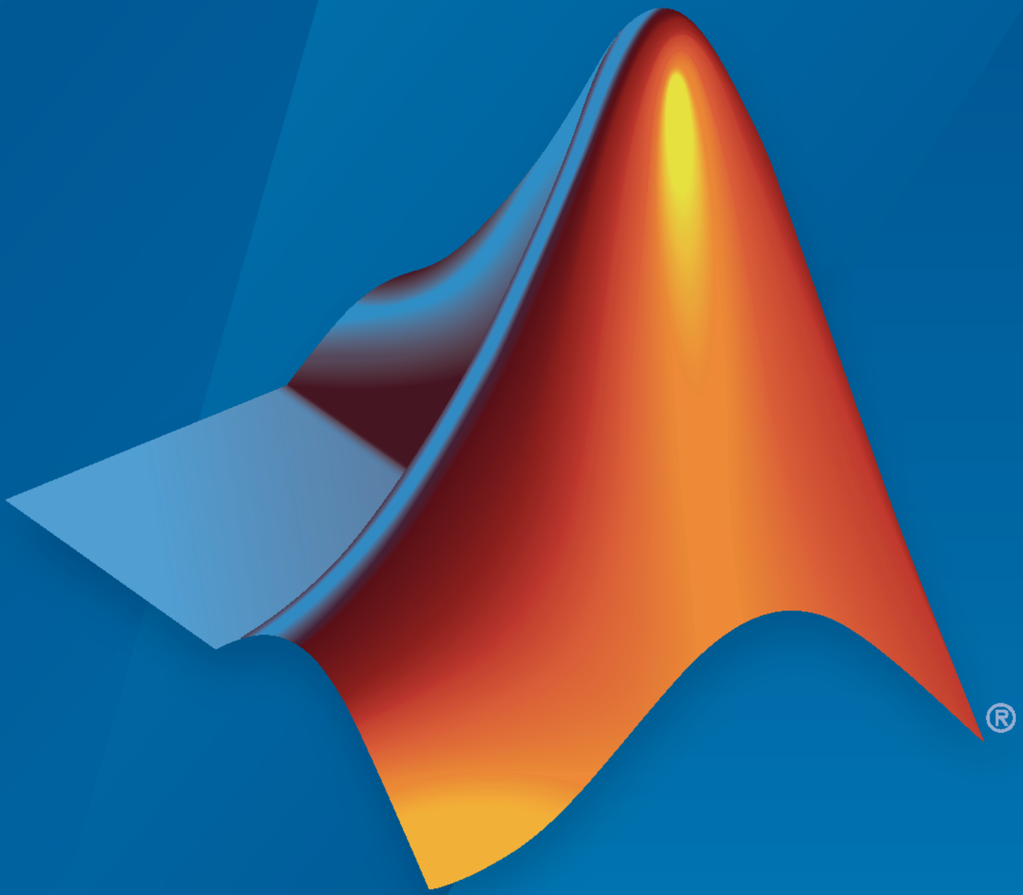
# Simulink® Requirements™

Reference



MATLAB® & SIMULINK®

R2020b

MathWorks®

*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) |
| September 2019 | Online Only | Revised for Version 1.4 (Release 2019b) |
| March 2020 | Online only | Revised for Version 1.5 (Release 2020a) |
| September 2020 | Online only | Revised for Version 1.6 (Release 2020b) |

# Contents

1

# Functions

## 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.cmConfigureVersion

Set version of linked requirements documents

## Syntax

```
prev_version = slreq.cmConfigureVersion(domain,doc_id,version)
prev_version = slreq.cmConfigureVersion(domain,doc_id,version,src)
```

## Description

`prev_version = slreq.cmConfigureVersion(domain,doc_id,version)` sets the configured version `version` of the linked requirements document `doc_id` of type `domain` and returns the previously configured version `prev_version`.

`prev_version = slreq.cmConfigureVersion(domain,doc_id,version,src)` sets the configured version `version` of the linked requirements document `doc_id` of type `domain` for all links from the Model-Based Design artifact `src` and returns the previously configured version `prev_version`.

## Examples

### Set Configured Version for All Links to IBM Rational DOORS Module Baseline

Use baseline version `2.2b` for all links to the IBM Rational DOORS module `546223g1`.

```
% Set configured version to 2.1b
versionA = slreq.cmConfigureVersion('linktype_rmi_doors','546223g1','2.1b')

versionA =

  0×0 empty char array

% versionA is empty because there is no previously configured version

versionB = slreq.cmConfigureVersion('linktype_rmi_doors','546223g1','2.2b')

versionB =

  '2.1b'

% 2.1b is the previously set configured version
```

### Set Configured Version for Links from Simulink Model to IBM Rational DOORS Module Baseline

Use baseline version `2.3b` for links from the Simulink® model `myModel.slx` to the IBM Rational DOORS module `00006a12`.

```
% Set configured version to 2.1b
versionA = slreq.cmConfigureVersion('linktype_rmi_doors', '00006a12', '2.1b', 'myModel.slx')
versionA =

  0×0 empty char array

% versionA is empty because there is no previously configured version

% Set the configured version to 2.3b

versionB = slreq.cmConfigureVersion('linktype_rmi_doors', '00006a12', '2.3b', 'myModel.slx')

versionB =

  '2.1b'

% 2.1b is the previously set configured version
```

## Input Arguments

**`domain` — Document type name**
`'linktype_rmi_doors'` | character vector | string

Registered document type name, specified as a character vector or a string. As of R2019b, link target version configuration is supported only for IBM® Rational® DOORS® with the value `'linktype_rmi_doors'`.

**`doc_id` — Requirements document identifier**
character vector | string

Unique identifier for a version-controlled requirements document, specified as a character vector or a string.

**`version` — Requirements document target version**
character vector | string

Target version of the requirements document, specified as a character vector or a string.

**`src` — Source artifact file name**
character vector | string

The file name of the Model-Based Design source artifact, specified as a character vector or a string.

## Output Arguments

**`prev_version` — Document version**
character vector

Previously configured version of the linked requirements document, returned as a character vector.

## See Also

`slreq.cmGetVersion`

**Introduced in R2019b**

# slreq.cmGetVersion

Get configured version of linked requirements documents

## Syntax

```
doc_version = slreq.cmGetVersion(domain,doc_id)
doc_version = slreq.cmGetVersion(domain,doc_id,src)
```

## Description

`doc_version = slreq.cmGetVersion(domain,doc_id)` queries the configured version `doc_version` of the linked requirements document `doc_id` of type `domain`.

`doc_version = slreq.cmGetVersion(domain,doc_id,src)` queries the configured version `doc_version` of the linked requirements document `doc_id` of type `domain` that is linked to the Model-Based Design artifact `src`.

## Examples

### Query Configured Version for IBM Rational DOORS Module

Get the configured baseline version for the IBM Rational DOORS module `1213424d`.

```
configVer = slreq.cmGetVersion('linktype_rmi_doors','1213424d')
```

```
configVer =

  '1.3a'
```

### Query Configured Version for Links from a Simulink Model to IBM Rational DOORS Module

Get the configured baseline version for links from the Simulink model `myModel.slx` for the IBM Rational DOORS module `1234a45a`.

```
configVer = slreq.cmGetVersion('linktype_rmi_doors', '1234a45a', 'myModel.slx')
```

```
configVer =

  '2.4c'
```

## Input Arguments

**domain — Document type name**
`'linktype_rmi_doors'` | character vector | string

Registered document type name, specified as a character vector or a string. As of R2019b, link target version configuration is supported only for IBM Rational DOORS with the value `'linktype_rmi_doors'`.

**doc_id — Requirements document identifier**
character vector | string

Unique identifier for a version-controlled requirements document, specified as a character vector or a string.

