG side, and impose strict requirements on Simulink session state for links to become visible on DNG side,
- 2-way links may become difficult to manage in large multi-user projects or when switching between DNG streams and changesets,
- you cannot control the Type of links from DNG to MATLAB/Simulink, the links are always of generic "Link To" type,
- built-in analysis in Simulink Requirements product cannot be applied to *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 DNG 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.

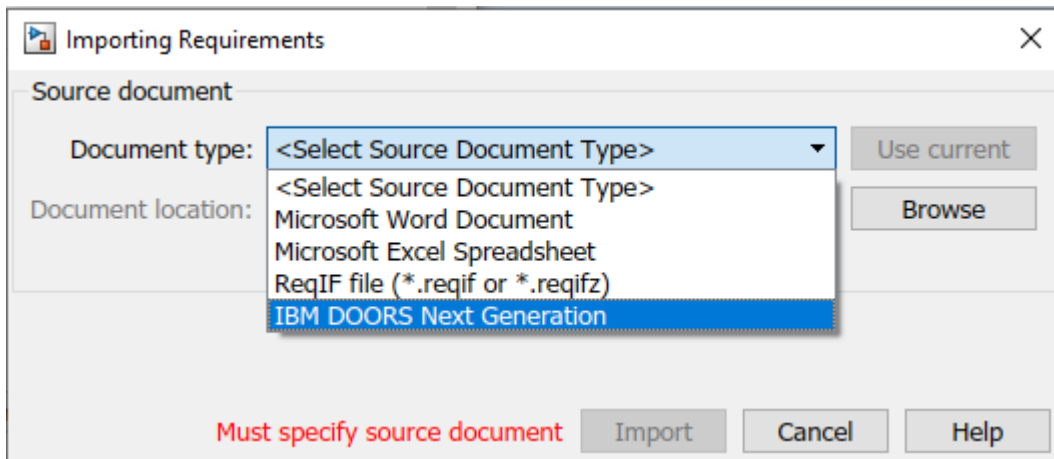
You will not be able to edit DNG contents locally, and you will not immediately see the updates when the sourced requirements are updated on the server, but you get the advantage of native linking support between Simulink artifacts without "dirtying" the server side, and you can use all the built-in analysis capabilities of Simulink Requirements product, including implementation and verification status, as well as change impact detection and management.

### Capture DNG Collections into Simulink Requirements Set

In the Requirements Editor, click **Import** in the main menu:



Select "IBM DOORS Next Generation" in **Document type** selector:



As before, you will be prompted for the DNG login password, and possibly for the DNG server address and login name, if this is your initial connection for the current MATLAB session.

**Document location** selector will populate with names of all DNG 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 DNG contents into Simulink Requirements. You can import the specified module, including the herarchical relationships between DNG 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:

OSLC Query Builder

Build an OSLC query.

Query by Built-in Property:

CustomID Equal  Add

CreatedBy Equal  Add

CreatedOn Before 16-Sep-2019 13:46:11 Add

Query by Enumerated Attribute:

Software Specification Query By Type

Artifact Format is equal to Diagram Add

OSLC query:

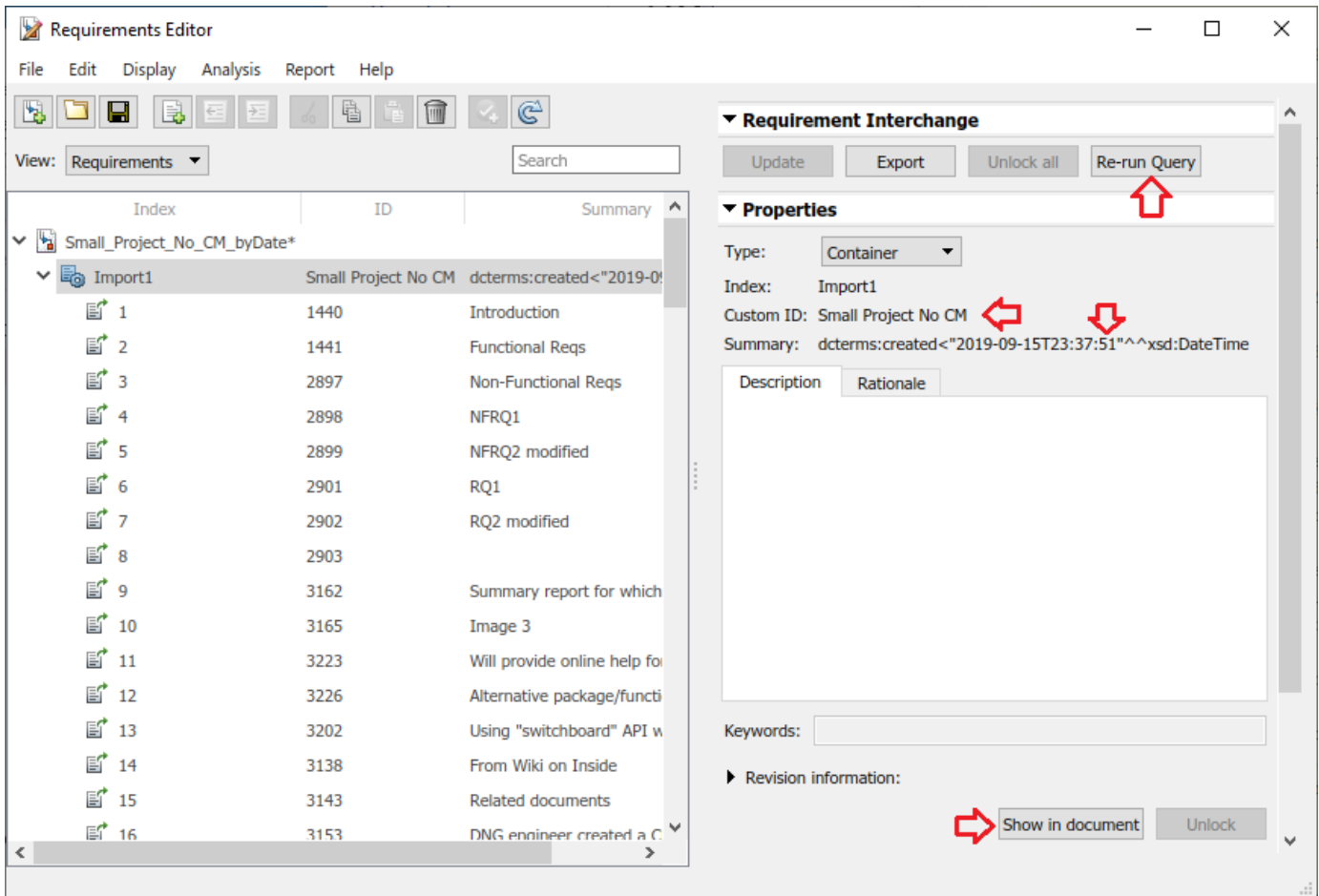
```
dcterms:created<"2019-09-16T13:46:11"^^xsd:DateTime
```

OK Cancel

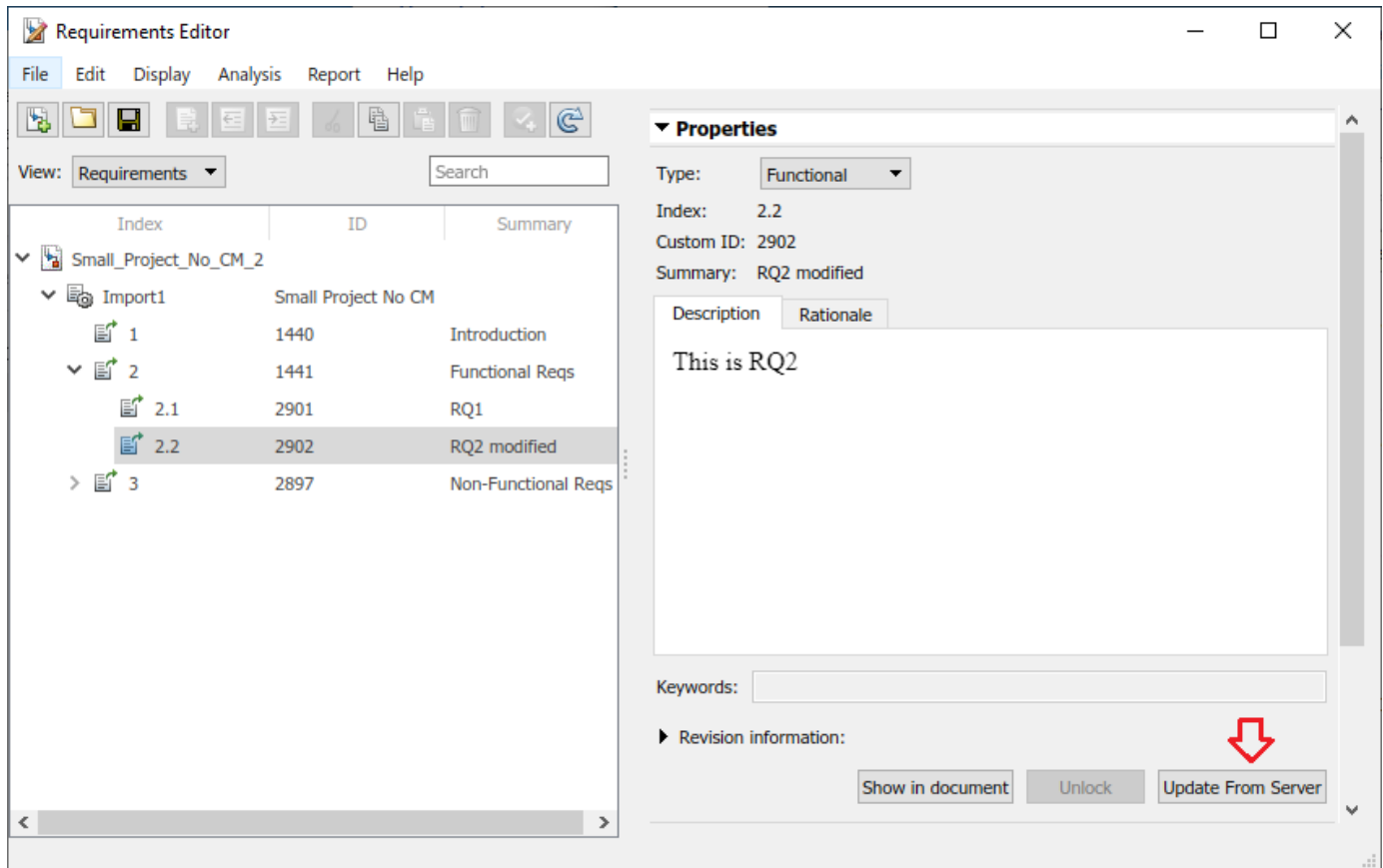
In both cases you get a top-level *Import node* with the ID that matches the name of your DNG project. The Summary text of the *Import node* will indicate the parameters used when capturing data from DNG. You can now work with the imported items as you would with the usual entries in Simulink Requirements:

- create links with related MBD artifacts and use all the built-in analysis capabilities.
- navigate to the original requirements in DNG 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 the imported 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.



### Linking with Captured DNG References

Now that you have captured DNG requirements collection of interest into a Requirement Set and saved it to .slreqx file, you can easily establish traceability between Requirements and design, then manage your Link Sets together with the rest of MBD artifacts, without affecting other users of sourced requirements on DNG server. For example, you can switch your Simulink design model into **Requirements perspective** view, then open the imported set of DNG 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 DNG references in the requirements browser.

The screenshot displays the Simulink Requirements tool interface for a project named 'rmi\_househeat'. The main workspace shows a Simulink model with various blocks like 'F2C', 'Thermostat', 'House', and 'Cost Calculator'. A red circle highlights a requirement link labeled '370: Environmental Considerations' within the 'House' block. Below the model is a 'Requirements - rmi\_househeat' table with the following data:

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

The 'Requirements Inspector' on the right shows details for requirement 370, including its type (Functional), index (2.2), and summary (Environmental Considerations). A red arrow points to the 'Environmental Considerations' row in the table.

## Reviewing and Analyzing Traceability Data

As with links to internally-managed requirements, you can switch into the **Links** view to access more details about links, and to 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 DNG requirements on server are updated or removed, you perform the automated update of captured 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 Rational DOORS interface with a Simulink model titled "rmi\_househeat". The model includes components like "Set Point", "Fahrenheit to Celsius", "Thermostat", "Room", "Heater", "HeatFlow", "Cost Calculator", "C2F", "Tindoor", "Toudoor", and "Temperatures". A requirement link is shown connecting the "House" block to "370: Environmental Considerations".

The "Requirement links - rmi\_househeat" table is shown below the model:

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

The "Property Inspector" on the right shows the details for the selected link:

- Link: 370: Environmental Considerations (A...
- Source: [House](#)
- Type: Implements
- Destination: [370 Environmental Consid...](#)

## Requirements Traceability with IBM Rational DOORS Next Generation

You can link and trace Simulink model elements to requirements in IBM Rational DOORS Next Generation. Before you begin, configure IBM Rational DOORS Next Generation for communication with MATLAB by following the instructions in “Install the Simulink Requirements Widget in IBM Rational DOORS Next Generation” on page 5-3. Enable bidirectional requirements traceability with IBM Rational DOORS Next Generation:

- 1 In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, select **Link Settings > Linking Options**.
- 2 Switch to the **Selection Linking** tab and select **DOORS** in the **Enabled applications** field.
- 3 Select **Modify destination for bidirectional linking**.

### Link to Requirements in IBM Rational DOORS Next Generation

To link and trace your Simulink model elements to requirements in IBM Rational DOORS Next Generation, use any of these workflows:

- “Link to Requirements by Using the Outgoing Links Editor Dialog Box” on page 7-25
- “Link to Selected Requirements in a Project by Using the Simulink Context Menu” on page 7-25 if you have installed the **Simulink Requirements** widget in IBM Rational DOORS Next Generation.
- “Link to the Requirements in a Project by Using the Numeric ID” on page 7-26 if you cannot use either of the two options.

#### Link to Requirements by Using the Outgoing Links Editor Dialog Box

- 1 Right-click the Simulink model element to which you want to link IBM Rational DOORS Next Generation requirements.
- 2 Select **Requirements > Open Outgoing Links dialog**.
- 3 In the **Outgoing Links** dialog box, click **New** and select OSLC Resource as the **Document type**.
- 4 Click **Browse**.
- 5 Enter your IBM Rational DOORS Next Generation login credentials. From the drop-down list, select the active project name in IBM Rational DOORS Next Generation.
- 6 Switch to the **Document Index** tab and select the requirements that you want to link to from the list of requirements. To create the link, click **OK**.

#### Link to Selected Requirements in a Project by Using the Simulink Context Menu

Install the **Simulink Requirements** widget in IBM Rational DOORS Next Generation. For more information, see “Install the Simulink Requirements Widget in IBM Rational DOORS Next Generation” on page 5-3

- 1 In IBM Rational DOORS Next Generation, open the **Mini Dashboard** and pin it to the screen.
- 2 Switch to the **Browse Artifacts** view.
- 3 Select the requirements that you want to link to by selecting the check box next to the requirement.

The requirements that you select for linking are displayed in the **Simulink Requirements** widget in the **Mini Dashboard**.

- 4 Right-click the Simulink model element to which you want to link IBM Rational DOORS Next Generation requirements.
- 5 Establish links to the requirements by selecting **Requirements > Link to Current Item in DNG**.

Click **List Projects** in the dialog box that appears and select the requirements from within IBM Rational DOORS Next Generation.

### Link to the Requirements in a Project by Using the Numeric ID

Use this option if you are unable to link to requirements by using the Outgoing Links dialog box or by using the Simulink context menu.

- 1 Right-click the Simulink model to which you want to link IBM Rational DOORS Next Generation requirements.
- 2 Select **Requirements > Link to Current Item in DNG**.
- 3 Click **Manual entry** in the dialog box that appears and enter the numeric ID for the link target. Establish links to the requirements by clicking **OK**.

### Navigate to Requirements from Simulink

Right-click the Simulink model element that requirements have been linked to. Select **Requirements** and navigate to the corresponding requirement in IBM Rational DOORS Next Generation by clicking the navigation shortcut at the top of the menu.

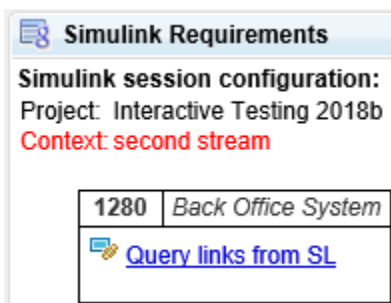
## Work with IBM Rational DOORS Next Generation Projects with Configuration Management Enabled

Projects with configuration management enabled in IBM Rational DOORS Next Generation support multiple branches called *streams* and *changesets*. Changesets are akin to shared development branches that can later be merged with the parent main stream. 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.

You can select the IBM Rational DOORS Next Generation Project and the configuration stream or changeset you want to work with. At the MATLAB command prompt, enter:

```
oslc.configure
```

The **Simulink Requirements** widget displays information about the current configuration stream context you work with in Simulink Requirements. The widget indicates if there is a mismatch between the active configuration stream contexts in Simulink Requirements and in IBM Rational DOORS Next Generation by highlighting the active configuration stream context in Simulink Requirements.



To resolve the mismatch, click the highlighted text in the widget. Click **Update** in the **DNG Configuration Context Mismatch** dialog box to update the configuration stream context in Simulink Requirements to be consistent with the current configuration stream context in IBM Rational DOORS Next Generation. Alternatively, you can change the active configuration stream in IBM Rational DOORS Next Generation.

