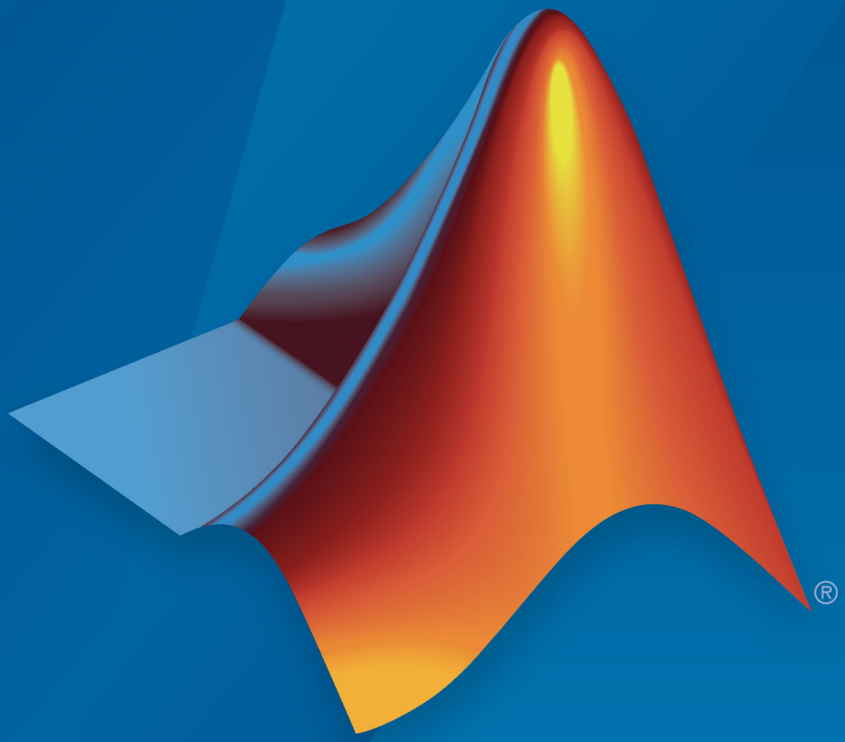
Simulink® Requirements™

Reference



MATLAB® & SIMULINK®

R2019a

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

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

Simulink® Requirements™ Reference

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 2018b)
March 2019	Online only	Revised for Version 1.3 (Release R2019a)

Contents

Functions — Alphabetical List

slreq.clear

Clear requirements and links from memory

Syntax

```
slreq.clear()
```

Description

`slreq.clear()` clears all requirements and links loaded in memory and closes the Requirements Editor, discarding all unsaved changes.

See Also

`slreq.LinkSet` | `slreq.ReqSet`

Introduced in R2018a

slreq.convertAnnotation

Convert annotations to requirement objects

Syntax

```
myReq = slreq.convertAnnotation(myAnnotation,myDestination)
myReq = slreq.convertAnnotation(myAnnotation,myDestination,Name,
Value)
```

Description

`myReq = slreq.convertAnnotation(myAnnotation,myDestination)` converts a Simulink® or a Stateflow® annotation `myAnnotation` into a requirement `myReq` and stores it in a destination entity `myDestination`.

`myReq = slreq.convertAnnotation(myAnnotation,myDestination,Name, Value)` converts a Simulink or a Stateflow annotation `myAnnotation` into a requirement `myReq` and stores it in a destination entity `myDestination` using additional options specified by one or more `Name, Value` pair arguments.

Examples

Convert Simulink Annotation to Requirement

```
% Find all annotations in a Simulink model
allAnnotations = find_system('controller_Model', 'FindAll', ...
'on', 'type', 'annotation');

% Create a new requirements set
newReqSet = slreq.new('myNewReqSet');

% Convert one annotation into a requirement newReq
% and add it to newReqSet
newReq = slreq.convertAnnotation(allAnnotations(1), ...
newReqSet);
```

Input Arguments

`myAnnotation` — Simulink or Stateflow annotation
`Simulink.Annotation` object

The annotation to be converted, specified as a `Simulink.Annotation` object.

`myDestination` — Converted annotation destination entity
`slreq.Requirement` object | `slreq.ReqSet` object

The destination entity for the converted annotation, specified either as an `slreq.Requirement` or as an `slreq.ReqSet` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'CreateLinks', true`

`CreateLinks` — Option to create links
`true` (default) | `false`

Option to create links when converting annotations, specified as a Boolean value.

`KeepAnnotation` — Option to retain annotation
`false` (default) | `true`

Option to retain the annotation after conversion, specified as a Boolean value.

`IgnoreCallback` — Option to force annotation conversion
`false` (default) | `true`

Option to specify annotation conversion even if a callback function is specified in the annotation, specified as a Boolean value.

`ShowMarkup` — Option to display requirements markup
`true` (default) | `false`

Option to display the Requirement markup after annotation conversion, specified as a Boolean value.

Output Arguments

myReq — Requirement
`slreq.Requirement` object

The converted annotation, returned as an `slreq.Requirement` object.

See Also

`slreq.ReqSet` | `slreq.Requirement`

Introduced in R2018a

slreq.createLink

Create traceable links

Syntax

```
myLink = slreq.createLink(src, dest)
```

Description

`myLink = slreq.createLink(src, dest)` creates an `slreq.Link` object `myLink` that serves as a link between the source artifact `src` and the destination artifact `dest`.

Examples

Create Links

```
% Create a link between the current Simulink Object and a requirement
link1 = slreq.createLink(gcb, REQ)

link1 =

  Link with properties:

          Type: 'Implement'
    Description: 'Plant Specs'
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 02-Sep-2017 15:49:28
      CreatedBy: 'Jane Doe'
     ModifiedOn: 21-Oct-2017 11:34:12
     ModifiedBy: 'John Doe'
       Comments: [0×0 struct]

% Create a link between a requirement and the current Stateflow object
link2 = slreq.createLink(REQ, sfgco);
```

Input Arguments

`src` — Link source artifact

structure

The link source artifact, specified as a MATLAB® structure.

`dest` — Link destination artifact

structure

The link destination artifact, specified as a MATLAB structure.

Output Arguments

`myLink` — Link artifact

`slreq.Link` object

The link between `src` and `dest`, specified as an `slreq.Link` object.

See Also

`slreq.Link` | `slreq.LinkSet`

Introduced in R2018a

slreq.dngCountLinks

Get number of links to IBM Rational DOORS Next Generation artifacts

Syntax

```
count = slreq.dngCountLinks(sourceArtifact)
count = slreq.dngCountLinks(sourceArtifact, config)
```

Description

`count = slreq.dngCountLinks(sourceArtifact)` returns the total number of links from `sourceArtifact` to IBM® Rational® DOORS® Next Generation artifacts.

`count = slreq.dngCountLinks(sourceArtifact, config)` returns the total number of links from `sourceArtifact` to the specified IBM Rational DOORS Next Generation configuration `config`.

Input Arguments

`sourceArtifact` — Link source artifact name
character vector | string | `slreq.LinkSet` object

The Simulink link source artifact, specified as a character vector or a string or as an `slreq.LinkSet` object.

`config` — Target project configuration identifier
string | character vector | structure

IBM Rational DOORS Next Generation Project configuration identifier. The configuration identifier can be the name, ID, or the configuration structure. The name and ID can be specified as a character vector or string. The configuration structure can be specified as a MATLAB structure.

Output Arguments

`count` — Link count
double

The total number of links from `sourceArtifact` to the IBM Rational DOORS Next Generation Project, returned as a `double`.

See Also

Introduced in R2018b

slreq.dngGetProjectConfig

Query known configurations from IBM Rational DOORS Next Generation server

Syntax

```
configs = slreq.dngGetProjectConfig()
configs = slreq.dngGetProjectConfig('project', ProjectName)
configs = slreq.dngGetProjectConfig('type', 'stream')
configs = slreq.dngGetProjectConfig('type', 'changeset')
configs = slreq.dngGetProjectConfig('name', ConfigName)
configs = slreq.dngGetProjectConfig('id', ConfigID)
```

Description

`configs = slreq.dngGetProjectConfig()` returns an array of structures representing all known configurations for the current IBM Rational DOORS Next Generation Project.

`configs = slreq.dngGetProjectConfig('project', ProjectName)` returns a structure representing the configuration for the IBM Rational DOORS Next Generation Project specified by `ProjectName` and switches the MATLAB session to `ProjectName`.

`configs = slreq.dngGetProjectConfig('type', 'stream')` returns a structure representing the known streams for the current IBM Rational DOORS Next Generation Project.

`configs = slreq.dngGetProjectConfig('type', 'changeset')` returns a structure representing the known changesets for the current IBM Rational DOORS Next Generation Project.

`configs = slreq.dngGetProjectConfig('name', ConfigName)` returns a structure representing the configuration for the stream or changeset specified by `ConfigName`.

`configs = slreq.dngGetProjectConfig('id', ConfigID)` returns a structure representing the configuration for the stream or changeset specified by `ConfigID`.

Input Arguments

`ProjectName` — Requirements project
character vector | string

IBM Rational DOORS Next Generation Project.

`ConfigName` — Stream or changeset name
character vector | string

The name of the IBM Rational DOORS Next Generation Project stream or changeset specified as a character vector or as a string.

`ConfigID` — Stream or changeset ID
character vector | string

The ID of the IBM Rational DOORS Next Generation Project stream or changeset specified as a character vector or as a string.

Output Arguments

`configs` — Server configurations
structure | array of structures

IBM Rational DOORS Next Generation Project configuration, returned as a structure or an array of structures containing these fields.

`id` — Configuration ID
character vector

IBM Rational DOORS Next Generation Project configuration ID, returned as a character vector.

`name` — Configuration name
character vector

IBM Rational DOORS Next Generation Project configuration name, returned as a character vector.

`type` — Configuration type
character vector

IBM Rational DOORS Next Generation Project configuration type, returned as a character vector.

`url` — Configuration URL
character vector

IBM Rational DOORS Next Generation Project configuration Uniform Resource Locator (URL), returned as a character vector.

See Also

Introduced in R2018b

slreq.dngGetUsedConfig

Query used IBM Rational DOORS Next Generation configurations from MATLAB/Simulink artifacts

Syntax

```
configs = slreq.dngGetUsedConfig()
configs = slreq.dngGetUsedConfig(sourceArtifact)
```

Description

`configs = slreq.dngGetUsedConfig()` returns all IBM Rational DOORS Next Generation configurations linked from loaded Simulink artifacts.

`configs = slreq.dngGetUsedConfig(sourceArtifact)` returns all IBM Rational DOORS Next Generation configurations linked from a given Simulink source, `sourceArtifact`.

Input Arguments

`sourceArtifact` — Link source artifact name
`slreq.LinkSet` object | character vector | string

The Simulink link source artifact, specified as a character vector or a string or as an `slreq.LinkSet` object.

Output Arguments

`configs` — Server configurations
array of structures

IBM Rational DOORS Next Generation Project configuration, returned as an array of structures containing these fields.

`id` — Configuration ID
character vector

IBM Rational DOORS Next Generation Project configuration ID, returned as a character vector.

`name` — Configuration name
character vector

IBM Rational DOORS Next Generation Project configuration name, returned as a character vector.

`type` — Configuration type
character vector

IBM Rational DOORS Next Generation Project configuration type, returned as a character vector.

`url` — Configuration URL
character vector

IBM Rational DOORS Next Generation Project configuration Uniform Resource Locator (URL), returned as a character vector.

See Also

Introduced in R2018b

slreq.dngUpdateConfig

Update links to IBM Rational DOORS Next Generation configuration

Syntax

```
count = slreq.dngUpdateConfig(sourceArtifact, oldConfig, newConfig)
```

Description

`count = slreq.dngUpdateConfig(sourceArtifact, oldConfig, newConfig)` updates the links to `oldConfig` originating from `sourceArtifact` to point to the same requirements in IBM Rational DOORS Next Generation under a different configuration, `newConfig`.

Input Arguments

`sourceArtifact` — Link source artifact name
`slreq.LinkSet` object | character vector | string

The Simulink link source artifact, specified as a character vector or a string or as an `slreq.LinkSet` object.

`oldConfig` — Stored project configuration name or ID
character vector

The original IBM Rational DOORS Next Generation Project configuration name or ID, specified as a character vector.

`newConfig` — New project configuration name or ID
character vector

The new IBM Rational DOORS Next Generation Project configuration name or ID, specified as a character vector.

Output Arguments

`count` — Link count
double

The total number of updated links from `sourceArtifact` to the IBM Rational DOORS Next Generation Project, returned as a `double`.

See Also

Introduced in R2018a

slreq.editor

Open Requirements Editor

Syntax

```
slreq.editor
```

Description

`slreq.editor` opens the Requirements Editor user interface (UI) dialog box.

See Also

`slreq.ReqSet`

Introduced in R2018a

slreq.exportViewSettings

Export view settings

Syntax

```
slreq.exportViewSettings(viewSettingsFile)
```

Description

`slreq.exportViewSettings(viewSettingsFile)` exports Simulink Requirements™ view settings to a MAT-file, `viewSettingsFile`.

Input Arguments

`viewSettingsFile` — View settings file
character vector

Simulink Requirements view settings file name, specified as a character vector.

See Also

`slreq.importViewSettings` | `slreq.resetViewSettings`

Introduced in R2018b

slreq.find

Find requirement, reference, and link set artifacts

Syntax

```
myArtifacts = slreq.find('Type',ArtifactType)
myArtifact = slreq.find('Type',ArtifactType,'PropertyName',
PropertyValue)
myReqs = slreq.find('Type',ArtifactType,'ReqType',ReqTypeValue)
myLinks = slreq.find('Type',ArtifactType,'LinkType',LinkTypeValue)
```

Description

`myArtifacts = slreq.find('Type',ArtifactType)` finds and returns all loaded Simulink Requirements artifacts `myArtifacts` of the type specified by `ArtifactType`.

`myArtifact = slreq.find('Type',ArtifactType,'PropertyName', PropertyValue)` finds and returns a Simulink Requirements artifact `myArtifact` of the type specified by `ArtifactType` matching the additional properties specified by `PropertyName` and `PropertyValue`.

`myReqs = slreq.find('Type',ArtifactType,'ReqType',ReqTypeValue)` finds and returns all requirements `myReqs` of the type specified by `ReqTypeValue`.

`myLinks = slreq.find('Type',ArtifactType,'LinkType',LinkTypeValue)` finds and returns all requirements `myLinks` of the type specified by `LinkTypeValue`.

Examples

Find Requirement Sets

```
% Find all requirement sets

allReqSets = slreq.find('Type', 'ReqSet')
```

```
allReqSets =

  1×8 ReqSet array with properties:

    Description
    Name
    Filename
    Revision
    Dirty
    CustomAttributeNames

% Find a requirement set with matching property values
myReqSet = slreq.find('Type', 'ReqSet', 'Name', 'My_Req_Set', 'Revision', 65)

myReqSet =

  ReqSet with properties:

            Description: ''
                   Name: 'My_Req_Set'
               Filename: 'C:\MATLAB\My_Req_Set.slreqx'
               Revision: 65
                  Dirty: 0
    CustomAttributeNames: {}
```

Find Requirements

```
% Find all requirements in all loaded requirement sets
allReqs = slreq.find('Type', 'Requirement')

allReqs =

  1×72 Requirement array with properties:

    Id
    Summary
    Keywords
    Description
    Rationale
    SID
    CreatedBy
    CreatedOn
    ModifiedBy
```

```
    ModifiedOn
    FileRevision
    Dirty
    Comments

% Find a requirement with matching property value
myReq = slreq.find('Type', 'Requirement', 'Id', '#19')

myReq =

  Requirement with properties:

            Id: '#19'
       Summary: 'Control Mode'
      Keywords: [0×0 char]
   Description: ''
     Rationale: ''
           SID: 19
     CreatedBy: 'Jane Doe'
     CreatedOn: 27-Feb-2017 10:15:38
    ModifiedBy: 'John Doe'
    ModifiedOn: 02-Aug-2017 15:18:55
  FileRevision: 52
         Dirty: 0
      Comments: [0×0 struct]
```

Find Referenced Requirements

```
% Find all referenced requirements in all loaded requirement sets
allRefs = slreq.find('Type', 'Reference')

allRefs =

  1×24 Reference array with properties:

    Keywords
    Artifact
    Id
    Summary
    Description
    SID
    Domain
    SynchronizedOn
    ModifiedOn
```

```
% Find a referenced requirement with matching property value
myRef = slreq.find('Type', 'Reference', 'Id', '#26')

myRef =

  Reference with properties:

          Keywords: [0×0 char]
          Artifact: 'My_req_doc.docx'
                Id: '#26'
           Summary: 'Overview'
       Description: ''
               SID: 2
            Domain: 'linktype_rmi_word'
    SynchronizedOn: 25-Jul-2017 11:34:02
        ModifiedOn: 16-Aug-2017 13:01:55
```

Find Link Sets

```
% Find all loaded link sets

allLinkSets = slreq.find('Type', 'LinkSet')

allLinkSets =

  1×2 LinkSet array with properties:

    Description
    Filename
    Artifact
    Domain
    Revision
    Dirty

% Find a link set with matching property values
myLinkSet = slreq.find('Type', 'LinkSet', 'Domain', 'linktype_rmi_slreq')

myLinkSet =

  LinkSet with properties:

    Description: ''
       Filename: 'C:\MATLAB\My_Reqs.slmx'
```

```
       Artifact: 'C:\MATLAB\My_Reqs.slreqx'
         Domain: 'linktype_rmi_slreq'
       Revision: 2
          Dirty: 0
```

Find Requirements and Links by Type

```
% Find all Functional requirements
myFunctionalReqs = slreq.find('Type', 'Requirement', 'ReqType', 'Functional')

myFunctionalReqs =

  1×70 Requirement array with properties:

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

% Find all Links of type Implement
myImplementLinks = slreq.find('Type', 'Link', 'LinkType', 'Implement')

myImplementLinks =

  1×95 Link array with properties:

    Type
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedOn
    ModifiedBy
```

```
Revision
Comments
```

Input Arguments

`ArtifactType` — Simulink Requirements artifact type
`'ReqSet'` | `'Requirement'` | `'Reference'` | `'LinkSet'`

The Simulink Requirements artifact to find.

`ReqTypeValue` — Requirement type
character vector

Requirement type. For more information, see “Requirement Types”.

`LinkTypeValue` — Link type
character vector

Link type. For more information, see “Link Types”.

Output Arguments

`myArtifacts` — Simulink Requirements artifact array
`slreq.ReqSet` array | `slreq.Requirement` array | `slreq.Reference` array | `slreq.LinkSet` array

Simulink Requirements artifacts, returned as arrays of the respective data type.

`myArtifact` — Simulink Requirements artifact
`slreq.ReqSet` | `slreq.Requirement` | `slreq.Reference` | `slreq.LinkSet`

Simulink Requirements artifact, returned as the respective data type.

`myReqs` — Requirement objects
`slreq.Requirement` object | array of `slreq.Requirement` objects

Requirement objects matching the requirement type specified by `ReqTypeValue`, returned as an `slreq.Requirement` object or as an array of `slreq.Requirement` objects.

`myLinks` — Link objects
`slreq.Link` object | array of `slreq.Link` objects

Link objects matching the link type specified by `LinkTypeValue`, returned as an `slreq.Link` object or as an array of `slreq.Link` objects.

See Also

`find` | `find` | `find` | `slreq.LinkSet` | `slreq.Reference` | `slreq.ReqSet` | `slreq.Requirement`

Introduced in R2018a

slreq.generateReport

Generate report for requirements set

Syntax

```
myReportPath = slreq.generateReport(reqSetList, reportOpts)
```

Description

`myReportPath = slreq.generateReport(reqSetList, reportOpts)` generates a report for the requirements sets specified by `reqSetList` using the options specified by `reportOpts` and returns the path `myReportPath` to the report.

Examples

Generate Requirement Report

```
% Generate a requirement report in Microsoft® Word
% format for all loaded requirements sets

% Get default report generation options structure
myReportOpts = slreq.getReportOptions();

% Specify the generated report path and file name
myReportOpts.reportPath = 'L:\My_Project\Reqs_Report.docx';

% Generate the report for all loaded requirements sets
myReport = slreq.generateReport('all', myReportOpts);
```

Note To generate reports in PDF and HTML formats, specify a `.pdf` or a `.html` file name as the `reportPath` value.

Input Arguments

`reqSetList` — Requirements set

character vector (default) | `slreq.ReqSet` object | array

Requirements sets for report generation. You can specify a single requirements set or an array of requirements sets. To generate a report for all the loaded requirements sets, specify `'all'` as the `reqSetList` value. If you do not specify a value for `reqSetList`, `'all'` is used as default.

`reportOpts` — Report generation options

structure

Report generation options specified as a MATLAB structure. If `reportOpts` is not specified, the report is generated using the default options specified in `slreq.getReportOptions`.

Options

Fields	Data Type	Description
`reportPath`	character vector	Generated report path.
`templatePath`	character vector	Report template path.
`titleText`	character vector	Report title.
`authors`	character vector	Report authors.
`includes.toc`	Boolean	Option to include table of contents in your report.
`includes.links`	Boolean	Option to include requirements links in your report.
`includes.rationale`	Boolean	Option to include requirements rationale in your report.
`includes.customAttributes`	Boolean	Option to include requirements set custom attributes in your report
`includes.comments`	Boolean	Option to include requirement comments in your report.
`includes.implementationStatus`	Boolean	Option to include requirement implementation status data in your report.
`includes.verificationStatus`	Boolean	Option to include requirement verification status data in your report.
`includes.keywords`	Boolean	Option to include requirement implementation status data in your report.
`includes.emptySections`	Boolean	Option to include empty sections in your report.
`includes.revision`	Boolean	Option to include requirement revision information in your report.

Output Arguments

`myReportPath` — Generated report path
character vector

The file path for the generated report, specified as a character vector.

See Also

`slreq.getReportOptions`

Topics

“Report Requirements Information”

Introduced in R2018a

slreq.getReportOptions

Get default report generation options

Syntax

```
myOptions = slreq.getReportOptions()
```

Description

`myOptions = slreq.getReportOptions()` returns a structure with the default options for generating reports for requirements sets.

Examples

Get Report Generation Options

```
myOptions = slreq.getReportOptions()

myOptions =

  struct with fields:

      reportPath: 'L:\slreqrpt_20170826.docx'
    templatePath: 'C:\matlab\toolbox\slrequirements\slrequirements\+slreq\+report\templ
       titleText: ''
         authors: 'Jane Doe'
        includes: [1×1 struct]
```

Output Arguments

`myOptions` — Report generation options
structure

Default options for report generation, returned as a MATLAB structure.

Options

Fields	Data Type	Description
`reportPath`	character vector	Generated report path.
`templatePath`	character vector	Report template path.
`titleText`	character vector	Report title.
`authors`	character vector	Report authors.
`includes.toc`	Boolean	Option to include table of contents in your report.
`includes.links`	Boolean	Option to include requirements links in your report.
`includes.linkGroup`	character vector	Option to group links by artifact or by link type.
`includes.rationale`	Boolean	Option to include requirements rationale in your report.
`includes.customAttributes`	Boolean	Option to include requirements set custom attributes in your report
`includes.comments`	Boolean	Option to include requirement comments in your report.
`includes.implementationStatus`	Boolean	Option to include requirement implementation status data in your report.
`includes.verificationStatus`	Boolean	Option to include requirement verification status data in your report.
`includes.keywords`	Boolean	Option to include requirement implementation status data in your report.
`includes.emptySections`	Boolean	Option to include empty sections in your report.

Fields	Data Type	Description
`includes.revision`	Boolean	Option to include requirement revision information in your report.