**`src` — Source artifact file name**
character vector | string

The file name of the Model-Based Design source artifact, specified as a character vector or a string.

## Output Arguments

**`doc_version` — Document version**
character vector

Configured version of the linked requirements document, returned as a character vector.

## See Also

`slreq.cmConfigureVersion`

**Introduced in R2019b**

# 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 allIBM 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 allrequirements `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` | `find` | `slreq.Justification` | `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(R) 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. |
| `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'
      openReport: 1
       titleText: ''
         authors: 'Jane Doe'
        includes: [1×1 struct]
```

## Output Arguments

**`myOptions` — Report generation options**
structure

Options for report generation, returned as a structure with the following fields:

**Options**

| Fields | Data Type | Description |
|---|---|---|
| `reportPath` | character vector | Report file path |
| `openReport` | Boolean | Option to open report automatically after generation |
| `titleText` | character vector | Report title |
| `authors` | character vector | Report authors |
| `includes.toc` | Boolean | Option to include table of contents in your report |
| `includes.publishedDate` | Boolean | Option to include the report publish date |
| `includes.revision` | Boolean | Option to include requirement revision information in your report |
| `includes.properties` | Boolean | Option to include requirement properties |
| `includes.links` | Boolean | Option to include requirements links in your report |
| `includes.changeInformation` | Boolean | Option to include change information such as change issues |
| `includes.groupLinksBy` | character vector | Option to group links by `Artifact` or `LinkType` |
| `includes.keywords` | Boolean | Option to include requirement implementation status data 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.emptySections` | Boolean | Option to include empty sections 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 |

## 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');
```

For more information on importing referenced requirements from third-party applications, see "Import Requirements from Third-Party Applications".

## 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'}`

**`columns` — Range of columns**
`double` array

Range of columns to import from Microsoft Excel spreadsheet, specified as a `double` array.

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

**`rows` — Range of rows**
`double` array

Range of rows to import from Microsoft Excel spreadsheet, 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.

**`attr2reqprop` — ReqIF™ attribute mapping**
`containers.Map` object

Import from ReqIF format, specifying the attribute mapping as a comma-separated pair consisting of `'attr2reqprop'` and a `containers.Map` object. For example:

```
attrMap = containers.Map('KeyType','char','ValueType','char')
attrMap('SourceID') = 'Custom ID'; % Built-in attribute
attrMap('ReqIF.ChapterName') = 'Summary'; % Built-in attribute
attrMap('Data Class') = 'MyDataClass'; % Custom attribute

slreq.import('myfile.reqif','attr2reqprop',attrMap);
```

Example: `slreq.import('myfile.reqif', 'attr2reqprop', attrMap);`

## 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.inLinks

Get incoming links for requirement or other linkable item

## Syntax

```
ks = slreq.inLinks(node)
```

## Description

`ks = slreq.inLinks(node)` returns incoming links `ks`, a `Link` or `Link` array, to `nodes`, a `Requirement`, `Reference`, or other linkable item.

## Examples

### Determine Incoming and Outgoing Links

This example shows how to determine the incoming link for a requirement and outgoing link for a model object. Click the **Open Live Script** button to get copies of the example files.

#### Load Model and Requirement Set

```
load_system('reqs_validation_property_proving_original_model');
rqset = slreq.load('original_thrust_reverser_requirements.slreqx');
```

#### Get a Requirement from the Set

```
req = slreq.find('Type','Requirement','Summary','Maximum Throttle Threshold');
```

#### Determine Incoming Links for the Requirement

```
lkIn = slreq.inLinks(req)
```

```
lkIn = 
  Link with properties:

           Type: 'Implement'
    Description: 'R11: Maximum Throttle Threshold (original_thrust_reverser_requirements#11)'
       Keywords: {}
      Rationale: ''
      CreatedOn: 25-Mar-2019 16:10:06
      CreatedBy: 'asriram'
     ModifiedOn: 25-Mar-2019 16:10:06
     ModifiedBy: 'asriram'
       Revision: 14
            SID: 52
       Comments: [0x0 struct]
```

#### Determine the Incoming Link Source

```
lkSrc = source(lkIn);
```

### Convert Link Source to Model Object

```
mo = slreq.structToObj(lkSrc);
```

### Determine Outgoing Link from the Model Object

```
lkOut = slreq.outLinks(mo)
```

```
lkOut = 
  Link with properties:

           Type: 'Implement'
    Description: 'R11: Maximum Throttle Threshold (original_thrust_reverser_requirements#11)'
       Keywords: {}
      Rationale: ''
      CreatedOn: 25-Mar-2019 16:10:06
      CreatedBy: 'asriram'
     ModifiedOn: 25-Mar-2019 16:10:06
     ModifiedBy: 'asriram'
       Revision: 14
            SID: 52
       Comments: [0x0 struct]
```

### Close Files

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

## Input Arguments

### node — Linkable item to get incoming links for
struct

A linkable item that may have incoming requirements links. Common examples include a `Requirement` or `Reference`. Can be the output of `find`.

Example: `Requirement with properties`

Data Types: `struct`

## Output Arguments

### ks — Link(s) incoming to node
`Link` or `Link` array

A `Link` or `Link` array incoming to the linkable item.

## See Also
`slreq.outLinks` | `slreq.structToObj`

**Introduced in R2017b**

## 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.outLinks

Get outgoing links for a block or other linkable item

## Syntax

```
ks = slreq.outLinks(node)
```

## Description

`ks = slreq.outLinks(node)`, returns outgoing links `ks`, a `Link` or `Link` array, from `node`, a block or other linkable item.

## Examples

### Determine Incoming and Outgoing Links

This example shows how to determine the incoming link for a requirement and outgoing link for a model object. Click the **Open Live Script** button to get copies of the example files.

#### Load Model and Requirement Set

```
load_system('reqs_validation_property_proving_original_model');
rqset = slreq.load('original_thrust_reverser_requirements.slreqx');
```

#### Get a Requirement from the Set

```
req = slreq.find('Type','Requirement','Summary','Maximum Throttle Threshold');
```

#### Determine Incoming Links for the Requirement

```
lkIn = slreq.inLinks(req)
```

```
lkIn = 
  Link with properties:

           Type: 'Implement'
    Description: 'R11: Maximum Throttle Threshold (original_thrust_reverser_requirements#11)'
       Keywords: {}
      Rationale: ''
      CreatedOn: 25-Mar-2019 16:10:06
      CreatedBy: 'asriram'
     ModifiedOn: 25-Mar-2019 16:10:06
     ModifiedBy: 'asriram'
       Revision: 14
            SID: 52
       Comments: [0x0 struct]
```

#### Determine the Incoming Link Source

```
lkSrc = source(lkIn);
```

#### Convert Link Source to Model Object

```
mo = slreq.structToObj(lkSrc);
```

#### Determine Outgoing Link from the Model Object

```
lkOut = slreq.outLinks(mo)
```

```
lkOut = 
  Link with properties:

           Type: 'Implement'
    Description: 'R11: Maximum Throttle Threshold (original_thrust_reverser_requirements#11)'
       Keywords: {}
      Rationale: ''
      CreatedOn: 25-Mar-2019 16:10:06
      CreatedBy: 'asriram'
     ModifiedOn: 25-Mar-2019 16:10:06
     ModifiedBy: 'asriram'
       Revision: 14
            SID: 52
       Comments: [0x0 struct]
```

#### Close Files

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

## Input Arguments

#### node — Linkable item to get outgoing links for
struct

A linkable item that may have outgoing requirements links. Common examples include a block, function, or `TestCase`.

Example: `Simulink.Gain`

Example: `TestCase with properties`

Data Types: `struct`

## Output Arguments

#### ks — Link(s) incoming to node
`Link` or `Link` array

A `Link` or `Link` array incoming to the linkable item.