## 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
```

---

**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, 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 7-28, 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	<b>1 Fuel rate controller requirements</b> The controller will use engine speed, throttle position and manifold pressure to airflow through the engine.
2	[Simulink reference: <code>sldemo_fuelsys/fuel_rate_control</code> (SubSystem)] 

- 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**.

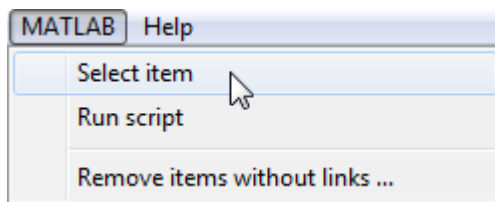
A single navigation object is inserted at the selected requirement.

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

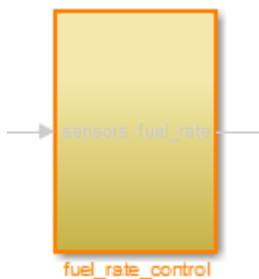
## Navigate Between IBM Rational DOORS Requirement and Model Object

In “Insert Navigation Objects into IBM Rational DOORS Requirements” on page 7-28, 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**.

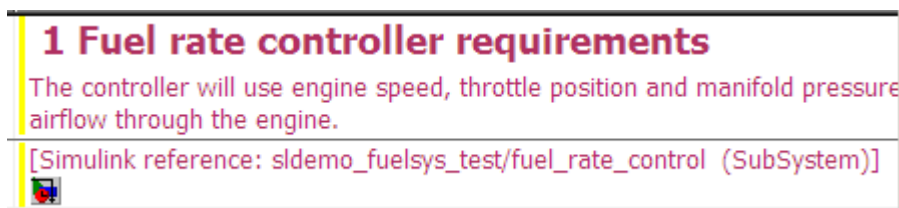


Your version of the `sldemo_fuelsys` model opens, with the `fuel_rate_control` subsystem highlighted.



- 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 7-28, 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.
- 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 Simulink Editor, select **Analysis > Requirements > Synchronize with DOORS**.

The DOORS synchronization settings dialog box opens.

- 7 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.

---

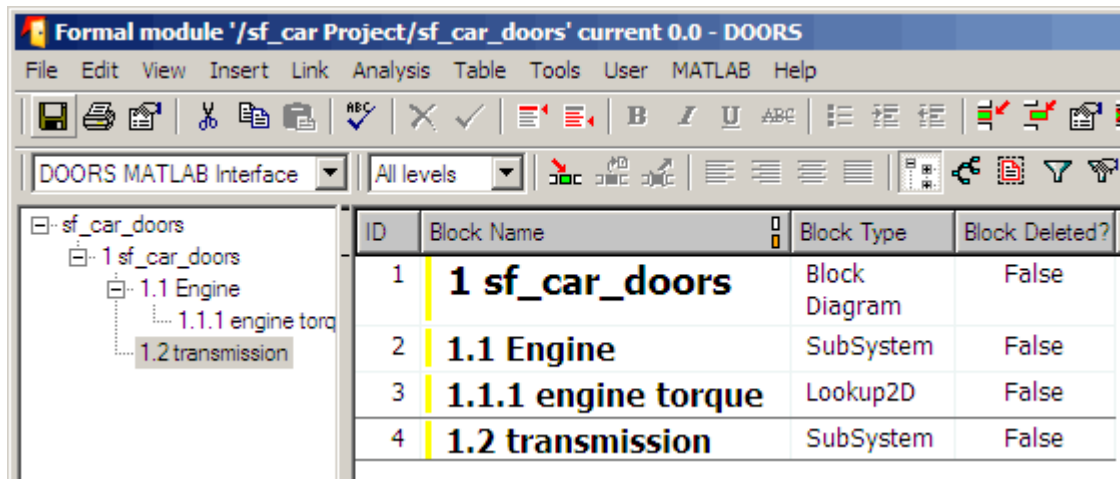
**Note** For more information about the synchronization options, see “Customize IBM Rational DOORS Synchronization” on page 7-36.

---

- 8 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).



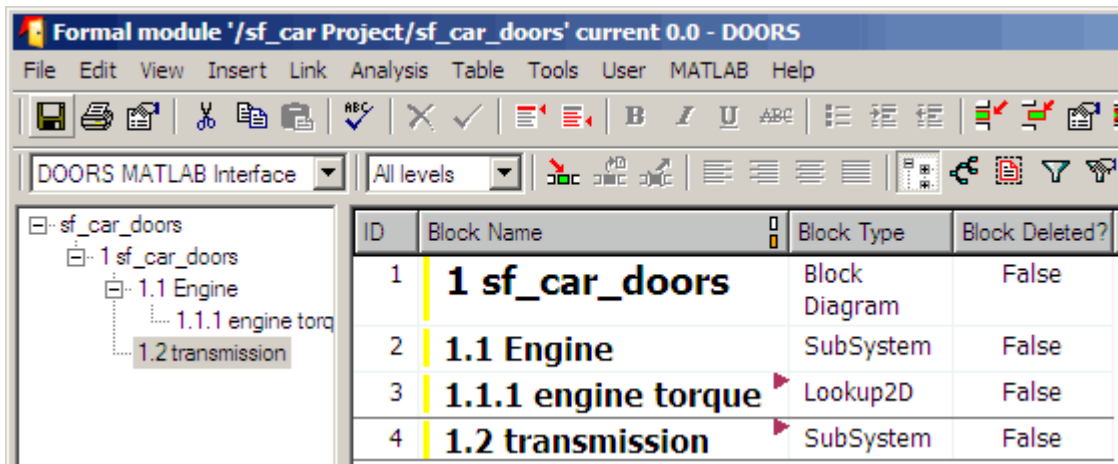
- 9 Save the surrogate module and the model.

## 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 Simulink Editor, select **Analysis > Requirements > 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.



## 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 Select **Analysis > Requirements > Synchronize with DOORS**.
- 3 In the DOORS synchronization settings dialog box, leave the **Extra mapping additionally to objects with links** option set to Complete - All blocks, subsystems, states, and transitions. Click **Synchronize**.

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

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

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

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

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

- 5 Delete the copied object (vehicle mph (yellow) & throttle %1) and resynchronize the model.
- 6 Save the surrogate module.
- 7 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.

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

- 2 Select the requirement name.

The formal module opens, at the Transmission Requirements object.

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

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

1	<b>1 Transmission Requirements</b>	/sf_car Project/sf_car_doors ▶ 4: transmission	
Should have five gears: Park, Reverse, D1, D2, and Lo. User cannot start engine unless the Park gear is engaged. Reverse cannot be entered from D1 unless the vehicle is not moving. Lo cannot be entered from D2 unless the speed of the vehicle is less than 15 mph.			

- 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
<b>DOORS surrogate module path and name</b>	<p>Specifies a unique DOORS path to a new or an existing surrogate module.</p> <p>For information about how the RMI resolves the path to the requirements document, see “Document Path Storage” on page 11-34.</p>
<b>Extra mapping additionally to objects with links</b>	<p>Determines the completeness of the Simulink model representation in the DOORS surrogate module. <b>None</b> specifies synchronizing only those Simulink objects that have linked requirements, and their parent objects. For more information about these synchronization options, see “Customize the Level of Detail in Synchronization” on page 7-38.</p>
<b>Update links during synchronization</b>	<p>Specifies updating any unmatched links the RMI encounters during synchronization, as designated in the <b>Copy unmatched links</b> and <b>Delete unmatched links</b> options.</p>
<b>Copy unmatched links</b>	<p>During synchronization, selecting the following options has the following results:</p> <ul style="list-style-type: none"> <li>• <b>from Simulink to DOORS:</b> For links between the model and the formal module, the RMI creates matching links between the DOORS surrogate and formal modules.</li> <li>• <b>from DOORS to Simulink:</b> For links between the DOORS surrogate and formal modules, the RMI creates matching links between the model and the DOORS modules.</li> </ul>
<b>Delete unmatched links</b>	<p>During synchronization, selecting the following options has the following results:</p> <ul style="list-style-type: none"> <li>• <b>Remove unmatched in DOORS:</b> For links between the formal and surrogate modules, when there is not a corresponding link between the model and the DOORS modules, the RMI deletes the link in DOORS.</li> </ul> <p>This option is available only if you select the <b>from Simulink to DOORS</b> option.</p> <ul style="list-style-type: none"> <li>• <b>Remove unmatched in Simulink:</b> 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.</li> </ul> <p>This option is available only if you select the <b>from DOORS to Simulink</b> option.</p>

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

### Resynchronize a Model with a Different Surrogate Module

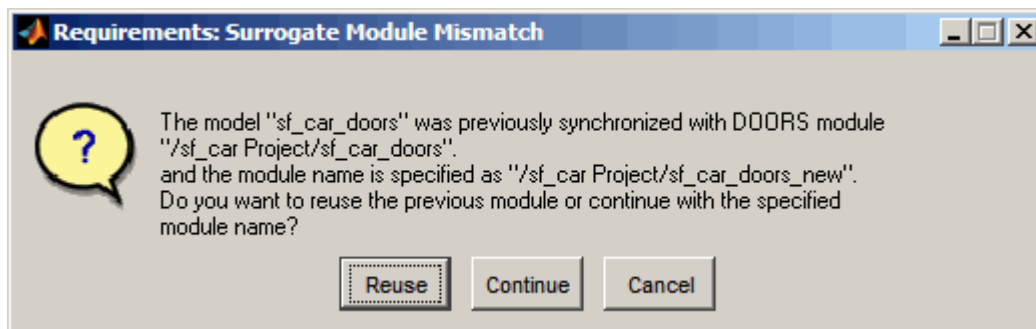
You can synchronize the same Simulink model with a new DOORS surrogate module. For example, you might want the surrogate module to contain only objects that have requirements to DOORS, rather than all objects in the model. In this case, you can change the synchronization options to reduce the level of 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 7-32, you synchronized the model with the **Extra mapping additionally to objects with links** option set to None. As a result, the surrogate module contains only Simulink objects that have requirement links, and their parent objects. Additional synchronization options, described in this section, can increase the level of surrogate detail. Increasing the level of surrogate detail can slow down synchronization.

The **Extra mapping additionally to objects with links** option can have one of the following values. Each subsequent option adds additional Simulink objects to the surrogate module. You choose None

to minimize the surrogate size or **Complete** to create a full representation of your model. The **Complete** option adds all Simulink objects to the surrogate module, creating a one-to-one mapping of the Simulink model in the surrogate module. The intermediate options provide more levels of detail.

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

### 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 7-32 and “Create Links Between Surrogate Module and Formal Module in an IBM Rational DOORS Database” on page 7-33.

- 1 Open the `sf_car_doors` model that you synchronized in “Synchronize a Simulink Model to Create a Surrogate Module” on page 7-32 and again in “Create Links Between Surrogate Module and Formal Module in an IBM Rational DOORS Database” on page 7-33.
- 2 In the Simulink Editor, select **Analysis > Requirements > Synchronize with DOORS**.

The DOORS synchronization settings dialog box opens.

- 3 Resynchronize with the same surrogate module, making sure that the **DOORS surrogate module path and name** specifies the surrogate module path and name that you used in “Synchronize a Simulink Model to Create a Surrogate Module” on page 7-32.

For information about how the RMI resolves the path to the requirements document, see “Document Path Storage” on page 11-34.

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

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

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

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

#### Detailed Information About The Surrogate Module You Created

Notice the following information about the surrogate module that you created in “Resynchronize to Include All Simulink Objects” on page 7-39:

- 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 Simulink model hierarchy.

When you synchronize a model for the first time, the DOORS software creates a surrogate module. The surrogate module contains a representation of the model, depending on your synchronization settings. (To learn how to customize the links and level of detail in the synchronization, see “Customize IBM Rational DOORS Synchronization” on page 7-36.)

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.

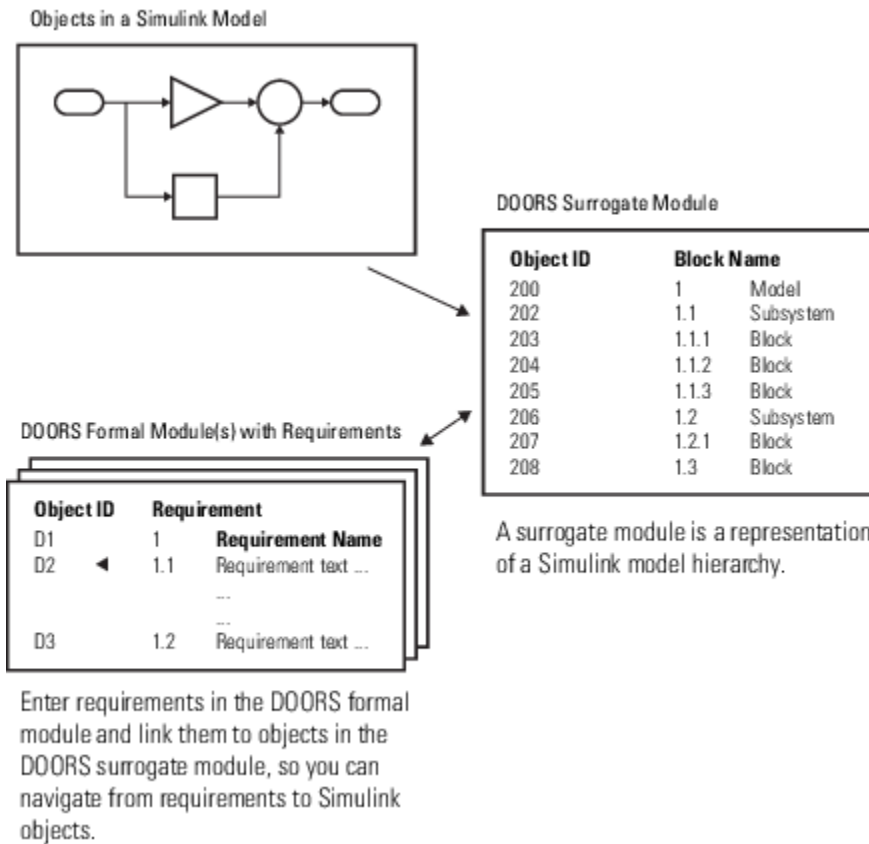
---

**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.

The screenshot displays the IBM Rational DOORS 9 interface. The main window shows a project titled 'HelicopterSystemRequirements' with a tree view on the left containing sections: 1 Helicopter Flight Control System, 2 Introduction, 3 System Description, and 4 System Requirements. The main pane shows a table of requirements with the following data:

ID	Summary
20	<b>3 System Description</b>
21	The flight control system consists of pilot input controls, cyclic and pedals, a flight control computer and hydraulic actuators to control the main and tail rotors. A diagram of the system is shown in the figure below.
22	3.0-2

Below the table is a diagram of the Helicopter Control System. It shows a flow from 'Pilot Input' to 'Flight Control', which then connects to two 'Hydraulic Actuator' blocks, which in turn control a helicopter. The diagram is titled 'Helicopter Control System'.

Overlaid on the bottom is the 'Requirements Editor' window. It shows a tree view of requirements with the following data:

Index	ID	Summary
HelicopterSystemRequirements		
Import1	0000da80	References to HelicopterSystemR
1	1	Helicopter Flight Control System I
2	18	Introduction
3	20	System Description
3.1	21	The flight control system consists
3.2	22	3.0-2
3.3	23	The cyclic controls the pitch of th
4	24	System Requirements

The 'Requirements Editor' also shows a diagram of the 'Helicopter Control System' with a 'Rationale' tab selected. The diagram shows 'Pilot Input Sensors' feeding into a 'Flight Control Computer', which outputs to three 'Hydraulic Actuator' blocks. These actuators control the helicopter's 'Attitude/Heading & Rate Sensors'. The diagram is titled 'Helicopter Control System'. Below the diagram are fields for 'Keywords:' and 'Revision information:', along with 'Show in document' and 'Unlock' buttons.

### Setup for IBM Rational DOORS

Working with DOORS 9 is supported on Microsoft Windows®. Configure the requirements management interface for interaction with IBM Rational DOORS by following the instructions in “Configure RMI 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™”.

### Import a DOORS Module

To import a DOORS module into Simulink Requirements:

1. Log in to DOORS and open the module to import.
2. In the Requirements Editor, select **File > Import**.
3. Select a DOORS module as the source document.
4. If your DOORS module includes pictures or tables, enable the **Include graphics and layout** option.
5. Click **Import** to complete the import process.
6. Check the results in Requirements Editor. The references should preserve the DOORS IDs and the requirements hierarchy.

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 need additional to 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)”.

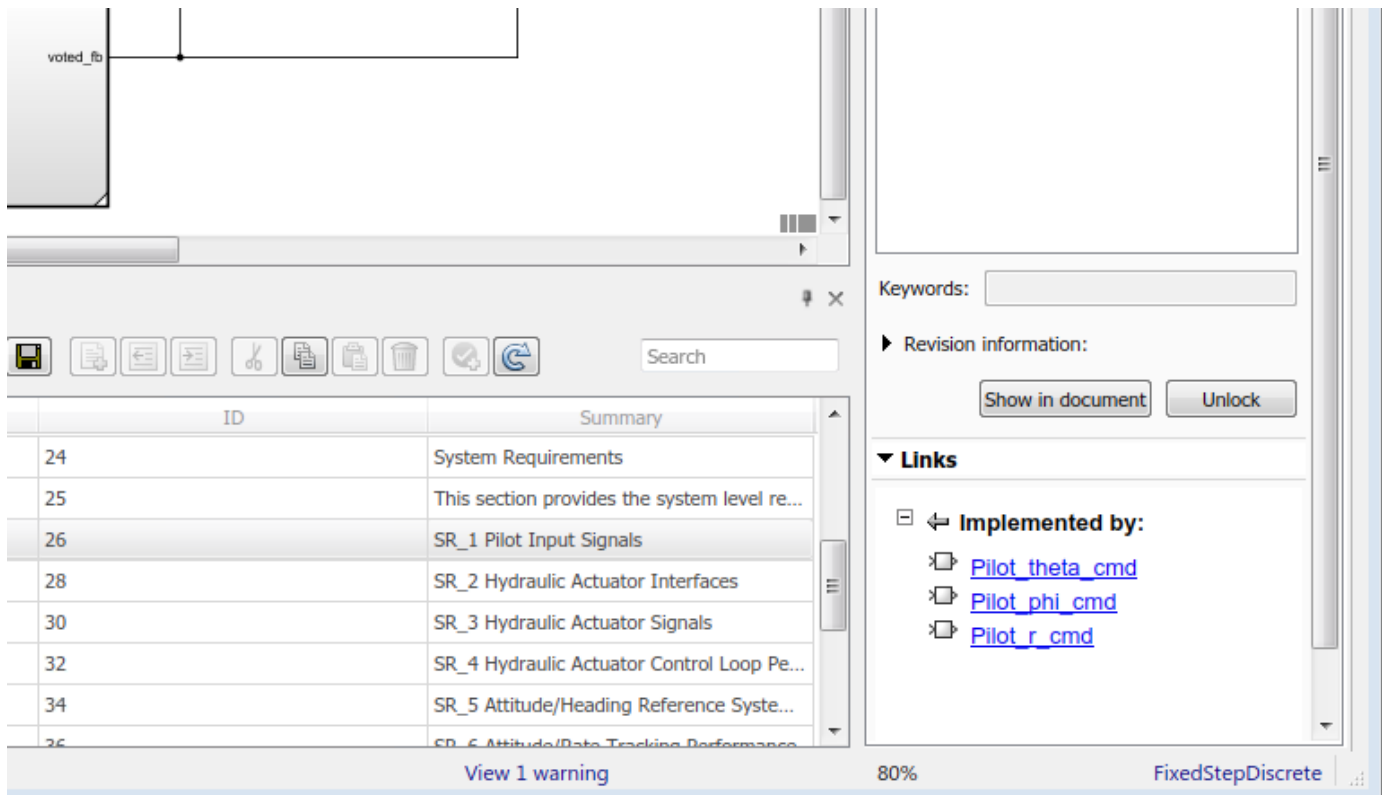
### Link to Your Model

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, the **Links** panel in the bottom right displays related 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 switching to the **Links View**.





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 7-43.

### Update Requirements to Reflect DOORS Changes

If the source requirements in DOORS change, you can update the linked requirements 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 1-17.

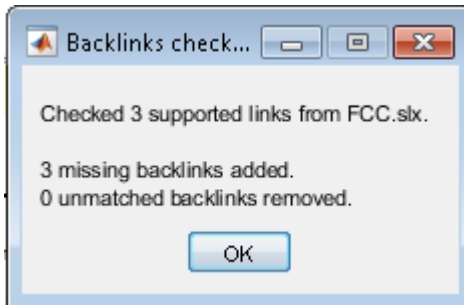
### Synchronizing Links and Navigation from DOORS

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:

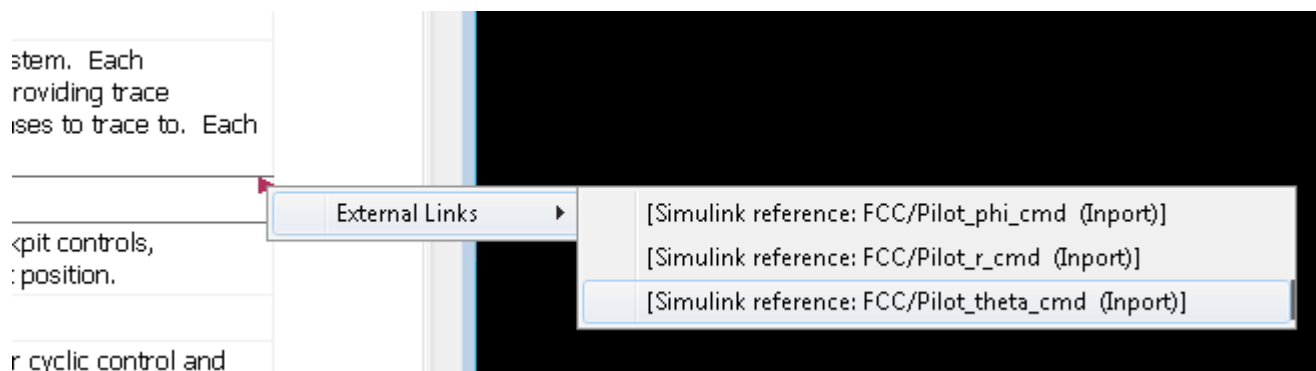
- Switch into the **Links View**.
- Locate and right-click the Link Set that has new links.
- Select **Update Backlinks** shortcut at the bottom of context menu.

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 an outgoing **External Link** and correspond to Simulink entities, such as a block name or test file 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 from 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:

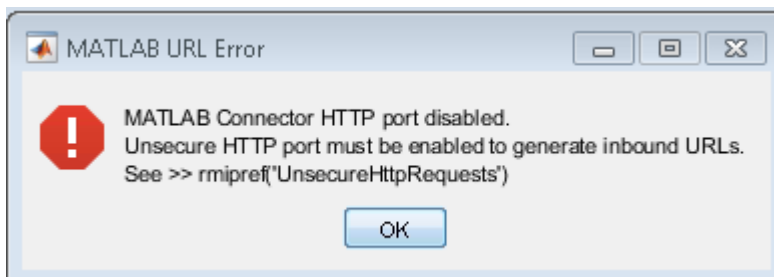


See Manage Navigation Backlinks in External Requirements Documents for general information about managing links from external documents.

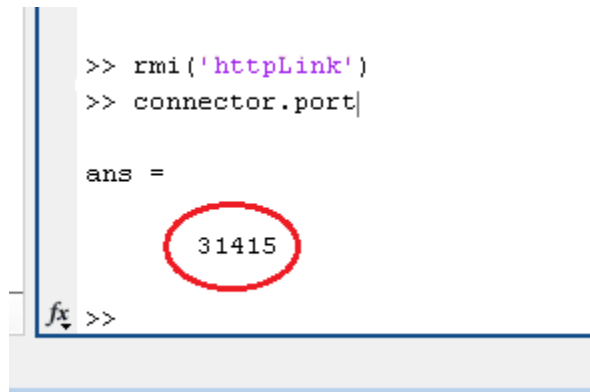
### Embedded HTTP Connector

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 the desired port number 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-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, depending on 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 these same 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 reenter 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 any Custom ID that is present in Simulink Requirement Set, the corresponding item is 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**,

The screenshot shows a Simulink model at the top and a Requirements Browser window below it. The browser window is titled "Requirements - f14\_Test" and shows a tree view of requirements. The table below is a representation of the data shown in the browser:

Index	ID	Summary
f14_doors		
Import1	00000022	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 Simulink model at the top and a 'Requirement links - f14\_Test' table below it. The Simulink model shows a 'Controller' block with inputs 'Stick Input (in)', 'alpha (rad)', and 'q (rad/sec)'. It outputs 'Elevator Command (deg)'. This is connected to a 'Model' block, which outputs 'Vertical Gust wGust (ft/sec)' and 'Pitch Rate q (rad/sec)'. The 'Pitch Rate q (rad/sec)' is connected to an 'Nz Pilot (g)' block, which outputs 'Angle of Attack'.

The 'Requirement links - f14\_Test' table shows the following data:

Label	Source	Type	Destination
f14_Test.slmx	Changed source: 0/4		Changed destination: 2/4
0000022: References to f14_doors (f...	Actuator Model	Implements	0000022 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
0000022: References to f14_doors (f...	Controller	Implements	0000022 References to f14_doors

To recover from this situation you need to: # identify the original DOORS IDs in LinkSet data, # construct the expected updated DOORS IDs based on your knowledge of the original and current module prefix, # rely on reconstructed IDs to locate the matching Requirement Set entry for each broken link destination, # 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: some text", and we know that item that used to have DOORS ID "KKK123" now has DOORS ID "QQQ123".