See Also

`slreq.generateReport`

Introduced in R2018a

slreq.import

Import requirements from external documents

Syntax

```
slreq.import(docPath)
[refCount, reqSetFilePath, reqSetObj] = slreq.import(docPath)
slreq.import(docType)
slreq.import(docPath, Name, Value)
slreq.import(reqifFile)
slreq.import(reqifFile, 'mappingFile', mapFilePath)
slreq.import('clearcache')
```

Description

`slreq.import(docPath)` imports requirements content as referenced requirements from an external document located at `docPath`. The imported requirements are saved in a new requirements set with the same name as the external document. Use this import method to import requirements content from Microsoft Office documents and from files in the Requirements Interchange Format (`.reqif` and `.reqifz`).

`[refCount, reqSetFilePath, reqSetObj] = slreq.import(docPath)` imports requirements content as referenced requirements from an external document located at `docPath` and returns the number of references imported `refCount`. The imported requirements are saved in the requirements set `reqSetObj` located at `reqSetFilePath` with the same name as the external document.

`slreq.import(docType)` imports requirements content as referenced requirements from an external document that is of a registered document type `docType`. The imported requirements are saved in a new requirements set with the same name as the external document.

`slreq.import(docPath, Name, Value)` imports requirements content as referenced requirements from an external document located at `docPath` with options specified by one or more `Name, Value` pair arguments.

`slreq.import(reqifFile)` imports requirement content from the ReqIF file `reqifFile` using a pre-configured attribute mapping.

`slreq.import(reqifFile, 'mappingFile', mapFilePath)` imports requirement content from the ReqIF file `reqifFile` using the attribute mapping specified by `mapFilePath`.

`slreq.import('clearcache')` cleans up temporary HTML files that are created when importing rich text requirements.

Examples

Import Referenced Requirements

```
% Import referenced requirements from Microsoft Office documents
slreq.import('Specification002.docx');
slreq.import('D:/Projects/Requirements/Safety321.xlsx');

% Import referenced requirements from an IBM Rational DOORS Module
slreq.import('linktype_rmi_doors');
```

Input Arguments

`docPath` — Document location

character vector

The file path of the external requirements document, specified as a character vector.

`docType` — Document type

character vector

The document type of the external requirements document, specified as a character vector.

Example: `'linktype_rmi_doors'`

`reqifFile` — ReqIF file location

character vector

The file path of the ReqIF file, specified as a character vector.

`mapFilePath` — Attribute mapping file location
character vector

The file path of the attribute mapping file, specified as a character vector.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'ReqSet', 'design_specs.slreqx'`

`AsReference` — Option to import as references
`true` (default) | `false`

Option to import requirements as references, specified as a Boolean value. The value `false` is supported only for import from Microsoft Office documents.

`ReqSet` — Requirements Set
character vector

The name for the existing requirements set that you import requirements into, specified as a character vector.

Example: `'ReqSet', 'My_Requirements_Set'`

`RichText` — Option to import rich text requirements
`false` (default) | `true`

Option to import requirements as rich text, specified as a Boolean value.

Example: `'RichText', true`

`bookmarks` — Option to import requirements using bookmarks
`false` | `true`

Option to import requirements content using user-defined bookmarks. This value is `true` by default for Microsoft Word documents and `false` by default for Microsoft Excel® spreadsheets.

Example: `'bookmarks', false`

`match` — Regular expression pattern
character vector

Regular expression pattern for ID search in Microsoft Office documents.

Example: `'match', '^REQ\d+'`

`attributes` — Attribute names
cell array

Attribute names to import, specified as a cell array.

Note When importing requirements from a Microsoft Excel spreadsheet, the length of this cell array must match the number of columns specified for import using the `'columns'` argument.

Example: `'attributes', {'Test Status', 'Test Procedure'}`

Pairs for Microsoft Excel Spreadsheets

`columns` — Range of columns
`double` array

Range of columns to import, specified as a `double` array.

Example: `'columns', [1 6]`

`rows` — Range of rows
`double` array

Range of rows to import, specified as a `double` array.

Example: `'rows', [3 35]`

`idColumn` — ID Column
`double`

Column in the Microsoft Excel spreadsheet that you want to correspond to the **ID** field in your requirement set, specified as a `double`.

Example: `'idColumn', 1`

summaryColumn — Summary Column
double

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Summary** field in your requirement set, specified as a `double`.

Example: `'summaryColumn', 4`

keywordsColumn — Keywords Column
double

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Keywords** field in your requirement set, specified as a `double`.

Example: `'keywordsColumn', 3`

descriptionColumn — Description Column
double

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Description** field in your requirement set, specified as a `double`.

Example: `'descriptionColumn', 2`

rationaleColumn — Rationale Column
double

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Rationale** field in your requirement set, specified as a `double`.

Example: `'rationaleColumn', 5`

attributeColumn — Custom Attributes Column
double

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Custom Attributes** field in your requirement set, specified as a `double`.

Example: `'attributeColumn', 6`

USDM — USDM Format Import Option
character vector

Import from Microsoft Excel spreadsheets specified in the USDM (Universal Specification Describing Manner) standard format. Specify values as a character vector with the ID prefix optionally followed by a separator character.

Example: `'RQ -'` will match entries with IDs similar to `RQ01`, `RQ01-2`, `RQ01-2-1` etc.

Output Arguments

`refCount` — Imported referenced requirements count
`double`

Number of referenced requirements imported, returned as a `double`.

`reqSetFilePath` — Requirement set file path
character vector

The file path of the requirement set to which you import requirements to, returned as a character vector.

`reqSetObj` — Requirement set object
`slreq.ReqSet` object

Handle to the requirement set to which you import requirements to, returned as an `slreq.ReqSet` object.

See Also

`createReferences` | `slreq.Reference`

Introduced in R2018a

slreq.importViewSettings

Import view settings

Syntax

```
slreq.importViewSettings(viewSettingsFile)
slreq.importViewSettings(viewSettingsFile, overwriteFlag)
```

Description

`slreq.importViewSettings(viewSettingsFile)` imports Simulink Requirements view settings from a MAT-file, `viewSettingsFile`.

`slreq.importViewSettings(viewSettingsFile, overwriteFlag)` imports Simulink Requirements view settings from a MAT-file, `viewSettingsFile`, with an optional argument to overwrite existing view settings, specified by `overwriteFlag`.

Input Arguments

`viewSettingsFile` — View settings file
character vector

Simulink Requirements view settings file name, specified as a character vector.

`overwriteFlag` — Overwrite flag
`false` (default) | true

Optional flag to specify whether the existing view settings are to be overwritten, specified as a Boolean.

See Also

`slreq.exportViewSettings` | `slreq.resetViewSettings`

Introduced in R2018b

slreq.load

Load requirements/link set

Syntax

```
myReqSet = slreq.load(reqSetArtifact)
myLinkSet = slreq.load(linkSetArtifact)
```

Description

`myReqSet = slreq.load(reqSetArtifact)` loads a requirements set `myReqSet` into memory.

`myLinkSet = slreq.load(linkSetArtifact)` loads a link set `myLinkSet` into memory.

Input Arguments

`reqSetArtifact` — Requirements set to load
character vector

The requirements set to load, specified as a character vector.

`linkSetArtifact` — Link set artifact name
character vector

The link set to load, specified as a character vector.

Output Arguments

`myReqSet` — Loaded requirements set
`slreq.ReqSet` object

The requirements set that was loaded, returned as an `slreq.ReqSet` object.

`myLinkSet` — Loaded link set
`slreq.LinkSet` object

The link set that was loaded, returned as an `slreq.LinkSet` object.

See Also

`slreq.LinkSet` | `slreq.ReqSet`

Introduced in R2018a

slreq.new

Create requirements set

Syntax

```
newReqSet = slreq.new(reqSetName)
newReqSet = slreq.new(reqSetPath)
```

Description

`newReqSet = slreq.new(reqSetName)` creates a requirements set `newReqSet` with the name specified by `reqSetName` in the current working folder.

`newReqSet = slreq.new(reqSetPath)` creates a requirements set `newReqSet` in the folder specified by `reqSetPath`.

Note The folder specified by `reqSetPath` must exist on disk.

Examples

Create Requirements Set

```
% Create requirements set in current working folder
myReqSet1 = slreq.new('New_Req_Set_1')

myReqSet1 =

  ReqSet with properties:

            Description: ''
                   Name: 'New_Req_Set_1'
               Filename: 'L:\New_Req_Set_1.slreqx'
               Revision: 1
                  Dirty: 1
```

```
    CustomAttributeNames: {}
                CreatedBy: 'John Doe'
                CreatedOn: 18-Feb-2008 20:54:52
               ModifiedBy: 'Jane Doe'
               ModifiedOn: 20-Jan-2016 12:44:12

% Create requirements set in a different directory
myReqSet2 = slreq.new('L:\Reqs_Work\New_Req_Set_2')

myReqSet2 =

  ReqSet with properties:

              Description: ''
                     Name: 'New_Req_Set_2'
                 Filename: 'L:\Reqs_Work\New_Req_Set_2.slreqx'
                 Revision: 1
                    Dirty: 1
    CustomAttributeNames: {}
                CreatedBy: 'Jane Doe'
                CreatedOn: 11-Jan-2009 11:33:01
               ModifiedBy: 'John Doe'
               ModifiedOn: 18-Jan-2018 09:07:32
```

Input Arguments

`reqSetName` — Requirements set name
character vector

Name of the requirements set to create, specified as a character vector.

`reqSetPath` — Requirements set path
character vector

Folder to create requirements set in, specified as a character vector.

Output Arguments

`newReqSet` — Created requirements set
`slreq.ReqSet` object

The created requirements set, specified as an `slreq.ReqSet` object.

See Also

`slreq.ReqSet`

Introduced in R2018a

slreq.open

Open requirements set

Syntax

```
myReqSet = slreq.open(ReqSetFilePath)
myReqSet = slreq.open(ReqSetName)
```

Description

`myReqSet = slreq.open(ReqSetFilePath)` loads the requirements set at `ReqSetFilePath` into memory. If the requirements set is already loaded into memory, the Requirements Editor opens. If the requirements set is already loaded and the Requirements Editor is open, the specified requirements set is selected in the Requirements Editor.

`myReqSet = slreq.open(ReqSetName)` loads the requirements set named `ReqSetName` if it can be located.

Input Arguments

`ReqSetFilePath` — Requirements set file path
character vector

The full file path of the requirements set to be loaded, specified as a character vector.

`ReqSetName` — Requirements set name
character vector

The name of the requirements set to be loaded, specified as a character vector.

Output Arguments

`myReqSet` — Requirements set object
`slreq.ReqSet` object

Handle to the requirements set you open, returned as an `slreq.ReqSet` object.

See Also

`slreq.ReqSet`

Introduced in R2018a

slreq.refreshLinkDependencies

Refresh requirement link dependencies

Syntax

```
slreq.refreshLinkDependencies()
```

Description

`slreq.refreshLinkDependencies()` recreates all requirement link dependencies. Use this command to:

- Refresh corrupted, missing, or incorrect requirement link dependencies if a project is open.
- Create dependency information when working with older projects and model files with embedded link sets.

See Also

Topics

“Review Requirement Links”

Introduced in R2018b

slreq.resetViewSettings

Reset saved view settings

Syntax

```
slreq.resetViewSettings('all')
slreq.resetViewSettings('editor')
slreq.resetViewSettings(ModelName)
```

Description

`slreq.resetViewSettings('all')` resets all saved view settings.

`slreq.resetViewSettings('editor')` resets all saved view settings for the Requirements Editor.

`slreq.resetViewSettings(ModelName)` resets all saved view settings for the model specified by `ModelName`.

Input Arguments

`ModelName` — Model name
character vector

Simulink model name, specified as a character vector.

Example: `'vdp', 'f14'`

See Also

Introduced in R2018b

rmi

Interact programmatically with Requirements Management Interface

Syntax

```
reqlinks = rmi('createEmpty')
reqlinks = rmi('get', object)
reqlinks = rmi('get', sig_builder, group_idx)
rmi('set', model, reqlinks)
rmi('set', sig_builder, reqlinks, group_idx)
rmi('cat', model, reqlinks)
cnt = rmi('count', object)
rmi('clearAll', object)
rmi('clearAll', object, 'deep')
rmi('clearAll', object, 'noprompt')
rmi('clearAll', object, 'deep', 'noprompt')

cmdStr = rmi('navCmd', object)
[cmdStr, titleStr] = rmi('navCmd', object)
object = rmi('guidlookup', model, guidStr)
rmi('highlightModel', object)
rmi('unhighlightModel', object)
rmi('view', object, index)
dialog = rmi('edit', object)
guidStr = rmi('guidget', object)

rmi('report', model)
rmi('report', matlabFilePath)
rmi('report', dictionaryFile)
rmi('projectreport')

rmi setup
rmi register linktypename
rmi unregister linktypename
rmi linktypelist

number_problems = rmi('checkdoc')
```

```
number_problems = rmi('checkdoc', docName)
rmi('check', matlabFilePath)
rmi('check', dictionaryFile)

rmi('doorssync', model)
[objHs, parentIdx, isSf, objSIDs] = rmi('getObjectsInModel', model)
[objName, objType] = rmi('getObjLabel', object)

rmi('setDoorsLabelTemplate', template)
template = rmi('getDoorsLabelTemplate')
label = rmi('doorsLabel', moduleID, objectID)
totalModifiedLinks = rmi('updateDoorsLabels', model)
```

Description

`reqlinks = rmi('createEmpty')` creates an empty instance of the requirement links data structure.

`reqlinks = rmi('get', object)` returns the requirement links data structure for `object`.

`reqlinks = rmi('get', sig_builder, group_idx)` returns the requirement links data structure for the Signal Builder group specified by the index `group_idx`.

`rmi('set', model, reqlinks)` sets `reqlinks` as the requirements links for `model`.

`rmi('set', sig_builder, reqlinks, group_idx)` sets `reqlinks` as the requirements links for the signal group `group_idx` in the Signal Builder block `sig_builder`.

`rmi('cat', model, reqlinks)` adds the requirements links in `reqlinks` to existing requirements links for `model`.

`cnt = rmi('count', object)` returns the number of requirements links for `object`.

`rmi('clearAll', object)` deletes all requirements links for `object`.

`rmi('clearAll', object, 'deep')` deletes all requirements links in the model containing `object`.

`rmi('clearAll', object, 'noprompt')` deletes all requirements links for `object` and does not prompt for confirmation.

`rmi('clearAll', object, 'deep', 'noprompt')` deletes all requirements links in the model containing `object` and does not prompt for confirmation.

`cmdStr = rmi('navCmd', object)` returns the MATLAB command `cmdStr` used to navigate to `object`.

`[cmdStr, titleStr] = rmi('navCmd', object)` returns the MATLAB command `cmdStr` and the title `titleStr` that provides descriptive text for `object`.

`object = rmi('guidlookup', model, guidStr)` returns the object name in `model` that has the globally unique identifier `guidStr`.

`rmi('highlightModel', object)` highlights all of the objects in the parent model of `object` that have requirement links.

`rmi('unhighlightModel', object)` removes highlighting of objects in the parent model of `object` that have requirement links.

`rmi('view', object, index)` accesses the requirement numbered `index` in the requirements document associated with `object`.

`dialog = rmi('edit', object)` displays the Requirements dialog box for `object` and returns the handle of the dialog box.

`guidStr = rmi('guidget', object)` returns the globally unique identifier for `object`. A globally unique identifier is created for `object` if it lacks one.

`rmi('report', model)` generates a Requirements Traceability report in HTML format for `model`.

`rmi('report', matlabFilePath)` generates a Requirements Traceability report in HTML format for the MATLAB code file specified by `matlabFilePath`.

`rmi('report', dictionaryFile)` generates a Requirements Traceability report in HTML format for the Simulink data dictionary specified by `dictionaryFile`.

`rmi('projectreport')` generates a Requirements Traceability report in HTML format for the current project. The master page of this report has HTTP links to reports for each project item that has requirements traceability associations. For more information, see “Create Requirements Traceability Report for A Project”.

`rmi setup` configures RMI for use with your MATLAB software and installs the interface for use with the IBM Rational DOORS software.

`rmi register linktypename` registers the custom link type specified by the function `linktypename`. For more information, see “Custom Link Type Registration”.

`rmi unregister linktypename` removes the custom link type specified by the function `linktypename`. For more information, see “Custom Link Type Registration”.

`rmi linktypelist` displays a list of the currently registered link types. The list indicates whether each link type is built-in or custom, and provides the path to the function used for its registration.

`number_problems = rmi('checkdoc')` checks validity of links to Simulink from a requirements document in Microsoft Word, Microsoft Excel, or IBM Rational DOORS. It prompts for the requirements document name, returns the total number of problems detected, and opens an HTML report in the MATLAB Web browser. For more information, see “Validate Requirements Links in a Requirements Document”.

`number_problems = rmi('checkdoc', docName)` checks validity of links to Simulink from the requirements document specified by `docName`. It returns the total number of problems detected and opens an HTML report in the MATLAB Web browser. For more information, see “Validate Requirements Links in a Requirements Document”.

`rmi('check', matlabFilePath)` checks consistency of traceability links associated with MATLAB code lines in the `.m` file `matlabFilePath`, and opens an HTML report in the MATLAB Web browser.

`rmi('check', dictionaryFile)` checks consistency of traceability links associated with the Simulink data dictionary `dictionaryFile`, and opens an HTML report in the MATLAB Web browser.

`rmi('doorssync', model)` opens the DOORS synchronization settings dialog box, where you can customize the synchronization settings and synchronize your model with an open project in an IBM Rational DOORS database.

`[objHs, parentIdx, isSf, objSIDs] = rmi('getObjectsInModel', model)` returns a list of Simulink objects that may be considered for inclusion in the IBM Rational DOORS surrogate module.

`[objName, objType] = rmi('getObjLabel', object)` returns Simulink object Name and Type information for the Simulink object that you link to with a third-party requirements management application.

`rmi('setDoorsLabelTemplate', template)` specifies a new custom template for labels of requirements links to IBM Rational DOORS. The default label template contains the section number and object heading for the DOORS requirement link target. To revert the link label template back to the default, enter `rmi('setDoorsLabelTemplate', '')` at the MATLAB command prompt.

`template = rmi('getDoorsLabelTemplate')` returns the currently specified custom template for labels of requirements links to IBM Rational DOORS.

`label = rmi('doorsLabel', moduleID, objectID)` generates a label for the requirements link to the IBM Rational DOORS object specified by `objectID` in the DOORS module specified by `moduleID`, according to the current template.

`totalModifiedLinks = rmi('updateDoorsLabels', model)` updates all IBM Rational DOORS requirements links labels in `model` according to the current template.

Examples

Requirements Links Management in Example Model

Get a requirement associated with a block in the `slvnvdemo_fuelsys_officereq` model, change its description, and save the requirement back to that block. Define a new requirement link and add it to the existing requirements links in the block.

Get requirement link associated with the Airflow calculation block in the `slvnvdemo_fuelsys_officereq` example model.

```
slvnvdemo_fuelsys_officereq;
blk_with_req = sprintf('slvnvdemo_fuelsys_officereq/fuel rate\ncontroller' ...
'/Airflow calculation');
reqts = rmi('get', blk_with_req);
```

Change the description of the requirement link.

```
reqts.description = 'Mass airflow estimation';
```

Save the changed requirement link description for the Airflow calculation block.

```
rmi('set', blk_with_req, reqts);
```

Create new requirement link to example document `fuelsys_requirements2.htm`.

```
new_req = rmi('createempty');
new_req.doc = 'fuelsys_requirements2.htm';
new_req.description = 'A new requirement';
```

Add new requirement link to existing requirements links for the Airflow calculation block.

```
rmi('cat', blk_with_req, new_req);
```

Requirements Traceability Report for Example Model

Create HTML report of requirements traceability data in example model.

Create an HTML requirements report for the `slvnvdemo_fuelsys_officereq` example model.

```
rmi('report', 'slvnvdemo_fuelsys_officereq');
```

The MATLAB Web browser opens, showing the report.

Labels for Requirements Links to IBM Rational DOORS

Specify a new label template for links to requirements in DOORS, and update labels of all DOORS requirements links in your model to fit the new template.

Specify a new label template for requirements links to IBM Rational DOORS so that new links to DOORS objects are labeled with the corresponding module ID, object absolute number, and the value of the ‘Backup’ attribute.

```
rmi('setDoorsLabelTemplate', '%m:%n [backup=%<Backup>]');
```

Specify a new label template for requirements links to IBM Rational DOORS and set the maximum label length to (for example) 200 characters.

```
rmi('setDoorsLabelTemplate', '%h %200');
```

Update existing DOORS requirements link labels to match the new specified template in your model `example_model`. When updating labels, DOORS must be running and all linked modules must be accessible for reading.

```
rmi('updateDoorsLabels', example_model);
```

Input Arguments

`model` — Simulink model or Stateflow chart with which requirements can be associated

name | handle

Simulink model or Stateflow chart with which requirements can be associated, specified as a character vector or handle.

Example: `'slvnvdemo_officereq'`

Data Types: `char`

`object` — Model object with which requirements can be associated

name | handle

Model object with which requirements can be associated, specified as a character vector or handle.

Example: `'slvnvdemo_fuelsys_officereq/fuel rate controller/Airflow calculation'`

Data Types: `char`

`sig_builder` — Signal Builder block containing signal group with requirements traceability associations

name | handle

Signal Builder block containing signal group with requirements traceability associations, specified as a character vector or handle.

Data Types: `char`

`group_idx` — Signal Builder group index

integer

Signal Builder group index, specified as a scalar.

Example: 2

Data Types: `char`

`matlabFilePath` — MATLAB code file with requirements traceability associations
path

MATLAB code file with requirements traceability associations, specified as the path to the file.

Example:

Data Types: `char`

`dictionaryFile` — Simulink data dictionary with requirements traceability associations
character vector

Simulink data dictionary with requirements traceability associations, specified as a character vector containing the file name and, optionally, path of the dictionary.

Example:

Data Types: `char`

`guidStr` — Globally unique identifier for model object
character vector

Globally unique identifier for model object `object`, specified as a character vector.

Example: `GIDa_59e165f5_19fe_41f7_abc1_39c010e46167`

Data Types: `char`

`index` — Index number of requirement linked to model object
integer

Index number of requirement linked to model object, specified as an integer.

`docName` — Requirements document in external application
file name | path

Requirements document in external application, specified as a character vector that represents one of the following:

- IBM Rational DOORS module ID.
- path to Microsoft Word requirements document.
- path to Microsoft Excel requirements document.

For more information, see “Validate Requirements Links in a Requirements Document”.

`label` — Label for links to requirements in IBM Rational DOORS
character vector

Example:

Data Types: `char`

`template` — Template label for links to requirements in IBM Rational DOORS
character vector

Template label for links to requirements in IBM Rational DOORS, specified as a character vector.

You can use the following format specifiers to include the associated DOORS information in your requirements links labels:

`%h`	Object heading
`%t`	Object text
`%p`	Module prefix
`%n`	Object absolute number
`%m`	Module ID
`%P`	Project name
`%M`	Module name
`%U`	DOORS URL
`%<ATTRIBUTE_NAME>`	Other DOORS attribute you specify

Example: `'%m:%n [backup=%<Backup>]'`

Data Types: `char`

`moduleID` — IBM Rational DOORS module
DOORS module ID

IBM Rational DOORS module, specified as the unique DOORS module ID.

Example:

Data Types: `char`

`objectID` — IBM Rational DOORS object
DOORS object ID

IBM Rational DOORS object in the DOORS module `moduleID`, specified as the locally unique DOORS ID.