## See Also
`slreq.inLinks` | `slreq.structToObj`

**Introduced in R2017b**

# 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**

# slreq.show

Navigate to link source or destination

## Syntax

```
slreq.show(tgt)
```

## Description

`slreq.show(tgt)` navigates to `tgt`, a link source or destination. The source or destination object opens in the corresponding interface, such as a block in a model, or test in the Test Manager.

## Examples

### Show Link Source

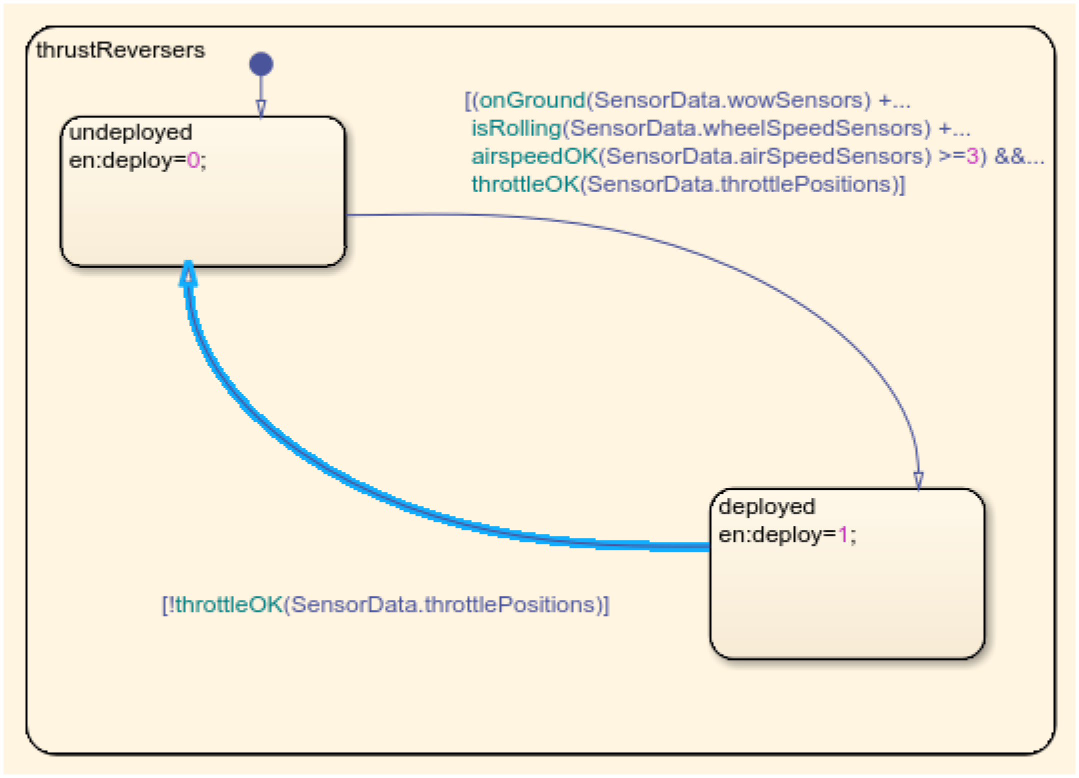
This example shows how to navigate to a link source.

#### Load Requirement Set and Links

```
rq = slreq.load('original_thrust_reverser_requirements.slreqx');
lk = slreq.load('reqs_validation_property_proving_original_model.slmx');
```

#### Navigate to a Link Source

```
sl = getLinks(lk);
sl2 = sl(2);
slreq.show(source(sl2))
```


thrustReversers
undeployed
en:deploy=0;
[(onGround(SensorData.wowSensors) +...
isRolling(SensorData.wheelSpeedSensors) +...
airspeedOK(SensorData.airSpeedSensors) >=3) &&...
throttleOK(SensorData.throttlePositions)]
deployed
en:deploy=1;
[!throttleOK(SensorData.throttlePositions)]

#### Cleanup

Cleanup commands. Clears open requirement sets without saving changes, and closes open models without saving changes.

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

## Input Arguments

### tgt — Link source or destination
`struct`

Link source or destination, as may be returned by `source` or `destination` for a `Link`.

Example: `struct with fields`

Data Types: `struct`

## See Also
`slreq.Link` | `slreq.inLinks` | `slreq.outLinks`

**Introduced in R2020a**

# slreq.structToObj

Convert link source or destination information from structure to model object type

## Syntax

```
ot = slreq.structToObj(linkinfo)
```

## Description

`ot = slreq.structToObj(linkinfo)` converts the source or destination link information in the structure `linkinfo` to the corresponding object type, `ot`. The object type returned can include Simulink blocks, Simulink Test™ test cases, or other object types compatible with Simulink Requirements.

## Examples

### Convert Link Source and Destination to Model Entity

This example shows how to get the structure containing unique requirement source and destination information, then convert the structure information to the specific source and destination model entity.

#### Load Model, Requirement Set, and Links

```
load_system('reqs_validation_property_proving_original_model');
reqset = slreq.load('original_thrust_reverser_requirements.slreqx');
linkset = slreq.load('reqs_validation_property_proving_original_model.slmx');
```

#### For a Link Set

Get sources from a link set, get a single source, and convert the structure to the model entity.

```
linkSources = sources(linkset);
linkSource1 = linkSources(1);
modelSource1 = slreq.structToObj(linkSource1);
```

#### For a Link

Get a link from the link set, get the source and destination for that link.

```
links = getLinks(linkset);
link2 = links(2);
linkSource2 = source(link2);
linkDest2 = destination(link2);
```

Convert the source and destination structure to the model entity.

```
modelSource2 = slreq.structToObj(linkSource2);
modelDest2 = slreq.structToObj(linkDest2);
```

### Clear Example Files

Cleanup commands -- close the open model, and clear and close the open requirement and link set.

```
slreq.clear;
close_system('reqs_validation_property_proving_original_model',0)
```

## Input Arguments

**`linkinfo` — Link information from a `slreq.Link` or `slreq.LinkSet`**
struct

`linkinfo` contains source artifact and unique identification information for particular links, as returned by

- `sources` for a `slreq.LinkSet`.
- `source` or `destination` for a `slreq.Link`.

Example: `struct with fields`

Data Types: `struct`

## Output Arguments

**`ot` — Source or destination object**
Requirement, model, or data entity

`ot` is the requirement, model, or data entity corresponding to the source artifact and unique identification in `linkinfo`. The value of `ot` depends on the type of entity the `Link` has as source or destination.

## See Also

`slreq.Link` | `slreq.LinkSet`

**Topics**
"Use Command-line API to Update or Repair Requirements Links"

**Introduced in R2018a**

# 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 = ['slvnvdemo_fuelsys_officereq/fuel rate controller/'...
'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.

```
addpath(fullfile(matlabroot,'toolbox','slrequirements',...
'slrequirementsdemos','fuelsys_req_docs'))
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 = '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.

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.

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

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

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.

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.

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

This example shows how to associate an external requirements traceability data file with a Simulink model

Open the model. Define the path to the requirement file.