```

resolveLinksByLabelMatch.m x DOORS9RequirementsExample.m x +
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

```

Ln 2 Col 1

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 model diagram with a 'Requirement links - f14\_Test' window open. The window shows a table of resolved links. The table has four columns: Label, Source, Type, and Destination. The 'Label' column contains links like '00000022: References to f14\_doors (f...)' and 'QQQ2: Aircraft Dynamics Model (f14\_...)' which are highlighted with a red box. The 'Source' column lists 'Actuator Model' and 'Controller'. The 'Type' column shows 'Implements'. The 'Destination' column lists '00000022 References to f14\_doors', 'QQQ2 Aircraft Dynamics Model', and 'QQQ3 Nz pilot calculation formula'.

Label	Source	Type	Destination
f14_Test.slmx*	Changed source: 0/4		Changed destination: 4/4
00000022: References to f14_doors (f...)	Actuator Model	Implements	00000022 References to f14_doors
QQQ2: Aircraft Dynamics Model (f14_...)	Controller	Implements	QQQ2 Aircraft Dynamics Model
QQQ3: Nz pilot calculation formula (f1...)	Controller	Implements	QQQ3 Nz pilot calculation formula
00000022: References to f14_doors (f...)	Controller	Implements	00000022 References to f14_doors

## See Also

### Related Examples

- “IBM Rational DOORS Traceability”
- “Managing Requirements for Fault-Tolerant Fuel Control System (IBM Rational DOORS)”





# 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 8-8
- “Requirements Links for Library Blocks and Reference Blocks” on page 8-11
- “Navigate to Requirements from Model” on page 8-15

## 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 Highlight the test case in the test browser.
- 3 Right-click the block or test step, and select **Requirements > Link to Current Test Case**.

#### Requirements Traceability Example

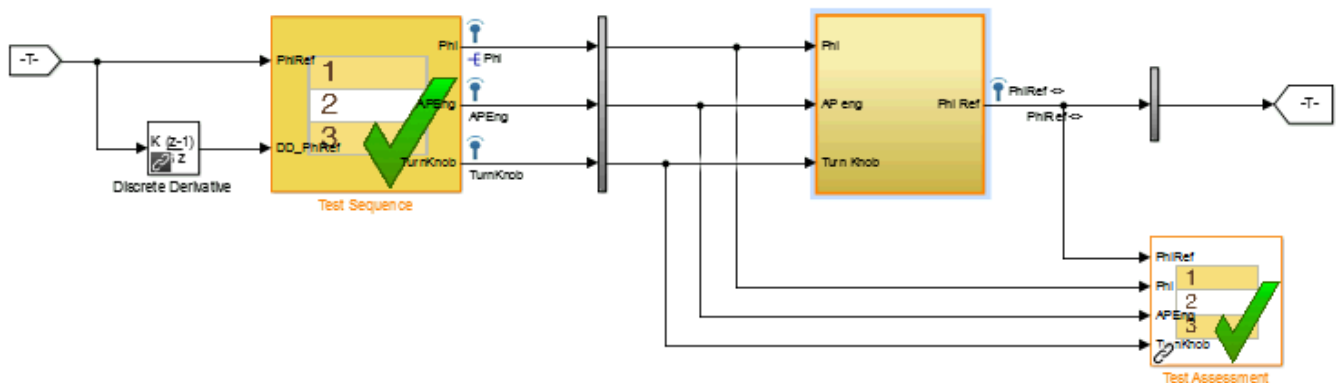
This example demonstrates adding requirements links to a test harness and test sequence. The model is a component of an autopilot roll control system. This example requires Simulink Test and Simulink Requirements.

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

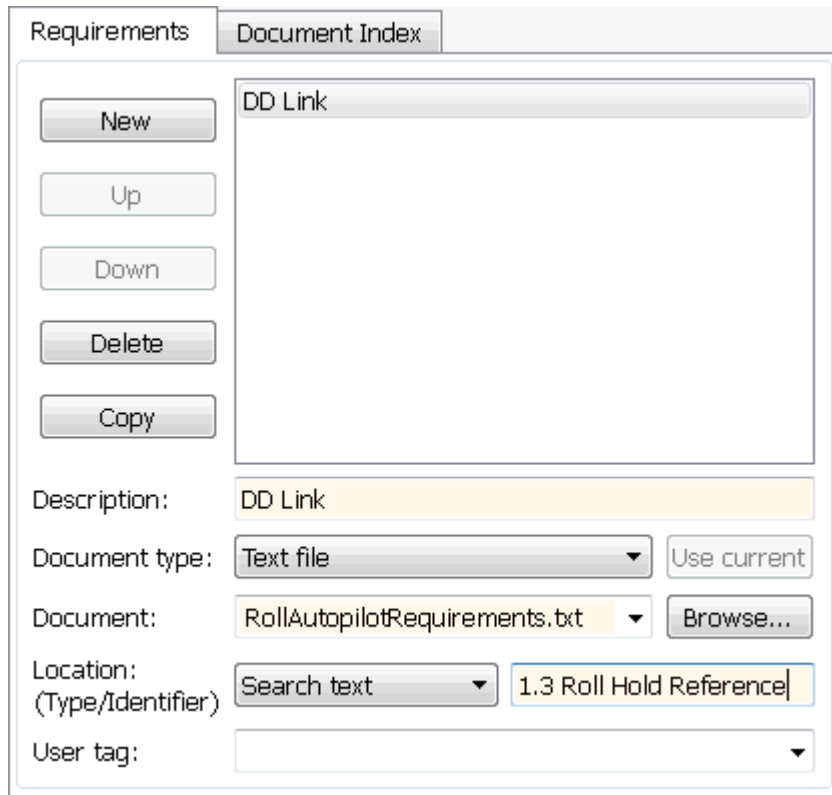
```
open AutopilotTestFile.mldatx,
open_system RollAutopilotMdlRef,
sltest.harness.open('RollAutopilotMdlRef/Roll Reference',...
'RollReference_Requirement1_3')
```

- 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 Open the Test Sequence block. Add a requirements link that links the InitializeTest step to the test case.
- a In the Test Manager, highlight Requirement 1.3 Test in the test browser.
- b Right-click the InitializeTest step in the Test Sequence Editor. Select **Requirements > Link to Current Test Case**.

When the requirements link is added, the Test Sequence Editor highlights the step.

Step	Transition
<pre>Initialize Test Phi = 0; APEng = false; TurnKnob = 0; % Initializes test sequence outputs</pre>	<pre>1. true</pre>

**See Also**

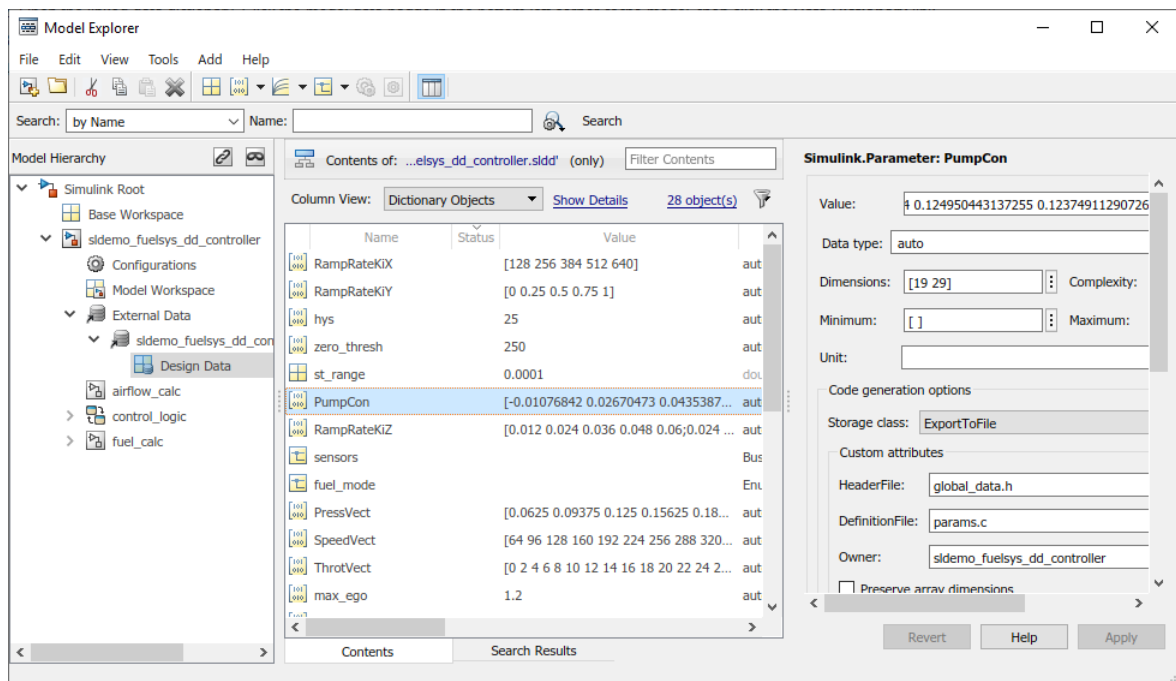
“Requirements-Based Testing for Model Development” (Simulink Test) | “Link to Test Cases from Requirements”

## 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 Enter `sldemo_fuelsys_dd_controller` at the command line to open `sldemo_fuelsys_dd_controller`.
- 2 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 You will link the PumpCon parameter to the Pumping Constant lookup table in the model.



- 6 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**.

The two objects are linked.

- 8 Check the link. Right-click the PumpCon parameter and select **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

### Link Signal Builder Blocks to Requirements Documents


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

- 1 Open the model:

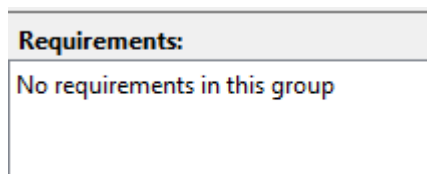
`sf_car`

- 2 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.

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

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



- 4 Place your cursor in the window, right-click, and select **Open Outgoing Links dialog**.

The Requirements Traceability Link Editor opens.

- 5 Click **New**. In the **Description** field, enter `User input requirements`.
- 6 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 11-34.

- 7 Browse to a requirements document and click **Open**.
- 8 In the **Location** drop-down list, select **Search text** to link to specified text in the document.
- 9 Next to the **Location** drop-down list, enter `User Input Requirements`.
- 10 Click **Apply** to create the link.
- 11 To verify that the RMI created the link, in the Simulink Editor, 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 Save the `sf_car_linking` model.

---

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



In this example, switch to a different active group in the drop-down list to link a requirement to another signal group.


---

## Link Signal Builder Blocks to Model Objects

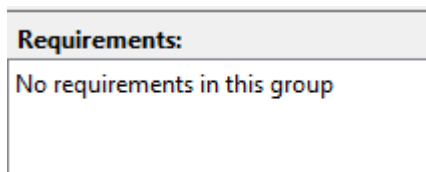
This example shows how to create links from a signal group in a Signal Builder block to a model object:

- 1 Open the `sf_car` model.
- 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 dialog box, click the **Gradual Acceleration** tab.
- 6 At the far-right end of the toolbar, click the **Show verification settings** button . (You might need to expand the Signal Builder dialog box for this button to become visible.)

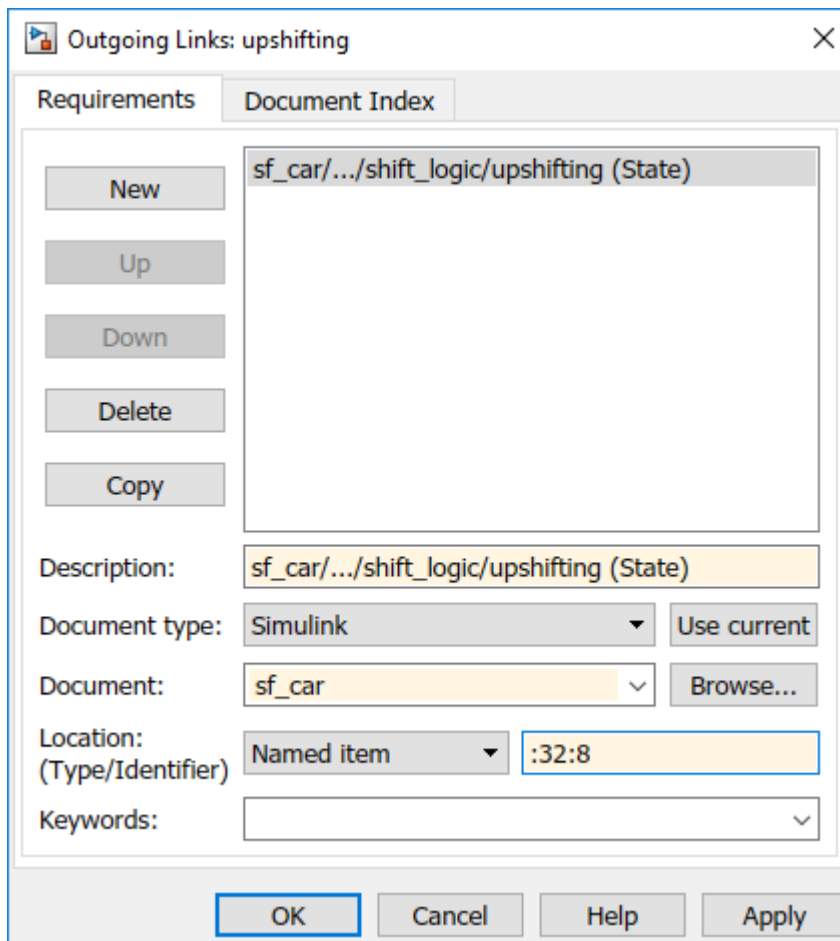
A **Requirements** pane opens on the right-hand side of the Signal Builder dialog box.



- 7 Place your cursor in the window, right-click, and select **Open Outgoing Links dialog**.

The Requirements Traceability Link Editor 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.

# Requirements Links for Library Blocks and Reference Blocks

## Introduction to Library Blocks and Reference Blocks

Simulink allows you to create your own block libraries. If you create a block library, you can reuse the functionality of a block, subsystem, or Stateflow atomic subchart in multiple models.

When you copy a library block to a Simulink model, the new block is called a reference block. You can create several instances of this library block in one or more models.

The reference block is linked to the library block using a library link. If you change a library block, any reference block that is linked to the library block is updated with those changes when you open or update the model that contains the reference block.

---

**Note** For more information about reference blocks and library links, see “Custom Libraries” (Simulink).

---

## Library Blocks and Requirements

Library blocks themselves can have links to requirements. In addition, if a library block is a subsystem or atomic subchart, the objects inside the library blocks can have library links. You use the Requirements Management Interface (RMI) to create and manage requirements links in libraries and in models.

The following sections describe how to manage requirements links on and inside library blocks and reference blocks.

## Copy Library Blocks with Requirements

When you copy a library subsystem or masked block to a model, you can highlight, view and navigate requirements links on the library block and on objects inside the library block. However, those links are not associated with that model. The links are stored with the library, not with the model.

You cannot add, modify, or delete requirements links on the library block from the context of the reference block. If you disable the link from the reference block to the library block, you can modify requirements on objects that are inside library blocks just as you can for other block attributes when a library link has been disabled.

## Manage Requirements on Reference Blocks

You use the RMI to manage requirements links on a reference block just like any other model object. You can view and navigate both local and library requirements on a reference block.

- Locally created requirements links — Can be modified or deleted without changing the library block:
  - **Manifold absolute pressure sensor**
  - **Mass airflow estimation**
- Requirements links on the library block — Cannot be modified or deleted from the context of the reference block:

- **Speed sensor**
- **Throttle sensor**
- **Oxygen sensor**

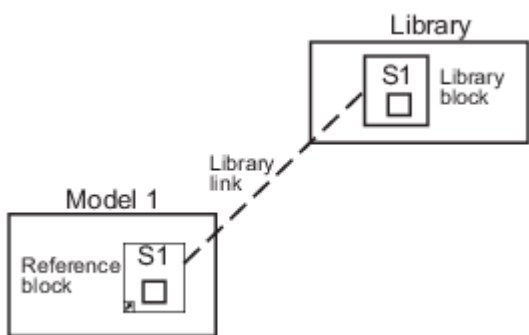
## Manage Requirements Inside Reference Blocks

If your library block is a subsystem or a Stateflow atomic subchart, you can create requirements links on objects *inside* the subsystem or subchart. If you disable the link from the reference block to the library, you can add, modify, or delete requirements links on objects inside a reference block. Once you have disabled the link, the RMI treats those links as locally created links.

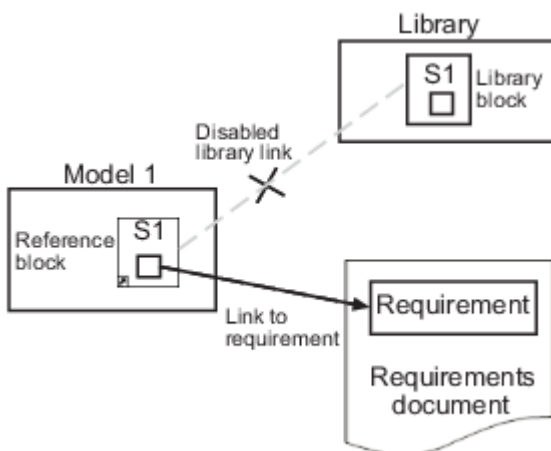
After you make changes to requirements links on objects inside a reference block, you can resolve the link so that those changes are pushed to the library block. The next time you create an instance of that library block, the changes you made are copied to the new instance of the library block.

The workflow for creating a requirement link on an object inside a reference block is:

- 1 Within a library you have a subsystem S1. Drag S1 to a model, creating a new subsystem. This subsystem is the reference block.



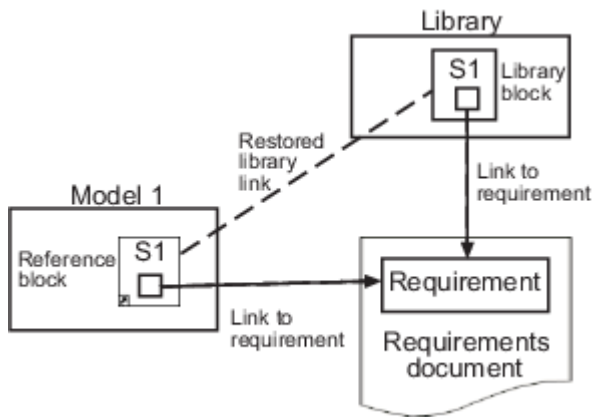
- 2 Disable the library link between the reference block and the library block. Keep the library loaded while you disable the link to maintain RMI data. To disable the link, select the reference block, and in the **Subsystem** tab, click **Disable Link**.
- 3 Create a link from the object inside the reference block to the requirements document.



**Note** When linking to a requirement from inside a reference block, you can create links only in one direction: from the model to the requirements document. The RMI does not support inserting navigation objects into requirements documents for objects inside reference blocks.

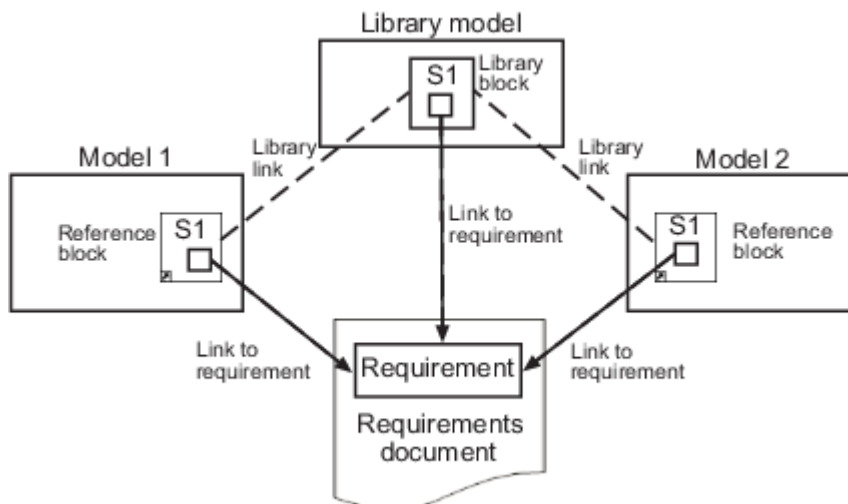
- 4 Resolve the library link between the reference block and the library block:
  - a Select the reference block.
  - b In the **Subsystem** tab, click **Restore Link**.
  - c In the **Action** column, click **Push**.
  - d Click **OK** to resolve the link to the library block and push the newly added requirement to the object inside the library block.