Example:

Data Types: `char`

Output Arguments

`reqlinks` — Requirement links data
`struct`

Requirement links data, returned as a structure array with the following fields:

`doc`	Character vector identifying requirements document

`id` Character vector defining location in requirements document. The first character specifies the identifier type:

First Character	Identifier	Example
`?`	Search text, the first occurrence of which is located in requirements document	`'?Requirement 1'`
`@`	Named item, such as bookmark in a Microsoft Word file or an anchor in an HTML file	`'@my_req'`
`#`	Page or item number	`'#21'`
`>`	Line number	`'>3156'`
`$`	Worksheet range in a spreadsheet	`'$A2:C5'`

`linked` Boolean value specifying whether the requirement link is accessible for report generation and highlighting:
`1` (default). Highlight model object and include requirement link in reports.
`0`

`description` Character vector describing the requirement

`keywords` Optional character vector supplementing `description`

`reqsys` Character vector identifying the link type registration name; `'other'` for built-in link types

`cmdStr` — Command used to navigate to model object
character vector

Command used to navigate to model object `object`, returned as a character vector.

Example: `rmiobjnavigate('slvnvdemo_fuelsys_officereq.slx', 'GIDa_59e165f5_19fe_41f7_abc1_39c010e46167');`

`titleStr` — Textual description of model object with requirements links
character vector

Textual description of model object with requirements links, returned as a character vector.

Example: `slvnvdemo_fuelsys_officereq/.../Airflow calculation/Pumping Constant (Lookup2D)`

`guidStr` — Globally unique identifier for model object
character vector

Globally unique identifier for model object `object`, returned as a character vector.

Example: `GIDa_59e165f5_19fe_41f7_abc1_39c010e46167`

`dialog` — Requirements dialog box for model object
handle

Requirements dialog box for model object `object`, returned as a handle to the dialog box.

`number_problems` — Total count of invalid links detected in external document
integer

Total count of invalid links detected in external document `docName`.

For more information, see “Validate Requirements Links in a Requirements Document”.

`totalModifiedLinks` — Total count of DOORS requirements links updated with new label template
integer

Total count of DOORS requirements links updated with new label template.

`objHs` — Numeric handles
array

List of numeric handles, returned as an array.

`parentIdx` — Model hierarchy indices
array

Model hierarchy indices, returned as an array.

`isSf` — List position to Stateflow object correspondence
array

Logical array that indicates which list positions correspond to which Stateflow objects.

`objSIDs` — Simulink IDs
array

Session-independent Simulink IDs, returned as an array.

See Also

`rmipref` | `rmiobjnavigate` | `rmidocrename` | `rmitag` | `rmimap.map` | `RptgenRMI.doorsAttribs`

Introduced in R2006b

rmidata.export

Move requirements traceability data to external `.req` file

Syntax

```
[total_linked,total_links] = rmidata.export
[total_linked,total_links] = rmidata.export(model)
```

Description

`[total_linked,total_links] = rmidata.export` moves requirements traceability data associated with the current Simulink model to an external file named *`model_name`*`.req`. `rmidata.export` saves the file in the same folder as the model. `rmidata.export` deletes the requirements traceability data stored in the model and saves the modified model.

`[total_linked,total_links] = rmidata.export(model)` moves requirements traceability data associated with `model` to an external file named *`model_name`*`.req`. `rmidata.export` saves the file in the same folder as `model`. `rmidata.export` deletes the requirements traceability data stored in the model and saves the modified model.

Input Arguments

`model`

Name or handle of a Simulink model

Output Arguments

`total_linked`

Integer indicating the number of objects in the model that have linked requirements

`total_links`

Integer indicating the total number of requirements links in the model

Examples

Move the requirements traceability data from the `slvnvdemo_fuelsys_officereq` model to an external file:

```
rmidata.export('slvnvdemo_fuelsys_officereq');
```

See Also

`rmi` | `rmidata.save` | `rmimap.map`

Topics

“Requirements Link Storage”

Introduced in R2011b

rmimap.map

Associate externally stored requirements traceability data with model

Syntax

```
rmimap.map(model,reqts_file)
rmimap.map(model,'undo')
rmimap.map(model,'clear')
```

Description

`rmimap.map(model,reqts_file)` associates the requirements traceability data from `reqts_file` with the Simulink model `model`.

`rmimap.map(model,'undo')` removes from the `.slmx` file associated with `model` the requirements traceability data that was most recently saved in the `.slmx` file.

`rmimap.map(model,'clear')` removes from the `.slmx` file associated with `model` all requirements traceability data.

Input Arguments

`model`

Name, handle, or full path for a Simulink model

`reqts_file`

Full path to the `.slmx` file that contains requirements traceability data for the model

Alternatives

To load a file that contains requirements traceability data for a model:

1. Open the model.
2. Select **Analysis** > **Requirements** > **Load Links**.

   Note The **Load Links** menu item appears only when your model is configured to store requirements data externally. To specify external storage of requirements data for your model, in the Requirements Settings dialog box under **Storage** > **Default storage location for requirements links data**, select **Store externally (in a separate *.slmx file)**.
3. Browse to the `.slmx` file that contains the requirements links.
4. Click **OK**.

Examples

Associate an external requirements traceability data file with a Simulink model. After associating the information with the model, view the objects with linked requirements by highlighting the model.

```
open_system('slvnvdemo_powerwindowController');
reqFile = fullfile(matlabroot, 'toolbox', 'slvnv', ...
    'rmidemos', 'powerwin_reqs', ...
    'slvnvdemo_powerwindowRequirements.slmx');
rmimap.map('slvnvdemo_powerwindowController', reqFile);
rmi('highlightModel', 'slvnvdemo_powerwindowController');
```

To clear the requirements you just associated with that model, run this `rmimap.map` command:

```
rmimap.map('slvnvdemo_powerwindowController','clear');
```

See Also

`rmi` | `rmidata.save` | `rmidata.export`

Topics

“Requirements Link Storage”

Introduced in R2015a

rmidata.save

Save requirements traceability data in external `.req` file

Syntax

```
rmidata.save(model)
```

Description

`rmidata.save(model)` saves requirements traceability data for a model in an external `.req` file. The model must be configured to store requirements traceability data externally. This function is equivalent to **Analysis > Requirements > Save Links** in the Simulink Editor.

Examples

Create New Requirement Link and Save Externally

Add a requirement link to an existing example model, and save the model requirements traceability data in an external file.

Open the example model, `slvnvdemo_powerwindowController`.

```
open_system('slvnvdemo_powerwindowController');
```

Specify that the model store requirements data externally.

```
rmipref('StoreDataExternally',1);
```

Create a new requirements link structure.

```
newReqLink = rmi('createEmpty');
newReqLink.description = 'newReqLink';
```

Specify the requirements document that you want to link to from the model. In this case, an example requirements document is provided.

```
newReqLink.doc = [matlabroot '\toolbox\slvnv\rmidemos\' ...
        'powerwin_reqs\PowerWindowSpecification.docx'];
```

Specify the text of the requirement within the document to which you want to link.

```
newReqLink.id = '?passenger input consists of a vector' ...
        'with three elements';
```

Specify that the new requirements link that you created be attached to the Mux4 block of the `slvnvdemo_powerwindowController` example model.

```
rmi('set', 'slvnvdemo_powerwindowController/Mux4', newReqLink);
```

Save the new requirement link that you just created in an external `.req` file associated with the model.

```
rmidata.save('slvnvdemo_powerwindowController');
```

This function is equivalent to the Simulink Editor option **Analysis > Requirements > Save Links**.

To highlight the Mux4 block, turn on requirements highlighting for the `slvnvdemo_powerwindowController` example model.

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

You can test your requirements link by right-clicking the Mux4 block. In the context menu, select **Requirements > 1. “newReqLink”**.

Close the example model.

```
close_system('slvnvdemo_powerwindowController', 0);
```

You are not prompted to save unsaved changes because you saved the requirements link data outside the model file. The model file remains unchanged.

Input Arguments

`model` — Name or handle of model with requirements links

character vector | handle

Name of model with requirements links, specified as a character vector, or handle to model with requirements links. The model must be loaded into memory and configured to store requirements traceability data externally.

If you have a new model with no existing requirements links, configure it for external storage as described in “Requirements Link Storage”. You can also use the `rmipref` command to specify storage settings.

If you have an existing model with internally stored requirements traceability data, convert that data to external storage as described in “Move Internally Stored Requirements Links to External Storage”. You can also use the `rmidata.export` command to convert existing requirements traceability data to external storage.

Example: `'slvnvdemo_powerwindowController'`

Example: `get_param(gcs,'Handle')`

See Also

`rmimap.map` | `rmidata.export`

Topics

“Requirements Link Storage”

Introduced in R2013b

rmidocrename

Update model requirements document paths and file names

Syntax

```
rmidocrename(model_handle, old_path, new_path)
rmidocrename(model_name, old_path, new_path)
```

Description

`rmidocrename(model_handle, old_path, new_path)` collectively updates the links from a Simulink model to requirements files whose names or locations have changed. `model_handle` is a handle to the model that contains links to the files that you have moved or renamed. `old_path` is a character vector that contains the existing full or partial file or path name. `new_path` is a character vector with the new full or partial file or path name.

`rmidocrename(model_name, old_path, new_path)` updates the links to requirements files associated with `model_name`. You can pass `rmidocrename` a model handle or a model file name.

When using the `rmidocrename` function, make sure to enter specific character vectors for the old document name fragments so that you do not inadvertently modify other links.

Examples

For the current Simulink model, update all links to requirements files that contain the character vector `'project_0220'`, replacing them with `'project_0221'`:

```
rmidocrename(gcs, 'project_0220', 'project_0221')
Processed 6 objects with requirements, 5 out of 13 links were modified.
```

Alternatives

To update the requirements links one at a time, for each model object that has a link:

1. For each object with requirements, open the Requirements Traceability Link Editor by right-clicking and selecting **Requirements Traceability > Open Link Editor**.
2. Edit the **Document** field for each requirement that points to a moved or renamed document.
3. Click **Apply** to save the changes.

See Also

`rmi`

Introduced in R2009b

rmiobjnavigate

Navigate to model objects using unique Requirements Management Interface identifiers

Syntax

```
rmiobjnavigate(modelPath, guId)
rmiobjnavigate(modelPath, guId, grpNum)
```

Description

`rmiobjnavigate(modelPath, guId)` navigates to and highlights the specified object in a Simulink model.

`rmiobjnavigate(modelPath, guId, grpNum)` navigates to the signal group number `grpNum` of a Signal Builder block identified by `guId` in the model `modelPath`.

Input Arguments

`modelPath`

A full path to a Simulink model file, or a Simulink model file name that can be resolved on the MATLAB path.

`guId`

A unique identifier that the RMI uses to identify a Simulink or Stateflow object.

`grpNum`

Integer indicating a signal group number in a Signal Builder block

Examples

Open the `slvnvdemo_fuelsys_officereq` example model and get the unique identifier for the MAP Sensor block:

```
% Open example model
slvnvdemo_fuelsys_officereq;
% Get the Ssession Independent Identifier of the MAP Sensor Block
targetSID = Simulink.ID.getSID('slvnvdemo_fuelsys_officereq/MAP sensor');
```

Navigate to the MAP Sensor block using `rmiobjnavigate` and the unique identifier returned in the previous step:

```
% Split targetSID into two components
[targetModel, targetObj] = strtok(targetSID,':');
% Navigate to the MAP sensor using the model name and model guID
rmiobjnavigate(targetModel, targetObj)
```

See Also

`rmi`

Topics

“Use the rmiobjnavigate Function”

Introduced in R2010b

rmipref

Get or set RMI preferences stored in `prefdir`

Syntax

```
rmipref

currentVal = rmipref(prefName)

previousVal = rmipref(Name,Value)
```

Description

`rmipref` returns list of `Name,Value` pairs corresponding to Requirements Management Interface (RMI) preference names and accepted values for each preference.

`currentVal = rmipref(prefName)` returns the current value of the preference specified by `prefName`.

`previousVal = rmipref(Name,Value)` sets a new value for the RMI preference specified by `Name`, and returns the previous value of that RMI preference.

Examples

References to Simulink Model in External Requirements Documents

Choose the type of reference that the RMI uses when it creates links to your model from external requirements documents. The reference to your model can be either the model file name or the full absolute path to the model file.

The value of the `'ModelPathReference'` preference determines how the RMI stores references to your model in external requirements documents. To view the current value of this preference, enter the following code at the MATLAB command prompt.

```
currentVal = rmipref('ModelPathReference')
```

The default value of the `'ModelPathReference'` preference is `'none'`.

```
currentVal =

none
```

This default value specifies that the RMI uses only the model file name in references to your model that it creates in external requirements documents.

Automatic Application of User Tags to Selection-Based Requirements Links

Configure the RMI to automatically apply a specified list of user tag keywords to new selection-based requirements links that you create.

Specify that the user tags `design` and `reqts` apply to new selection-based requirements links that you create.

```
previousVal = rmipref('SelectionLinkTag','design,reqts')
```

When you specify a new value for an RMI preference, `rmipref` returns the previous value of that RMI preference. In this case, `previousVal` is an empty character vector, the default value of the `'SelectionLinkTag'` preference.

```
previousVal =

     ''
```

View the currently specified value for the `'SelectionLinkTag'` preference.

```
currentVal = rmipref('SelectionLinkTag')
```

The function returns the currently specified comma-separated list of user tags.

```
currentVal =

design,reqts
```

These user tags apply to all new selection-based requirements links that you create.

Internal Storage of Requirements Traceability Data

Configure the RMI to embed requirements links data in the model file instead of in a separate `.req` file.

Note If you have existing requirements links for your model that are stored internally, you need to move these links into an external `.req` file before you change the storage settings for your requirements traceability data. See “Move Internally Stored Requirements Links to External Storage” for more information.

If you would like to embed requirements traceability data in the model file, set the `'StoreDataExternally'` preference to `0`.

```
previousVal = rmipref('StoreDataExternally',0)
```

When you specify a new value for an RMI preference, `rmipref` returns the previous value of that RMI preference. By default, the RMI stores requirements links data externally in a separate `.req` file, so the previous value of this preference was `1`.

```
previousVal =

     1
```

After you set the `'StoreDataExternally'` preference to `0`, your requirements links are embedded in the model file.

```
currentVal = rmipref('StoreDataExternally')

currentVal =

     0
```

Input Arguments

`prefName` — RMI preference name

`'BiDirectionalLinking'` | `'FilterRequireTags'` | `'CustomSettings'` | ...

RMI preference name, specified as the corresponding `Name` character vector listed in “Name-Value Pair Arguments” on page 1-77.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`).

Example: `'BiDirectionalLinking',true` enables bidirectional linking for your model, so that when you create a selection-based link to a requirements document, the RMI creates a corresponding link to your model from the requirements document.

`BiDirectionalLinking` — Bidirectional selection linking preference
`false` (default) | `true`

Bidirectional selection linking preference, specified as a logical value.

This preference specifies whether to simultaneously create return link from target to source when creating link from source to target. This setting applies only for requirements document types that support selection-based linking.

Data Types: `logical`

`DocumentPathReference` — Preference for path format of links to requirements documents from model
`'modelRelative'` (default) | `'absolute'` | `'pwdRelative'` | `'none'`

Preference for path format of links to requirements documents from model, specified as one of the following values.

Value	Document reference contains...
`'absolute'`	full absolute path to requirements document.
`'pwdRelative'`	path relative to MATLAB current folder.
`'modelRelative'`	path relative to model file.
`'none'`	document file name only.

For more information, see “Document Path Storage”.

Data Types: `char`

`ModelPathReference` — Preference for path format in links to model from requirements documents
`'none'` (default) | `'absolute'`

Preference for path format in links to model from requirements documents, specified as one of the following values.

Value	Model reference contains...
`'absolute'`	full absolute path to model.
`'none'`	model file name only.

Data Types: `char`

`LinkIconFilePath` — Preference to use custom image file as requirements link icon
empty character vector (default) | full image file path

Preference to use custom image file as requirements link icon, specified as full path to icon or small image file. This image will be used for requirements links inserted in external documents.

Data Types: `char`

`FilterEnable` — Preference to enable filtering by user tag keywords
`false` (default) | `true`

Preference to enable filtering by user tag keywords, specified as a logical value. When you filter by user tag keywords, you can include or exclude subsets of requirements links in highlighting or reports. You can specify user tag keywords for requirements links filtering in the `'FilterRequireTags'` and `'FilterExcludeTags'` preferences. For more information about requirements filtering, see “Filter Requirements with User Tags”.

Data Types: `logical`

`FilterRequireTags` — Preference for user tag keywords for requirements links
empty character vector (default) | comma-separated list of user tag keywords

Preference for user tag keywords for requirements links, specified as a comma-separated list of words or phrases in a character vector. These user tags apply to all new requirements links you create. Requirements links with these user tags are included in model highlighting and reports. For more information about requirements filtering, see “Filter Requirements with User Tags”.

Data Types: `char`

`FilterExcludeTags` — Preference to exclude certain requirements links from model highlighting and reports
empty character vector (default) | comma-separated list of user tag keywords

Preference to exclude certain requirements links from model highlighting and reports, specified as a comma-separated list of user tag keywords. Requirements links with these user tags are excluded from model highlighting and reports. For more information about requirements filtering, see “Filter Requirements with User Tags”.

Data Types: `char`

`FilterMenusByTags` — Preference to disable labels of requirements links with designated user tags
`false` (default) | `true`

Preference to disable labels of requirements links with designated user tags, specified as a logical value. When set to `true`, if a requirement link has a user tag designated in `'FilterExcludeTags'` or `'FilterRequireTags'`, that requirements link will be disabled in the Requirements context menu. For more information about requirements filtering, see “Filter Requirements with User Tags”.

Data Types: `logical`

`FilterConsistencyChecking` — Preference to filter Model Advisor requirements consistency checks with designated user tags
`false` (default) | `true`

Preference to filter Model Advisor requirements consistency checks with designated user tags, specified as a logical value. When set to `true`, Model Advisor requirements consistency checks include requirements links with user tags designated in `'FilterRequireTags'` and excludes requirements links with user tags designated in `'FilterExcludeTags'`. For more information about requirements filtering, see “Filter Requirements with User Tags”.

Data Types: `logical`

`KeepSurrogateLinks` — Preference to keep DOORS surrogate links when deleting all requirements links
empty (default) | `false` | `true`

Preference to keep DOORS surrogate links when deleting all requirements links, specified as a logical value. When set to `true`, selecting **Requirements > Delete All Links** deletes all requirements links including DOORS surrogate module requirements links.

When not set to `true` or `false`, selecting **Requirements** > **Delete All Links** opens a dialog box with a choice to keep or delete DOORS surrogate links.

Data Types: `logical`

`ReportFollowLibraryLinks` — Preference to include requirements links in referenced libraries in generated report
`false` (default) | `true`

Preference to include requirements links in referenced libraries in generated report, specified as a logical value. When set to `true`, generated requirements reports include requirements links in referenced libraries.

Data Types: `logical`

`ReportHighlightSnapshots` — Preference to include highlighting in model snapshots in generated report
`true` (default) | `false`

Preference to include highlighting in model snapshots in generated report, specified as a logical value. When set to `true`, snapshots of model objects in generated requirements reports include highlighting of model objects with requirements links.

Data Types: `logical`

`ReportNoLinkItems` — Preference to include model objects with no requirements links in generated requirements reports
`false` (default) | `true`

Preference to include model objects with no requirements links in generated requirements reports, specified as a logical value. When set to `true`, generated requirements reports include lists of model objects that have no requirements links.

Data Types: `logical`

`ReportUseDocIndex` — Preference to include short document ID instead of full path to document in generated requirements reports
`false` (default) | `true`

Preference to include short document ID instead of full path to document in generated requirements reports, specified as a logical value. When set to `true`, generated requirements reports include short document IDs, when specified, instead of full paths to requirements documents.

Data Types: `logical`

`ReportIncludeTags` — Preference to list user tags for requirements links in generated reports
`false` (default) | `true`

Preference to list user tags for requirements links in generated reports, specified as a logical value. When set to `true`, generated requirements reports include user tags specified for each requirement link. For more information about requirements filtering, see “Filter Requirements with User Tags”.

Data Types: `logical`

`ReportDocDetails` — Preference to include extra detail from requirements documents in generated reports
`false` (default) | `true`

Preference to include extra detail from requirements documents in generated reports, specified as a logical value. When set to `true`, generated requirements reports load linked requirements documents to include additional information about linked requirements. This preference applies to Microsoft Word, Microsoft Excel, and IBM Rational DOORS requirements documents only.

Data Types: `logical`

`ReportLinkToObjects` — Preference to include links to model objects in generated requirements reports
`false` (default) | `true`

Preference to include links to model objects in generated requirements reports, specified as a logical value. When set to `true`, generated requirements reports include links to model objects. These links work only if the MATLAB internal HTTP server is active.

Data Types: `logical`

`SelectionLinkWord` — Preference to include Microsoft Word selection link option in Requirements context menu
`true` (default) | `false`

Preference to include Microsoft Word selection link option in Requirements context menu, specified as a logical value.

Data Types: `logical`

`SelectionLinkExcel` — Preference to include Microsoft Excel selection link option in Requirements context menu
`true` (default) | `false`

Preference to include Microsoft Excel selection link option in Requirements context menu, specified as a logical value.

Data Types: `logical`

`SelectionLinkDoors` — Preference to include IBM Rational DOORS selection link option in Requirements context menu
`true` (default) | `false`

Preference to include IBM Rational DOORS selection link option in Requirements context menu, specified as a logical value.

Data Types: `logical`

`SelectionLinkTag` — Preference for user tags to apply to new selection-based requirements links
empty character vector (default) | comma-separated list of user tag keywords

Preference for user tags to apply to new selection-based requirements links, specified as a comma-separated list of words or phrases in a character vector. These user tags automatically apply to new selection-based requirements links that you create. For more information about requirements filtering, see “Filter Requirements with User Tags”.

Data Types: `char`

`StoreDataExternally` — Preference to store requirements links data in external `.req` file
`false` (default) | `true`

Preference to store requirements links data in external `.req` file, specified as a logical value. This setting applies to all new models and to existing models that do not yet have requirements links. For more information about storage of requirements links data, see “Requirements Link Storage”.

Data Types: `logical`

`UseActiveXButtons` — Preference to use legacy ActiveX® buttons in Microsoft Office requirements documents
`false` (default) | `true`

Preference to use legacy ActiveX buttons in Microsoft Office requirements documents, specified as a logical value. The default value of this preference is `false`; requirements links are URL-based by default. ActiveX requirements navigation is supported for backward compatibility.

Data Types: `logical`

`CustomSettings` — Preference for storing custom settings
`inUse: 0` (default) | structure array of custom field names and settings

Preference for storing custom settings, specified as a structure array. Each field of the structure array corresponds to the name of your custom preference, and each associated value corresponds to the value of that custom preference.

Data Types: `struct`

Output Arguments

`currentVal` — Current value of the RMI preference specified by `prefName`
`true` | `false` | `'absolute'` | `'none'` | ...

Current value of the RMI preference specified by `prefName`. RMI preference names and their associated possible values are listed in “Name-Value Pair Arguments” on page 1-77.

`previousVal` — Previous value of the RMI preference specified by `prefName`
`true` | `false` | `'absolute'` | `'none'` | ...

Previous value of the RMI preference specified by `prefName`. RMI preference names and their associated possible values are listed in “Name-Value Pair Arguments” on page 1-77.