```
open_system('slvnvdemo_powerwindowController');
reqFile = fullfile('slvnvdemo_powerwindowRequirements.slmx');
```

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.

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

```
Mapping ...\slrequirements-ex91255337\slvnvdemo_powerwindowController.slx to slvnvdemo_powerwind
```

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

#### Cleanup

Clean up commands. Clear the open requirement sets and link sets without saving changes and close the open models without saving changes.

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

## 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 **Save** > **Save Links Only** in the **Requirements** tab.

### 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 **Save** > **Save Links Only** in the **Requirements** tab.

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 1-69.

### 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 numeric or logical `1` (`true`) or `0` (`false`).

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`

**`DuplicateOnCopy` — Preference for copying requirements links with model objects**
`true` (default) | `false`

Preference for copying requirements links along with model objects, specified as a numeric or logical `1` (`true`) or `0` (`false`).

This preference specifies whether requirements links should be duplicated when copying Simulink and Stateflow objects. When set to `false`, links are duplicated only when you highlight links in the source model where the model objects are copied from.

Data Types: `logical`

**`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 numeric or logical `1` (`true`) or `0` (`false`). 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 numeric or logical `1` (`true`) or `0` (`false`). 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 numeric or logical `1` (`true`) or `0` (`false`). 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 numeric or logical `1` (`true`) or `0` (`false`). When set to `true`, right-clicking **Requirements at This Level** > **Delete All Outgoing Links** deletes all requirements links including DOORS surrogate module requirements links. When not set to `true` or `false`, right-clicking **Requirements at This Level** > **Delete All Outgoing 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 numeric or logical `1` (`true`) or `0` (`false`). 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 numeric or logical `1` (`true`) or `0` (`false`). 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 numeric or logical `1` (`true`) or `0` (`false`). 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 numeric or logical `1` (`true`) or `0` (`false`). 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 numeric or logical `1` (`true`) or `0` (`false`). 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 numeric or logical `1` (`true`) or `0` (`false`). 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 numeric or logical `1` (`true`) or `0` (`false`). 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 numeric or logical `1` (`true`) or `0` (`false`).

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 numeric or logical `1` (`true`) or `0` (`false`).

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 numeric or logical `1` (`true`) or `0` (`false`).

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 numeric or logical `1` (`true`) or `0` (`false`). 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 numeric or logical `1` (`true`) or `0` (`false`). 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 1-69.

**`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 1-69.

## 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 file name 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.<br><br>The default configuration includes the **Object Heading** and **Object Text** attributes, and all other attributes, except:<br>• **Created Thru**<br>• System attributes with empty string values<br>• System attributes that are false |
| `'show'` | Display the current settings for the DOORS attributes to include in the report. |
| `'type'` | Include or omit groups of DOORS attributes from the report.<br><br>If you specify `'type'` for the first argument, valid values for the second argument are:<br>• `'all'` — Include all DOORS attributes in the report.<br>• `'user'` — Include only user-defined DOORS in the report.<br>• `'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. |

| Value | Description |
|---|---|
| `'nonempty'` | Enable or disable the empty attribute filter:<br>• Enter `RptgenRMI.doorsAttribs('nonempty', 'off')` to omit all empty attributes from the report.<br>• 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: `'ShowProgressBar','off'`

**`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**

2

# Classes

## 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 = 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 |
| copy | Copy and paste justification |
| demote | Demote justifications |
| find | Find children of parent justification |
| getAttribute | Get justification attributes |
| isHierarchical | Check if justification is hierarchical |
| move | Move justification in hierarchy |
| moveDown | Move justification down in hierarchy |
| moveUp | Move justification up in hierarchy |
| 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 Requirements creates 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.

**`SID` — Link Session Independent Identifier**
character vector

The Session Independent Identifier corresponding to the link.

## Methods

| | |
|---|---|
| destination | Get link destination artifact |
| getAttribute | Get link custom attributes |
| 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 |
| setAttribute | Set link custom attributes |
| setDestination | Set requirement link destination |
| setSource | Set requirement link source |
| 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(myReq);
```

### Get Outgoing Links

```
% Load a link set and get link sources
myLinkSet = slreq.load('c5.slx');
```

```
allSrcs = myLinkSet.sources();

% Get outgoing links
myLink = 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 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 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.

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

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

## Methods

| | |
|---|---|
| addAttribute | Add custom attribute to link set |
| deleteAttribute | Delete custom attribute from link set |
| find | Find links in link set with matching attribute values |
| getLinks | Get links from link set |
| inspectAttribute | Get information about link set custom attribute |
| save | Save link set |
| sources | Get link sources |
| updateAttribute | Update information for link set custom attribute |

## 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 children of parent 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

| | |
|---|---|
| addAttribute | Add custom attribute to requirement set |
| addJustification | Add justifications to requirement set |
| close | Close a requirements set |
| createReferences | Create read-only references to requirement items in third-party documents |
| deleteAttribute | Delete custom attribute from requirement set |
| 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 |
| inspectAttribute | Get information about requirement set custom attribute |
| save | Save a requirements set |
| updateAttribute | Update information for requirement set custom attribute |
| 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('Design_Requirements');

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

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

% 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.

**Index — Requirement index**
character array

The index of the requirement, returned as a character array.

## Methods

| | |
|---|---|
| add | Add requirement to requirements set |
| children | Find child requirements of a requirement |
| copy | Copy and paste requirement |
| demote | Demote requirements |
| find | Find children of parent requirements |
| 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 |
| move | Move requirement in hierarchy |
| moveDown | Move requirement down in hierarchy |
| moveUp | Move requirement up in hierarchy |
| parent | Find parent item of requirement |
| promote | Promote requirements |
| remove | Remove requirement from requirement set |
| 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**

## slreq.verification.services.TAP class

**Package:** `slreq.verification.services`

Work with external results sources

### Description

Instances of the `slreq.verification.services.TAP` provides utilities for interpreting TAP (Test Anything Protocol) result files for verification.

### Construction

Service objects used in the custom logic of `GetResultFcn` to script up result fetching logic.

`tapService = slreq.verification.services.TAP()` directs the result fetching logic to the TAP file.

#### Output Arguments

**`tapService` — services used for TAP files**
character vector

Service used in `GetResultFcn` to script up result fetching logic

### Methods

The output is `result` that is an instance of the `tapService` object. For the `resultFile` with `testID`, the `GetResultFcn` function returns the result for that `testID`:

```
result = tapService.getResult(testID, resultFile);
```

The `GetResultFcn` fetches the `result` for the `testID` with test points in the `resultFile` using:

```
result = tapService.getAllResults(resultFile);
```

### Example

#### Service Usage in a `GetResultFcn` of Link Type

```
 function result = GetResultFcn(link)
    testID = link.destination.id;
    testFile = link.destination.artifact;
    resultFile = getResultFile(testFile);

    if ~isempty(resultFile) && isfile(resultFile)
        tapService = slreq.verification.services.TAP();
        result = tapService.getResult(testID, resultFile);
    else
        result.status = slreq.verification.Status.Unknown;
    end
end
```

## See Also
“Link Type Properties” | `slreq.Link`

**Introduced in R2020a**

# slreq.verification.services.JUnit class

**Package:** `slreq.verification.services`

Work with external results sources

## Description

Instances of the `slreq.verification.services.JUnit` provides utilities for interpreting JUnit result files for verification.

## Construction

`JUnitService = slreq.verification.services.JUnit()` directs the result fetching logic to the XML file.

### Output Arguments

**`JUnitService` — Services used for XML files**
character vector

Services used in `GetResultFcn` to script up result fetching logic

## Methods

The output is `result` that is an instance of the `JUnitService` object. For the `resultFile` with `testID`, the `GetResultFcn` function returns the result for that `testID`:

```
result = JUnitService.getResult(testID, resultFile);
```

The `GetResultFcn` fetches the `result` for the `testID` with test points in the `resultFile` using:

```
result = JUnitService.getAllResults(resultFile);
```

## Example

### Service Usage in a `GetResultFcn` of Link Type

```
 function result = GetResultFcn(link)
    testID = link.destination.id;
    testFile = link.destination.artifact;
    resultFile = getResultFile(testFile);

    if ~isempty(resultFile) && isfile(resultFile)
        JUnitService = slreq.verification.services.JUnit();
        result = JUnitService.getResult(testID, resultFile);
    else
        result.status = slreq.verification.Status.Unknown;
    end
end
```

## See Also

“Link Type Properties” | `slreq.Link`

**Introduced in R2020a**

3

# Methods

## 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**

# copy

**Class:** `slreq.Justification`
**Package:** `slreq`

Copy and paste justification

## Syntax