When you resolve the library link between the library block and the subsystem, Simulink pushes the new requirement link to the library block S1. The following graphic shows the new link from inside the library block S1 to the requirement.



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

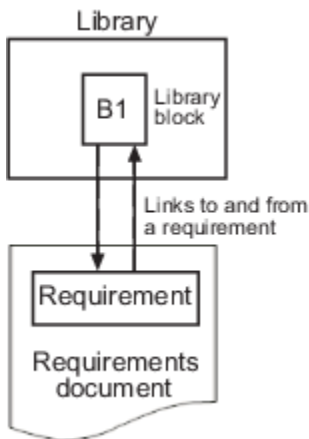
- 5 If you reuse library block S1, which now has an object with a requirement link, in another model, the new subsystem contains an object that links to that requirement.



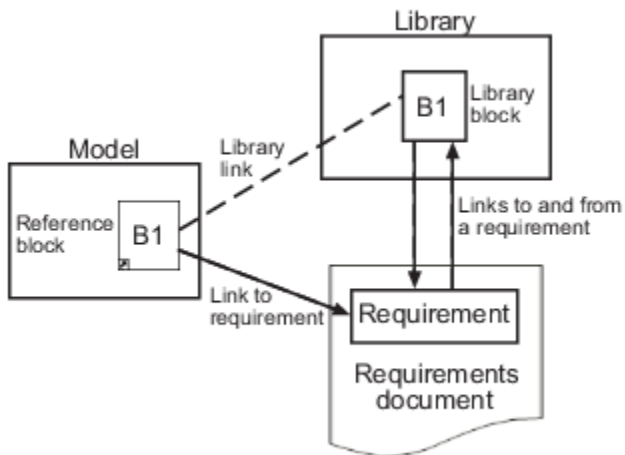
## Links from Requirements to Library Blocks

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

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



When you use library block B1 in a model, you can navigate from the reference block to the requirement. However, the link from the requirement still points only to library block B1, not to the reference block.



As discussed in the previous section, you can create requirements links on objects inside instances of library block after disabling library links. However, the RMI prohibits you from creating a link from the requirements document to such an object because that link would become invalid when you restored the library link.

## Navigate to Requirements from Model

### Navigate from Model Object

You can navigate directly from a model object to that object's associated requirement. When you take these steps, the external requirements document opens in the application, with the requirements text highlighted.

- 1 Open the example model:  
`slvndemo_fuelsys_officereq`
- 2 Open the fuel rate controller subsystem.
- 3 To open the linked requirement, right-click the Airflow calculation subsystem and select **Requirements > 1. "Mass airflow estimation"**.

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

---

**Note** If you are running a 64-bit version of MATLAB, when you navigate to a requirement in a PDF file, the file opens at the top of the page, not at the bookmark location.

---

### Navigate from System Requirements Block

Sometimes you want to see all the requirements links at a given level of the model hierarchy. In such cases, you can insert a System Requirements block to collect all requirements links in a model or subsystem. The System Requirements block lists requirements links for the model or subsystem in which it resides; it does not list requirements links for model objects inside that model or subsystem, because those are at a different level of the model hierarchy.

In the following example, you insert a System Requirements block at the top level of the `slvndemo_fuelsys_officereq` model, and navigate to the requirements using the links inside the block.

- 1 Open the example model:  
`slvndemo_fuelsys_officereq`
- 2 Enable **Model Highlighting** in the **Coverage** app.
- 3 Open the fuel rate controller subsystem.  
  
The Airflow calculation subsystem has a requirements link.
- 4 Open the Airflow calculation subsystem.
- 5 In the Simulink Editor, select **View > Library Browser**.
- 6 On the **Libraries** pane, 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, `slvndemo_FuelSys_DesignDescription.docx`, opens, with the section **2.1 Mass airflow estimation** selected.



# MATLAB Code Traceability

---

## 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 using the navigation links available from the **Links** pane in the Requirements View of 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 using the navigation links available from the **Links** pane in the Requirements View of the Requirements Editor.

### 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 9-3.
- 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:

- 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 11-7.

## 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

### Overlapping Linked Ranges

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.

### Cut and Paste Operations

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.

### Drag Operation

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.

### MATLAB Function Block Code Traceability in Web View

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” (Simulink Report Generator).

### Traceability for MATLAB Live Editor

Requirements traceability is not supported for MATLAB Live Editor.

## See Also

### More About

- “Generate Code for Models with Requirements Links” on page 11-7



# URL and Custom Traceability

---

- “Requirement Links and Link Types” on page 10-2
- “Custom Link Types” on page 10-8

## 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 <b>Document type</b> 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"> <li>• The software uses the standard method for resolving the path.</li> <li>• In the Outgoing Links Editor, when you click <b>Browse</b>, the file selection dialog box opens.</li> </ul>
Extensions	An array of file extensions. Use these file extensions as filter options in the Outgoing Links Editor when you click <b>Browse</b> . The file extensions infer the link type based on the document name. If you registered more than one link type for the same file extension, the link type that you registered takes first priority.
LocDelimiters	A string containing the list of supported navigation delimiters. The first character in the ID of a requirement specifies the type of identifier. For example, an identifier can refer to a specific page number (#4), a named bookmark (@my_tag), or some searchable text (?search_text). The valid location delimiters determine the possible entries in the Outgoing Links Editor <b>Location</b> 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 <b>Document Index</b> 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 <b>Document</b> field contents. <p>The function returns three outputs:</p> <ul style="list-style-type: none"> <li>• Labels</li> <li>• Depths</li> <li>• Locations</li> </ul>
BrowseFcn	The MATLAB callback invoked when you click <b>Browse</b> 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"> <li>• Full path name to the requirements document</li> <li>• Info about creating a URL to the document (if applicable)</li> <li>• Location of the requirement in the document</li> </ul> <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"> <li>• Fully qualified name for the requirements document</li> <li>• Location of the requirement in the document</li> </ul> <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"> <li>• Full path to the requirements document</li> <li>• Location of the requirement in the document</li> <li>• Requirement description label as stored in Simulink</li> </ul> <p>IsValidDescFcn returns two outputs:</p> <ul style="list-style-type: none"> <li>• True if the description matches the requirement, false otherwise.</li> <li>• The requirement label in the document, if not matched in Simulink.</li> </ul>

Property	Description
DetailsFcn	<p>The MATLAB callback invoked when you generate the requirements report with the <b>Include details from linked documents</b> option. This function returns detailed content associated with the requirement and has three input arguments:</p> <ul style="list-style-type: none"> <li>• Full path to the requirements document</li> <li>• Location of the requirement in the document</li> <li>• Level of details to include in report (Unused)</li> </ul> <p>The DetailsFcn returns two outputs:</p> <ul style="list-style-type: none"> <li>• Numeric array that describes the hierarchical relationship among the fragments in the cell array</li> <li>• Cell array of formatted fragments (paragraphs, tables, et al.) from the requirement</li> </ul>
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"> <li>• Handle to the model object that will have the requirement link</li> <li>• True if a navigation object is inserted into the requirements document, or false if no navigation object is inserted</li> </ul> <p>SelectionLinkFcn returns the requirements link structure for the selected requirement.</p>
GetResultFcn	<p>The MATLAB callback invoked when you link external test cases with the requirements to the custom link type file. It is used in the custom link type file and fetches external results to integrate with verification statuses.</p> <p>This function has one input argument:</p> <ul style="list-style-type: none"> <li>• <code>link</code>: This is a <code>slreq.Link</code> object. The function identifies the source and destination of the link.</li> </ul> <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"> <li>• <code>status</code> (Required): This is a value from <code>slreq.verification.Status</code> (Pass, Fail, Stale, or Unknown)</li> <li>• <code>timestamp</code> (Optional): Skip this field or mark <code>NaN</code> to avoid stale result detection.</li> <li>• <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.</li> <li>• <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.</li> </ul>

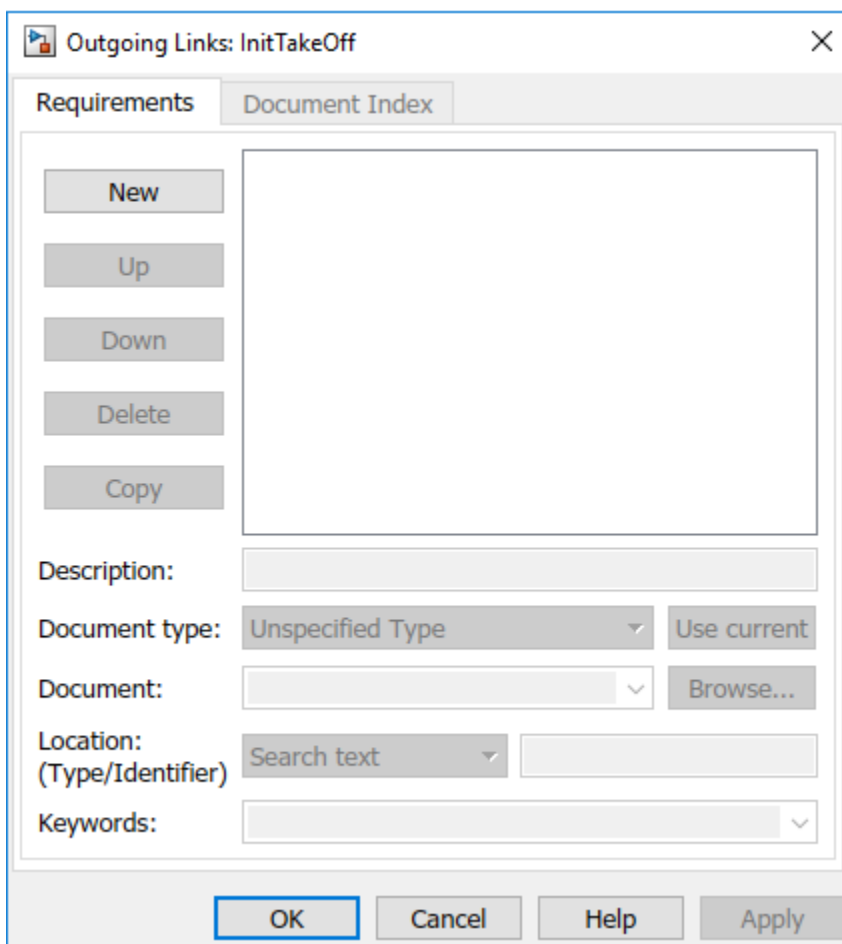
## Outgoing Links Editor

### 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"

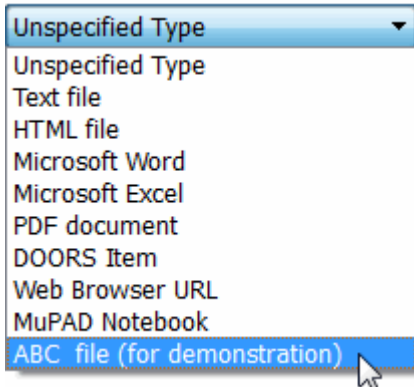
- 5 Open the following example model:

aero\_dap3dof

- 6 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:

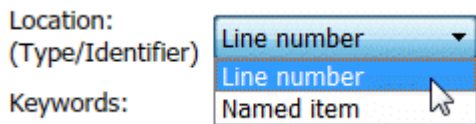
- > — Line number in a file
- @ — Named item, such as a bookmark, function, or HTML anchor

---

**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).



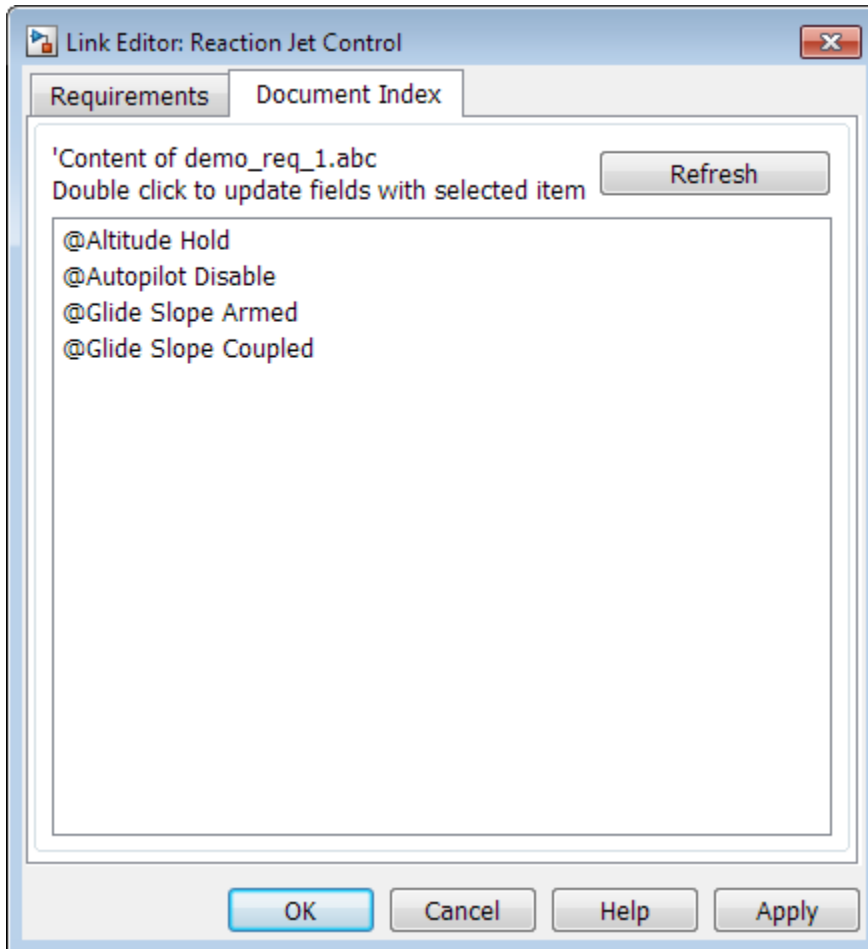
- 10 Select **Line number**. Enter the number 26, which corresponds with the line number for the Altitude Hold requirement in `demo_req_1.abc`.
- 11 In the **Description** field, enter Altitude Hold, to identify the requirement by name.
- 12 Click **Apply**.
- 13 Verify that the Altitude Hold requirement 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 `rmiustabcinterface.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 10-14.

## 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 `slvnvdemo_fuelsys_officereq` to generate the Excel workbook surrogate for the model.

# 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 11-7
- “Create and Customize Requirements Traceability Reports” on page 11-9
- “Create Requirements Traceability Report for A Project” on page 11-24
- “Validate Requirements Links” on page 11-25
- “Delete Requirements Links from Simulink Objects” on page 11-33
- “Document Path Storage” on page 11-34

## 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:

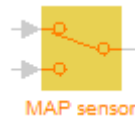
- 1 Open the example model:

```
slvndemo_fuelsys_officereq
```

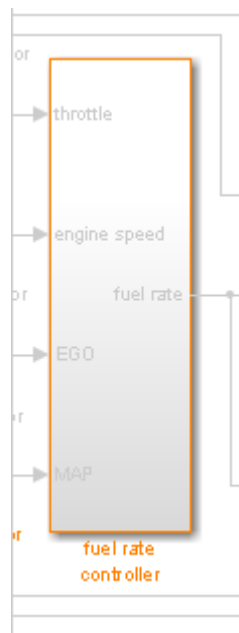
- 2 Select **Coverage Highlighting** from the **Coverage** app.

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\shared\reqmgt\+linktypes` contain examples of how to implement this functionality:

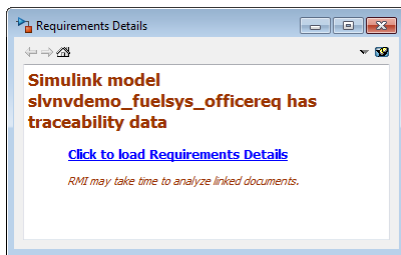
```
linktype_rmi_doors.m
linktype_rmi_excel.m
linktype_rmi_html.m
linktype_rmi_text.m
```

## View Requirements Details for a Selected Block

### Requirements Details Workflow

When you highlight model objects with requirements, you can use the RMI Informer window to view the requirements details for a selected block. Follow this workflow:

- 1 In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Highlight Links**.
- 2 In your model, highlights indicate model objects with requirements links. The RMI Informer window opens.



- 3 In the RMI Informer window, click the link to load the requirements details. If your Simulink model has links to Microsoft Word, Microsoft Excel, or IBM Rational DOORS documents, the RMI Informer displays requirements context from the requirements documents and additional link labels stored by Simulink.
- 4 Select highlighted model objects to display associated RMI links in the RMI Informer window.
- 5 Close the requirements details window to remove highlights from the Simulink model.

For more information see "Navigate to Requirements from Model" on page 8-15.

### Requirements Details Limitations

Security restrictions in Microsoft Office can interrupt the requirements details loading process. Requirement details loaded from read-only Microsoft Office documents, documents stored on network drives, and documents with Microsoft Office Trust Center ActiveX control restrictions may not work with RMI. Consider enabling ActiveX controls without prompting and using requirements stored in a writable location on the MATLAB path.

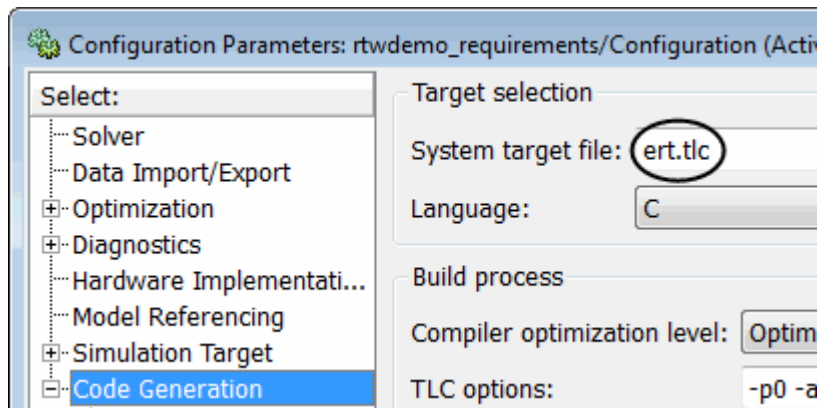
Before loading requirements details for IBM Rational DOORS links, you must be logged in to the IBM Rational DOORS Client.

## 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>&lt;model&gt;.h</code>
Nonvirtual subsystem	At the call site for the subsystem
Virtual subsystem	At the call site of the closest nonvirtual parent subsystem. If a virtual subsystem does not have a nonvirtual parent, requirement descriptions appear in the main header file for the model, <code>&lt;model&gt;.h</code> .
Nonsubsystem block	In the generated code for the block
MATLAB code line in MATLAB Function block	In the generated code for the MATLAB code line(s)

# Create and Customize Requirements Traceability Reports

## Create Requirements Traceability Report for Model

To create the default requirements report for a Simulink model:

- 1 Open the example model:  
`slvndemo_fuelsys_officereq`
- 2 Make sure that your current working folder is writable.
- 3 In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, select **Share > Generate Model Traceability Report**.

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

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

## Requirements Report for slvndemo\_fuelsys\_officereq

`username`

17-Jun-2010 10:57:04

---

### Table of Contents

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

A typical requirements report includes:

- Table of contents
- List of tables
- Per-subsystem sections that include:
  - Tables that list objects with requirements and include links to associated requirements documents

- Graphic images of objects with requirements
- Lists of objects with no requirements
- MATLAB code lines with requirements in MATLAB Function blocks

For detailed information about requirements reports, see “Customize Requirements Traceability Report for Model” on page 11-10.

### **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 11-18.

### **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 11-17.

## **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 11-17.

---

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:

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



The Requirements Management Interface (RMI) searches through all the blocks and subsystems in the model for associated requirements. The RMI generates and displays a complete report in HTML format.

The report is saved with the default name, *model\_name\_requirements.html*. If you generate a subsequent report on the same model, the new report file overwrites any earlier report file.

The report contains the following content:

### Table of Contents

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

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

## Requirements Report for slvndemo\_fuelsys\_officereq

username

17-Jun-2010 10:57:04

---

### Table of Contents

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

### List of Tables

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

**List of Tables**

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

**Model Information**

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

**Chapter 1. Model Information for "slvndemo\_fuelsys\_officereq"**

Table 1.1. slvndemo\_fuelsys\_officereq Version Information

<i>ModelVersion</i>	1.159	<i>ConfigurationManager</i>	none
<i>Created</i>	Tue Jun 02 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	<a href="#">fuelsys_req_docs\slvnvdemo_FuelSys_DesignDescription.docx</a>	29-Oct-2009 10:56:01	8
DOC3	<a href="#">fuelsys_req_docs\slvnvdemo_FuelSys_RequirementsSpecification.docx</a>	29-Oct-2009 10:56:02	6
DOC4	<a href="#">fuelsys_req_docs\slvnvdemo_FuelSys_TestScenarios.xlsx</a>	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.

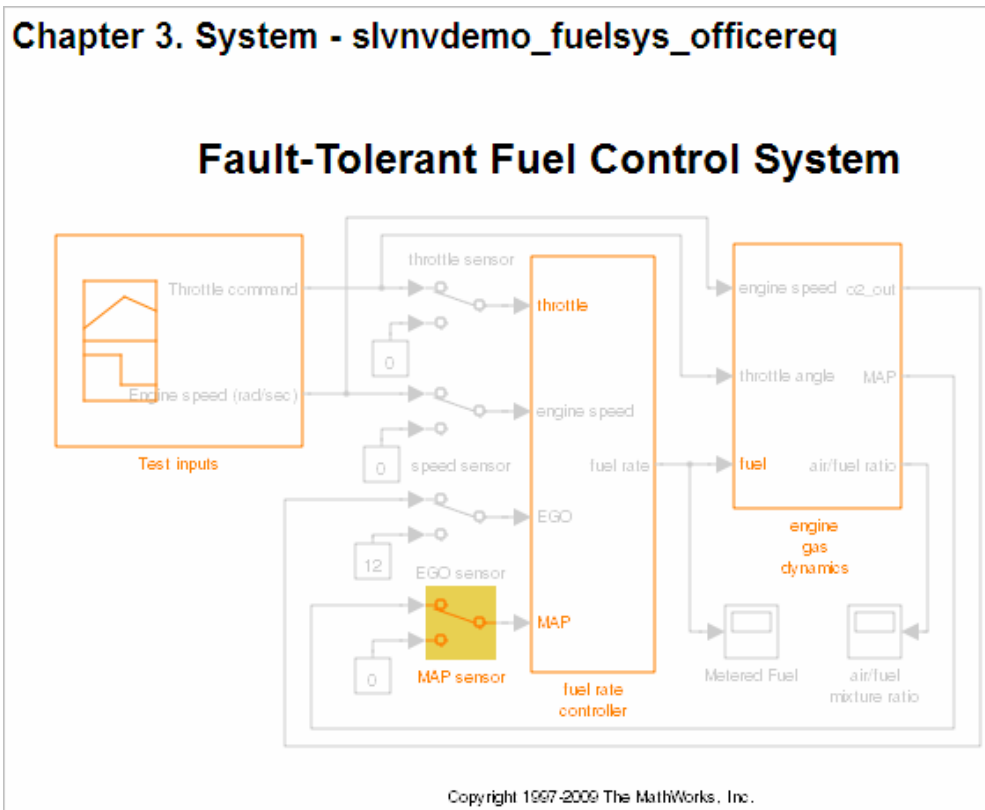
When your requirements documents have long path names that can clutter the report, select the **User document IDs in requirements tables** option. This option is disabled by default, as you can see in the examples in this section.

- **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	<a href="#">slvndemo FuelSys DesignDescription.docx</a>

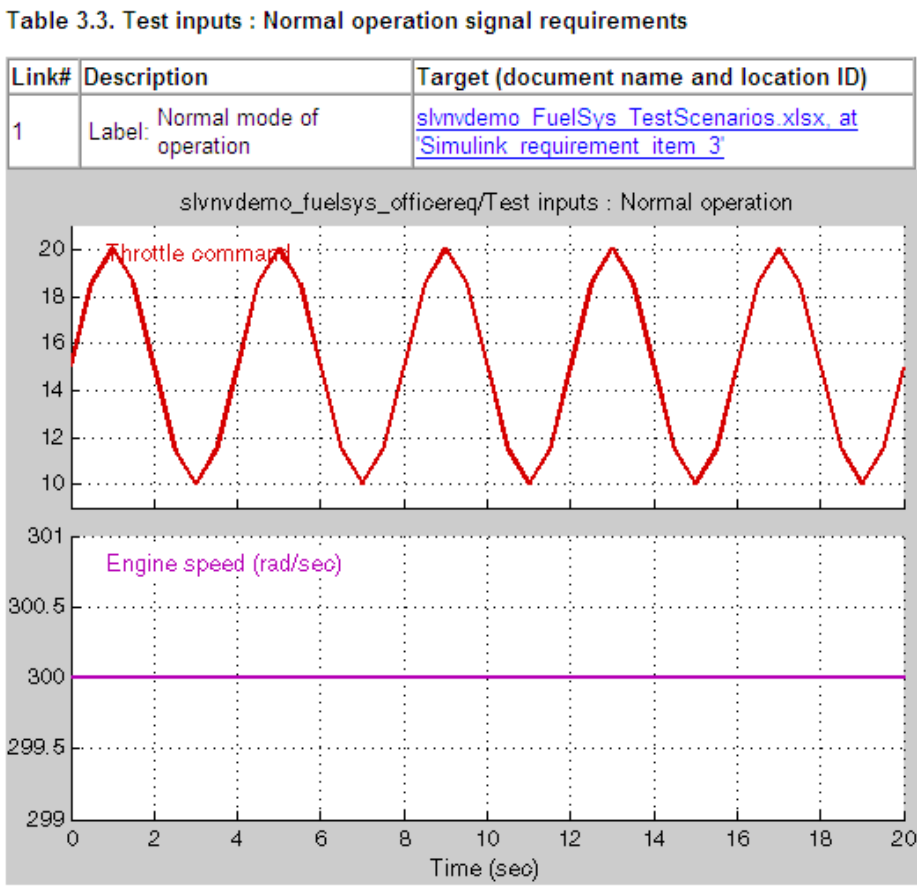
- A list of blocks in the top-level model that have requirements. In this example, only the MAP sensor block has a requirement at the top level. Click the link next to **Target:** to open the requirements document associated with the MAP sensor block.

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

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

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

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



**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

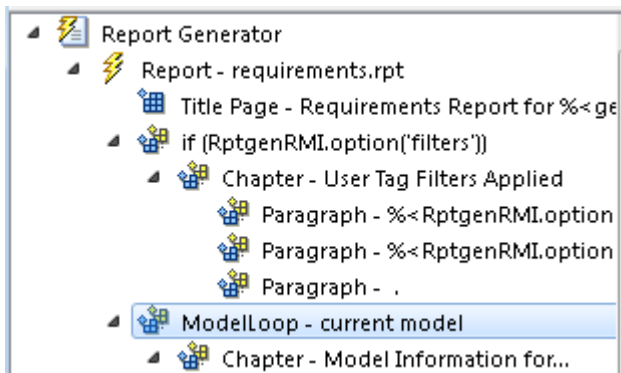
Name	Requirements
warmup	Link#1 label: "During a calibratable warm up period the oxygen sensor correction shall be disabled." Target: <a href="#">fuelsys_req_docs\slvndemo_FuelSys_RequirementsSpecification.docx, at 'Simulink requirement item 3'</a>
[speed==0 & press < zero_thresh]/...	Link#1 label: "Speed sensor failure detection" Target: <a href="#">fuelsys_req_docs\slvndemo_FuelSys_DesignDescription.docx, at 'Simulink requirement item 6'</a>
Rich_Mixture	Link#1 label: "Enriched mixture usage" Target: <a href="#">fuelsys_req_docs\slvndemo_FuelSys_DesignDescription.docx, at 'Simulink requirement item 4'</a>
Warmup	Link#1 label: "During a calibratable warm up period the oxygen sensor correction shall be disabled." Target: <a href="#">fuelsys_req_docs\slvndemo_FuelSys_RequirementsSpecification.docx, at 'Simulink requirement item 3'</a>

### Report for Requirements in Model Blocks

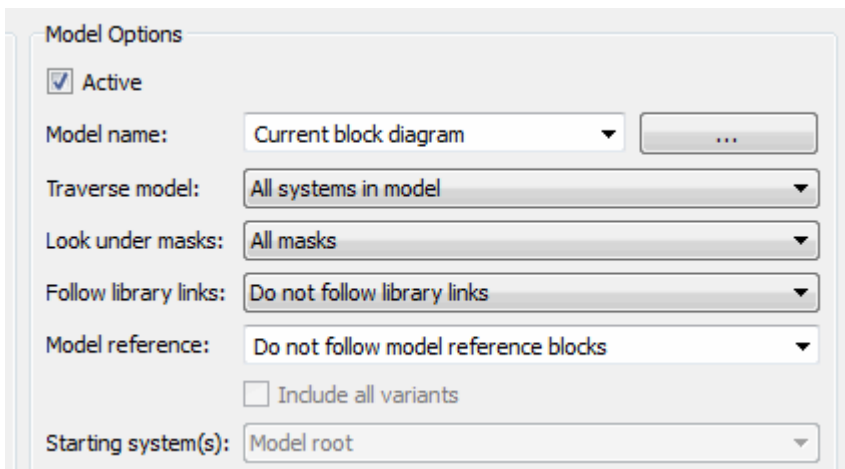
If your model contains Model blocks that reference external models, the default report does not include information about requirements in the referenced models. To generate a report that includes requirements information for referenced models, you must have a license for the Simulink Report Generator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


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

- 1 Open the model for which you want to create a requirements report. This workflow uses the example model `slvndemo_fuelsys_officereq`.
- 2 To open the template for the default requirements report, at the MATLAB command prompt, enter:  
`setedit requirements`
- 3 In the Simulink Report Generator software window, in the far-left pane, click the **Model Loop** component.



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



- 5 In the **Model reference** field drop-down list, select Follow all model reference blocks.
- 6 To generate a requirements report for the open model that includes information about referenced models, click the **Report** icon .

### Customize Requirements Report

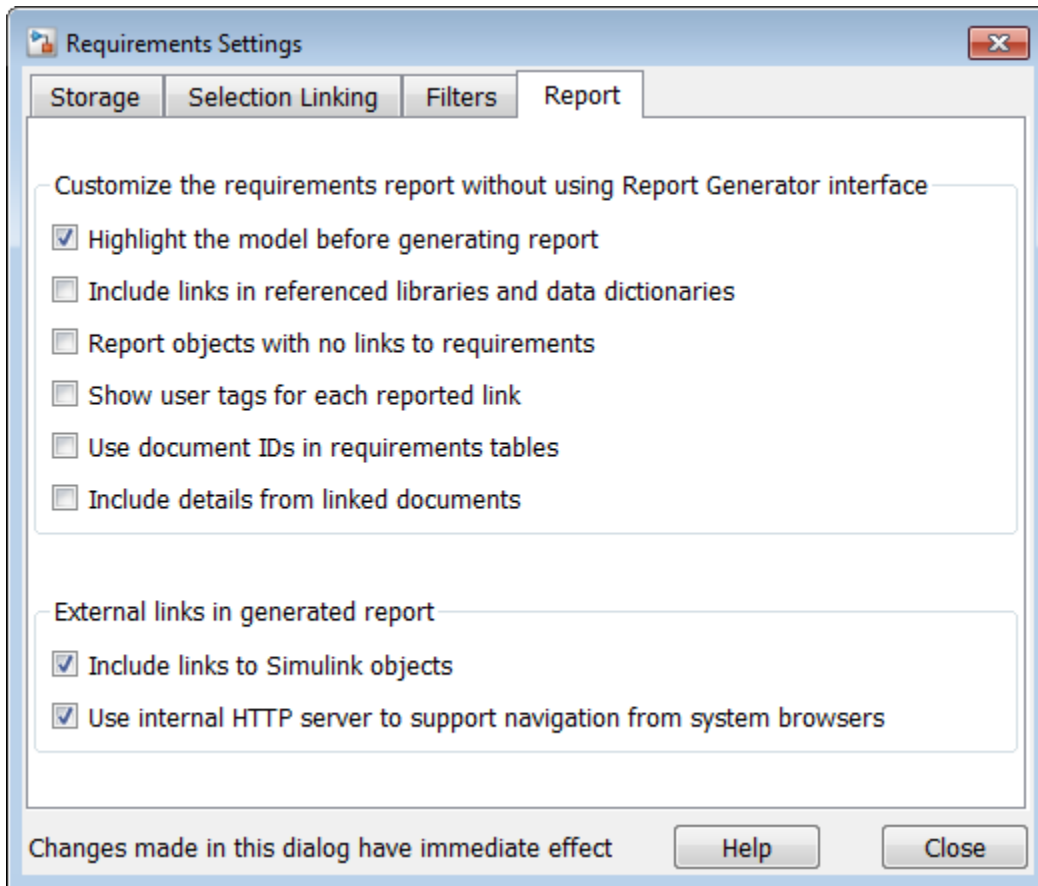
The Requirements Management Interface (RMI) uses the Simulink Report Generator software to generate the requirements report. You can customize the requirements report using the RMI or the Simulink Report Generator software:

- “Customize Requirements Report Using the RMI Settings” on page 11-18
- “Customize Requirements Report Using Simulink Report Generator” on page 11-21

### Customize Requirements Report Using the RMI Settings

There are several options for customizing a requirements report using the Requirements Settings dialog box.





On the **Report** tab, select options that specify the contents that you want in the report.

Requirements Settings Report Option	Description
<b>Highlight the model before generating report</b>	Enables highlighting of Simulink objects with requirements in the report graphics.
<b>Include links in referenced libraries and data dictionaries</b>	Includes requirements links in referenced libraries in the generated report.
<b>Report objects with no links to requirements</b>	Includes lists of model objects that have no requirements.
<b>Show user tags for each reported link</b>	Lists the user tags, if any, for each reported link.
<b>Use document IDs in requirements tables</b>	Uses a document ID, if available, instead of a path name in the tables of the requirements report. This capability prevents long path names to requirements documents from cluttering the report tables.

Requirements Settings Report Option	Description
<b>Include details from linked documents</b>	Includes additional content from linked requirements. The following requirements documents are supported: <ul style="list-style-type: none"> <li>• Microsoft Word</li> <li>• Microsoft Excel</li> <li>• IBM Rational DOORS</li> </ul>
<b>Include links to Simulink objects</b>	Includes links from the report to objects in Simulink.
<b>Use internal HTTP server to support navigation from system browsers</b>	Specifies use of internal MATLAB HTTP server for navigation from generated report to documents and model objects. By selecting this setting, this navigation is available from system browsers as long as the MATLAB internal HTTP server is active on your local host. To start the internal HTTP server, at the MATLAB command prompt, type <code>rmi('httpLink')</code> .

To see how these options affect the content of the report:

- 1 Open the `slvndemo_fuelsys_officereq` model:

```
slvndemo_fuelsys_officereq
```
- 2 In the **Requirements Viewer** tab, click **Link Settings**.
- 3 In the Requirements Settings dialog box, click the **Report** tab.
- 4 For this example, select **Highlight the model before generating report**.

When you select this option, before generating the report, the graphics of the model that are included in the report are highlighted so that you can easily see which objects have requirements.

- 5 To close the Requirements Settings dialog box, click **Close**.
- 6 Generate a requirements report. In the Requirements tab, select S.

The requirements report opens in a browser window so that you can review the content of the report.

- 7 If you do not want to overwrite the current report when you regenerate the requirements report, rename the HTML file, for example, `slvndemo_fuelsys_officereq_requirements_old.html`.

The default report file name is `model_name_requirements.html`.

- 8 In the **Apps** tab, select **Requirements Manager**.
- 9 In the **Requirements** tab, select **Share > Generate Model Traceability Report**.
  - **Show user 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.
IBMRationalDOORS	By default, the report includes: <ul style="list-style-type: none"> <li>• <b>DOORS Object Heading</b></li> <li>• <b>DOORS Object Text</b></li> <li>• All other attributes except <b>Created Thru</b>, attributes with empty string values, and system attributes that are false.</li> </ul> Use the <code>RptgenRMI.doorsAttribs</code> function to include or exclude specific attributes or groups of attributes.

- 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 the following command at the MATLAB command prompt:

```
slvndemo_powerwindow_report
```

### Customize Requirements Report Using Simulink Report Generator

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

At the MATLAB command prompt, enter the following command:

```
setedit requirements
```

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

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

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

<b>Simulink Report Generator Component</b>	<b>Report Information</b>
<b>Missing Requirements Block Loop</b>	Applies all child components to blocks that have no requirements
<b>Missing Requirements System Loop</b>	Applies all child components to systems that have no requirements
<b>Requirements Block Loop</b>	Applies all child components to blocks that have requirements
<b>Requirements Documents Table</b>	Inserts a table that lists requirements documents
<b>Requirements Signal Loop</b>	Applies all child components to signal groups with requirements
<b>Requirements Summary Table</b>	Inserts a property table that lists requirements information for blocks with associated requirements
<b>Requirements System Loop</b>	Applies all child components to systems with requirements
<b>Requirements Table</b>	Inserts a table that lists system and subsystem requirements
<b>Data Dictionary Traceability Table</b>	Inserts a table that links data dictionary information to requirements
<b>MATLAB Code Traceability Table</b>	Inserts a table that links MATLAB code to requirements
<b>Simulink Test Suite Traceability Table</b>	Inserts a table that links a Simulink test suite to requirements

To customize the requirements report, you can:

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

For more information, see the Simulink Report Generator documentation.

### **Generate Requirements Reports Using Simulink**

When you have a model open in Simulink, the Model Editor provides two options for creating requirements reports:

#### **System Design Description Report**

The System Design Description report describes a system design represented by the current Simulink model.

You can use the System Design Description report to:

- Review a system design without having the model open.

- Generate summary and detailed descriptions of the design.
- Assess compliance with design requirements.
- Archive the system design in a format independent of the modeling environment.
- Build a customized version of the report using the Simulink Report Generator software.

To generate a System Design Description report that includes requirements information:

- 1 Open the model for which you want to create a report.
- 2 In the **Modeling** tab, select **Compare > System Design Description Report**.
- 3 In the Design Description dialog box, select **Requirements traceability**.
- 4 Select any other options that you want for this report.
- 5 Click **Generate**.

As the software is generating the report, the status appears in the MATLAB command window.

The report name is the model name, followed by a numeral, followed by the extension that reflects the document type (.pdf, .html, etc.).

If your model has linked requirements, the report includes a chapter, **Requirements Traceability**, that includes:

- Lists of model objects that have requirements with hyperlinks to display the objects
- Images of each subsystem, highlighting model objects with requirements

### **Design Requirements Report**

In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Share > Generate Model Traceability Report**. This option creates a requirements report, as described in “Create Default Requirements Report” on page 11-10.

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 11-18.

## Create Requirements Traceability Report for A Project

To create a report for requirements traceability data in a project:

- 1 Open your project.
- 2 At the MATLAB command prompt, enter the following:

```
rmi('projectreport')
```

The MATLAB Web browser opens, showing the traceability report for the project.

This top-level HTML report contains a separate section for Simulink model files, MATLAB code files, and other files included in the project. For each individual file with one or more associated requirements links, a separate HTML report, or sub-report, shows the requirements traceability data for that file. The top-level report contains links to each sub-report.

If you have a MATLAB file with requirements traceability links that is not part of a project, you can create a separate report for the MATLAB file using the `rmi('report', matlabfilepath)` command. For more information, see `rmi`.

# Validate Requirements Links

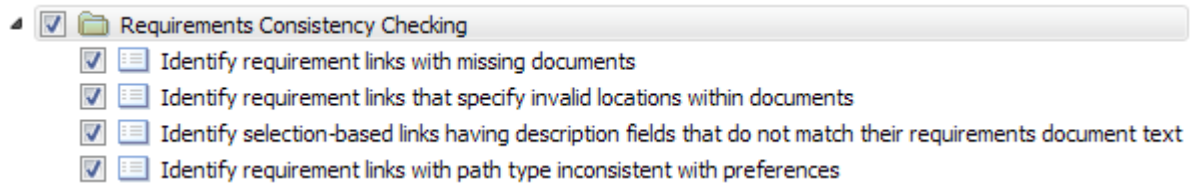
## Validate Requirements Links in a Model

### Check Requirements Links with the Model Advisor

To make sure that every requirements link in your Simulink model has a valid target in a requirements document, run the Model Advisor Requirements consistency checks:

- 1 Open the example model:  
slvndemo\_fuelsys\_officereq
- 2 Open the Model Advisor to run a consistency check. In the **Apps** tab, click **Requirements Manager**. In the **Requirements** tab, click **Check Consistency**.

In the **Requirements Consistency Checking** category, all the checks are selected. For this tutorial, keep all the checks selected.



These checks identify the following problems with your model requirements.

Consistency Check	Problem Identified
<b>Identify requirement links with missing documents</b>	The Model Advisor cannot find the requirements document. This might indicate a problem with the path to the requirements document.
<b>Identify requirement links that specify invalid locations within documents</b>	The Model Advisor cannot find the designated location for the requirement. This check is implemented for: <ul style="list-style-type: none"> <li>• Microsoft Word documents</li> <li>• Microsoft Excel documents</li> <li>• IBM Rational DOORS documents</li> <li>• Simulink objects</li> </ul>

Consistency Check	Problem Identified
<p><b>Identify selection-based links having description fields that do not match their requirements document text</b></p>	<p>The <b>Description</b> field for the link does not match the requirements document text. When you create selection-based links, the Requirements Management Interface (RMI) saves the selected text in the link <b>Description</b> field. This check is implemented for:</p> <ul style="list-style-type: none"> <li>• Microsoft Word documents</li> <li>• Microsoft Excel documents</li> <li>• IBM Rational DOORS documents</li> <li>• Simulink objects</li> </ul>
<p><b>Identify requirement links with path type inconsistent with preferences</b></p>	<p>The path to the requirements document does not match the <b>Document file reference</b> field in the Requirements Settings dialog box <b>Selection Linking</b> tab. This might indicate a problem with the path to the requirements document.</p> <p>On Linux systems, this check is named <b>Identify requirement links with absolute path type</b>. The check reports a warning for each requirements links that uses an absolute path.</p> <hr/> <p><b>Note</b> For information about how the RMI resolves the path to the requirements document, see “Document Path Storage” on page 11-34.</p>

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

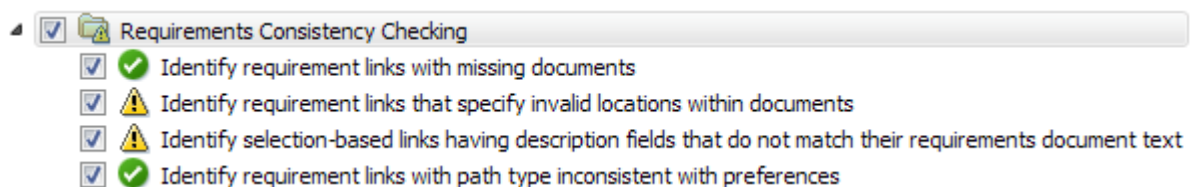
- If your model has links to Microsoft Word or Microsoft Excel documents, the consistency check requires that you close all instances of those applications. If you have one of these applications open, it displays a warning and does not continue the checks. The consistency checks must verify up-to-date 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) 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 11-25, three requirements consistency checks generate warnings in the `slvndemo_fuelsys_officereq` model.

#### Resolve Warning: Identify requirement links that specify invalid locations within documents

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

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

**Inconsistencies:**

The following requirements link to invalid locations within their documents. The specified location (e.g., bookmark, line number, anchor) within the requirements document could not be found. To resolve this issue, edit each requirement and specify a valid location within its requirements document.

<b>Block</b>	<b>Requirements</b>
<a href="#">slvndemo_fuelsys_officereq/fuel_rate_controller/Sensor_correction_and Fault Redundancy/Terminator 1</a>	<a href="#">This section will be deleted</a>

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

- 2 Get more information about this link:
  - a To navigate to the Terminator1 block, under **Block**, click the hyperlink.
  - b To open the “Outgoing Links Editor” on page 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	
<a href="#">slvndemo_fuelsys_officereq/Test inputs</a>	<a href="#">Normal mode of operation</a>	The simulation is run with a throttle input that ramps from 10 to 20 degrees over a period of two seconds, then back to 10 degrees over the next two seconds. This cycle repeats continuously while the engine is held at a constant speed.	<a href="#">Update</a>
<a href="#">slvndemo_fuelsys_officereq/fuel rate controller/Sensor correction and Fault Redundancy/MAP Estimate</a>	<a href="#">Manifold pressure failure</a>	Manifold pressure failure mode	<a href="#">Update</a>

The first message indicated that the model contains a link to a bookmark named **Simulink\_requirement\_item\_7** in the requirements document that does not exist.

In addition, this check identified the following mismatching text between the requirements blocks and the requirements document:

- The **Description** field in the Test inputs Signal Builder block link is **Normal mode of operation**. The requirement text is **The simulation is run with a throttle input that ramps from 10 to 20 degrees over a period of two seconds, then back to 10 degrees over the next two seconds. This cycle repeats continuously while the engine is held at a constant speed.**
- The **Description** field in the MAP Estimate block link is **Manifold pressure failure**. The requirement text in `slvndemo_FuelSys_DesignDescription.docx` is **Manifold pressure failure mode**.

- 2 Get more information about this link:
  - a To navigate to a block, under **Block**, click the hyperlink.
  - b To open the “Outgoing Links Editor” on page 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:

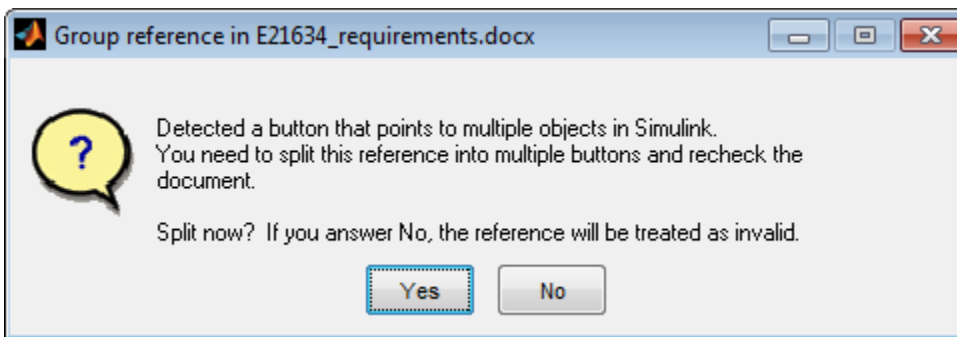
If you want to...	Do the following...
Fix the invalid links	Follow the instructions in “Fix Invalid Links in a Requirements Document” on page 11-30.
Keep the changes to the navigation controls without fixing the invalid links	Save the requirements document.

If you want to...	Do the following...
Ignore the invalid links	Close the requirements document without saving it.

### When Multiple Objects Have Links to the Same Requirement

When you link multiple objects to the same requirement, only one navigation object is inserted into the requirements document. When you double-click that navigation object, all of the linked model objects are highlighted.

If you check the requirements document using the 'checkdoc' option of the rmi function and the check detects a navigation object that points to multiple objects, the check stops and displays the following dialog box.



You have two options:

- If you click **Yes**, or you close this dialog box, the RMI creates additional navigation objects, one for each model object that links to that requirement. The document check continues, but the RMI does not recheck that navigation; the report only shows one link for that requirement. To rerun the check so that all requirements are checked, at the top of the report, click **Refresh**.
- If you click **No**, the document check continues, and the report identifies that navigation object as a broken link.

### Fix Invalid Links in a Requirements Document

Using the report that the rmi function creates, you may be able to fix the invalid links in your requirements document.

In the following example, rmi cannot locate the model specified in two links.

**References with Unresolved Models - 1 unique problem in 2 links**

Document content	Target model
<a href="#">Transmission Requirements</a>	sf_car_doors.mdl
<a href="#">Engine Torque Requirements</a>	

To fix invalid links:

- 1 In the report, under **Document content**, click the hyperlink associated with the invalid requirement link.

The requirements document opens with the requirement text highlighted.

- 2 In the requirements document, depending on the document format, take these steps:
  - In DOORS:
    - a Select the navigation control for an invalid link.
    - b Select **MATLAB > Select item**.
  - In Microsoft Word, double-click the navigation control.

A dialog box opens that allows you to fix, reset, or ignore all the invalid links with a given problem.

- 3 Click one of the following options.

To...	Click...
Navigate to and select a new target model or new target objects for these broken links.	<b>Fix all</b>
Reset the navigation controls for these invalid links to their original state, the state before you checked the requirements document.	<b>Reset all</b>
Make no changes to the requirements document. Any modifications rmi made to the navigation controls remain in the requirements document.	<b>Cancel</b>

- 4 Save the requirements document to preserve the changes made by the rmi function.

## Validation of Requirements Links

Requirements links in a model can become outdated when requirements change over time. Similarly, links in requirements documents may become invalid when your Simulink model changes, for example, when the model, or objects in the model, are renamed, moved, or deleted. The 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 11-31
- “How the rmi Function Checks a Requirements Document” on page 11-32

### 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 11-25.



---

**Note** For more information about inserting navigation controls in requirements documents, see:

- “Insert Navigation Objects in Microsoft Office Documents” on page 6-9
  - “Insert Navigation Objects into IBM Rational DOORS Requirements” on page 7-28
- 

### 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



# Requirements Management Interface

---



# Verification and Validation

---

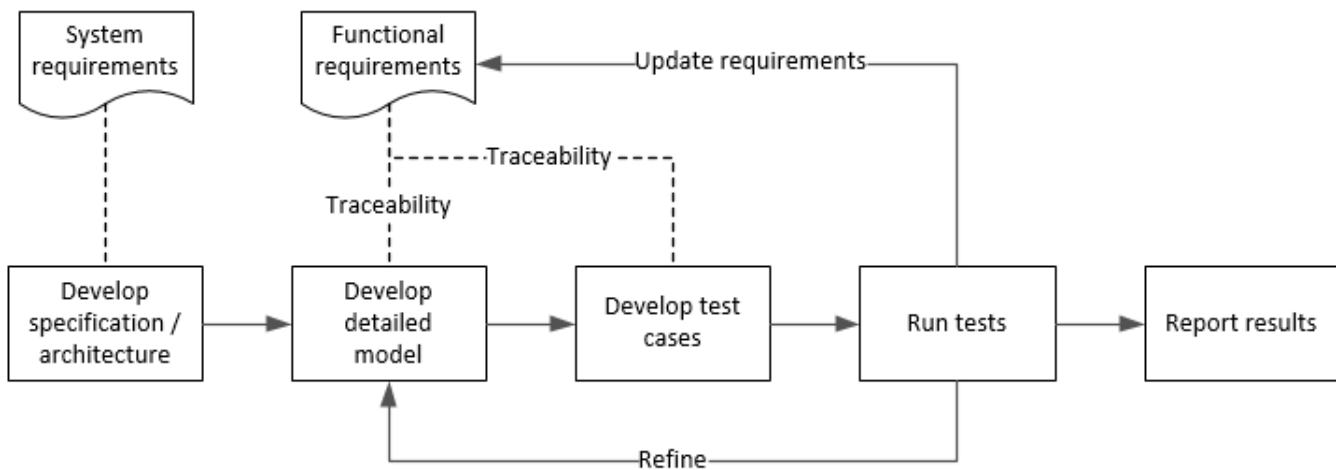
- “Test Model Against Requirements and Report Results” on page 13-2
- “Analyze a Model for Standards Compliance and Design Errors” on page 13-7
- “Perform Functional Testing and Analyze Test Coverage” on page 13-9
- “Analyze Code and Test Software-in-the-Loop” on page 13-12

## 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.

- Expand the requirements information to include verification and implementation status. Right-click a requirement and select **Verification Status** and **Implementation Status**.

The screenshot displays the Simulink Requirements tool interface. The top window shows a Simulink model with several input blocks: CruiseOnOff (boolean), Brake (boolean), Speed (single), CoastSetSw (boolean), and AccelResSw (boolean). These inputs feed into a central block labeled 'Compute target speed'. The bottom window shows a table of requirements with the following data:

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		

The right-hand window shows the Property Inspector for Requirement A 1.2. It includes details such as Type (Functional), Index (1.2), and Summary (Set Speed / Decelerate Button). The Description tab is active, showing the rationale for the requirement:

**Set Speed / Decelerate Button**

The controller shall have an input button to:

- set the target speed to the current vehicle speed when the cruise control is **not engaged (active)**
- decelerate (reduce) the target speed when the cruise control is **engaged (active)**

- In the Project window, open the Simulink Test file `s1ReqTests.mldatx` from the `tests` folder. The test file opens in the Test Manager.

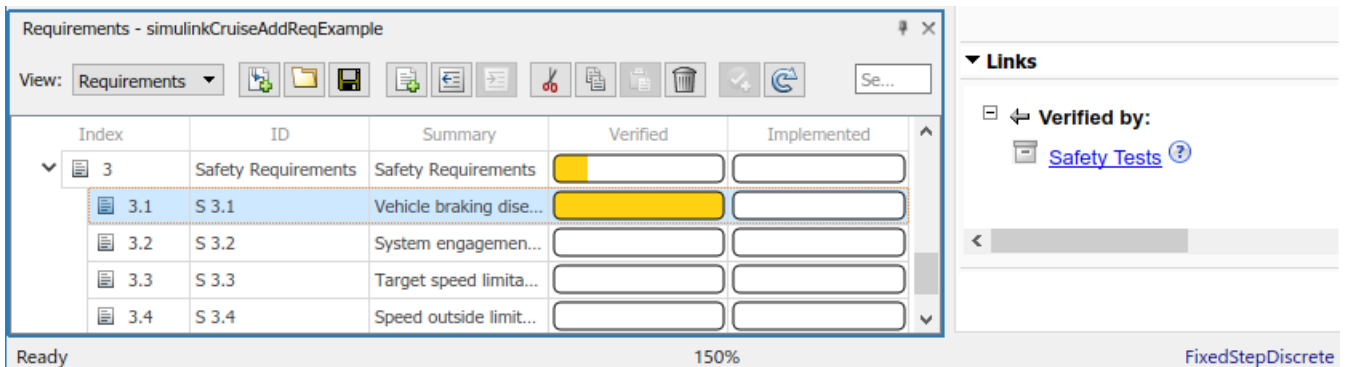
## Link Requirements to Tests

Link the requirements to the test case.

- 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

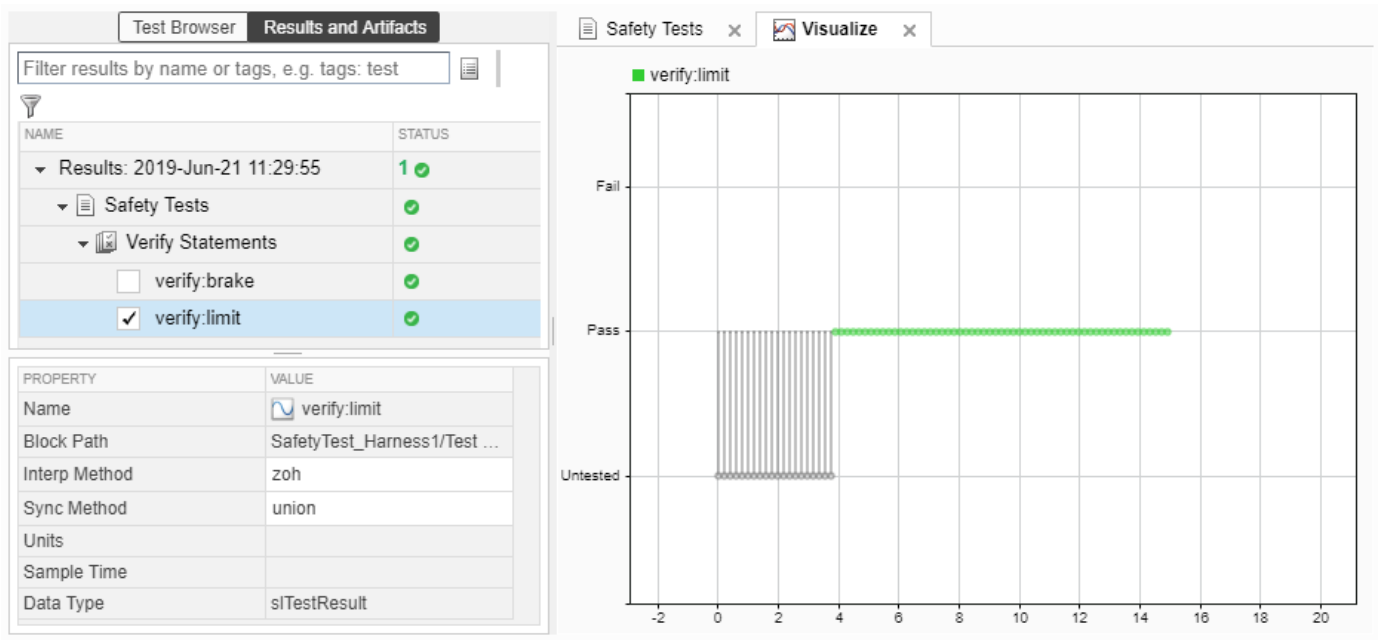
```
verify(engaged == false,...
      'verify:brake',...
      'system must disengage when brake applied')
```

- The `LimitTest` sequence engages the cruise control, then ramps up the vehicle speed until it exceeds the upper limit. It includes the `verify` statement.

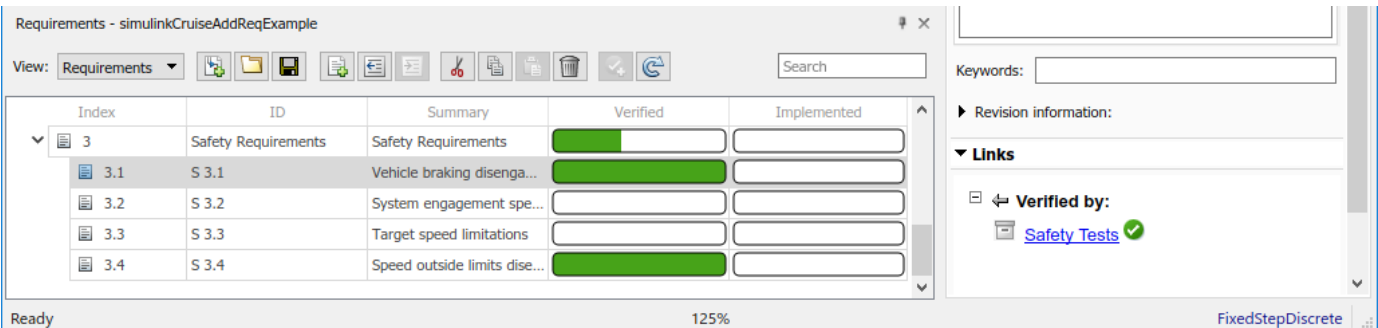
```
verify(engaged == false,...
      'verify:limit',...
      'system must disengage when limit exceeded')
```

- 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

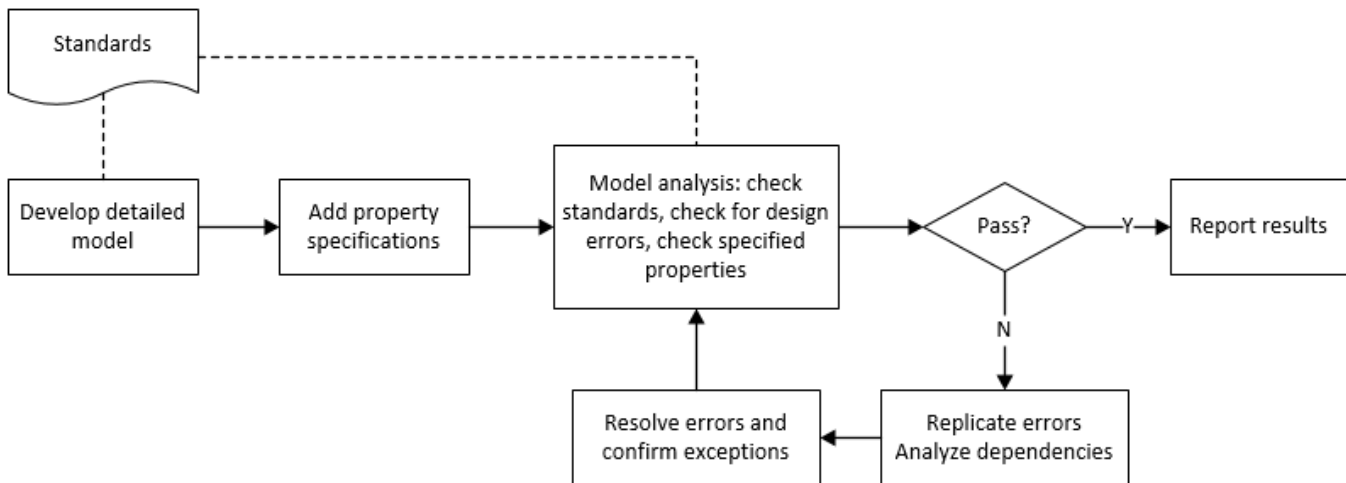
### Related Examples

- “Link to Requirements” (Simulink Test)
- “Validate Requirements Links in a Model” on page 11-25
- “Customize Requirements Traceability Report for Model” on page 11-10

# 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 Click **Check model diagnostic parameters** to review the configuration parameter settings that violate MAB style guidelines.
- d In the right pane, click the parameter links to update the values in the Configuration Parameters dialog box.
- 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 Model Advisor, you can also check your model for hidden design errors using Simulink Design Verifier.

- 1 In the left pane, in the **By Product > Simulink Design Verifier** folder, select **Design Error Detection**. All the checks in the folder are selected.
- 2 In the right pane, click **Run Selected Checks**.
- 3 After the analysis is complete, expand the **Design Error Detection** folder, then select checks to review warnings or errors.
- 4 In the right pane, click **Simulink Design Verifier Results Summary**. The dialog box provides tools to help you diagnose errors and warnings in your model.
  - a Review the 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. The Simulink Design Verifier Results Inspector window displays information that an overflow error occurred. To see the test cases that demonstrate the errors, click **View test case**.
  - c Review the analysis report. In the Simulink Design Verifier Results Inspector window, click **Back to summary**. To see a detailed analysis report, click **HTML** or **PDF**.

### See Also

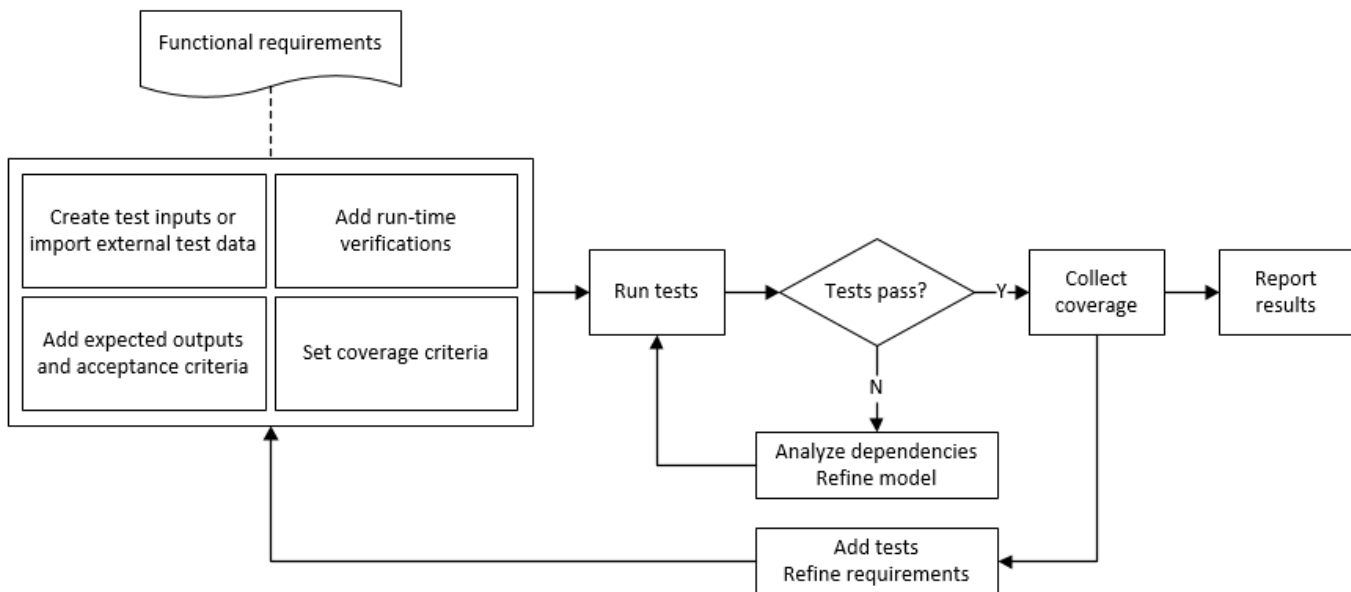
#### Related Examples

- “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

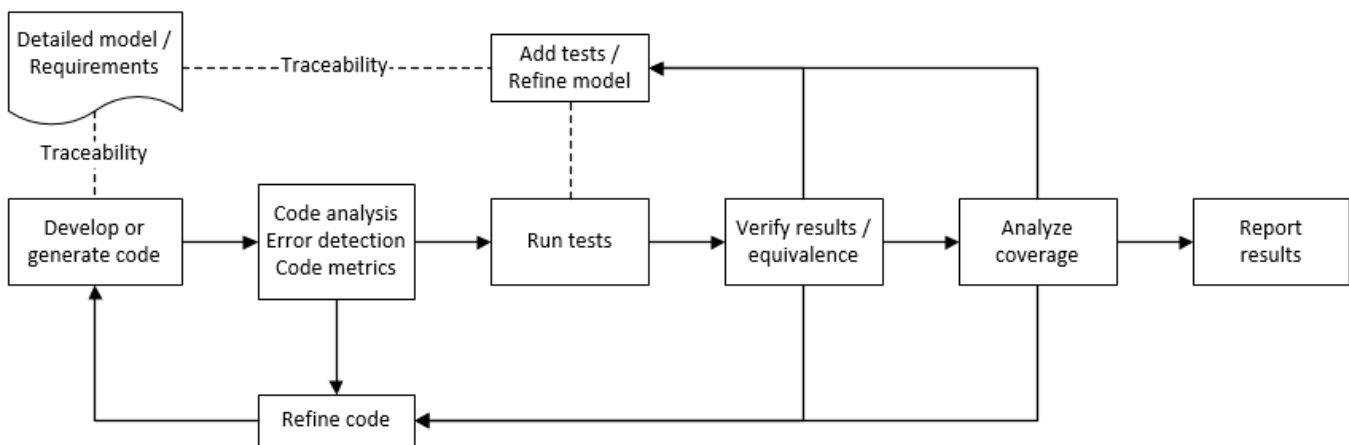
### Related Examples

- “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

Analyze code to detect errors, check standards compliance, and evaluate key metrics such as length and cyclomatic complexity. Typically for handwritten code, you 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, refine the code and add tests. For generated code, demonstrate that code execution produces equivalent results to the model by using the same test cases and baseline results. 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, code defects, and MISRA compliance. 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:

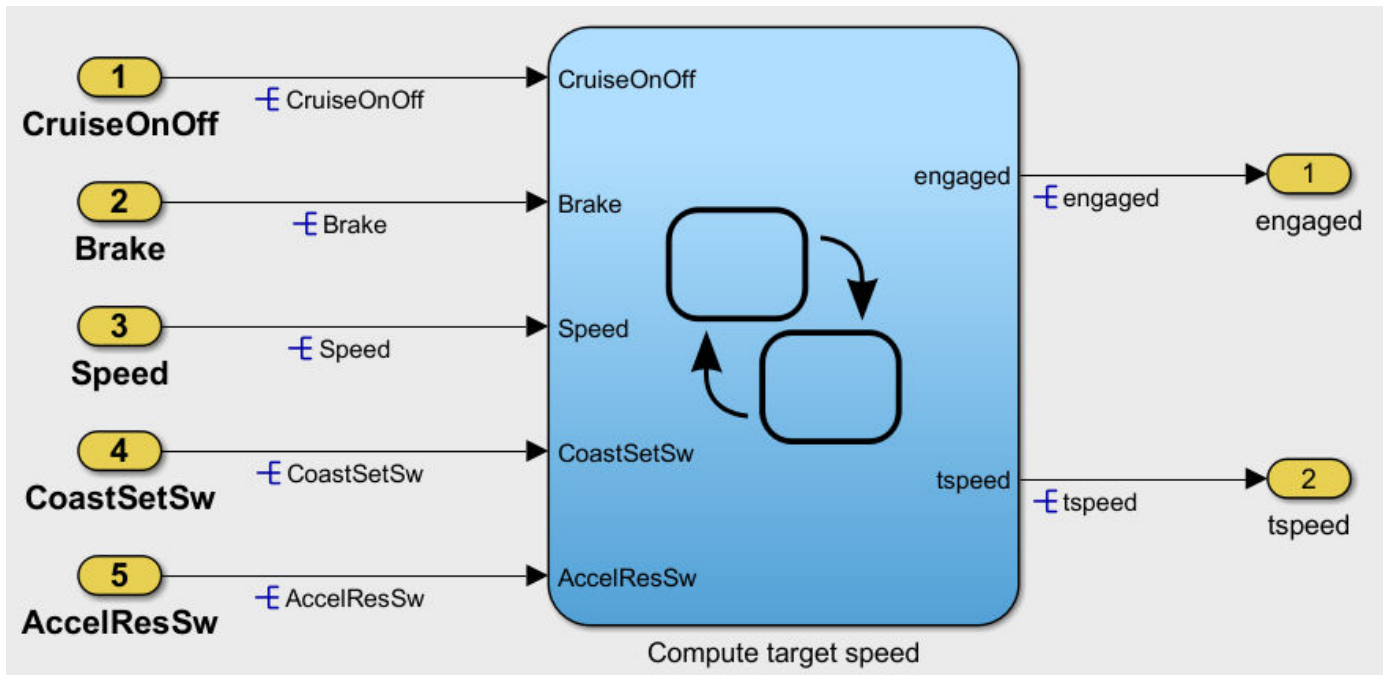
- 1 Open the project.