See Also
`rmi`

Topics
“Requirements Settings”

Introduced in R2013a

rmiref.insertRefs

Insert links to models into requirements documents

Syntax

```
[total_links, total_matches, total_inserted] = rmiref.insertRefs(
model_name, doc_type)
```

Description

`[total_links, total_matches, total_inserted] = rmiref.insertRefs( model_name, doc_type)` inserts ActiveX controls into the open, active requirements document of type `doc_type`. These controls correspond to links from `model_name` to the document. With these controls, you can navigate from the requirements document to the model.

Input Arguments

`model_name`

Name or handle of a Simulink model

`doc_type`

A character vector that indicates the requirements document type:

- `'word'`
- `'excel'`

Examples

Remove the links in an example requirements document, and then reinsert them:

1. Open the example model:

   ```
   slvnvdemo_fuelsys_officereq
   ```

2. Open the example requirements document:

   ```
   open([matlabroot strcat('/toolbox/slrequirements/slrequirementsdemos/fuelsys_req_docs/',...
       'slvnvdemo_FuelSys_DesignDescription.docx')])
   ```

3. Remove the links from the requirements document:

   ```
   rmiref.removeRefs('word')
   ```

4. Enter `y` to confirm the removal.

5. Reinsert the links from the requirements document to the model:

   ```
   [total_links, total_matches, total_inserted] = ...
       rmiref.insertRefs(gcs, 'word')
   ```

See Also

`rmiref.removeRefs`

Introduced in R2011a

rmiref.removeRefs

Remove links to models from requirements documents

Syntax

```
rmiref.removeRefs(doc_type)
```

Description

`rmiref.removeRefs(doc_type)` removes all links to models from the open, active requirements document of type `doc_type`.

Input Arguments

`doc_type`

A character vector that indicates the requirements document type:

- `'word'`
- `'excel'`
- `'doors'`

Examples

Remove the links in this example requirements document:

```
open([matlabroot strcat('/toolbox/slvnv/rmidemos/fuelsys_req_docs/', ...
    'slvnvdemo_FuelSys_DesignDescription.docx')])
rmiref.removeRefs('word')
```

See Also

`rmiref.insertRefs`

Introduced in R2011a

rmitag

Manage user tags for requirements links

Syntax

```
rmitag(model, 'list')
rmitag(model, 'add', tag)
rmitag(model, 'add', tag, doc_pattern)
rmitag(model, 'delete', tag)
rmitag(model, 'delete', tag, doc_pattern)
rmitag(model, 'replace', tag, new_tag)
rmitag(model, 'replace', tag, new_tag, doc_pattern)
rmitag(model, 'clear', tag)
rmitag(model, 'clear', tag, doc_pattern)
```

Description

`rmitag(model, 'list')` lists all user tags in `model`.

`rmitag(model, 'add', tag)` adds `tag` as a user tag for all requirements links in `model`.

`rmitag(model, 'add', tag, doc_pattern)` adds `tag` as a user tag for all links in `model`, where the full or partial document name matches the regular expression `doc_pattern`.

`rmitag(model, 'delete', tag)` removes the user tag, `tag` from all requirements links in `model`.

`rmitag(model, 'delete', tag, doc_pattern)` removes the user tag, `tag`, from all requirements links in `model`, where the full or partial document name matches `doc_pattern`.

`rmitag(model, 'replace', tag, new_tag)` replaces `tag` with `new_tag` for all requirements links in `model`.

`rmitag(model, 'replace', tag, new_tag, doc_pattern)` replaces `tag` with `new_tag` for links in `model`, where the full or partial document name matches the regular expression `doc_pattern`.

`rmitag(model, 'clear', tag)` deletes all requirements links that have the user tag, `tag`.

`rmitag(model, 'clear', tag, doc_pattern)` deletes all requirements links that have the user tag, `tag`, and link to the full or partial document name specified in `doc_pattern`.

Input Arguments

`model`

Name of or handle to Simulink or Stateflow model with which requirements are associated.

`tag`

Character vector specifying user tag for requirements links.

`doc_pattern`

Regular expression to match in the linked requirements document name. Not case sensitive.

`new_tag`

Character vector that indicates the name of a user tag for a requirements link. Use this argument when replacing an existing user tag with a new user tag.

Examples

Open the `slvnvdemo_fuelsys_officereq` example model, and add the user tag `tmptag` to all objects with requirements links:

```
open_system('slvnvdemo_fuelsys_officereq');
rmitag(gcs, 'add', 'tmptag');
```

Remove the user tag `test` from all requirements links:

```
open_system('slvnvdemo_fuelsys_officereq');
rmitag(gcs, 'delete', 'test');
```

Delete all requirements links that have the user tag `design`:

```
open_system('slvnvdemo_fuelsys_officereq');
rmitag(gcs, 'clear', 'design');
```

Change all instances of the user tag `tmptag` to `safety requirement`, where the document filename extension is `.docx`:

```
open_system('slvnvdemo_fuelsys_officereq');
rmitag(gcs, 'replace', 'tmptag', ...
       'safety requirements', '\.docx');
```

See Also

`rmi` | `rmidocrename`

Topics

“User Tags and Requirements Filtering”

Introduced in R2010a

RptgenRMI.doorsAttribs

IBM Rational DOORS attributes in requirements report

Syntax

```
RptgenRMI.doorsAttribs (action,attribute)
```

Description

`RptgenRMI.doorsAttribs (action,attribute)` specifies which DOORS object attributes to include in the generated requirements report.

Input Arguments

`action`

Character vector that specifies the desired action for what content to include from a DOORS record in the generated requirements report. Valid values for this argument are as follows.

Value	Description
`'default'`	Restore the default settings for the DOORS system attributes to include in the report. The default configuration includes the **Object Heading** and **Object Text** attributes, and all other attributes, except: • **Created Thru** • System attributes with empty string values • System attributes that are false
`'show'`	Display the current settings for the DOORS attributes to include in the report.

Value	Description
`'type'`	Include or omit groups of DOORS attributes from the report. If you specify `'type'` for the first argument, valid values for the second argument are: • `'all'` — Include all DOORS attributes in the report. • `'user'` — Include only user-defined DOORS in the report. • `'none'` — Omit all DOORS attributes from the report.
`'remove'`	Omit specified DOORS attributes from the report.
`'all'`	Include specified DOORS attributes in the report, even if that attribute is currently excluded as part of a group.
`'nonempty'`	Enable or disable the empty attribute filter: • Enter `RptgenRMI.doorsAttribs('nonempty', 'off')` to omit all empty attributes from the report. • Enter `RptgenRMI.doorsAttribs('nonempty', 'on')` to include empty user-defined attributes. The report never includes empty system attributes.

Default:

`attribute`

Character vector that qualifies the `action` argument.

Output Arguments

`result`

- True if `RptgenRMI.doorsAttribs` modifies the current settings.
- For `RptgenRMI.doorsAttribs('show')`, this argument is a cell array of character vectors that indicate which DOORS attributes to include in the requirements report, for example:

```
>> RptgenRMI.doorsAttribs('show')

ans =
```

```
'Object Heading'
'Object Text'
'$AllAttributes$'
'$NonEmpty$'
'-Created Thru'
```

- The **Object Heading** and **Object Text** attributes are included by default.
- `'$AllAttributes$'` specifies to include all attributes associated with each DOORS object.
- `'$Nonempty$'` specifies to exclude all empty attributes.
- `'-Created Thru'` specifies to exclude the **Created Thru** attribute for each DOORS object.

Examples

Limit the DOORS attributes in the requirements report to user-defined attributes:

```
RptgenRMI.doorsAttribs('type', 'user');
```

Omit the content of the **Last Modified By** attribute from the requirements report:

```
RptgenRMI.doorsAttribs('remove', 'Last Modified By');
```

Include the content of the **Last Modified On** attribute in the requirements report, even if system attributes are not included as a group:

```
RptgenRMI.doorsAttribs('add', 'Last Modified On');
```

Include empty system attributes in the requirements report:

```
RptgenRMI.doorsAttribs('nonempty', 'off');
```

Omit the **Object Heading** attribute from the requirements report. Use this option when the link label is always the same as the **Object Heading** for the target DOORS object and you do not want duplicate information in the requirements report:

```
RptgenRMI.doorsAttribs('remove', 'Object Heading');
```

See Also

`rmi`

Introduced in R2011b

slwebview_req

Export Simulink system to Web views with requirements

Syntax

```
filename = slwebview_req(sysname)
filename = slwebview_req(sysname,Name,Value)
```

Description

`filename = slwebview_req(sysname)` exports the system `sysname` and its children to a web page `filename` with contextual requirements information for the system displayed on a separate panel of the layered model structure Web view.

`filename = slwebview_req(sysname,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Note You can use `slwebview_req` only if you have also installed Simulink Report Generator™.

Examples

Export All Layers

Export all the layers (including libraries and masks) from the system `gcs` to the file `filename`

```
filename = slwebview_req(gcs, 'LookUnderMasks', 'all',
'FollowLinks', 'on')
```

Input Arguments

sysname — The system to export to a Web view file
character vector containing the path to the system | handle to a subsystem or block diagram | handle to a chart or subchart

Exports the specified system or subsystem and its child systems to a Web view file, with contextual requirements information for the system displayed on a separate panel of the layered model structure Web view. By default, child systems of the `sysname` system are also exported. Use the `SearchScope` name-value pair to export other systems, in relation to `sysname`.

Example: 'sysname'

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example:

SearchScope — Systems to export, relative to the sysname system
`'CurrentAndBelow'` (default) | `'Current'` | `'CurrentAndAbove'` | `'All'`

`'CurrentAndBelow'` exports the Simulink system or the Stateflow chart specified by `sysname` and all systems or charts that it contains.

`'Current'` exports only the Simulink system or the Stateflow chart specified by `sysname`.

`'CurrentAndAbove'` exports the Simulink system or the Stateflow chart specified by the `sysname` and all systems or charts that contain it.

`'All'` exports all Simulink systems or Stateflow charts in the model that contains the system or chart specified by `sysname`.

Data Types: `char`

`LookUnderMasks` — Specifies whether to export the ability to interact with masked blocks
`'none'` (default) | `'all'`

`'none'` does not export masked blocks in the Web view. Masked blocks are included in the exported systems, but you cannot access the contents of the masked blocks.

`'all'` exports all masked blocks.

Data Types: `char`

`FollowLinks` — Specifies whether to follow links into library blocks
`'off'` (default) | `'on'`

`'off'` does not allow you to follow links into library blocks in a Web view.

`'on'` allows you to follow links into library blocks in a Web view.

Data Types: `char`

`FollowModelReference` — Specifies whether to access referenced models in a Web view
`'off'` (default) | `'on'`

`'off'` does not allow you to access referenced models in a Web view.

`'on'` allows you to access referenced models in a Web view.

Data Types: `char`

`ViewFile` — Specifies whether to display the Web view in a Web browser when you export the Web view
`'on'` (default) | `'off'`

`'on'` displays the Web view in a Web browser when you export the Web view.

`'off'` does not display the Web view in a Web browser when you export the Web view.

Data Types: `char`

`ShowProgressBar` — Specifies whether to display the status bar when you export a Web view
'on' (default) | 'off'

`'on'` displays the status bar when you export a Web view.

`'off'` does not display the status bar when you export a Web view.

Data Types: `char`

Output Arguments

`filename` — The name of the HTML file for displaying the Web view
character vector

Reports the name of the HTML file for displaying the Web view. Exporting a Web view creates the supporting files, in a folder.

Tips

A Web view is an interactive rendition of a model that you can view in a Web browser. You can navigate a Web view hierarchically to examine specific subsystems and to see properties of blocks and signals.

You can use Web views to share models with people who do not have Simulink installed.

Web views require a Web browser that supports Scalable Vector Graphics (SVG).

See Also

`slwebview_cov`

Introduced in R2015a

Classes — Alphabetical List

slreq.Justification class

Package: slreq

Work with `slreq.Justification` objects

Description

Use `slreq.Justification` objects to work with requirements that you exclude from the implementation and verification status metrics roll-up for your requirements sets. Justify a requirement by creating an outgoing link from the `slreq.Justification` object to the requirement and setting the link type to **Implement** or **Verify**.

Construction

`jst = slreq.find(rs, 'Type', 'Justification', 'PropertyName', PropertyValue)` finds and returns an `slreq.Justification` object `jst` in the requirements set `rs` with additional properties specified by `PropertyName` and `PropertyValue`.

`jst = slreq.add(jt, 'PropertyName', PropertyValue)` adds a child justification `jst` to the parent justification `jt` with additional properties specified by `PropertyName` and `PropertyValue`.

Input Arguments

`rs` — Requirement set
`slreq.ReqSet` object

Requirement set, specified as an `slreq.ReqSet` object.

`jt` — Justification object
`slreq.Justification` object

Justification, specified as an `slreq.Justification` object.

Output Arguments

`jst` — Justification object
`slreq.Justification` object

Justification, returned as an `slreq.Justification` object.

Properties

`Id` — Justification custom ID
character vector

Custom ID of the justification, returned as a character vector. You cannot use spaces and `'#'` in custom IDs.

`Summary` — Justification summary
character vector

Justification summary text, specified as a one-line, plain text character vector.

`Description` — Justification description
character vector

Justification description text, specified as a multiline character vector.

`Rationale` — Justification rationale
character vector

Justification rationale text, specified as a multiline character vector.

`Keywords` — Justification keywords
character array

Justification keywords, specified as a character array.

`SID` — Justification Session Independent Identifier
character vector

The Session Independent Identifier corresponding to the justification.

`CreatedOn` — Date justification was created
`datetime` value

The date on which the justification was created, specified as a `datetime` value. The software populates this property.

`CreatedBy` — Justification creator
character vector

The name of the individual or organization who created the requirement.

`ModifiedOn` — Date justification was modified
`datetime` value

The date on which the justification was last modified, specified as a `datetime` value. The software populates this property.

`ModifiedBy` — Justification modifier
character vector

The name of the individual or organization who last modified the justification.

`FileRevision` — Justification revision number
scalar

Justification revision number, specified as a scalar.

`Dirty` — Unsaved changes indicator
`0` | `1`

Indicates if the justification has unsaved changes. `0` for no unsaved changes and `1` for unsaved changes.

Methods

add	Add child justification
children	Find children justifications
demote	Demote justifications
find	Find justifications
getAttribute	Get justification attributes
isHierarchical	Check if justification is hierarchical
parent	Find parent item of justification
promote	Promote justifications
remove	Remove justification items
reqSet	Return parent requirement set
setAttribute	Set justification attributes
setHierarchical	Change hierarchical justification status

Examples

```
% Find justification objects in a requirement set Project_reqs
myJustifications = find(Project_reqs, 'Type', 'Justification')

myJustifications =

  1×2 Justification array with properties:

    Id
    Summary
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedBy
    SID
    FileRevision
    ModifiedOn
    Dirty
```

```
    Comments

% Add a child justification to the first justification in the array
myChildJustification = add(myJustifications(1), 'Id', '2.1', ...
'Summary', 'New Child Justification')

myChildJustification =

  Justification with properties:

            Id: '2.1'
       Summary: 'New Child Justification'
   Description: ''
      Keywords: [0×0 char]
     Rationale: ''
     CreatedOn: 25-Aug-2017 14:37:29
     CreatedBy: 'Jane Doe'
    ModifiedBy: 'John Doe'
           SID: 73
  FileRevision: 1
    ModifiedOn: 26-Aug-2017 17:30:20
         Dirty: 0
      Comments: [0×0 struct]
```

See Also

`slreq.Reference` | `slreq.ReqSet` | `slreq.Requirement`

Introduced in R2018b

slreq.Link class

Package: slreq

Work with link objects

Description

When you establish a traceable association between artifacts, Simulink Requirementscreates an `slreq.Link` object to store source and destination data of the link.

Construction

`link = slreq.createLink(src, dest)` creates an `slreq.Link` object `link` with source and destination artifacts specified by `src` and `dest` respectively. The `slreq.Link` object is stored in the Link set file that belongs to `src`.

`outLinks = slreq.outLinks(src)` returns an array of `slreq.Link` objects `outLinks` that contains the outgoing links from the source artifact `src`.

`inLinks = slreq.inLinks(dest)` returns an array of `slreq.Link` objects `inLinks` that contains the incoming links to the destination artifact `dest`.

Input Arguments

`src` — Link source artifact
`struct`

Link source artifact, specified as a MATLAB structure.

`dest` — Link destination artifact
`struct`

Link destination artifact, specified as a MATLAB structure.

Output Arguments

`link` — Link object
`slreq.Link` object

Handle to a link, returned as an `slreq.Link` object.

`outLinks` — Outgoing links
`slreq.Link` object array

Array of outgoing links.

`inLinks` — Incoming links
`slreq.Link` object array

Array of incoming links.

Properties

`CreatedOn` — Date link was created
`datetime` value

The date on which the link was created, specified as a `datetime` value. The software populates this property.

`CreatedBy` — Link creator
character vector

The name of the individual or organization who created the link.

`ModifiedOn` — Date link was modified
`datetime` value

The date on which the link was last modified, specified as a `datetime` value. The software populates this property.

`ModifiedBy` — Link modifier
character vector

The name of the individual or organization who last modified the link.

`Comments` — Link comments
`struct`

The comments that are attached with the link, returned as a structure.

`Type` — Link type enumeration
`'Implement'` | `'Verify'` | `'Relate'` | `'Derive'` | `'Refine'`

The relationship between the source and the destination artifacts. For more information, see “Link Types”.

`Description` — Link description
character vector

Link descriptive text, specified as a multi-line character vector.

`Keywords` — Link keywords
character array

Link keywords, specified as character array.

`Rationale` — Link rationale
character vector

Link rationale text, specified as a multiline character vector.

Methods

destination	Get link destination artifact
isResolved	Check if the link is resolved
isResolvedDestination	Check if the link destination is resolved
isResolvedSource	Check if the link source is resolved
linkSet	Return parent link set
remove	Delete links
source	Get link source artifact

Examples

Create Links

```
% Create a link between the current Simulink Object and a requirement
link1 = slreq.createLink(gcb, REQ)

link1 =

  Link with properties:

          Type: 'Implement'
    Description: 'Plant Specs'
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 02-Sep-2017 15:49:28
      CreatedBy: 'Jane Doe'
     ModifiedOn: 21-Oct-2017 11:34:12
     ModifiedBy: 'John Doe'
       Comments: [0×0 struct]

% Create a link between a requirement and the current Stateflow object
link2 = slreq.createLink(REQ, sfgco);
```

Get Incoming Links

```
% Get the handle to a requirements set
myReqSet = slreq.find('Type', 'ReqSet', 'Name', 'Design_Spec');

% Get the handle to a requirement in myReqSet
myReq = find(myReqSet, 'Type', 'Requirement', 'Id', 'R1.1');

% Query incoming links to myReq
inLinks = slreq.inLinks(myReqs);
```

Get Outgoing Links

```
% Load a link set and get link sources
myLinkSet = slreq.load('fuelsys.slx');
allSrcs = myLinkSet.sources();
```

```
% Get outgoing links
myLinks1 = slreq.outLinks(allSrcs(1));
```

See Also

`slreq.LinkSet` | `slreq.createLink`

Introduced in R2018a

slreq.LinkSet class

Package: slreq

Work with link sets

Description

Instances of `slreq.LinkSet` are Link Set objects. Links are organized in Link Sets. Each Link Set is associated with a source artifact such as a Simulink model or a data dictionary and is serialized into a separate file which stores the links associated with it. The default location and name of the Link set file matches that of the source artifact.

Construction

`allLinkSets = slreq.find('Type', 'LinkSet')` finds and returns an array of all loaded `slreq.LinkSet` objects `allLinkSets`.

`myLinkSet = slreq.find('Type', 'LinkSet', 'Name', ArtifactName)` finds and returns an `slreq.LinkSet` object `myLinkSet` matching the artifact name specified by `ArtifactName`.

`myLinkSet = slreq.load(ArtifactName)` loads an `slreq.LinkSet` object `myLinkSet` matching the artifact name specified by `ArtifactName`.

Input Arguments

`ArtifactName` — Link set artifact name
character vector

The name of the link set artifact, specified as a character vector.

Output Arguments

`allLinkSets` — Link sets
`slreq.LinkSet` array

Array of all loaded link sets.

`myLinkSet` — Link set
`slreq.LinkSet` object

Link set, returned as an `slreq.LinkSet` object.

Properties

`Filename` — Link set file path
character vector

The file path of the link set, specified as a character vector.

`Artifact` — Container identifier
character vector

Top-level container identifier, such as a Microsoft Office document name, an IBM Rational DOORS Module unique ID, Simulink model name, or Simulink Test™ Test Manager file name.

`Domain` — Link set custom link type
character vector

The custom link type of the links in the link set. For more information, see “Custom Link Types”.

Example: `linktype_rmi_excel`, `linktype_rmi_doors`

`Revision` — Link set revision number
scalar

Link set revision number, specified as a scalar.

`Dirty` — Unsaved changes indicator
`0` | `1`

Indicates if the link set has unsaved changes. `0` for no unsaved changes and `1` for unsaved changes.

`Description` — Link set description
character vector

Link set description text, specified as a character vector.

Methods

save	Save link set
sources	Get link sources

Examples

```
% Find a link set
myLinkSet1 = slreq.find('Type', 'LinkSet', 'Name', 'Project_req')

myLinkSet1 =

  LinkSet with properties:

    Description: ''
       Filename: 'Project_req.slmx'
       Artifact: 'Project_req.slreqx'
         Domain: 'linktype_rmi_slreq'
       Revision: 2
          Dirty: 0

myLinkSet2 = slreq.load('fuelsys.slx')

myLinkset2 =

  LinkSet with properties:

    Description: ''
       Filename: 'C:\MATLAB\My_Files\fuelsys_linkset.slmx'
       Artifact: 'D:\Work\Design_Specs\fuelsys.slx'
         Domain: 'linktype_rmi_simulink'
       Revision: 2
          Dirty: 0

% Set the link set description
myLinkset2.Description = 'Link set for the fuel system'

myLinkset2 =
```

```
LinkSet with properties:

  Description: 'Link set for the fuel system'
     Filename: 'C:\MATLAB\My_Files\fuelsys_linkset.slmx'
     Artifact: 'D:\Work\Design_Specs\fuelsys.slx'
       Domain: 'linktype_rmi_simulink'
     Revision: 2
        Dirty: 1
```

See Also

`rmimap.map` | `slreq.Link`

Introduced in R2018a

slreq.Reference class

Package: slreq

Work with external requirement proxy objects

Description

Instances of `slreq.Reference` are proxies for external requirement objects that a third-party external application manages and maintains. Referenced requirement objects are read-only but can be synchronized from an external application and can exist only within a requirements set.

Construction

`ref = find(rs, 'Type', 'Reference', 'PropertyName', PropertyValue)` finds and returns a referenced requirement or a set of referenced requirements `ref` in the requirements set `rs` specified by the properties matching `PropertyName` and `PropertyValue`.

`ref = add(rs, 'Artifact', FileName, 'PropertyName', PropertyValue)` adds a referenced requirement `ref` to a requirements set `rs` which references requirements from the external document specified by `FileName` with properties and custom attributes specified by `PropertyName` and `PropertyValue`.

Input Arguments

`rs` — Requirement set object
`slreq.ReqSet` object

Requirement set, specified as an `slreq.ReqSet` object.

`FileName` — Container identifier
character vector

File name for a top-level container identifier, such as a Microsoft Office document name or an IBM Rational DOORS Module unique ID.

Output Arguments

`ref` — Referenced requirement

`slreq.Reference` object

Referenced requirement, specified as an `slreq.Reference` object.

Properties

`Id` — Referenced requirement ID

character vector

Referenced requirement ID, returned as a character vector.

`CustomId` — Referenced requirement Custom ID

character vector

Referenced requirement custom ID, returned as a character vector.