```
tf = copy(just1,location,just2)
```

## Description

`tf = copy(just1,location,just2)` copies justification `just1` and pastes it under, before, or after justification `just2` depending on the location specified by `location`. The function returns `1` if the copy and paste is performed successfully.

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

## Input Arguments

**`just1` — Justification to copy**
`slreq.Justification` object

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

**`location` — Justification paste location**
`'under'` | `'before'` | `'after'`

Paste location, specified as `'under'`, `'before'`, or `'after'`.

**`just2` — Justification to paste original justification near**
`slreq.Justification` object

Justification to paste original justification near, specified as an `slreq.Justification` object.

## Output Arguments

**`tf` — Paste success status**
`0` | `1`

Paste success status, returned as a `0` or `1` of data type `logical`.

## ExamplesCopy and Paste a Justification

This example shows how to copy a justification and paste it under, before, or after another justification.

Load the `crs_req_justs` requirement file, which describes a cruise control system, and assign it to a variable. Find two justifications by index. The first justification will be copied and pasted in relation to the second justification.

```
rs = slreq.load('crs_req_justs');
jt1 = find(rs,'Type','Justification','Index','5.1');
jt2 = find(rs,'Type','Justification','Index','5.2');
```

### Paste Under a Justification

Copy and paste the first justification, `jt1`, under the second justification, `jt2`. The first justification becomes the last child justification of `jt2`, which you can verify by finding the children of `jt2` and comparing the summary of the last child and `jt1`.

```
tf = copy(jt1,'under',jt2);
```

```
Warning: Error occurred while executing the listener callback for event ReqDataChange defined fo
Dot indexing is not supported for variables of this type.

Error in slreq.das.ReqRoot/recAddDasObjctsIfNeeded

Error in slreq.das.ReqRoot/onReqDataChange

Error in slreq.das.ReqRoot>@(varargin)this.onReqDataChange(varargin{:})

Error in slreq.data.ReqData/pasteFromClipboard

Error in slreq.data.ReqData/copyRequirement

Error in slreq.BaseEditableItem/copy

Error in CopyAndPasteAJustificationExample (line 4)
tf = copy(jt1,'under',jt2);

Error in matlab.internal.editor.evaluateRegions

Error in matlab.internal.editor.EvaluationOutputsService.evalRegions

Error in matlab.internal.liveeditor.LiveEditorUtilities.execute (line 43)
matlab.internal.editor.EvaluationOutputsService.evalRegions(editorId, uuid, regionDataList, full

Error in mwtools.liveCodeToDocbook>doRun (line 364)
    LiveEditorUtilities.execute(editorId, source);

Error in mwtools.liveCodeToDocbook>doRunConvert (line 314)
    [exampleTime,exampleWarnings, exampleErrors] = doRun(javaRichDocument, source);

Error in mwtools.liveCodeToDocbook (line 143)
        [exampleTime,exampleRunWarnings,exampleRunErrors] = doRunConvert(...

Error in BML (line 13)
        evalin('base', s);
```

```
childJusts = children(jt2);
lastChild = childJusts(numel(childJusts));
lastChild.Summary
```

```
ans =
'Non-functional requirement'
```

```
jt1.Summary
```

```
ans = 
'Non-functional requirement'
```

**Paste Before a Justification**

Copy and paste the first justification, `jt1`, before the second justification, `jt2`. Confirm that the justification was pasted before `jt2` by checking the index and summary. The old index of `jt2` was `5.2`. The index of the pasted justification should be `5.2` and the index of `jt2` should be `5.3`.

```
tf = copy(jt1,'before',jt2);
```

```
Warning: Error occurred while executing the listener callback for event ReqDataChange defined fo
Index exceeds the number of array elements (2).

Error in slreq.das.BaseObject/insertChildObject

Error in slreq.das.ReqRoot/recAddDasObjctsIfNeeded

Error in slreq.das.ReqRoot/recAddDasObjctsIfNeeded

Error in slreq.das.ReqRoot/recAddDasObjctsIfNeeded

Error in slreq.das.ReqRoot/onReqDataChange

Error in slreq.das.ReqRoot>@(varargin)this.onReqDataChange(varargin{:})

Error in slreq.data.ReqData/copyRequirement

Error in slreq.BaseEditableItem/copy

Error in CopyAndPasteAJustificationExample (line 9)
tf = copy(jt1,'before',jt2);

Error in matlab.internal.editor.evaluateRegions

Error in matlab.internal.editor.EvaluationOutputsService.evalRegions

Error in matlab.internal.liveeditor.LiveEditorUtilities.execute (line 43)
matlab.internal.editor.EvaluationOutputsService.evalRegions(editorId, uuid, regionDataList, full

Error in mwtools.liveCodeToDocbook>doRun (line 364)
    LiveEditorUtilities.execute(editorId, source);

Error in mwtools.liveCodeToDocbook>doRunConvert (line 314)
    [exampleTime,exampleWarnings, exampleErrors] = doRun(javaRichDocument, source);

Error in mwtools.liveCodeToDocbook (line 143)
        [exampleTime,exampleRunWarnings,exampleRunErrors] = doRunConvert(...

Error in BML (line 13)
        evalin('base', s);
```

```
pastedJust1 = find(rs,'Type','Justification','Index','5.2');
pastedJust1.Summary
```

```
ans = 
'Non-functional requirement'
```

```
jt2.Index
```

```
ans =
'5.3'
```

### Paste After a Justification

Copy and paste the first justification, `jt1`, after the second justification, `jt2`. Confirm that the justification was pasted after `jt2` by checking the index. The index of `jt2` is `5.3` and should not change, which means the index of the pasted justification should be `5.4`.

```
tf = copy(jt1,'after',jt2);
```

```
Warning: Error occurred while executing the listener callback for event ReqDataChange defined fo
Index exceeds the number of array elements (2).

Error in slreq.das.BaseObject/insertChildObject

Error in slreq.das.ReqRoot/recAddDasObjctsIfNeeded

Error in slreq.das.ReqRoot/recAddDasObjctsIfNeeded

Error in slreq.das.ReqRoot/recAddDasObjctsIfNeeded

Error in slreq.das.ReqRoot/onReqDataChange

Error in slreq.das.ReqRoot>@(varargin)this.onReqDataChange(varargin{:})

Error in slreq.data.ReqData/copyRequirement

Error in slreq.BaseEditableItem/copy

Error in CopyAndPasteAJustificationExample (line 13)
tf = copy(jt1,'after',jt2);

Error in matlab.internal.editor.evaluateRegions

Error in matlab.internal.editor.EvaluationOutputsService.evalRegions

Error in matlab.internal.liveeditor.LiveEditorUtilities.execute (line 43)
matlab.internal.editor.EvaluationOutputsService.evalRegions(editorId, uuid, regionDataList, full

Error in mwtools.liveCodeToDocbook>doRun (line 364)
    LiveEditorUtilities.execute(editorId, source);

Error in mwtools.liveCodeToDocbook>doRunConvert (line 314)
    [exampleTime,exampleWarnings, exampleErrors] = doRun(javaRichDocument, source);

Error in mwtools.liveCodeToDocbook (line 143)
        [exampleTime,exampleRunWarnings,exampleRunErrors] = doRunConvert(...