```

path = fullfile(matlabroot,'toolbox','shared','examples',...
'verification','src','cruise')
run(fullfile(path,'slVerificationCruiseStart'))
  
```

- 2 From the project, open the model `simulinkCruiseErrorAndStandardsExample`.



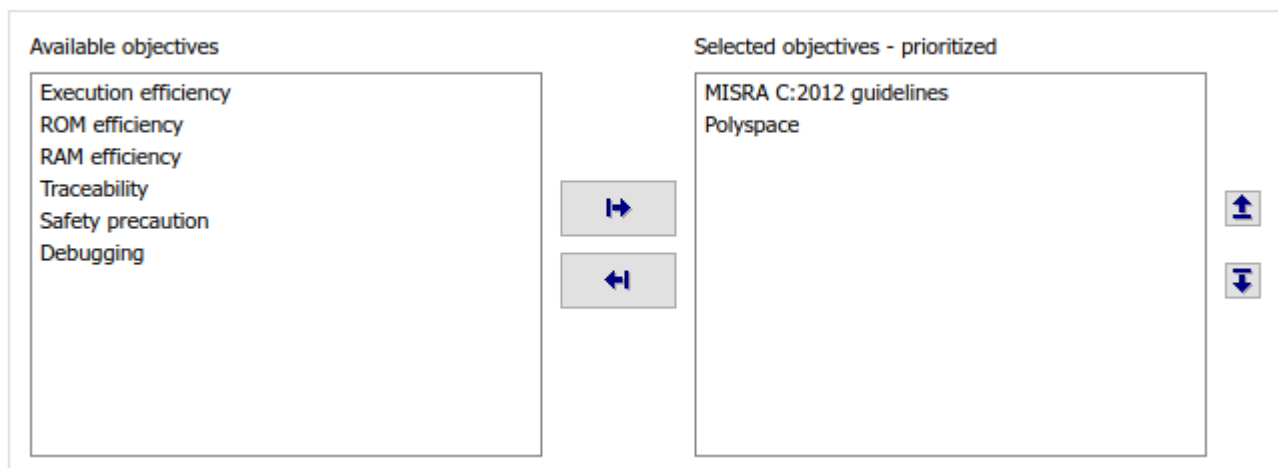


### Run Code Generator Checks

Before you generate code from your model, there are steps that you can take to generate code more compliant with MISRA C and more compatible with Polyspace. This example shows how to use the Code Generation Advisor to check your model before generating code.

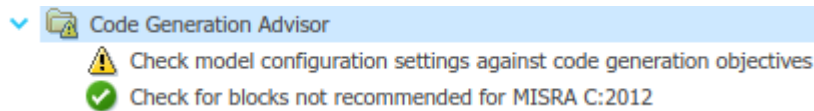
- 1 Right-click Compute target speed and select **C/C++ Code > Code Generation Advisor**.
- 2 Select the Code Generation Advisor folder. In the right pane, move Polyspace to **Selected objectives - prioritized**. The MISRA C:2012 guidelines objective is already selected.

Code Generation Objectives (System target file: ert.tlc)



- 3 Click **Run Selected Checks**.

The Code Generation Advisor checks whether there are any 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 there are some configuration settings that are incompatible with MISRA compliance and Polyspace checking.



- 4 Click on 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

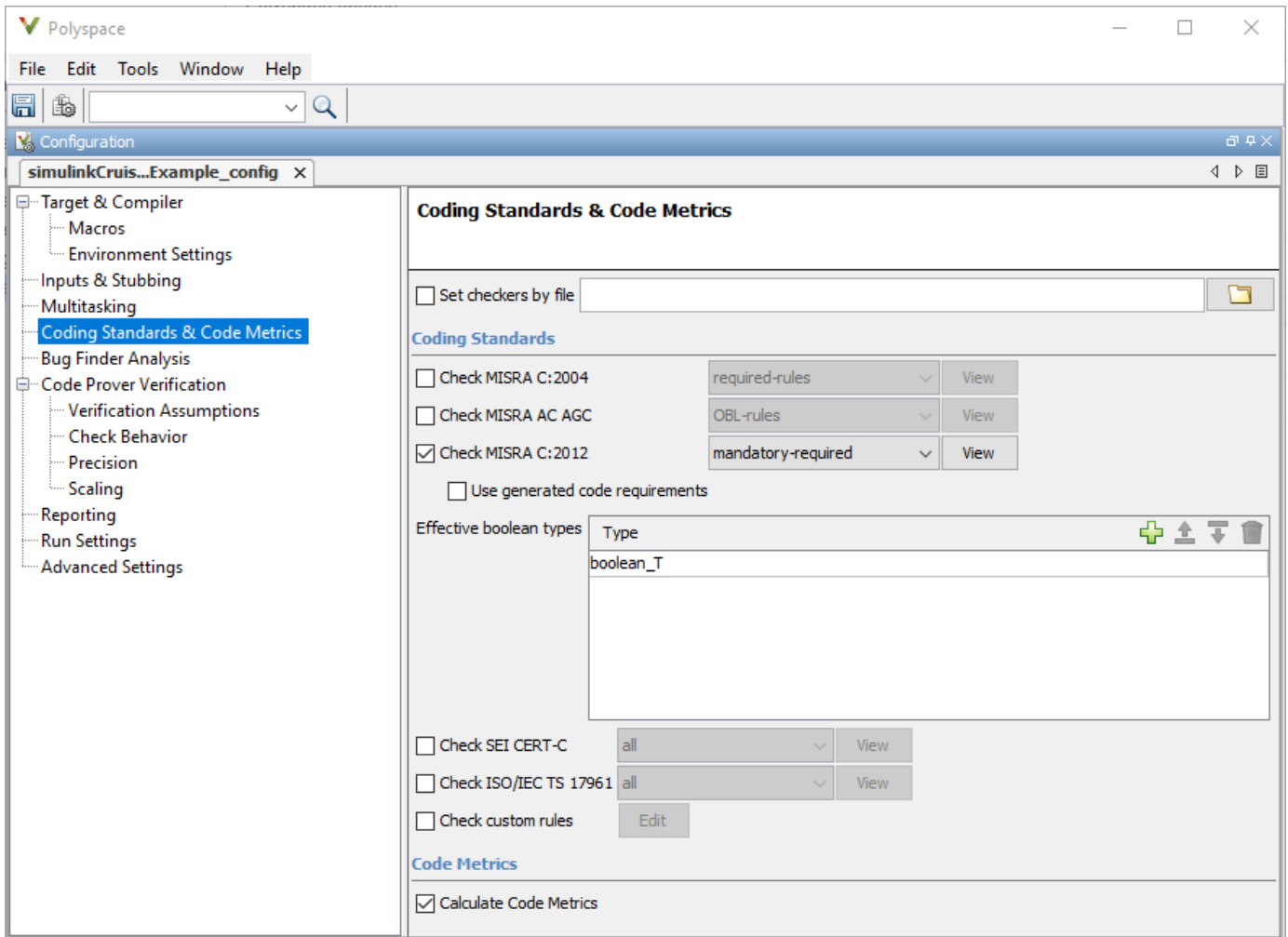
Before you generate code from your model, there are steps you can take to generate code that is more compliant with MISRA C and more compatible with Polyspace. This example shows you how to use the Model Advisor to check your model before generating code.

- 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, right-click Compute target speed and select **Polyspace > Options**.
- 4 Click the **Configure** (Polyspace Bug Finder) button. This option allows you to choose more advanced Polyspace analysis options in the Polyspace configuration window.



- 5 On the same pane, select **Calculate Code Metrics**. This option turns on 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 and defect checks. You can see the progress of the analysis in the MATLAB Command Window. Once the analysis is finished, the Polyspace environment opens.

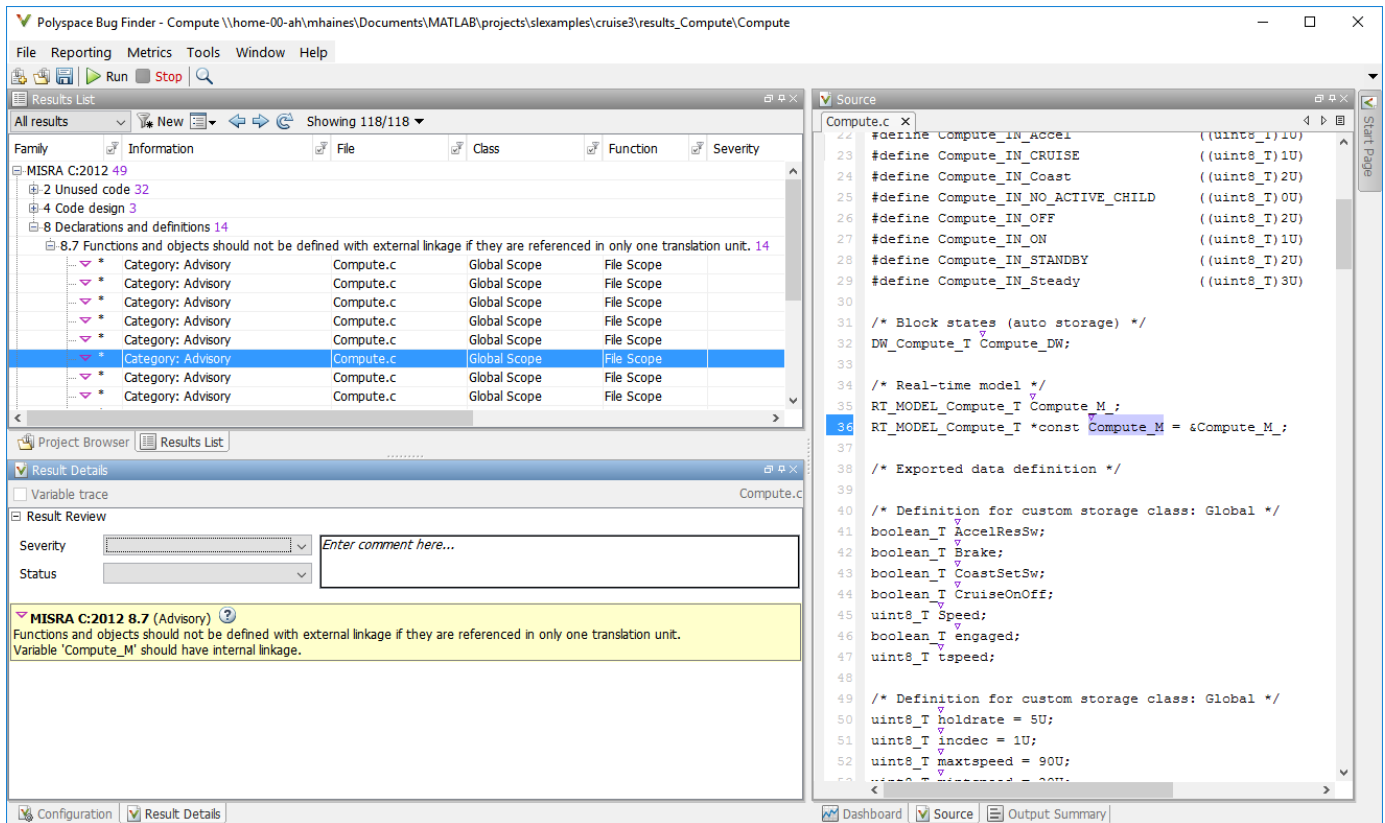
## Review Results

After you run a Polyspace analysis of your generated code, 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.

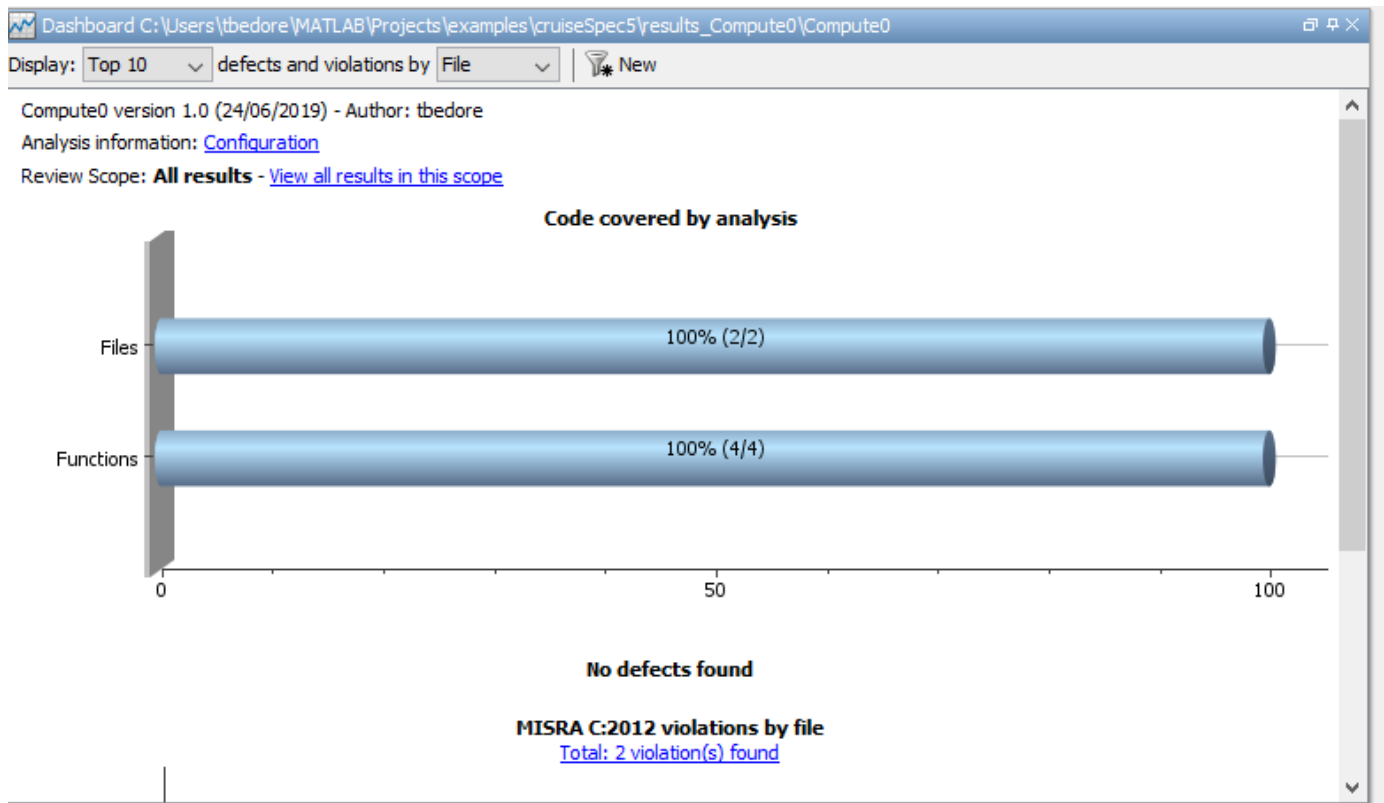
Rule 8.7 states that functions and objects should not be global if the function or object is local. As you click through the 8.7 violations, you can see that these results refer to variables that other components also use, such as `CruiseOnOff`. You can annotate your code or your model to justify

every result. But, 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** (Polyspace Bug Finder) option to **Project configuration**. This option allows you to choose a subset of MISRA rules in the Polyspace configuration.
- 4 Click the **Configure** button.
- 5 In the Polyspace Configuration window, on the **Coding Standards & Code Metrics** pane, select the check box **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, only two violations were found.



When this model is integrated 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. This section shows you how to generate a report after the analysis. If you want to generate a report every time you run an analysis, see [Generate report](#).

- 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**.

### 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 and Generate Test Results Reports” (Simulink Test)