`Artifact` — Container identifier

character vector

Top-level container identifier, like a Microsoft Office document name or an IBM Rational DOORS Module unique ID.

`ArtifactId` — Requirement identifier

character vector

Unique requirement identifier in the source requirements document. For requirements imported from IBM Rational DOORS, the **ArtifactId** is the Numeric Object Id. For requirements imported from Microsoft Word, the bookmark names are used as the **ArtifactId**.

`Domain` — Requirements document custom link type

character vector

The custom link type of the requirements document. For more information, see “Custom Link Types”.

Example: `'linktype_rmi_doors', 'linktype_rmi_excel'`

`UpdatedOn` — Date and time referenced requirement was last updated
`datetime`

The date and time the referenced requirement was last synchronized with the external document, specified as a `datetime` value. The software automatically populates this property.

`IsLocked` — Referenced requirement lock indicator
1 (default) | 0

Indicates if the referenced requirement is locked. `1` for locked and `0` for unlocked.

`Summary` — Referenced requirement summary
character vector

Referenced requirement summary text, returned as a character vector.

`Description` — Referenced requirement description
character vector

Referenced requirement description text, returned as a multiline character vector.

`Rationale` — Referenced requirement rationale
character vector

Referenced requirement rationale text, returned as a multiline character vector.

`Keywords` — Referenced requirement keywords
character array

Referenced requirement keywords, specified as a character array.

`Type` — Referenced requirement type
character vector

Referenced requirement type. For more information, see “Requirement Types”.

`SID` — Referenced requirement Session Independent Identifier
character vector

The Session Independent Identifier corresponding to the referenced requirement.

`FileRevision` — Referenced requirement revision number
scalar

Referenced requirement revision number, specified as a scalar.

`ModifiedOn` — Date referenced requirement was modified
`datetime`

The date the referenced requirement was last modified, specified as a `datetime` value. The software automatically populates this property.

`ModifiedBy` — Referenced requirement modifier
character vector

The name of the individual or organization who last modified the referenced requirement.

`CreatedOn` — Date referenced requirement was created
`datetime`

The date the referenced requirement was created, specified as a `datetime` value. The software automatically populates this property.

`CreatedBy` — Referenced requirement creator
character vector

The name of the individual or organization who created the referenced requirement.

`Dirty` — Unsaved changes indicator
`0` | `1`

Indicates if the referenced requirement has unsaved changes. `0` for no unsaved changes and `1` for unsaved changes.

`Comments` — Referenced requirement comments
structure array

The comments that are attached with the referenced requirement, returned as a structure.

Methods

add	Add referenced requirements
addComment	Add comments to referenced requirements
children	Find children references
find	Find referenced requirements
getAttribute	Get referenced requirement custom attributes
getImplementationStatus	Query referenced requirement implementation status summary
getVerificationStatus	Query referenced requirement verification status summary
isJustifiedFor	Check if referenced requirement is justified
justifyImplementation	Justify referenced requirements for implementation
justifyVerification	Justify referenced requirements for verification
parent	Find parent item of referenced requirement
remove	Remove referenced requirements
reqSet	Return parent requirements set
setAttribute	Set referenced requirement custom attributes
unlock	Unlock referenced requirements
unlockAll	Unlock all child referenced requirements for editing
updateFromDocument	Update referenced requirements from external requirements document

Examples

Get the Handle to a Referenced Requirement

```
% Find a referenced requirement with Id R9 in a requirement set rs
ref = find(rs, 'Type', 'Reference', 'Id', 'R9')

ref =

  Reference with properties:
```

```
        Keywords: [0×0 char]
        Artifact: 'Req_doc.docx'
              Id: 'R9'
         Summary: 'System overview'
     Description: ''
             SID: 3
          Domain: 'linktype_rmi_word'
SynchronizedOn: 25-Jul-2017 11:34:02
```

See Also

`slreq.ReqSet` | `slreq.Requirement` | `slreq.import`

Introduced in R2018a

slreq.ReqSet class

Package: slreq

Work with Requirements sets

Description

Instances of `slreq.ReqSet` are Requirement Set objects.

Construction

`newReqSet = slreq.new(reqSetName)` creates a requirement set named `reqSetName` in the current working folder.

`newReqSet = slreq.new(reqSetPath)` creates a requirement set on the specified path.

Input Arguments

`reqSetName` — Requirement set name
character vector

Name of the requirement set, specified as a character vector.

Example: `'Design Requirements'`

`reqSetPath` — Requirement set file name and path
character vector

The file name and path of the requirement set, specified as a character vector.

Example: `'C:\MATLAB\myReqSet.slreqx'`

Output Arguments

`newReqSet` — Requirements set
`slreq.ReqSet` object

An instance of the `slreq.ReqSet` object.

Properties

`Name` — Requirements set name
character vector

Name of the requirements set, specified as a character vector.

`Filename` — Requirements set file path
character vector

The file path of the requirements set, specified as a character vector.

`Revision` — Requirements set revision number
scalar

Requirements set revision number, specified as a scalar.

`CreatedBy` — Requirements set creator
character vector

The name of the individual or organization who created the requirements set.

`CreatedOn` — Date requirements set was created
`datetime` value

The date the requirements set was created, specified as a `datetime` value. The software automatically populates this property.

`ModifiedBy` — Requirements set modifier
character vector

The name of the individual or organization who last modified the requirements set.

`ModifiedOn` — Date requirements set was modified
`datetime` value

The date the requirements set was last modified, specified as a `datetime` value. The software automatically populates this property.

`Description` — Requirements set description
character vector

Requirements set description text, specified as a character vector.

`Dirty` — Unsaved changes indicator
`0` | `1`

Indicates if the requirements set has unsaved changes. `0` for no unsaved changes, and `1` for unsaved changes.

`CustomAttributeNames` — Custom attributes associated with the requirements set
cell array of character vectors

Requirements set custom attribute names, specified as a cell array of character vectors.

Methods

addJustification	Add justifications to requirement set
close	Close a requirements set
createReferences	Create read-only references to requirement items in third-party documents
find	Find requirements in requirements set that have matching attribute values
getImplementationStatus	Query requirement set implementation status summary
getVerificationStatus	Query requirement set verification status summary
importFromDocument	Import editable requirements from external documents
save	Save a requirements set
updateImplementationStatus	Update requirement set implementation status summary
updateVerificationStatus	Update requirement set verification status summary

Examples

Create and Instantiate a Requirements Set Object

```
% Create a new requirements set
rs = slreq.new('My_Requirements_Set');

% Save and close the requirements set - saving creates a .slreqx file
save(rs);
close(rs);

% Load an existing requirements set
rs1 = slreq.load('Design_Specifications');

% Open the requirements set in the Requirements Editor
slreq.open(rs1);
```

See Also

`slreq.Link` | `slreq.LinkSet` | `slreq.Reference` | `slreq.Requirement`

Introduced in R2018a

slreq.Requirement class

Package: slreq

Work with Requirement objects

Description

Instances of `slreq.Requirement` are Requirement objects that you manage solely inside Simulink Requirements and that do not have a persistent association with artifacts managed by external applications. Requirement objects can exist only within a requirements set.

Construction

`req = find(rs, 'PropertyName', PropertyValue)` finds and returns a requirement `req` in the requirements set `rs` with additional requirement properties specified by `PropertyName` and `PropertyValue`.

`req = add(rs, 'PropertyName', PropertyValue)` adds a requirement `req` to the requirement set `rs` with additional requirement properties specified by `PropertyName` and `PropertyValue`.

Input Arguments

`rs` — Requirements set object
`slreq.ReqSet` object

Requirements set, specified as an `slreq.ReqSet` object.

Output Arguments

`req` — Requirement object
`slreq.Requirement` object

Handle to a requirement, returned as an `slreq.Requirement` object.

Properties

`Type` — Requirement type
character vector

Requirement type. For more information, see “Requirement Types”.

`Id` — Requirement custom ID
character vector

Custom ID of the requirement, returned as a character vector. You cannot use spaces and `'#'` in custom IDs.

`Summary` — Requirement summary
character vector

Requirement summary text, specified as a one-line, plain text character vector.

`Keywords` — Requirement keywords
character array

Requirement keywords, specified as character array.

`Description` — Requirement description
character vector

Requirement description text, specified as a multiline character vector.

`Rationale` — Requirement rationale
character vector

Requirement rationale text, specified as a multiline character vector.

`SID` — Requirement Session Independent Identifier
character vector

The Session Independent Identifier corresponding to the requirement.

`CreatedOn` — Date requirement was created
`datetime` value

The date on which the requirement was created, specified as a `datetime` value. The software populates this property.

`CreatedBy` — Requirement creator
character vector

The name of the individual or organization who created the requirement.

`ModifiedOn` — Date requirement was modified
`datetime` value

The date on which the requirement was last modified, specified as a `datetime` value. The software populates this property.

`ModifiedBy` — Requirement modifier
character vector

The name of the individual or organization who last modified the requirement.

`FileRevision` — Requirement revision number
scalar

Requirement revision number, specified as a scalar.

`Dirty` — Unsaved changes indicator
`0` | `1`

Indicates if the requirement has unsaved changes. `0` for no unsaved changes and `1` for unsaved changes.

`Comments` — Requirement comments
structure array

The comments that are attached with the requirement, returned as a structure.

Methods

add	Add requirement to requirements set
children	Find child requirements of a requirement
demote	Demote requirements
find	Find requirements that have matching attribute values
getAttribute	Get requirement custom attributes
getImplementationStatus	Query requirement implementation status summary
getVerificationStatus	Query requirement verification status summary
isJustifiedFor	Check if requirement is justified
justifyImplementation	Justify requirements for implementation
justifyVerification	Justify requirements for verification
parent	Find parent item of requirement
promote	Promote requirements
reqSet	Return parent requirements set
setAttribute	Set requirement custom attributes

Examples

Find a Requirement in a Requirements Set

```
% Find a requirement with ID 77 in a requirements set rs
req = find(rs, 'Type', 'Requirement', 'ID', '77');

req =

  Requirement with properties:

            Id: '77'
       Summary: 'Test Spec'
      Keywords: [0×0 char]
   Description: ''
     Rationale: ''
           SID: 80
```

```
     CreatedBy: 'John Doe'
     CreatedOn: 05-Oct-2007 16:09:38
    ModifiedBy: 'Jane Doe'
    ModifiedOn: 21-Dec-2016 11:10:05
      Comments: [0×0 struct]
```

See Also

`slreq.Link` | `slreq.Reference` | `slreq.ReqSet`

Introduced in R2018a

Methods — Alphabetical List

add

Class: slreq.Justification
Package: slreq

Add child justification

Syntax

```
childJustification = add(jt, 'PropertyName', PropertyValue)
```

Description

`childJustification = add(jt, 'PropertyName', PropertyValue)` adds a child justification `childJustification` to a justification object `jt` with additional properties specified by `PropertyName` and `PropertyValue`.

Input Arguments

`jt` — Justification object

`slreq.Justification` object

Justification, specified as an `slreq.Justification` object.

Output Arguments

`childJustification` — Requirement justification

`slreq.Justification` object

The child justification that was added, returned as an `slreq.Justification` object.

Examples

Add a Child Justification to a Parent Justification

```
% Load a requirement set file
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

% Add a justification object to the requirement set
just1 = addJustification(rs, 'Id', 'J1', ...
'Summary', 'Non-functional requirement justification');

% Add a child justification to the justification just1
childJust1 = add(just1, 'Id', 'J1.1', ...
'Summary', 'Justification for non-functional requirement')

childJust1 =

  Justification with properties:

             Id: 'J1.1'
        Summary: 'Justification for non-functional requirement'
    Description: ''
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 25-Aug-2017 11:21:29
      CreatedBy: 'John Doe'
     ModifiedBy: 'Jane Doe'
            SID: 11
   FileRevision: 2
     ModifiedOn: 25-Aug-2017 14:00:29
          Dirty: 0
       Comments: [0×0 struct]
```

See Also

children | remove

Introduced in R2018b

children

Class: slreq.Justification
Package: slreq

Find children justifications

Syntax

```
childJusts = children(jt)
```

Description

`childJusts = children(jt)` returns the child justifications `childJusts` of the `slreq.Justification` object `jt`.

Input Arguments

`jt` — Justification object

`slreq.Justification` object

Justification, specified as an `slreq.Justification` object.

Output Arguments

`childJusts` — Child justifications

`slreq.Justification` object | `slreq.Justification` object array

The child justifications belonging to the justification `jt`, returned as `slreq.Justification` objects.

Examples

Find Child Justifications

```
% Load a requirement set file and find justification objects
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
allJusts = find(rs, 'Type', 'Justification')

allJusts =

  1×20 Justification array with properties:

    Id
    Summary
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedBy
    SID
    FileRevision
    ModifiedOn
    Dirty
    Comments

jt1 = allJusts(1);

% Find the children of jt1
childJusts = children(jt1)

childJusts =

   1×10 Justification array with properties:

    Id
    Summary
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedBy
```

```
SID
FileRevision
ModifiedOn
Dirty
Comments
```

See Also

add | parent

Introduced in R2018b

demote

Class: slreq.Justification
Package: slreq

Demote justifications

Syntax

```
demote(jt)
```

Description

`demote(jt)` demotes the `slreq.Justification` object `jt` down one level in the hierarchy.

Input Arguments

`jt` — Justification object

`slreq.Justification` object

Justification, specified as an `slreq.Justification` object.

Examples

Demote a Justification

```
% Load a requirement set file and find justification objects
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

allJusts = find(rs, 'Type', 'Justification')

allJusts =
```

```
  1×20 Justification array with properties:

    Id
    Summary
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedBy
    SID
    FileRevision
    ModifiedOn
    Dirty
    Comments

jt1 = allJusts(1);

% Find the children of jt1
childJusts = children(jt1)

childJusts =

   1×10 Justification array with properties:

    Id
    Summary
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedBy
    SID
    FileRevision
    ModifiedOn
    Dirty
    Comments

% Demote the first child of jt1
demotedJustification = demote(childJusts(1));

% Find the parent of demotedJustification
parentJustification = parent(demotedJustification)
```

```
parentJustification =

    Justification with properties:

              Id: 'J1.1'
         Summary: 'Justifications'
     Description: ''
        Keywords: [0×0 char]
       Rationale: ''
       CreatedOn: 27-Feb-2014 10:15:38
       CreatedBy: 'Jane Doe'
      ModifiedBy: 'John Doe'
             SID: 34
    FileRevision: 21
      ModifiedOn: 02-Aug-2017 13:49:40
           Dirty: 1
        Comments: [0×0 struct]
```

See Also

children | parent | promote

Introduced in R2018b

find

Class: slreq.Justification
Package: slreq

Find justifications

Syntax

```
justs = find(rs, 'Type', 'Justification', 'PropertyName',
PropertyValue)
```

Description

`justs = find(rs, 'Type', 'Justification', 'PropertyName', PropertyValue)` finds and returns a justification or a set of justifications `justs` in the requirements set `rs` specified by the properties matching `PropertyName` and `PropertyValue`. Property name matching is case-insensitive.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

Requirements set specified as an `slreq.ReqSet` object.

Output Arguments

`justs` — Justification objects

`slreq.Justification` object | `slreq.Justification` object array

Justifications, returned as `slreq.Justification` objects.

Examples

Find Justifications That Have Matching Attribute Values

```
% Load a requirement set file
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

% Find a justification with Id J1.1 in the requirements set
matchedJust = find(rs, 'Type', 'Justification', 'Id', 'J1.1')

matchedJust =

  Justification with properties:

             Id: 'J1.1'
        Summary: 'Justifications'
    Description: 'Non-functional requirement justifications'
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 27-Feb-2017 10:34:22
      CreatedBy: 'Jane Doe'
     ModifiedBy: 'John Doe'
            SID: 71
   FileRevision: 1
     ModifiedOn: 03-Aug-2018 17:14:16
          Dirty: 0
       Comments: [0×0 struct]
```

Find Justifications by Using Regular Expression Matching

Construct regular expression search patterns by using the tilde (~) symbol to find justifications in your requirement set.

```
% Load a requirement set file
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

% Find all justifications that correspond to subsystems
subsystemJusts = find(rs, 'Type', 'Justification', 'Description', ...
'~Subsystems(?i)\w*')

subsystemJusts =
```

```
1×15 Justification array with properties:

  Id
  Summary
  Description
  Keywords
  Rationale
  CreatedOn
  CreatedBy
  ModifiedBy
  SID
  FileRevision
  ModifiedOn
  Dirty
  Comments
```

For more information on constructing regular expression search patterns, see “Steps for Building Expressions” (MATLAB).

See Also

`slreq.find`

Introduced in R2018b

getAttribute

Class: slreq.Justification
Package: slreq

Get justification attributes

Syntax

```
val = getAttribute(jt, propertyName)
```

Description

`val = getAttribute(jt, propertyName)` gets a justification property `propertyName` of the justification `jt`.

Input Arguments

`jt` — Justification object

`slreq.Justification` object

Justification, specified as an `slreq.Justification` object.

`propertyName` — Justification property

character vector

Justification property name.

Example: `'SID', 'CreatedOn', 'Summary'`

Examples

Get Justification Attributes

```
% Load a requirement set file and get the handle to one justification
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
jt1 = find(rs, 'Type', 'Justification', 'ID', 'J3.5');

% Get the Summary of jt1
summaryJt1 = getAttribute(jt1, 'Summary')

summaryJt1 =

  'Requirement Justification'
```

See Also

setAttribute

Introduced in R2018b

isHierarchical

Class: slreq.Justification
Package: slreq

Check if justification is hierarchical

Syntax

```
tf = isHierarchical(jt)
```

Description

`tf = isHierarchical(jt)` checks if the `slreq.Justification` object `jt` is part of a hierarchy of justifications and returns the Boolean `tf`.

Input Arguments

`jt` — Justification object

`slreq.Justification` object

Justification, specified as an `slreq.Justification` object.

Output Arguments

`tf` — Hierarchical justification status

`true` | `false`

The hierarchical justification status of the `slreq.Justification` object, returned as a Boolean.

Examples

Query Hierarchical Justification Status

```
% Load a requirement set file and find justification objects
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

allJusts = find(rs, 'Type', 'Justification')

allJusts =

  1×9 Justification array with properties:

    Id
    Summary
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedBy
    SID
    FileRevision
    ModifiedOn
    Dirty
    Comments

% Check if the first justification in allJusts is hierarchically justified
tf = isHierarchical(allJusts(1))

tf =

  logical

   0
```

See Also

children | setHierarchical

Introduced in R2018b

parent

Class: slreq.Justification
Package: slreq

Find parent item of justification

Syntax

```
parentObj = parent(jt)
```

Description

`parentObj = parent(jt)` returns the parent object `parentObj` of the `slreq.Justification` object `jt`.

Input Arguments

`jt` — Justification object

`slreq.Justification` object

Justification, specified as an `slreq.Justification` object.

Output Arguments

`parentObj` — Parent object

`slreq.Justification` object | `slreq.ReqSet` object

The parent of the justification `jt`, returned as an `slreq.Justification` object or as an `slreq.ReqSet` object.

Examples

Find Parent Justification

```
% Load a requirement set file and find justification objects
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
myJustifications = find(rs, 'Type', 'Justification')

myJustifications =

  1×13 Justification array with properties:

    Id
    Summary
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedBy
    SID
    FileRevision
    ModifiedOn
    Dirty
    Comments

% Find the parent of the first justification object
parentJust1 = parent(myJustifications(1))

parentJust1 =

  ReqSet with properties:

             Description: ''
                    Name: 'My_Requirements_Set_1'
                Filename: 'C:\MATLAB\My_Requirements_Set_1.slreqx'
                Revision: 6
                   Dirty: 1
    CustomAttributeNames: {}

% Find the parent of the third justification object
parentJust3 = parent(myJustifications(3))
```

```
parentJust3 =

  Justification with properties:

             Id: 'J1'
        Summary: 'Justifications'
    Description: ''
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 27-Feb-2014 10:15:38
      CreatedBy: 'Jane Doe'
     ModifiedBy: 'John Doe'
            SID: 35
   FileRevision: 11
     ModifiedOn: 02-Aug-2017 13:49:40
          Dirty: 1
       Comments: [0×0 struct]
```

See Also

children | demote | promote

Introduced in R2018b

promote

Class: slreq.Justification
Package: slreq

Promote justifications

Syntax

```
promote(jt)
```

Description

`promote(jt)` promotes the `slreq.Justification` object `jt` up one level in the hierarchy.

Input Arguments

`jt` — Justification object

`slreq.Justification` object

Justification, specified as an `slreq.Justification` object.

Examples

Promote a Justification

```
% Load a requirement set file and find justification objects
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

allJusts = find(rs, 'Type', 'Justification')

allJusts =
```

```
  1×20 Justification array with properties:

    Id
    Summary
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedBy
    SID
    FileRevision
    ModifiedOn
    Dirty
    Comments

jt1 = allJusts(1);

% Find the children of jt1
childJusts = children(jt1)

childJusts =

   1×10 Justification array with properties:

    Id
    Summary
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedBy
    SID
    FileRevision
    ModifiedOn
    Dirty
    Comments

% Promote the first child of jt1
promote(childJusts(1));

% Find the parent of childJusts(1)
parentJustification = parent(childJusts(1))
```

```
parentJustification =

  ReqSet with properties:

            Description: ''
                   Name: 'My_Requirements_Set_1'
               Filename: 'C:\MATLAB\My_Requirements_Set_1.slreqx'
               Revision: 81
                  Dirty: 1
    CustomAttributeNames: {}
```

See Also

children | demote | parent

Introduced in R2018b

remove

Class: slreq.Justification
Package: slreq

Remove justification items

Syntax

```
count = remove(jt, 'PropertyName', PropertyValue)
```

Description

`count = remove(jt, 'PropertyName', PropertyValue)` removes all child justification items belonging to the parent justification `jt` with additional properties specified by `PropertyName` and `PropertyValue`. Returns the number of items removed as `count`.

Input Arguments

`jt` — Parent justification object

`slreq.Justification` object

Parent justification, specified as an `slreq.Justification` object.

Output Arguments

`count` — Removed justification count

`double`

Number of justification items removed, returned as a double.

Examples

Remove Justification Items

```
% Load a requirement set file
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

% Find all justification objects in the requirement set
myJustifications = find(rs, 'Type', 'Justification')

myJustifications =

  1×10 Justification array with properties:

    Id
    Summary
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedBy
    SID
    FileRevision
    ModifiedOn
    Dirty
    Comments

% Remove all justification objects that were created by Jane Doe
count = remove(myJustifications, 'CreatedBy', 'Jane Doe')

count =

  5
```

See Also

add

Introduced in R2018b

reqSet

Class: slreq.Justification
Package: slreq

Return parent requirement set

Syntax

```
rsout = reqSet(jt)
```

Description

`rsout = reqSet(jt)` returns the parent requirement set `rsout`. The justification `jt` belongs to `rsout`.

Input Arguments

`jt` — Justification object

`slreq.Justification` object

Justification, specified as an `slreq.Justification` object.

Output Arguments

`rsout` — Parent requirement set

`slreq.ReqSet` object

The parent requirement set of the justification `jt`, returned as an `slreq.ReqSet` object.

Examples

Query Requirements Set Information

```
% Load a new requirement set file and select one justification
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
allJustifications = find(rs, 'Type', 'Justification');
jt = allJustifications(1);

% Query which requirement set jt belongs to
reqSet(jt)

ans =

  ReqSet with properties:

            Description: ''
                   Name: 'My_Requirements_Set_1'
               Filename: 'C:\MATLAB\My_Requirements_Set_1.slreqx'
               Revision: 65
                  Dirty: 0
    CustomAttributeNames: {}
              CreatedBy: 'John Doe'
              CreatedOn: 17-Dec-2016 10:02:30
             ModifiedBy: 'Jane Doe'
             ModifiedOn: 01-May-2016 11:20:21
```

See Also

parent | promote

Introduced in R2018b

setAttribute

Class: slreq.Justification
Package: slreq

Set justification attributes

Syntax

```
setAttribute(jt, propertyName, propertyValue)
```

Description

`setAttribute(jt, propertyName, propertyValue)` sets a justification property.