Error in BML (line 13)
        evalin('base', s);
```

```
pastedJust2 = find(rs,'Type','Justification','Index','5.4');
pastedJust2.Summary
```

```
ans =
'Non-functional requirement'
```

```
jt2.Index

ans =
'5.3'
```

#### Cleanup

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

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

## See Also

`move` | `moveDown` | `moveUp` | `slreq.Justification`

**Introduced in R2020b**

# 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 children of parent justification

## Syntax

```
childJusts = find(jt,'PropertyName1',PropertyValue1,...,'PropertyNameN',
PropertyValueN)
```

## Description

`childJusts = find(jt,'PropertyName1',PropertyValue1,...,'PropertyNameN', PropertyValueN)` finds and returns child justifications `childJusts` of the parent justification `jt` that match the properties specified by `PropertyName` and `PropertyValue`.

## Input Arguments

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

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

**`PropertyName` — Justification property**
character vector

Justification property name, specified as a character vector. See the valid property names in the properties section of `slreq.Justification`.

Example: `'Type','Keywords','SID'`

**`PropertyValue` — Justification property value**
character vector | character array | `datetime` value | scalar | `logical` | structure array

Justification property value, specified as a character vector, character array, `datetime` value, scalar, `logical`, or structure array. The data type depends on the specified `propertyName`. See the valid property values in the properties section of `slreq.Justification`.

## Output Arguments

**`childJusts` — Child justifications**
`slreq.Justification` object | `slreq.Justification` object array

Child justifications, returned as `slreq.Justification` objects.

## Examples

### Find Child Justifications

This example shows how to find child justifications that match property values.

Load the `crs_req_justs` requirement file, which describes a cruise control system, and assign it to a variable. Find the justification with index `5`, as this justification has child justifications.

```
rs = slreq.load('crs_req_justs');
parentReq = find(rs,'Type','Justification','Index','5');
```

Find all the child justifications of `parentReq` that were modified in revision `1`.

```
childReqs1 = find(parentReq,'FileRevision',1)
```

```
childReqs1=1×6 object
  1x6 Justification array with properties:

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

Find all the child justifications of `parentReq` that were modified in revision `1` and whose summary says `Non-functional requirement`.

```
childReqs2 = find(parentReq,'FileRevision',1,'Summary','Non-functional requirement')
```

```
childReqs2 = 
  Justification with properties:

              Id: '#72'
         Summary: 'Non-functional requirement'
     Description: '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-htm
        Keywords: {}
       Rationale: ''
       CreatedOn: 27-Feb-2017 10:34:22
       CreatedBy: 'itoy'
      ModifiedBy: 'asriram'
             SID: 72
    FileRevision: 1
      ModifiedOn: 03-Aug-2017 17:14:44
           Dirty: 0
        Comments: [0x0 struct]
           Index: '5.1'
```

### Cleanup

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

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

## See Also

`slreq.Justification` | `slreq.ReqSet` | `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**

# move

**Class:** `slreq.Justification`
**Package:** `slreq`

Move justification in hierarchy

## Syntax

```
tf = move(jt1,location,jt2)
```

## Description

`tf = move(jt1,location,jt2)` moves justification `jt1` under, before, or after justification `jt2` depending on the location specified by `location`. The function returns `1` if the move is performed successfully.

## Input Arguments

**`jt1` — Justification to move**
`slreq.Justification` object

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

**`location` — Justification move location**
`'under'` | `'before'` | `'after'`

Justification move location, specified as `'under'`, `'before'`, or `'after'`.

**`jt2` — Justification**
`slreq.Justification` object

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

## Output Arguments

**`tf` — Paste success status**
`0` | `1`

Paste success status, returned as a `0` or `1` of data type `logical`.

## Examples

### Move a Justification

This example shows how to move a justification under, before, or after another justification.

Load the `crs_req_justs`requirement file, which describes a cruise control system, and assign it to a variable. Find two justifications by index. The first justification will be moved in relation to the second justification.

```
rs = slreq.load('crs_req_justs');
jt1 = find(rs,'Type','Justification','Index','5.1');
jt2 = find(rs,'Type','Justification','Index','5.2');
```

### Move Under a Justification

Move the first justification, `jt1`, under the second justification, `jt2`. The first justification becomes the last child justification of justification `jt2`, and `jt2` moves up one in the hierarchy, which you can verify by checking the index of `jt1` and `jt2`. The old indices of `jt1` and `jt2` were `5.1` and `5.2`, respectively.

```
tf = move(jt1,'under',jt2);
jt1.Index

ans =
'5.1.3'

jt2.Index

ans =
'5.1'
```

### Move Before a Justification

Move the first justification, `jt1`, before the second justification, `jt2`. Confirm that the justification was moved correctly by checking the indices of `jt1` and `jt2`. The indices of `jt1` and `jt2` are now the same as they were originally: `5.1` and `5.2`, respectively.

```
tf = move(jt1,'before',jt2);
jt1.Index

ans =
'5.1'

jt2.Index

ans =
'5.2'
```

### Move After a Justification

Move the first justification, `jt1`, after the second justification, `jt2`. When you move justification `jt1` down in the hierarchy, justification `jt2` also moves up, which you can verify by checking the indices of `jt1` and `jt2`.

```
tf = move(jt1,'after',jt2);
jt1.Index

ans =
'5.2'

jt2.Index

ans =
'5.1'
```

#### Cleanup

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

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

## See Also
copy | moveDown | moveUp | slreq.Justification

**Introduced in R2020b**

## moveDown

**Class:** `slreq.Justification`
**Package:** `slreq`

Move justification down in hierarchy

### Syntax

```
tf = moveDown(jt)
```

### Description

`tf = moveDown(jt)` moves the justification `jt` down one spot in the hierarchy, and returns `1` if the move is successful. The justification `jt` cannot be moved to a new level in the hierarchy.

### Input Arguments

**`jt` — Justification**
`slreq.Justification`

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

### Output Arguments

**`tf` — Paste success status**
`0 | 1`

Paste success status, returned as a `0` or `1` of data type `logical`.

### Examples

#### Move a Justification Down

This example shows how to move a justification down in the hierarchy.

Load the `crs_req_justs` requirement file, which describes a cruise control system, and assign it to a variable. Find the justification with index `5.3`.

```
rs = slreq.load('crs_req_justs');
jt1 = find(rs,'Type','Justification','Index','5.3');
```

Move the justification down one spot in the hierarchy. Confirm the move by checking the success status, `tf1`, and the index.

```
tf1 = moveDown(jt1)
```

```
tf1 = logical
   1
```

```
jt1.Index

ans = 
'5.4'
```

Find the justification with index `5.2.2`. This justification is already at the bottom of its level in the hierarchy and cannot be moved down further, which you can verify by trying to move it down. Confirm that the move failed by checking the success status, `tf2`, and the index.

```
jt2 = find(rs,'Type','Justification','Index','5.2.2');
tf2 = moveDown(jt2)

tf2 = logical
   0


jt2.Index

ans = 
'5.2.2'
```

### Cleanup

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

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

## See Also
`copy` | `move` | `moveUp` | `slreq.Justification`

**Introduced in R2020b**

# moveUp

**Class:** `slreq.Justification`
**Package:** `slreq`

Move justification up in hierarchy

## Syntax

```
tf = moveUp(jt)
```

## Description

`tf = moveUp(jt)` moves the justification `jt` up one spot in the hierarchy, and returns `1` if the move is successful. The justification `jt` cannot be moved to a new level in the hierarchy.

## Input Arguments

**`jt` — Justification**
`slreq.Justification`

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

## Output Arguments

**`tf` — Paste success status**
`0 | 1`

Paste success status, returned as a `0` or `1` of data type `logical`.

## Examples

### Move a Justification Up

This example shows how to move a justification up in the hierarchy.