Input Arguments

`jt` — Justification object

`slreq.Justification` object

Justification, specified as an `slreq.Justification` object.

`propertyName` — Justification property

character vector

Justification property name.

Example: `'SID', 'CreatedOn', 'Summary'`

`propertyValue` — Justification property value

character vector

Justification property value.

Example: `'Test Justification', 'J4.5.4'`

Examples

Set Justification Attributes

```
% Load a requirement set file and get the handle to one justification
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
jt1 = find(rs, 'Type', 'Justification', 'ID', 'J2.1');

% Set the Summary of req1
setAttribute(jt1, 'Summary', 'Controller Requirement Justification');

jt1

jt1 =

  Justification with properties:

             Id: 'J2.1'
        Summary: 'Controller Requirement Justification'
    Description: ''
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 27-Feb-2014 10:15:38
      CreatedBy: 'Jane Doe'
     ModifiedBy: 'John Doe'
            SID: 37
   FileRevision: 25
     ModifiedOn: 02-Aug-2017 13:49:40
          Dirty: 1
       Comments: [0×0 struct]
```

See Also

getAttribute

Introduced in R2018b

setHierarchical

Class: slreq.Justification
Package: slreq

Change hierarchical justification status

Syntax

```
setHierarchical(jt, tf)
```

Description

`setHierarchical(jt, tf)` changes the hierarchical justification status of the `slreq.Justification` object `jt` as specified by the Boolean `tf`.

Input Arguments

`jt` — Justification object

`slreq.Justification` object

Justification, specified as an `slreq.Justification` object.

`tf` — Hierarchical justification status flag

`true` | `false`

The hierarchical justification status of the `slreq.Justification` object, specified as a Boolean.

Examples

Change Hierarchical Justification Status

```
% Load a requirement set file and find justification objects
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
```

```
allJusts = find(rs, 'Type', 'Justification')

allJusts =

  1×10 Justification array with properties:

    Id
    Summary
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedBy
    SID
    FileRevision
    ModifiedOn
    Dirty
    Comments

% Check if the first justification in allJusts is hierarchically justified
tf = isHierarchical(allJusts(1))

tf =

  logical

   0

% Change the first justification in allJusts to be hierarchically justified
setHierarchical(allJusts(1), true);
```

See Also

`isHierarchical` | `parent`

Introduced in R2018b

destination

Class: slreq.Link
Package: slreq

Get link destination artifact

Syntax

```
dest = destination(myLink)
```

Description

`dest = destination(myLink)` returns the source artifact `dest` of the link `myLink`.

Input Arguments

`myLink` — Link object

`slreq.Link` object

Link, specified as an `slreq.Link` object.

Output Arguments

`dest` — Destination artifact

`struct`

The link destination artifact, returned as a MATLAB structure.

Examples

Get Link Destination

```
% Load a requirement set file and select one link
rs = slreq.load('C:\MATLAB\My_Req_Set.slreqx');
allReqs = find(rs, 'Type', 'Requirement');
req = allReqs(1);
allIncomingLinks = inLinks(req);
myLink = allIncomingLinks(1);

% Get link destination
myDestination = destination(myLink)

myDestination =

  struct with fields:

     reqSet: 'My_Req_Set'
     domain: 'linktype_rmi_slreq'
    summary: 'My Requirement'
    details: ''
         id: ''
        sid: 12
```

See Also

`LinkSet` | `slreq.Link` | `source`

Introduced in R2018a

isResolved

Class: slreq.Link
Package: slreq

Check if the link is resolved

Syntax

```
tf = isResolved(myLink)
```

Description

`tf = isResolved(myLink)` checks if the link `myLink` is resolved. A link is unresolved if either its source or destination are not resolved.

Input Arguments

`myLink` — Link object

`slreq.Link` object

Handle to a link, specified as an `slreq.Link` object.

Output Arguments

`tf` — Link resolution status

`0` | `1`

The resolution status of the `slreq.Link` object, returned as a Boolean.

Examples

Check if Link is Resolved

```
isResolvedDestination(myLink)

ans =

  logical

   1

isResolvedSource(myLink)

ans =

  logical

   0

isResolved(myLink)

ans =

  logical

   0
```

See Also

isResolvedDestination | isResolvedSource

Introduced in R2019a

isResolvedDestination

Class: slreq.Link
Package: slreq

Check if the link destination is resolved

Syntax

```
tf = isResolvedDestination(myLink)
```

Description

`tf = isResolvedDestination(myLink)` checks if the destination of the link `myLink` is resolved.

Input Arguments

`myLink` — Link object

`slreq.Link` object

Handle to a link, specified as an `slreq.Link` object.

Output Arguments

`tf` — Link destination resolution status

`0` | `1`

The destination resolution status of the `slreq.Link` object, returned as a Boolean.

Examples

Check if Link Destination is Resolved

```
isResolvedDestination(myLink)

ans =

  logical

   1
```

See Also

isResolved | isResolvedSource

Introduced in R2019a

isResolvedSource

Class: slreq.Link
Package: slreq

Check if the link source is resolved

Syntax

```
tf = isResolvedSource(myLink)
```

Description

`tf = isResolvedSource(myLink)` checks if the source of the link `myLink` is resolved.

Input Arguments

`myLink` — Link object

`slreq.Link` object

Handle to a link, specified as an `slreq.Link` object.

Output Arguments

`tf` — Link source resolution status

`0` | `1`

The source resolution status of the `slreq.Link` object, returned as a Boolean.

Examples

Check if Link Source is Resolved

```
isResolved(myLink)

ans =

  logical

   0
```

See Also

isResolved | isResolvedDestination

Introduced in R2019a

linkSet

Class: slreq.Link
Package: slreq

Return parent link set

Syntax

```
lks = linkSet(myLink)
```

Description

`lks = linkSet(myLink)` returns the parent link set `lks` to which the link `myLink` belongs.

Input Arguments

`myLink` — Link object

`slreq.Link` object

Link, specified as an `slreq.Link` object.

Output Arguments

`lks` — Parent link set

`slreq.LinkSet` object

Parent link set of the link `myLink`, returned as an `slreq.LinkSet` object.

Examples

Query Link Set Information

```
% Load a requirement set file and select one requirement
rs = slreq.load('C:\MATLAB\My_Req_Set.slreqx');
allReqs = find(rs, 'Type', 'Requirement');
req = allReqs(1);

% Find the incoming links that belong to req
allInLinks = inLinks(req);

% Query link set information
myParentLinkSet = linkSet(allInLinks)

myParentLinkSet =

  LinkSet with properties:

    Description: ''
       Filename: 'model_controller.slmx'
       Artifact: 'model_controller.slx'
         Domain: 'linktype_rmi_simulink'
       Revision: 4
          Dirty: 0
```

See Also

`destination` | `slreq.Link` | `source`

Introduced in R2018a

remove

Class: slreq.Link
Package: slreq

Delete links

Syntax

```
remove(myLink)
```

Description

`remove(myLink)` deletes the link `myLink`.

Input Arguments

`myLink` — Link to delete

`slreq.Link` object

Link to delete, specified as an `slreq.Link` object.

Examples

Delete Link

```
% Delete a link myLink

remove(myLink);
```

See Also

`slreq.Link`

Introduced in R2019a

source

Class: slreq.Link
Package: slreq

Get link source artifact

Syntax

```
src = source(myLink)
```

Description

`src = source(myLink)` returns the source artifact `src` of the link `myLink`.

Input Arguments

`myLink` — Link object

`slreq.Link` object

Link, specified as an `slreq.Link` object.

Output Arguments

`src` — Source artifact

`struct`

The link source artifact, returned as a MATLAB structure.

Examples

Get Link Source

```
% Load a requirement set file and select one link
rs = slreq.load('C:\MATLAB\My_Req_Set.slreqx');
allReqs = find(rs, 'Type', 'Requirement');
req = allReqs(1);
allIncomingLinks = inLinks(req);
myLink = allIncomingLinks(1);

% Get link source
mySource = source(myLink)

mySource =

  struct with fields:

      domain: 'linktype_rmi_simulink'
    artifact: 'controller_model.slx'
          id: ':241'
```

See Also

`LinkSet` | `destination` | `slreq.Link`

Introduced in R2018a

save

Class: slreq.LinkSet
Package: slreq

Save link set

Syntax

```
save(lks)
save(lks, filePath)
```

Description

`save(lks)` saves the link set `lks` by using its file name.

`save(lks, filePath)` saves the link set `lks` and updates its Name and Filename properties.

Input Arguments

`lks` — Link set file

`slreq.LinkSet` object

Link set file, specified as an `slreq.LinkSet` object.

`filePath` — File name and path

character vector

The file name and path of the link set, specified as a character vector.

Example: `'C:\MATLAB\myLinkSet.slmx'`

Examples

Save Link Set File

```
% Load a link set file
myLinkSet = slreq.load(get_param('fuelsys', 'Name'));

% Save the link set file
save(myLinkSet);

% Save the link set file by another name
save(myLinkSet, 'C:\MATLAB\Files\MyLinkSet1.slmx');
```

See Also

`slreq.LinkSet` | `sources`

Introduced in R2018a

sources

Class: slreq.LinkSet
Package: slreq

Get link sources

Syntax

```
linkSetSources = sources(lks)
```

Description

`linkSetSources = sources(lks)` returns an array of structures `linkSetSources` that contains the link sources of all the links in the link set `lks`.

Input Arguments

`lks` — Link set

`slreq.LinkSet` object

Instance of an `slreq.LinkSet` object.

Output Arguments

`linkSetSources` — Link set sources

structure

Link set source data, returned as a MATLAB structure.

Examples

Get Link Sources

```
% Load a link set and get link sources
myLinkSet = slreq.load('fuelsys.slx');
mySources = sources(myLinkSet)

mySources =

  1×16 struct array with fields:

    domain
    artifact
    id
```

See Also

save | slreq.LinkSet

Introduced in R2018a

add

Class: slreq.Reference
Package: slreq

Add referenced requirements

Syntax

```
refNew = add(rs, 'Artifact',FileName,'PropertyName',PropertyValue)
refChild = add(ref,'Artifact',FileName,'PropertyName',PropertyValue)
```

Description

`refNew = add(rs, 'Artifact',FileName,'PropertyName',PropertyValue)` adds a referenced requirement `refNew` to a requirements set `rs` which references requirements from the external document specified by `FileName` with properties and custom attributes specified by `PropertyName` and `PropertyValue`.

`refChild = add(ref,'Artifact',FileName,'PropertyName',PropertyValue)` adds a referenced child requirement `refChild` to a referenced requirement `ref` which references requirements from the external document specified by `FileName` with properties and custom attributes specified by `PropertyName` and `PropertyValue`.

Input Arguments

`rs` — Requirements set file

`slreq.ReqSet` object

Requirements set file, specified as an `slreq.ReqSet` object.

`ref` — Referenced requirement

`slreq.Reference` object

Referenced requirement, specified as an `slreq.Reference` object.

`FileName` — Container identifier
character vector

File name for a top-level container identifier, such as a Microsoft Office document name or an IBM Rational DOORS Module unique ID.

Output Arguments

`refNew` — Referenced requirement
`slreq.Reference` object

The referenced requirement that was added, returned as an `slreq.Reference` object.

`refChild` — Referenced child requirement
`slreq.Reference` object

The referenced child requirement that was added, returned as an `slreq.Reference` object.

Examples

Add a Referenced Requirement

```
% Load a requirement set file

rs = slreq.load('C:\MATLAB\My_Requirement_Set_1.slreqx');

% The parent external document for rs is Req_doc.docx
% Add a top-level referenced requirement to rs
newRef1 = add(rs, 'Artifact', 'crs_req.docx', 'Id', '5.0', 'Summary', ...
 'Additional Requirement');

% Add a child referenced requirement to newRef1
newRef2 = add(newRef1, 'Artifact', 'crs_req.docx', 'Id', '5.1', 'Summary', ...
'Additional Child Requirement');
```

See Also

`slreq.Reference` | `slreq.ReqSet`

Introduced in R2018a

addComment

Class: slreq.Reference
Package: slreq

Add comments to referenced requirements

Syntax

```
newComment = addComment(myRef, 'myComment')
```

Description

`newComment = addComment(myRef, 'myComment')` adds a comment `newComment` to the referenced requirement `myRef`.

Input Arguments

`myRef` — Referenced requirement
`slreq.Reference` object

The referenced requirement to which you add a comment to, specified as an `slreq.Reference` object.

Output Arguments

`newComment` — Comment
`struct`

Comment added to the referenced requirement, returned as a structure containing these fields.

`CommentedBy` — Referenced requirement commenter
character vector

The name of the individual or organization who commented on the referenced requirement, returned as a character vector.

`CommentedOn` — Date comment was added
`datetime`

The date on which the comment was added to the referenced requirement, returned as a `datetime` value.

`CommentedRevision` — Comment revision number
scalar

Referenced requirement comment revision number, specified as a scalar.

`Text` — Comment text
character vector

The text of the added comment, returned as a character vector.

Examples

Add a Comment to a Referenced Requirement

```
myComment = addComment(myRef, 'New comment')

myComment =

  struct with fields:

          CommentedBy: 'Jane Doe'
          CommentedOn: 21-Dec-2018 13:39:11
    CommentedRevision: 1
                 Text: 'New comment'
```

See Also

`getAttribute`

Introduced in R2019a

children

Class: slreq.Reference
Package: slreq

Find children references

Syntax

```
childRefs = children(ref)
```

Description

`childRefs = children(ref)` returns the child referenced requirements `childRefs` of the `slreq.Reference` object `ref`.

Input Arguments

`ref` — Referenced requirement instance

`slreq.Reference` object

Reference to a requirement specified as an `slreq.Reference` object.

Output Arguments

`childRef` — Child references

`slreq.Reference` object | `slreq.Reference` object array

The child referenced requirements belonging to the referenced requirement `ref`, returned as `slreq.Reference` objects.

Examples

Find Child References

```
% Load a requirements set file and find referenced requirements
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
allRefs = find(rs, 'Type', 'Reference')

allRefs =

  1×32 Reference array with properties:

    Keywords
    Artifact
    Id
    Summary
    Description
    SID
    Domain
    SynchronizedOn
    ModifiedOn

ref1 = allRefs(1);

% Find the children of ref1
childRef = children(ref1)

childRef =

  Reference with properties:

          Keywords: [0×0 char]
          Artifact: 'Req_doc.docx'
                Id: 'R1.1'
           Summary: 'References'
       Description: ''
               SID: 2
            Domain: 'linktype_rmi_word'
    SynchronizedOn: 26-Jul-2015 15:45:22
        ModifiedOn: 27-Jul-2015 12:00:13
```

See Also

`parent` | `slreq.Reference` | `slreq.ReqSet`

Introduced in R2018a

find

Class: slreq.Reference
Package: slreq

Find referenced requirements

Syntax

```
refs = find(rs, 'Type', 'Reference', 'PropertyName', PropertyValue)
```

Description

`refs = find(rs, 'Type', 'Reference', 'PropertyName', PropertyValue)` finds and returns a referenced requirement or a set of referenced requirements `refs` in the requirements set `rs` specified by the properties matching `PropertyName` and `PropertyValue`. Property name matching is case-insensitive.

Input Arguments

`rs` — Requirements set

`slreq.ReqSet` object

Requirements set specified as an `slreq.ReqSet` object.

Output Arguments

`refs` — Referenced requirements

`slreq.Reference` object | `slreq.Reference` object array

Referenced requirements, returned as `slreq.Reference` objects.

Examples

Find Referenced Requirements That Have Matching Attribute Values

```
% Load a requirement set file
rs = slreq.load('C:\MATLAB\My_Requirement_Set_1.slreqx');

% Find all referenced requirements with Id R10 in the requirement set
matchedRefs = find(rs, 'Type', 'Reference', 'Id', 'R10')

matchedRefs =

  Reference with properties:

         Keywords: [0×0 char]
         Artifact: 'Req_doc.docx'
               Id: 'R10'
          Summary: 'System overview'
      Description: ''
              SID: 3
           Domain: 'linktype_rmi_word'
    SynchronizedOn: 23-Jul-2017 12:47:23
```

Find Referenced Requirements by Using Regular Expression Matching

You can search for referenced requirements in your requirements sets by constructing regular expression search patterns by using the tilde (~) symbol.

```
% Load a requirement set file
rs = slreq.load('C:\MATLAB\My_Requirement_Set_1.slreqx');

% Find all referenced requirements that correspond to subsystems
subsystemReqs = find(rs, 'Type', 'Reference', 'Description', '~Subsystems(?i)\w*')

subsystemReqs =

  1×40 Reference array with properties:

    Keywords
```

```
Artifact
Id
Summary
Description
SID
Domain
SynchronizedOn
ModifiedOn
```

For more information on constructing regular expression search patterns, see “Steps for Building Expressions” (MATLAB).

See Also

`slreq.Reference` | `slreq.ReqSet` | `slreq.find`

Introduced in R2018a

getAttribute

Class: slreq.Reference
Package: slreq

Get referenced requirement custom attributes

Syntax

```
val = getAttribute(ref, propertyName)
```

Description

`val = getAttribute(ref, propertyName)` gets a referenced requirement property.

Input Arguments

`ref` — Referenced requirement instance
`slreq.Reference` object

Reference to a requirement specified as an `slreq.Reference` object.

`propertyName` — Referenced requirement property
character vector

Referenced requirement property name.

Example: `'SID', 'CreatedOn', 'Summary'`

Examples

Get Referenced Requirement Attributes

```
% Load a requirement set file and get the handle to
% one referenced requirement
```

```
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
ref1 = find(rs, 'Type', 'Reference', 'Id', 'R10.1');

% Get the Priority (custom attribute) of ref1
summaryRef1 = getAttribute(ref1, 'Priority')

summaryRef1 =

  'Medium'
```

See Also

setAttribute | slreq.Reference | slreq.ReqSet

Introduced in R2018a

getImplementationStatus

Class: slreq.Reference
Package: slreq

Query referenced requirement implementation status summary

Syntax

```
status = getImplementationStatus(ref)
status = getImplementationStatus(ref, 'self')
```

Description

`status = getImplementationStatus(ref)` returns the implementation status summary for the referenced requirement `ref` and all its child references.

`status = getImplementationStatus(ref, 'self')` returns the implementation status summary for just the referenced requirement `ref`.

Input Arguments

`ref` — Referenced requirement instance

`slreq.Reference` object

Referenced requirement instance, specified as an `slreq.Reference` object.

Output Arguments

`status` — Referenced requirement implementation status summary

structure

The implementation status summary for the referenced requirement and its child references, returned as a MATLAB structure containing these fields.

`total` — Total number of referenced requirements
`double`

The total number of Functional referenced requirements (including child references), returned as a `double`.

`implemented` — Implemented referenced requirements
`double`

The total number of implemented referenced requirements (including child references), returned as a `double`.

`justified` — Justified referenced requirements
`double`

The total number of referenced requirements (including child references), justified for implementation, returned as a `double`.

`none` — Unimplemented referenced requirements
`double`

The total number of unimplemented referenced requirements (including child references), returned as a `double`.

Examples

Get Implementation Status Summary of a Referenced Requirement

```
% Get the implementation status summary of the referenced requirement ref
% and all its child references
refImplStatus = getImplementationStatus(ref)

refImplStatus =

  struct with fields:

         total: 35
    implemented: 23
      justified: 9
```

```
           none: 3

% Get the implementation status summary of only the referenced requirement myRef
myRefImplStatus = getImplementationStatus(myRef, 'self')

myRefImplStatus =

  struct with fields:

   implemented: 0
     justified: 0
          none: 0
```

See Also

`updateImplementationStatus`

Introduced in R2018b

getVerificationStatus

Class: slreq.Reference
Package: slreq

Query referenced requirement verification status summary

Syntax

```
status = getVerificationStatus(ref)
status = getVerificationStatus(ref, 'self')
```

Description

`status = getVerificationStatus(ref)` returns the verification status summary for the referenced requirement `ref` and all its child references.

`status = getVerificationStatus(ref, 'self')` returns the verification status summary for just the referenced requirement `ref`.

Input Arguments

`ref` — Referenced requirement instance

`slreq.Reference` object

Referenced requirement instance, specified as an `slreq.Reference` object.

Output Arguments

`status` — Referenced requirement verification status summary

structure

The verification status summary for the referenced requirement and its child references, returned as a MATLAB structure containing these fields.

`total` — Total number of referenced requirements
`double`

The total number of referenced requirements (including child references) with Verify links, returned as a `double`.

`passed` — Passed referenced requirements
`double`

The total number of referenced requirements (including child references) that passed the tests associated with them, returned as a `double`.

`failed` — Failed referenced requirements
`double`

The total number of referenced requirements (including child references) that failed the tests associated with them, returned as a `double`.

`unexecuted` — Unexecuted requirements
`double`

The total number of referenced requirements (including child references) with unexecuted associated tests, returned as a `double`.

`justified` — Justified referenced requirements
`double`

The total number of referenced requirements (including child references) that are justified for verification, returned as a `double`.

`none` — Unlinked referenced requirements
`double`

The total number of referenced requirements (including child references) without links to verification objects, returned as a `double`.

Examples

Get Verification Status Summary of Referenced Requirements

```
% Get the verification status summary of the referenced requirement ref
% and all its child references
```

```
refVerifStatus = getVerificationStatus(ref)

refVerifStatus =

  struct with fields:

          total: 70
         passed: 45
         failed: 7
     unexecuted: 10
      justified: 1
           none: 7

% Get the verification status summary of only the referenced requirement myRef
myRefVerifStatus = getVerificationStatus(myRef, 'self')

myRefVerifStatus =

  struct with fields:

        passed: 1
        failed: 0
    unexecuted: 0
     justified: 0
          none: 0
```

See Also

updateVerificationStatus

Introduced in R2018b

isJustifiedFor

Class: slreq.Reference
Package: slreq

Check if referenced requirement is justified

Syntax

```
tf = isJustifiedFor(ref, linkType)
```

Description

`tf = isJustifiedFor(ref, linkType)` checks if the referenced requirement `ref` is justified for the link type specified by `linkType`.

Input Arguments

`ref` — Referenced requirement instance

`slreq.Reference` object

Referenced requirement to check for justification, specified as an `slreq.Reference` object.

`linkType` — Justification link type

`'Implement'` | `'Verify'`

Justification link type, specified as a character vector.

Output Arguments

`tf` — Justification status

`0` | `1`

The justification status of the referenced requirement, returned as a Boolean.

Examples

Check if Referenced Requirements Are Justified

```
% Check if referenced requirement ref1 is justified for Implementation
ref1_Status = isJustifiedFor(ref1, 'Implement')

ref1_Status =

  logical

   1

% Check if referenced requirement ref2 is justified for Verification
ref2_Status = isJustifiedFor(ref2, 'Verify')

ref2_Status =

  logical

   0
```

See Also

`getImplementationStatus` | `getVerificationStatus`

Introduced in R2018b

justifyImplementation

Class: slreq.Reference
Package: slreq

Justify referenced requirements for implementation

Syntax

```
implementationJustLink = justifyImplementation(ref, jt)
```

Description

`implementationJustLink = justifyImplementation(ref, jt)` justifies the referenced requirement `ref` for implementation by creating a link `implementationJustLink` from the justification `jt` to `ref`.

Input Arguments

`ref` — Referenced requirement instance
`slreq.Reference` object

Referenced requirement to justify for implementation, specified as an `slreq.Reference` object.

`jt` — Justification object
`slreq.Justification` object

Justification object to justify `ref` for implementation, specified as an `slreq.Justification` object.

Output Arguments

`implementationJustLink` — Justification link
`slreq.Link` object

Link to justification object `jt` of type **Implement**, returned as an `slreq.Link` object.

Examples

```
% Justify referenced requirement myRef for implementation
% by using a justification object myJust

myImplJustification = justifyImplementation(myRef, myJust)

myImplJustification =

  Link with properties:

          Type: 'Implement'
    Description: 'Cruise Control Mode (crs_req_func_spec#1)'
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 13-Jan-2017 13:45:12
      CreatedBy: 'John Doe'
     ModifiedOn: 24-Oct-2018 12:25:30
     ModifiedBy: 'Jane Doe'
       Revision: 6
       Comments: [0×0 struct]
```

See Also

`addJustification` | `getImplementationStatus`

Introduced in R2018b

justifyVerification

Class: slreq.Reference
Package: slreq

Justify referenced requirements for verification

Syntax

```
verificationJustLink = justifyVerification(ref, jt)
```

Description

`verificationJustLink = justifyVerification(ref, jt)` justifies the referenced requirement `ref` for verification by creating a link `verificationJustLink` from the justification `jt` to `ref`.

Input Arguments

`ref` — Referenced requirement instance

`slreq.Reference` object

Referenced requirement to justify for verification, specified as an `slreq.Reference` object.

`jt` — Justification object

`slreq.Justification` object

Justification object to justify `ref` for verification, specified as an `slreq.Justification` object.

Output Arguments

`verificationJustLink` — Justification link

`slreq.Link` object

Link to justification object `jt` of type **Verify**, returned as an `slreq.Link` object.

Examples

```
% Justify referenced requirement myRef for verification
% by using a justification object myJust

myVerifJustification = justifyVerification(myRef, myJust)

myVerifJustification =

  Link with properties:

          Type: 'Verify'
    Description: 'Brake Test (crs_req_func_spec#73)'
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 25-Nov-2017 10:11:35
      CreatedBy: 'John Doe'
     ModifiedOn: 26-Feb-2018 17:16:09
     ModifiedBy: 'Jane Doe'
       Revision: 7
       Comments: [0×0 struct]
```

See Also

`addJustification` | `getVerificationStatus`

Introduced in R2018b

parent

Class: slreq.Reference
Package: slreq

Find parent item of referenced requirement

Syntax

```
parentObj = parent(ref)
```

Description

`parentObj = parent(ref)` returns the parent object `parentObj` of the `slreq.Reference` object `req`.

Input Arguments

`ref` — Referenced requirement instance

`slreq.Reference` object

Referenced requirement specified as an `slreq.Reference` object.

Output Arguments

`parentObj` — Parent object

`slreq.Reference` object | `slreq.ReqSet` object

The parent of the referenced requirement `ref`, returned as an `slreq.Reference` object or as an `slreq.ReqSet` object.

Examples

Find Parent References

```
% Load a requirements set file and find referenced requirements
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
refs = find(rs, 'Type', 'Reference')

refs =

  1×32 Reference array with properties:

    Keywords
    Artifact
    Id
    Summary
    Description
    SID
    Domain
    SynchronizedOn
    ModifiedOn

% Find the parent of the first reference element
parentRef1 = parent(refs(1));

parentRef1 =

  ReqSet with properties:

            Description: ''
                   Name: 'My_Requirements_Set_1'
               Filename: 'C:\MATLAB\My_Requirements_Set_1.slreqx'
               Revision: 6
                  Dirty: 1
    CustomAttributeNames: {}
```

See Also

children | slreq.Reference | slreq.ReqSet

Introduced in R2018a

remove

Class: slreq.Reference
Package: slreq

Remove referenced requirements

Syntax

```
count = remove(topRef)
```

Description

`count = remove(topRef)` removes all the child referenced requirements under the Import node `topRef` and returns the number of referenced requirements removed `count`.

Input Arguments

`topRef` — Import node

`slreq.Reference` object

Import node, specified as an `slreq.Reference` object.

Output Arguments

`count` — Removed referenced requirements count

double

The number of referenced requirements removed, returned as a double.

Examples

Remove Referenced Requirements

```
% Load a requirement set file
rs = slreq.load('C:\MATLAB\My_Requirement_Set_1.slreqx');

% Find all referenced requirements in the requirement set
allRefs = find(rs, 'Type', 'Reference')

allRefs =

  1×46 Reference array with properties:

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

% Remove the top Import node and all child referenced requirements under it
count = remove(allRefs(1))

count =

    46
```

See Also

add

Introduced in R2019a

reqSet

Class: slreq.Reference
Package: slreq

Return parent requirements set

Syntax

```
rsout = reqSet(ref)
```

Description

`rsout = reqSet(ref)` returns the parent requirements set `rsout` to which the referenced requirement `ref` belongs.

Input Arguments

`ref` — Referenced requirement object

`slreq.Reference` object

Referenced requirement, specified as a `slreq.Reference` object.

Output Arguments

`rsout` — Parent requirements set

`slreq.ReqSet` object

The parent requirements set of the referenced requirement `ref`, returned as an `slreq.ReqSet` object.

Examples

Query Requirements Set Information

```
% Load a new requirements set file and select one referenced requirement
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
allRefs = find(rs,'Type','Reference');
ref = allRefs(1);

% Query which requirements set ref belongs to
reqSet(ref)

ans =

  ReqSet with properties:

            Description: ''
                   Name: 'My_Requirements_Set_1'
               Filename: 'C:\MATLAB\My_Requirements_Set_1.slreqx'
               Revision: 65
                  Dirty: 0
    CustomAttributeNames: {}
```

See Also

`parent` | `slreq.Reference` | `slreq.ReqSet`

Introduced in R2018a

setAttribute

Class: slreq.Reference
Package: slreq

Set referenced requirement custom attributes

Syntax

```
setAttribute(ref, propertyName, propertyValue)
```

Description

`setAttribute(ref, propertyName, propertyValue)` sets a referenced requirement property. Use this method to set the values of custom attributes that you define for your requirements set.

Input Arguments

`ref` — Referenced requirement instance
`slreq.Reference` object

Referenced requirement specified as an `slreq.Reference` object.

`propertyName` — Referenced requirement custom attribute
character vector

Referenced requirement custom attribute name.

Example: `'Priority'`

`propertyValue` — Referenced requirement custom attribute value
character vector

Referenced requirement custom attribute name, specified as a character vector.

Example: `'High', 'Medium'`

Examples

Set Referenced Requirement Custom Attribute

```
% Load a requirements set file and get the handle to one requirement
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
ref1 = find(rs, 'Type', 'Reference', 'ID', 'R20.1');

% Set the Priority (custom attribute) of ref1
setAttribute(ref1, 'Priority', 'Low');
```

See Also

`getAttribute` | `slreq.Reference` | `slreq.ReqSet`

Introduced in R2018a

unlock

Class: slreq.Reference
Package: slreq

Unlock referenced requirements

Syntax

```
unlock(ref)
```

Description

`unlock(ref)` unlocks a referenced requirement for editing.

Input Arguments

`ref` — Referenced requirement

`slreq.Reference` object

Referenced requirement to unlock, specified as an `slreq.Reference` object.

Examples

Unlock an Imported Referenced Requirement

```
% Load a requirement set file
rs = slreq.load('C:\MATLAB\My_Requirement_Set_1.slreqx');

% Find all referenced requirements in the requirement set
allRefs = find(rs, 'Type', 'Reference')

allRefs =
```

```
  1×73 Reference array with properties:

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

% Unlock a referenced requirement
unlock(allRefs(25))
```

See Also

unlockAll

Introduced in R2019a

unlockAll

Class: slreq.Reference
Package: slreq

Unlock all child referenced requirements for editing

Syntax

```
unlockAll(topRef)
```

Description

`unlockAll(topRef)` unlocks all the child referenced requirements of the top Import node `topRef`.

Input Arguments

`topRef` — Import node
`slreq.Reference` object

Import node, specified as an `slreq.Reference` object.

Examples

Unlock all the Children of a Parent Referenced Requirement

```
% Load a requirement set file
rs = slreq.load('C:\MATLAB\My_Requirement_Set_1.slreqx');

% Find all referenced requirements in the requirement set
allRefs = find(rs, 'Type', 'Reference')

allRefs =
```

```
  1×25 Reference array with properties:

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

```
% Unlock all child referenced requirements of the top Import node
unlockall(allRefs(1))
```

See Also

unlock

Introduced in R2019a

updateFromDocument

Class: slreq.Reference
Package: slreq

Update referenced requirements from external requirements document

Syntax

```
result = updateFromDocument(topRef)
```

Description

`result = updateFromDocument(topRef)` updates all the referenced requirements under the Import node `topRef`.

Input Arguments

`topRef` — Import node

`slreq.Reference` object

Import node, specified as an `slreq.Reference` object.

Output Arguments

`result` — Update confirmation

character vector

The result of the Update operation (pass or fail), returned as a character vector.

Examples

Update Referenced Requirements

```
% Load a requirement set file
rs = slreq.load('C:\MATLAB\My_Requirement_Set_1.slreqx');

% Find all referenced requirements in the requirement set
allRefs = find(rs, 'Type', 'Reference')

allRefs =

  1×46 Reference array with properties:

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

% Update all child referenced requirements of the top Import node
result = updateFromDocument(allRefs(1))

result =

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

See Also

`slreq.import`

Topics

“Update Imported Requirements”

Introduced in R2019a

addJustification

Class: slreq.ReqSet
Package: slreq

Add justifications to requirement set

Syntax

```
jt = addJustification(rs)
jt = addJustification(rs, 'PropertyName', PropertyValue)
```

Description

`jt = addJustification(rs)` adds a justification `jt` to the requirement set `rs`.

`jt = addJustification(rs, 'PropertyName', PropertyValue)` adds a justification `jt` to the requirement set `rs` with additional properties specified by `PropertyName` and `PropertyValue`.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

Requirement set, specified as an `slreq.ReqSet` object.

Output Arguments

`jt` — Justification object

`slreq.Justification` object

Justification added to the requirement set `rs`, returned as an `slreq.Justification` object.

Examples

Add Justifications to Requirement Set

```
% Add a justification jt1 to a requirement set rs
jt1 = addJustification(rs)

jt1 =

  Justification with properties:

             Id: '70'
        Summary: ''
    Description: ''
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 16-Jan-2018 10:53:28
      CreatedBy: 'John Doe'
     ModifiedBy: 'Jane Doe'
            SID: 76
   FileRevision: 1
     ModifiedOn: 16-Feb-2018 12:50:43
          Dirty: 0
       Comments: [0×0 struct]

% Add a justification jt2 to a requirement set rs and specify properties
jt2 = addJustification(rs, 'Summary', 'New justification', ...
'Description', 'Justify safety requirement')

jt2 =

  Justification with properties:

             Id: '71'
        Summary: 'New justification'
    Description: 'Justify safety requirement'
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 11-Feb-2018 11:45:12
      CreatedBy: 'John Doe'
     ModifiedBy: 'Jane Doe'
            SID: 77
   FileRevision: 1
```

```
      ModifiedOn: 12-Feb-2018 13:01:08
           Dirty: 0
        Comments: [0×0 struct]
```

See Also

`justifyImplementation` | `justifyImplementation` | `justifyVerification` | `justifyVerification`

Introduced in R2018b

close

Class: slreq.ReqSet
Package: slreq

Close a requirements set

Syntax

```
close(rs)
```

Description

`close(rs)` closes a requirements set.

Input Arguments

`rs` — Requirements set file

`slreq.ReqSet` object

Requirements set file, specified as an `slreq.ReqSet` object.

Examples

Close a Requirement Set

```
% Create a new requirements set file
rs1 = slreq.new('C:\MATLAB\My_Requirements_Set_1.slreqx');

% Save the requirements set file
save(rs1);

% Close the requirements set file
close(rs1);
```

See Also

`slreq.ReqSet`

Introduced in R2018a

createReferences

Class: slreq.ReqSet
Package: slreq

Create read-only references to requirement items in third-party documents

Syntax

```
createReferences(rs, pathToFile, Name, Value)
createReferences(rs, reqFormat, Name, Value)
```

Description

`createReferences(rs, pathToFile, Name, Value)` creates read-only references to requirements content in an external document at `pathToFile` by using additional `Name, Value` arguments to specify import options.

`createReferences(rs, reqFormat, Name, Value)` creates read-only references to requirements content in an external document corresponding to the specified registered document type specified by `reqFormat` by using additional `Name, Value` arguments to specify import options.

Input Arguments

`rs` — Requirements set file

`slreq.ReqSet` object

Requirements set file, specified as a `slreq.ReqSet` object.

`pathToFile` — File path

character vector

Path to the requirements document.

Example: `'C:\MATLAB\System_Requirements.docx'`

`reqFormat` — Registered document type label
character vector

Custom registered document type label that you create by using a Custom Document Type extension API.

Example: `'linktype_rmi_doors'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'columns', '[1 8]', 'RichText', true`

`ReqSet` — Requirements Set
`slreq.ReqSet` object

The name of the existing requirements set that you import references to requirements into, specified as the comma-separated pair of `'ReqSet'` and a valid requirements set file name.

Example: `'ReqSet', 'My_Requirements_Set'`

`RichText` — Requirements content imported as rich text
`false` (default) | `true`

Option to import requirements content as rich text, specified as the comma-separated pair consisting of `'RichText'` and `true` or `false`.

Example: `'RichText', true`

`bookmarks` — Use custom bookmarks in Microsoft Word and Microsoft Excel
`true` | `false`

Option to use custom bookmarks in Microsoft Word documents and Microsoft Excel spreadsheets to import requirements content, specified as the comma-separated pair consisting of `'bookmarks'` and `true` or `false`.

Example: `'bookmarks', false`

`match` — Regular expression
character vector

Import requirements by using regular expression pattern matching, specified as the comma-separated pair consisting of `'match'` and a regular expression pattern.

Example: `'match', '^REQ\d+'`

`columns` — Range of columns
double array

Range of columns to import. This option is applicable only for Microsoft Excel spreadsheets.

Example: `'columns', [1 6]`

`rows` — Range of rows
double array

Range of rows to import. This option is applicable only for Microsoft Excel spreadsheets.

Example: `'rows', [3 35]`

`attributes` — Attribute names
cell array

Attribute names to import, specified as a cell array.

Note When importing requirements from a Microsoft Excel spreadsheet, the length of this cell array must match the number of columns that you specified for import by using the `'columns'` option.

Example: `'attributes', {'Test Status', 'Test Procedure'}`

`idColumn` — ID Column
double

Column in the Microsoft Excel spreadsheet that you want to correspond to the **ID** field in the requirements set.

Example: `'idColumn', 1`

`summaryColumn` — Summary Column
double

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Summary** field in the requirements set.

Example: `'summaryColumn', 4`

`keywordsColumn` — Keywords Column
double

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Keywords** field in the requirements set.

Example: `'keywordsColumn', 3`

`descriptionColumn` — Description Column
double

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Description** field in the requirements set.

Example: `'descriptionColumn', 2`

`rationaleColumn` — Rationale Column
double

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Rationale** field in the requirements set.

Example: `'rationaleColumn', 5`

Examples

Create Read-Only References to Requirements in Microsoft Office Documents

```
% Create a new requirements set and save it

rs = slreq.new('newReqSet');
save(rs);
```

```
% Create read-only rich text references to requirements
% in a Word document
createReferences(rs, 'C:\Work\Requirements_Spec.docx', ...
'RichText', true);

% Create read-only plain text references to requirements
% in an Excel spreadsheet
createReferences(rs, 'C:\Work\Design_Spec.xlsx', ...
'columns', [2 6], 'rows', [3 32], 'idColumn', 2, ...
'summaryColumn', 3);
```

See Also

`slreq.Reference` | `slreq.ReqSet` | `slreq.import`

Introduced in R2018a

find

Class: slreq.ReqSet
Package: slreq

Find requirements in requirements set that have matching attribute values

Syntax

```
myReq = find(rs, 'PropertyName', 'PropertyValue')
```

Description

`myReq = find(rs, 'PropertyName', 'PropertyValue')` finds and returns an `slreq.Requirement` object `myReq` in the requirements set `rs` specified by the properties matching `PropertyName` and `PropertyValue`. Property name matching is case-insensitive.

Input Arguments

`rs` — Requirements set
`slreq.ReqSet` object

Requirements set, specified as a `slreq.ReqSet` object.

Output Arguments

`myReq` — Requirement object
`slreq.Requirement` object

Requirement, returned as an `slreq.Requirement` object.

Examples

Find Requirements That Have Matching Attribute Values

```
% Load a requirements set file
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

% Find all editable requirements in the requirement set
allReqs = find(rs, 'Type', 'Requirement');

% Find all referenced requirements in the requirement set
allRefs = find(rs, 'Type', 'Reference');

% Find all requirements with a certain ID
matchedReqs = find(rs, 'ID', 'R1.1');
```

Find Requirements by Using Regular Expression Matching

You can search for requirements in your requirements sets by constructing regular expression search patterns by using the tilde (~) symbol.

```
% Load a requirements set file
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

% Find all requirements that correspond to the controller
controllerReqs = find(rs, 'Type', 'Requirement', 'Summary', '~Controller(?i)\w*')

controllerReqs =

  1×19 Requirement array with properties:

    Id
    Summary
    Keywords
    Description
    Rationale
    SID
    CreatedBy
    CreatedOn
    ModifiedBy
    ModifiedOn
    FileRevision
```

```
    Dirty
    Comments
```

For more information on constructing regular expression search patterns, see “Steps for Building Expressions” (MATLAB).

See Also

`slreq.ReqSet` | `slreq.find`

Introduced in R2018a

getImplementationStatus

Class: slreq.ReqSet
Package: slreq

Query requirement set implementation status summary

Syntax

```
status = getImplementationStatus(rs)
```

Description

`status = getImplementationStatus(rs)` returns the implementation status for the requirement set `rs`.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

Requirement set, specified as an `slreq.ReqSet` object.

Output Arguments

`status` — Requirement set implementation status summary

structure

The implementation status summary for the requirements in the requirement set, returned as a MATLAB structure containing these fields.

`total` — Total number of requirements

`double`

The total number of Functional requirements in the requirement set, returned as a `double`.

`implemented` — Implemented requirements
`double`

The total number of implemented requirements in the requirement set, returned as a `double`.

`justified` — Justified requirements
`double`

The total number of requirements justified for implementation in the requirement set, returned as a `double`.

`none` — Unimplemented requirements
`double`

The total number of unimplemented requirements in the requirement set, returned as a `double`.

Examples

Get Implementation Status Summary of a Requirement Set

```
% Load a requirements set file
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

% Get the implementation status summary of the requirement set rs
implStatus = getImplementationStatus(rs)

implStatus =

  struct with fields:

         total: 25
    implemented: 18
      justified: 5
           none: 2
```

See Also

`updateImplementationStatus`

Introduced in R2018b

getVerificationStatus

Class: slreq.ReqSet
Package: slreq

Query requirement set verification status summary

Syntax

```
status = getVerificationStatus(rs)
```

Description

`status = getVerificationStatus(rs)` returns the verification status summary of all requirements in the requirement set `rs`.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

Requirement set, specified as an `slreq.ReqSet` object.

Output Arguments

`status` — Requirement set verification status summary

structure

The verification status summary for the requirement set, returned as a MATLAB structure containing these fields.

`total` — Total number of requirements

`double`

The total number of requirements in the requirement set with Verify links, returned as a `double`.

`passed` — Passed requirements
`double`

The total number of requirements in the requirement set that passed the tests associated with them, returned as a `double`.

`failed` — Failed requirements
`double`

The total number of requirements in the requirement set that failed the tests associated with them, returned as a `double`.

`unexecuted` — Unexecuted requirements
`double`

The total number of requirements in the requirement set with unexecuted associated tests, returned as a `double`.

`justified` — Justified requirements
`double`

The total number of requirements justified for verification in the requirement set, returned as a `double`.

`none` — Unlinked requirements
`double`

The total number of requirements without links to verification objects in the requirement set, returned as a `double`.

Examples

Get Verification Status Summary of a Requirement Set

```
% Load a requirements set file
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
```

```
% Get the verification status summary of the requirements in rs
verifStatus = getVerificationStatus(rs)

verifStatus =

  struct with fields:

          total: 25
         passed: 10
         failed: 5
     unexecuted: 4
      justified: 1
           none: 5
```

See Also

updateVerificationStatus

Introduced in R2018b

importFromDocument

Class: slreq.ReqSet
Package: slreq

Import editable requirements from external documents

Syntax

```
importFromDocument(rs, pathToFile, Name, Value)
```

Description

`importFromDocument(rs, pathToFile, Name, Value)` imports editable requirements with contents duplicated from an external document at `pathToFile` using by additional `Name, Value` arguments to specify import options.

Input Arguments

`rs` — Requirements set file

`slreq.ReqSet` object

Requirements set file, specified as a `slreq.ReqSet` object.

`pathToFile` — File path

character vector

Path to the requirements document that you want to import editable requirements from.

Example: `'C:\MATLAB\System_Requirements.docx'`

`ReqSet` — Requirements Set

character vector

The name for the existing requirements set that you import requirements into, specified as a character vector.

Example: `'ReqSet', 'My_Requirements_Set'`

`RichText` — Option to import rich text requirements
`false` (default) | `true`

Option to import requirements as rich text, specified as a Boolean value.

Example: `'RichText', true`

`bookmarks` — Option to import requirements using bookmarks
`false` | `true`

Option to import requirements content using user-defined bookmarks. This value is `true` by default for Microsoft Word documents and `false` by default for Microsoft Excel spreadsheets.

Example: `'bookmarks', false`

`match` — Regular expression pattern
character vector

Regular expression pattern for ID search in Microsoft Office documents.

Example: `'match', '^REQ\d+'`

`attributes` — Attribute names
cell array

Attribute names to import, specified as a cell array.

Note When importing requirements from a Microsoft Excel spreadsheet, the length of this cell array must match the number of columns specified for import using the `'columns'` argument.

Example: `'attributes', {'Test Status', 'Test Procedure'}`

Pairs for Microsoft Excel Spreadsheets

`columns` — Range of columns
`double` array

Range of columns to import, specified as a `double` array.

Example: `'columns', [1 6]`

`rows` — Range of rows
`double` array

Range of rows to import, specified as a `double` array.

Example: `'rows', [3 35]`

`idColumn` — ID Column
`double`

Column in the Microsoft Excel spreadsheet that you want to correspond to the **ID** field in your requirement set, specified as a `double`.

Example: `'idColumn', 1`

`summaryColumn` — Summary Column
`double`

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Summary** field in your requirement set, specified as a `double`.

Example: `'summaryColumn', 4`

`keywordsColumn` — Keywords Column
`double`

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Keywords** field in your requirement set, specified as a `double`.

Example: `'keywordsColumn', 3`

`descriptionColumn` — Description Column
`double`

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Description** field in your requirement set, specified as a `double`.

Example: `'descriptionColumn', 2`

`rationaleColumn` — Rationale Column
`double`

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Rationale** field in your requirement set, specified as a `double`.

Example: `'rationaleColumn', 5`

`attributeColumn` — Custom Attributes Column
`double`

Column in the Microsoft Excel spreadsheet that you want to correspond to the **Custom Attributes** field in your requirement set, specified as a `double`.

Example: `'attributeColumn', 6`

`USDM` — USDM Format Import Option
character vector

Import from Microsoft Excel spreadsheets specified in the USDM (Universal Specification Describing Manner) standard format. Specify values as a character vector with the ID prefix optionally followed by a separator character.