Load the `crs_req_justs` requirement file, which describes a cruise control system, and assign it to a variable. Find the justification with index `5.3`.

```
rs = slreq.load('crs_req_justs');
jt1 = find(rs,'Type','Justification','Index','5.3');
```

Move the justification up one spot in the hierarchy. Confirm the move by checking the success status, `tf1`, and the index.

```
tf1 = moveUp(jt1)
```

```
tf1 = logical
   1
```

```
jt1.Index
```

```
ans = 
'5.2'
```

Find the justification with index `5.1`. This justification is already at the top of its level in the hierarchy and cannot be moved up further, which you can verify by trying to move it up. Confirm that the move failed by checking the success status, `tf2`, and the index.

```
jt2 = find(rs,'Type','Justification','Index','5.1');
tf2 = moveUp(jt2)
```

```
tf2 = logical
   0
```

```
jt2.Index
```

```
ans = 
'5.1'
```

### Cleanup

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

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

## See Also

`copy` | `move` | `moveDown` | `slreq.Justification`

**Introduced in R2020b**

# 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 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 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 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 destination 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**

# getAttribute

**Class:** `slreq.Link`
**Package:** `slreq`

Get link custom attributes

## Syntax

`val = getAttribute(myLink,name)`

## Description

`val = getAttribute(myLink,name)` returns the custom attribute value of the custom attribute specified by `name` for the link `myLink`.

## Input Arguments

**`myLink` — Link**
`slreq.Link` object

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

**`name` — Custom attribute name**
character array

Custom attribute name, specified as a character array.

## Output Arguments

**`val` — Custom attribute value**
character array | `double` | `logical` | `datetime`

Custom attribute value, returned as a character array, `double`, `logical`, or `datetime`. The data type depends on the custom attribute type.

## Examples

### Get Link Attribute Value

This example shows how to get the attribute value of a specified custom attribute for a link.

Load the `crs_req` requirement files, which contain links for a cruise control system. Find the link set.

```
slreq.load('crs_req');
ls = slreq.find('Type','LinkSet');
```

Create a links array containing all the links from link set `ls`. Get one link from the array. Get the attribute value of the custom attribute called `Target Speed Change`, which tracks whether linked requirements are related to incrementing or decrementing the speed.

```
linksArray = find(ls);
myLink = linksArray(7);
val = getAttribute(myLink,'Target Speed Change')
```

```
val = 
'Decrement'
```

#### Cleanup

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

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

## See Also

`setAttribute` | `slreq.Link` | `slreq.LinkSet`

**Topics**

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

**Introduced in R2020b**

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

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

### 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` | `setDestination` | `setSource`

**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.

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

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

### 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` | `setDestination`

**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.

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

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

### 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` | `setSource`

**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**

# setAttribute

**Class:** `slreq.Link`
**Package:** `slreq`

Set link custom attributes

## Syntax

```
setAttribute(myLink,name,value)
```

## Description

`setAttribute(myLink,name,value)` sets the value specified by `value` of the custom attribute specified by `name` for the link `myLink`.

## Input Arguments

**`myLink` — Link**
`slreq.Link` object

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

**name — Custom attribute name**
character array

Custom attribute name, specified as a character array.

**value — Custom attribute value**
character array | `double` | `logical`, | `datetime`

Custom attribute value, specified as a character array, `double`, `logical` or `datetime`. The data type depends on the custom attribute type.

## Examples

### Set Link Attribute Value

This example shows how to set the attribute value of a specified custom attribute for a link.

Load the `crs_req` requirement files, which contain links for a cruise control system.

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

Create a links array containing all links. Get one link from the array.

```
linksArray = slreq.find('Type','Link')
```

```
linksArray=1×12 object
  1x12 Link array with properties:
    Type
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedOn
    ModifiedBy
    Revision
    SID
    Comments
```

```
lk = linksArray(1);
```

Custom attribute `Target Speed Change`, tracks whether the linked requirements are related to incrementing or decrementing the speed, or not related at all. Set the value of `Target Speed Change` to `Unset` for your link. Then use `getAttribute` to confirm that the value was set correctly.

```
setAttribute(lk,'Target Speed Change','Unset');
value = getAttribute(lk,'Target Speed Change')
```

```
value =
'Unset'
```

### Cleanup

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

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

## See Also

`getAttribute` | `slreq.Link` | `slreq.LinkSet`

### Topics

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

**Introduced in R2020b**

# setDestination

**Class:** `slreq.Link`
**Package:** `slreq`

Set requirement link destination

## Syntax

`setDestination(myLink,dest)`

## Description

`setDestination(myLink,dest)` sets the link destination artifact `dest` for the `slreq.Link` object `myLink`.

## Input Arguments

**`myLink` — Link object**
`slreq.Link` object

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

**dest — Link destination**
Simulink Requirements linkable item

Artifact to serve as the link destination, specified as a Simulink Requirements linkable item. See “Linkable Items”.

## Examples

### Set Simulink Blocks as Link Destinations

```
% Set the Gain block in model myModel as the destination for link myLink
setDestination(myLink, 'myModel/Gain');
```

### Set Simulink Test Objects as Link Destinations

```
% Create a Simulink Test test file, test suite, and a test case
myTestfile = sltest.testmanager.TestFile('my_test_file.mldatx');
myTestsuite = sltest.testmanager.TestSuite(myTestfile,'My Test Suite');
myTestcase = sltest.testmanager.TestCase(myTestsuite,'equivalence','Equivalence Test Case');

% Create a link from the test case to requirement myReq
myLink = slreq.createLink(req, myTestcase);

% Set the link destination to the test suite
setDestination(myLink, myTestsuite);
```

### Set Stateflow Objects as Link Destinations

```
% Get Stateflow Root Handle
rt = sfroot;
```

```
% Find the state with the name 'Intermediate'
myState = rt.find('-isa', 'Stateflow.State', 'Name', 'Intermediate');

% Set the destination for link myLink to myState
setDestination(myLink, myState);
```

**Set Simulink Data Dictionary Entries as Link Destinations**

```
% Get handle to Simulink data dictionary entry
myDict = Simulink.data.dictionary.open('myDictionary.sldd');
dataSectObj = getSection(myDict,'Design Data');
myDictEntry = getEntry(dataSectObj,'myEntry');

% Set the destination for link myLink to myDictEntry
setDestination(myLink, myDictEntry);
```

## See Also

`setSource`

**Introduced in R2019b**

# setSource

**Class:** `slreq.Link`
**Package:** `slreq`

Set requirement link source

## Syntax

`setSource(myLink,src)`

## Description

`setSource(myLink,src)` sets the link source artifact `src` for the `slreq.Link` object `myLink`. You can set a link source only to a linkable artifact that belongs to the original link source artifact.

## Input Arguments

**`myLink` — Link object**
`slreq.Link` object

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

**`src` — Link source**
Simulink Requirements linkable artifact

Artifact to serve as the link source, specified as a Simulink Requirements linkable artifact. See “Linkable Items”.