Example: `'RQ -'` will match entries with IDs similar to `RQ01`, `RQ01-2`, `RQ01-2-1` etc.

Examples

Import Editable Requirements from Microsoft Office Documents

```
% Create a new requirements set and save it
rs = slreq.new('newReqSet');
save(rs);

% Import editable requirements as rich text from a Word document
importFromDocument(rs, 'C:\Work\Requirements_Spec.docx', ...
 'RichText', true);

% Import editable requirements from an Excel spreadsheet
importFromDocument(rs, 'C:\Work\Design_Spec.xlsx', ...
'columns', [2 6], 'rows', [3 32], 'idColumn', 2, ...
'summaryColumn', 3);
```

See Also

`createReferences` | `slreq.ReqSet`

Introduced in R2018a

save

Class: slreq.ReqSet
Package: slreq

Save a requirements set

Syntax

```
save(rs)
save(rs, filePath)
```

Description

`save(rs)` saves a requirements set by using its file name.

`save(rs, filePath)` saves a requirements set and updates its Name and Filename properties.

Input Arguments

`rs` — Requirements set file

`slreq.ReqSet` object

Requirements set file, specified as a `slreq.ReqSet` object.

`filePath` — File name and path

character vector

The file name and path of the requirements set, specified as a character vector.

Example: `'C:\MATLAB\myReqSet.slreqx'`

Examples

Save Requirements Set File

```
% Create the requirements set file
rs = slreq.new('C:\MATLAB\My Requirements Set.slreqx');

% Save the requirements set file
save(rs);

% Save the requirements set file as another requirements set file
save(rs, 'C:\MATLAB\Another Requirements Set.slreqx');
```

See Also

`slreq.ReqSet`

Introduced in R2018a

updateImplementationStatus

Class: slreq.ReqSet
Package: slreq

Update requirement set implementation status summary

Syntax

```
updateImplementationStatus(rs)
```

Description

`updateImplementationStatus(rs)` updates the implementation status summary of the requirement set `rs`.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

Requirement set, specified as an `slreq.ReqSet` object.

See Also

`getImplementationStatus`

Introduced in R2018b

updateVerificationStatus

Class: slreq.ReqSet
Package: slreq

Update requirement set verification status summary

Syntax

```
updateVerificationStatus(rs)
```

Description

`updateVerificationStatus(rs)` updates the verification status summary of the requirement set `rs`.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

Requirement set, specified as an `slreq.ReqSet` object.

See Also

`getVerificationStatus`

Introduced in R2018b

add

Class: slreq.Requirement
Package: slreq

Add requirement to requirements set

Syntax

```
req = add(reqObj, 'PropertyName', PropertyValue)
```

Description

`req = add(reqObj, 'PropertyName', PropertyValue)` adds a requirement `req` to a requirements object `reqObj` with properties and custom attributes specified by `PropertyName` and `PropertyValue`.

Input Arguments

`reqObj` — Requirements object

`slreq.ReqSet` object | `slreq.Requirement` object

Requirements set or requirement objects, specified as an `slreq.ReqSet` or as an `slreq.Requirement`object.

Output Arguments

`req` — Requirement

`slreq.Requirement` object

The requirement that was added, returned as an `slreq.Requirement` object.

Examples

Add a Requirement to a Requirements Set

```
% Load a requirements set file

rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

% Add a top-level requirement to the requirements set
req1 = add(rs, 'Id', '5', 'Summary', 'Additional Requirement');

% Add a child requirement to the requirement req1
req2 = add(req1, 'Id', '5.1', 'Summary', 'Additional Child Requirement');
```

See Also

`slreq.ReqSet` | `slreq.Requirement`

Introduced in R2018a

children

Class: slreq.Requirement
Package: slreq

Find child requirements of a requirement

Syntax

```
childReqs = children(req)
```

Description

`childReqs = children(req)` returns the child requirements `childReqs` of the `slreq.Requirement` object `req`.

Input Arguments

`req` — Requirement instance

`slreq.Requirement` object

Requirement specified as an `slreq.Requirement` object.

Output Arguments

`childReqs` — Child requirements

`slreq.Requirement` object | `slreq.Requirement` object array

The child requirements belonging to the requirement `req`, returned as `slreq.Requirement` objects.

Examples

Find Child Requirements

```
% Load a requirements set file and add three new requirements

rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
req1 = add(rs, 'Id', '5', 'Summary' , 'Additional Requirement');
req2 = add(req1, 'Id', '5.1', 'Summary', 'Additional Child Requirement 1');
req3 = add(req1, 'Id', '5.2', 'Summary', 'Additional Child Requirement 2');

% Find the children of req1
childReqs = children(req1);

childReqs =

  1×2 Requirement array with properties:

    Id
    Summary
    Keywords
    Description
    Rationale
    SID
    CreatedBy
    CreatedOn
    ModifiedBy
    ModifiedOn
    FileRevision
    Comments
```

See Also

parent | slreq.ReqSet | slreq.Requirement

Introduced in R2018a

demote

Class: slreq.Requirement
Package: slreq

Demote requirements

Syntax

```
deomote(req)
```

Description

`deomote(req)` demotes the `slreq.Requirement` object `req` one level down in the hierarchy.

Input Arguments

`req` — Requirement instance
`slreq.Requirement` object

Requirement specified as an `slreq.Requirement` object.

Examples

Demote Requirements

```
% Load a requirements set file and add two new requirements
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
req1 = add(rs, 'Id', '5', 'Summary' , 'Additional Requirement');
req2 = add(req1, 'Id', '5.1', 'Summary' , 'Child Requirement');

% Demote req2
demote(req2);
```

```
% Find the parent of req2
parentReq = parent(req2);

parentReq =

  ReqSet with properties:

            Description: ''
                   Name: 'My_Requirements_Set_1'
               Filename: 'C:\MATLAB\My_Requirements_Set_1.slreqx'
               Revision: 6
                  Dirty: 1
    CustomAttributeNames: {}
```

See Also

promote | slreq.ReqSet | slreq.Requirement

Introduced in R2018a

find

Class: slreq.Requirement
Package: slreq

Find requirements that have matching attribute values

Syntax

```
reqs = find(rs, 'PropertyName', PropertyValue)
```

Description

`reqs = find(rs, 'PropertyName', PropertyValue)` returns a requirement or set of requirements `reqs` in the requirements set `rs` specified by the properties that match `PropertyName` and `PropertyValue`. Property name matching is case-insensitive.

Input Arguments

`rs` — Requirements set

`slreq.ReqSet` object

Requirements set specified as an `slreq.ReqSet` object.

Output Arguments

`reqs` — Requirements

`slreq.Requirement` object | `slreq.Requirement` object array

Requirements, returned as `slreq.Requirement` objects.

Examples

Find Requirements That Have Matching Attribute Values

```
% Load a requirements set file
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

% Find all editable requirements with ID R1.1 in the requirements set
matchedReqs = find(rs, 'Type', 'Requirement', 'ID', 'R1.1');
```

Find Requirements by Using Regular Expression Matching

You can search for requirements in your requirements sets by constructing regular expression search patterns by using the tilde (~) symbol.

```
% Load a requirements set file
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');

% Find all requirements that correspond to the plant
plantReqs = find(rs, 'Type', 'Requirement', 'Summary', '~Plant(?i)\w*')

plantReqs =

  1×11 Requirement array with properties:

    Id
    Summary
    Keywords
    Description
    Rationale
    SID
    CreatedBy
    CreatedOn
    ModifiedBy
    ModifiedOn
    FileRevision
    Dirty
    Comments
```

For more information on constructing regular expression search patterns, see “Steps for Building Expressions” (MATLAB).

See Also

`slreq.Requirement` | `slreq.find`

Introduced in R2018a

getAttribute

Class: slreq.Requirement
Package: slreq

Get requirement custom attributes

Syntax

```
val = getAttribute(req, propertyName)
```

Description

`val = getAttribute(req, propertyName)` gets a requirement property that is specified by `propertyName`.

Input Arguments

`req` — Requirement instance
`slreq.Requirement` object

Requirement specified as an `slreq.Requirement` object.

`propertyName` — Requirement property
character vector

Requirement property name.

Example: `'SID', 'CreatedOn', 'Summary'`

Examples

Get Requirement Attributes

```
% Load a requirements set file and get the handle to one requirement
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
req1 = find(rs, 'Type', 'Requirement', 'ID', 'R1.1');

% Get the Priority (custom attribute) of req1
summaryReq1 = getAttribute(req1, 'Priority')

summaryReq1 =

  'High'
```

See Also

`setAttribute` | `slreq.Requirement`

Introduced in R2018a

getImplementationStatus

Class: slreq.Requirement
Package: slreq

Query requirement implementation status summary

Syntax

```
status = getImplementationStatus(req)
status = getImplementationStatus(req, 'self')
```

Description

`status = getImplementationStatus(req)` returns the implementation status summary for the requirement `req` and all its child requirements.

`status = getImplementationStatus(req, 'self')` returns the implementation status summary for just the requirement `req`.

Input Arguments

`req` — Requirement instance
`slreq.Requirement` object

Requirement instance, specified as an `slreq.Requirement` object.

Output Arguments

`status` — Requirement implementation status summary
structure

The implementation status summary for the requirement and its child requirements, returned as a MATLAB structure containing these fields.

`total` — Total number of requirements
`double`

The total number of Functional requirements (including child requirements), returned as a `double`.

`implemented` — Implemented requirements
`double`

The total number of implemented requirements (including child requirements), returned as a `double`.

`justified` — Justified requirements
`double`

The total number of requirements (including child requirements), justified for implementation, returned as a `double`.

`none` — Unimplemented requirements
`double`

The total number of unimplemented requirements (including child requirements), returned as a `double`.

Examples

Get Implementation Status Summary of a Requirement

```
% Get the implementation status summary of the requirement req
% and all its child requirements
reqImplStatus = getImplementationStatus(req)

reqImplStatus =

  struct with fields:

         total: 20
    implemented: 16
      justified: 3
           none: 1
```

```
% Get the implementation status summary of only the requirement myReq
myReqImplStatus = getImplementationStatus(myReq, 'self')

myReqImplStatus =

  struct with fields:

    implemented: 16
      justified: 3
           none: 1
```

See Also

updateImplementationStatus

Introduced in R2018b

getVerificationStatus

Class: slreq.Requirement
Package: slreq

Query requirement verification status summary

Syntax

```
status = getVerificationStatus(req)
status = getVerificationStatus(req, 'self')
```

Description

`status = getVerificationStatus(req)` returns the verification status summary for the requirement `req` and all its child requirements.

`status = getVerificationStatus(req, 'self')` returns the verification status summary for just the requirement `req`.

Input Arguments

`req` — Requirement instance
`slreq.Requirement` object

Requirement instance, specified as an `slreq.Requirement` object.

Output Arguments

`status` — Requirement verification status summary
structure

The verification status for the requirement and its child requirements, returned as a MATLAB structure containing these fields.

`total` — Total number of requirements
`double`

The total number of requirements (including child requirements) with Verify links, returned as a `double`.

`passed` — Passed requirements
`double`

The total number of requirements (including child requirements) that passed the tests associated with them, returned as a `double`.

`failed` — Failed requirements
`double`

The total number of requirements (including child requirements) that failed the tests associated with them, returned as a `double`.

`unexecuted` — Unexecuted requirements
`double`

The total number of requirements (including child requirements) with unexecuted associated tests, returned as a `double`.

`justified` — Justified requirements
`double`

The total number of requirements (including child requirements) that are justified for verification in the requirement set, returned as a `double`.

`none` — Unlinked requirements
`double`

The total number of requirements (including child requirements) without links to verification objects, returned as a `double`.

Examples

Get Verification Status Summary of a Requirement

```
% Get the verification status summary of the requirement req
% and all its child requirements
```

```
reqVerifStatus = getVerificationStatus(req)

reqVerifStatus =

  struct with fields:

          total: 34
         passed: 14
         failed: 15
     unexecuted: 4
      justified: 1
           none: 0


% Get the verification status summary of only the requirement myReq
myReqVerifStatus = getVerificationStatus(myReq, 'self')

myReqVerifStatus =

  struct with fields:

        passed: 0
        failed: 1
    unexecuted: 0
     justified: 0
          none: 0
```

See Also

updateVerificationStatus

Introduced in R2018b

isJustifiedFor

Class: slreq.Requirement
Package: slreq

Check if requirement is justified

Syntax

```
tf = isJustifiedFor(req, linkType)
```

Description

`tf = isJustifiedFor(req, linkType)` checks if the requirement `req` is justified for the link type specified by `linkType`.

Input Arguments

`req` — Requirement instance

`slreq.Requirement` object

Requirement to check for justification, specified as an `slreq.Requirement` object.

`linkType` — Justification link type

`'Implement'` | `'Verify'`

Justification link type, specified as a character vector.

Output Arguments

`tf` — Justification status

`0` | `1`

The justification status of the requirement, returned as a Boolean.

Examples

Check if Requirements Are Justified

```
% Check if requirement req1 is justified for Implementation
req1_Status = isJustifiedFor(req1, 'Implement')

req1_Status =

  logical

   1

% Check if requirement req2 is justified for Verification
req2_Status = isJustifiedFor(req2, 'Verify')

req2_Status =

  logical

   0
```

See Also

`getImplementationStatus` | `getVerificationStatus`

Introduced in R2018b

justifyImplementation

Class: slreq.Requirement
Package: slreq

Justify requirements for implementation

Syntax

```
implementationJustLink = justifyImplementation(req, jt)
```

Description

`implementationJustLink = justifyImplementation(req, jt)` justifies the requirement `req` for implementation by creating a link `implementationJustLink` from the justification `jt` to `req`.

Input Arguments

`req` — Requirement instance
`slreq.Requirement` object

Requirement to justify for implementation, specified as an `slreq.Requirement` object.

`jt` — Justification object
`slreq.Justification` object

Justification object to justify `req` for implementation, specified as an `slreq.Justification` object.

Output Arguments

`implementationJustLink` — Justification link
`slreq.Link` object

Link to justification object `jt` of type **Implement**, returned as an `slreq.Link` object.

Examples

```
% Justify requirement myReq for implementation by using a justification object myJust

myImplJustification = justifyImplementation(myReq, myJust)

myImplJustification =

  Link with properties:

          Type: 'Implement'
    Description: 'Cruise Control Mode (crs_req_func_spec#1)'
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 13-Jan-2017 13:45:12
      CreatedBy: 'John Doe'
     ModifiedOn: 24-Oct-2018 12:25:30
     ModifiedBy: 'Jane Doe'
       Revision: 6
       Comments: [0×0 struct]
```

See Also

`addJustification` | `getImplementationStatus`

Introduced in R2018b

justifyVerification

Class: slreq.Requirement
Package: slreq

Justify requirements for verification

Syntax

```
verificationJustLink = justifyVerification(req, jt)
```

Description

`verificationJustLink = justifyVerification(req, jt)` justifies the requirement `req` for verification by creating a link `verificationJustLink` from the justification `jt` to `req`.

Input Arguments

`req` — Requirement object
`slreq.Requirement` object

Requirement to justify for verification, specified as an `slreq.Requirement` object.

`jt` — Justification object
`slreq.Justification` object

Justification object to justify `req` for verification, specified as an `slreq.Justification` object.

Output Arguments

`verificationJustLink` — Justification link
`slreq.Link` object

Link to justification object `jt` of type **Verify**, returned as an `slreq.Link` object.

Examples

```
% Justify requirement myReq for verification by using a justification object myJust

myVerifJustification = justifyVerification(myReq, myJust)

myVerifJustification =

  Link with properties:

          Type: 'Verify'
    Description: 'Cruise mode detection (crs_req_func_spec#67)'
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 30-Oct-2017 09:10:34
      CreatedBy: 'John Doe'
     ModifiedOn: 02-Feb-2018 17:08:09
     ModifiedBy: 'Jane Doe'
       Revision: 5
       Comments: [0×0 struct]
```

See Also

`addJustification` | `getVerificationStatus`

Introduced in R2018b

parent

Class: slreq.Requirement
Package: slreq

Find parent item of requirement

Syntax

```
parentObj = parent(req)
```

Description

`parentObj = parent(req)` returns the parent object `parentObj` of the `slreq.Requirement` object `req`.

Input Arguments

`req` — Requirement instance

`slreq.Requirement` object

Requirement specified as an `slreq.Requirement` object.

Output Arguments

`parentObj` — Parent object

`slreq.Requirement` object | `slreq.ReqSet` object

The parent of the requirement `req`, returned as an `slreq.Requirement` object or as an `slreq.ReqSet` object.

Examples

Find Parent Objects of Requirements

```
% Load a requirements set file and add two new requirements

rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
req1 = add(rs, 'Id', '5', 'Summary' , 'Additional Requirement');
req2 = add(req1, 'Id', '5.1', 'Summary' , 'Additional Child Requirement');

% Find the parent of req2
parentReq1 = parent(req2)

parentReq1 =

  Requirement with properties:

            Id: '5'
       Summary: 'Additional Requirement'
      Keywords: [0×0 char]
    Description: ''
      Rationale: ''
            SID: 10
      CreatedBy: 'John Doe'
      CreatedOn: 05-Oct-2007 16:09:38
     ModifiedBy: 'Jane Doe'
     ModifiedOn: 21-Dec-2016 11:10:05
       Comments: [0×0 struct]

% Find the parent of req1
parentReq2 = parent(req1)

parentReq2 =

  ReqSet with properties:

             Description: ''
                    Name: 'My_Requirements_Set_1'
                Filename: 'C:\MATLAB\My_Requirements_Set_1.slreqx'
                Revision: 6
                   Dirty: 1
    CustomAttributeNames: {}
```

See Also

`children` | `slreq.ReqSet` | `slreq.Requirement`

Introduced in R2018a

promote

Class: slreq.Requirement
Package: slreq

Promote requirements

Syntax

```
promote(req)
```

Description

`promote(req)` promotes the `slreq.Requirement` object `req` one level up in the hierarchy.

Input Arguments

`req` — Requirement instance
`slreq.Requirement` object

Requirement specified as an `slreq.Requirement` object.

Examples

Find Requirements with Matching Attribute Values

```
% Load a requirements set file and add two new requirements
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
req1 = add(rs, 'Id', '5', 'Summary' , 'Additional Requirement');
req2 = add(req1, 'Id', '5.1', 'Summary' , 'Child Requirement');

% Promote req2
promote(req2);
```

```
% Find the parent of req2
parentReq = parent(req2);

parentReq =

  ReqSet with properties:

            Description: ''
                   Name: 'My_Requirements_Set_1'
               Filename: 'C:\MATLAB\My_Requirements_Set_1.slreqx'
               Revision: 6
                  Dirty: 1
    CustomAttributeNames: {}
```

See Also

demote | slreq.ReqSet | slreq.Requirement

Introduced in R2018a

reqSet

Class: slreq.Requirement
Package: slreq

Return parent requirements set

Syntax

```
rsout = reqSet(req)
```

Description

`rsout = reqSet(req)` returns the parent requirements set `rsout` to which the requirement `req` belongs.

Input Arguments

`req` — Requirement object

`slreq.Requirement` object

Requirement, specified as an `slreq.Requirement` object.

Output Arguments

`rsout` — Parent requirements set

`slreq.ReqSet` object

The parent requirements set of the requirement `req`, returned as an `slreq.ReqSet` object.

Examples

Query Requirements Set Information

```
% Load a new requirements set file and select one requirement
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
allReqs = find(rs, 'Type', 'Requirement');
req = allReqs(1);

% Query which requirements set req belongs to
reqSet(req)

ans =

  ReqSet with properties:

            Description: ''
                   Name: 'My_Requirements_Set_1'
               Filename: 'C:\MATLAB\My_Requirements_Set_1.slreqx'
               Revision: 63
                  Dirty: 0
    CustomAttributeNames: {}
              CreatedBy: 'Jane Doe'
              CreatedOn: 27-Feb-2017 10:20:39
             ModifiedBy: 'John Doe'
             ModifiedOn: 08-Mar-2017 09:27:31
```

See Also

parent | slreq.ReqSet | slreq.Requirement

Introduced in R2018a

setAttribute

Class: slreq.Requirement
Package: slreq

Set requirement custom attributes

Syntax

```
setAttribute(req, propertyName, propertyValue)
```

Description

`setAttribute(req, propertyName, propertyValue)` sets a requirement property.

Input Arguments

`req` — Requirement instance
`slreq.Requirement` object

Requirement specified as an `slreq.Requirement` object.

`propertyName` — Requirement property
character vector

Requirement property name.

Example: `'SID', 'CreatedOn', 'Summary'`

`propertyValue` — Requirement property value
character vector

Requirement property value.

Example: `'Test Requirement', 'R1.3.1'`

Examples

Set Requirement Custom Attributes

```
% Load a requirement set file and get the handle to one requirement
rs = slreq.load('C:\MATLAB\My_Requirements_Set_1.slreqx');
req1 = find(rs, 'Type', 'Requirement', 'ID', 'R2.1');

% Set the Priority (custom attribute) of req1
setAttribute(req1, 'Priority', 'Low');

req1

req1 =

  Requirement with properties:

             Id: 'R2.1'
        Summary: 'Controller Requirement'
       Keywords: [0×0 char]
    Description: ''
      Rationale: ''
            SID: 21
      CreatedBy: 'Jane Doe'
      CreatedOn: 27-Feb-2014 10:15:38
     ModifiedBy: 'John Doe'
     ModifiedOn: 02-Aug-2017 13:49:40
   FileRevision: 43
          Dirty: 1
       Comments: [0×0 struct]
       Priority: Low
```

See Also

`getAttribute` | `slreq.ReqSet` | `slreq.Requirement`

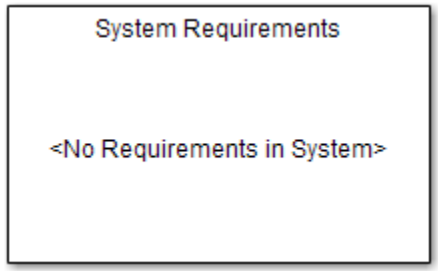
Introduced in R2018a

4

Block Reference

System Requirements

List system requirements in Simulink models



Library

Simulink Requirements

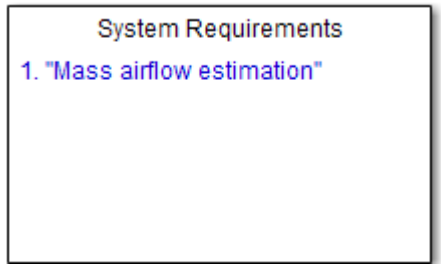
Description

The System Requirements block lists the system-level requirements associated with a model or subsystem. This block is dynamically populated. It displays system requirements associated with the level of hierarchy in which the block appears in the model. It does not list requirements associated with individual blocks in the model. To ensure that all requirement links are listed in the System Requirements block:

1. Right-click the background of your model.
2. Select **Requirements at This Level**.
3. From the top of the context menu, verify that all the requirements you want to list appear in the System Requirements block.

You can place this block anywhere in your model. It does not connect to other Simulink blocks. You can have only one System Requirements block in a given subsystem.

When you insert this block into your Simulink model, it is populated with the system requirements, as shown in the `Airflow Calculation` subsystem of the `slvnvdemo_fuelsys_officereq` example.



Each of the listed requirements is an active link to the requirements document. When you double-click a requirement label, the associated requirements document opens in its editor window, scrolled to the target location.

Parameters

Block Title

The title of the system requirements list in the model. The default title is `System Requirements`. You can enter a customized title, for example, `Engine Requirements`.

Introduced before R2006a