## Examples

### Set Simulink Blocks as Link Sources

```
% Set the Gain block in model myModel as the source for link myLink
setSource(myLink, 'myModel/Gain');
```

### Set Simulink Test Objects as Link Source

```
% Create a test file, test suite, and a test case
myTestfile = sltest.testmanager.TestFile('my_test_file.mldatx');
myTestsuite = sltest.testmanager.TestSuite(myTestfile,'My Test Suite');
myTestcase = sltest.testmanager.TestCase(myTestsuite,'equivalence','Equivalence Test Case');

% Create a link from the test case to requirement myReq
myLink = slreq.createLink(myTestcase, req);

% Set the link source to the test suite
setSource(myLink, myTestsuite);
```

### Set Stateflow Objects as Link Sources

```
% Get Stateflow Root Handle
rt = sfroot;
```

```
% Find the state with the name 'Intermediate'
myState = rt.find('-isa', 'Stateflow.State', 'Name', 'Intermediate');

% Set the source for link myLink to myState
setSource(myLink, myState);
```

**Set Simulink Data Dictionary Entries as Link Sources**

```
% Get handle to Simulink data dictionary entry
myDict = Simulink.data.dictionary.open('myDictionary.sldd');
dataSectObj = getSection(myDict,'Design Data');
myDictEntry = getEntry(dataSectObj,'myEntry');

% Set the source for link myLink to myDictEntry
setSource(myLink, myDictEntry);
```

**Change a Link Source to a Different Source Artifact**

```
% Get destination of link link_1
dest = destination(link_1);

% Create a new link, link_2, with source newSrc and destination dest
link_2 = slreq.createLink(newSrc, dest);

% Copy link properties
link_2.Description = link_1.Description;
link_2.Rationale = link_1.Rationale;
link_2.Keywords = link_1.Keywords;
comments = link_1.Comments;
for i = 1:length(comments)
  link_2.addComment(comments(i).Text);
end

% Delete link_1
remove(link_1);
```

## See Also

`setDestination`

**Introduced in R2019b**

# 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
`destination` | `linkSet` | `slreq.Link`

**Introduced in R2018a**

# addAttribute

**Class:** `slreq.LinkSet`
**Package:** `slreq`

Add custom attribute to link set

## Syntax

```
addAttribute(myLinkSet,name,type)
addAttribute(myLinkSet,name,'Checkbox','DefaultValue',value)
addAttribute(myLinkSet,name,'Combobox','List',options)
addAttribute(myLinkSet, ___ ,'Description',descr)
```

## Description

`addAttribute(myLinkSet,name,type)` adds a custom attribute with the name specified by `name` and the custom attribute type specified by `type` to the link set `myLinkSet`.

`addAttribute(myLinkSet,name,'Checkbox','DefaultValue',value)` adds a `Checkbox` custom attribute with the name specified by `name` and the default value specified by `value` to the link set `myLinkSet`.

`addAttribute(myLinkSet,name,'Combobox','List',options)` adds a `Combobox` custom attribute with name specified by `name`, and the list options specified by `options` to the link set `myLinkSet`.

`addAttribute(myLinkSet, ___ ,'Description',descr)` adds a custom attribute with the name specified by `name`, the type specified by `type`, and the description specified by `descr` to the link set `myLinkSet`.

## Input Arguments

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

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

**name — Custom attribute name**
character array

Custom attribute name, specified as a character array.

**type — Custom attribute type**
`'Edit'` | `'Checkbox'` | `'Combobox'` | `'DateTime'`

Custom attribute type, specified as a character array. The valid custom attribute types are `'Edit'`, `'Checkbox'`, `'Combobox'`, and `'DateTime'`.

**`descr` — Custom attribute description**
character array

Custom attribute description, specified as a character array.

**value — Checkbox default value**
false (default) | true

Checkbox default value, specified as a logical 1 (true) or 0 (false).

**options — Combobox list options**
cell array

Combobox list options, specified as a cell array. The list of options is valid only if 'Unset' is the first entry. 'Unset' indicates that the user hasn't chosen an option from the combo box. If the list does not start with 'Unset', it will be automatically appended as the first entry.

Example: {'Unset','A','B','C'}

## Examples

### Add Custom Attribute to Link Set

This example shows how to add a custom attribute to of all four available types, Edit, Checkbox, Combobox, and DateTime, and how to add a custom attribute with a description.

#### Add an Edit Custom Attribute

Load the crs_req requirement files, which describes for a cruise control system. Find a link set in the files and assign it to a variable.

```
slreq.load('crs_req');
ls = slreq.find('Type','LinkSet');
```

Add an Edit custom attribute. Confirm that the attribute added by using inspectAttribute.

```
addAttribute(ls,'MyEditAttribute','Edit');
atrb = inspectAttribute(ls,'MyEditAttribute')
```

```
atrb = struct with fields:
           name: 'MyEditAttribute'
           type: Edit
    description: ''
```

#### Add a Checkbox Custom Attribute

Add a Checkbox custom attribute with the default value true. Confirm that the attribute was added successfully by using inspectAttribute.

```
addAttribute(ls,'MyCheckbox','Checkbox','DefaultValue',true);
atrb2 = inspectAttribute(ls,'MyCheckbox')
```

```
atrb2 = struct with fields:
           name: 'MyCheckbox'
           type: Checkbox
    description: ''
        default: 1
```

### Add a Combobox Custom Attribute

Add a `ComboBox` custom attribute with the options `Unset`, `A`, `B`, and `C`. Confirm that the attribute was added successfully by using `inspectAttribute`.

```
addAttribute(ls,'MyCombobox','Combobox','List',{'Unset','A','B','C'});
atrb3 = inspectAttribute(ls,'MyCombobox')
```

```
atrb3 = struct with fields:
           name: 'MyCombobox'
           type: Combobox
    description: ''
           list: {'Unset'  'A'  'B'  'C'}
```

### Add a DateTime Custom Attribute

Add a `DateTime` custom attribute. Confirm that the attribute was added successfully by using `inspectAttribute`.

```
addAttribute(ls,'MyDateTime','DateTime');
atrb4 = inspectAttribute(ls,'MyDateTime')
```

```
atrb4 = struct with fields:
           name: 'MyDateTime'
           type: DateTime
    description: ''
```

### Add a Custom Attribute with a Description

Add an `Edit` custom attribute. Add a description to the custom attribute. Confirm that the attribute was added successfully by using `inspectAttribute`.

```
addAttribute(ls,'MyEditAttribute2','Edit','Description',...
    'You can enter text as the custom attribute value.');
atrb5 = inspectAttribute(ls,'MyEditAttribute2')
```

```
atrb5 = struct with fields:
           name: 'MyEditAttribute2'
           type: Edit
    description: 'You can enter text as the custom attribute value.'
```

Add a `ComboBox` custom attribute with the options `Unset`, `A`, `B`, and `C`. Add a description to the custom attribute. Confirm that the attribute was added successfully by using `inspectAttribute`.

```
addAttribute(ls,'MyCombobox2','Combobox','List',{'Unset','A','B','C'},'Description',...
    'This combo box attribute has 4 options.');
atrb6 = inspectAttribute(ls,'MyCombobox2')
```

```
atrb6 = struct with fields:
           name: 'MyCombobox2'
           type: Combobox
    description: 'This combo box attribute has 4 options.'
           list: {'Unset'  'A'  'B'  'C'}
```

### Cleanup

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

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

## See Also

`deleteAttribute` | `inspectAttribute` | `slreq.LinkSet` | `updateAttribute`

**Topics**

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

**Introduced in R2020b**

# deleteAttribute

**Class:** `slreq.LinkSet`
**Package:** `slreq`

Delete custom attribute from link set

## Syntax

```
deleteAttribute(myLinkSet,name,'Force',true)
deleteAttribute(myLinkSet,name,'Force',false)
```

## Description

`deleteAttribute(myLinkSet,name,'Force',true)` deletes the custom attribute specified by `name` from the link set `myLinkSet`, even if the custom attribute is used by links in the link set.

`deleteAttribute(myLinkSet,name,'Force',false)` deletes the custom attribute specified by `name` from the link set `myLinkSet` only if the custom attribute is not used by links in the link set.

## Input Arguments

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

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

**name — Custom attribute name**
character array

Custom attribute name, specified as a character array.

## Examples

### Delete Custom Attribute

This example shows how to delete a custom attribute.

Load the `crs_req` requirement files, which contain links for a cruise control system. Find a link set in the files.

```
slreq.load('crs_req');
ls = slreq.find('Type','LinkSet');
```

Delete the custom attribute named `Target Speed Change` from the link set. Because the `Target Speed Change` attribute is used by links, it can only be deleted by setting `Force` to `true`.Confirm that it was deleted successfully by accessing the `CustomAttributeNames` property for the link set.

```
deleteAttribute(ls,'Target Speed Change','Force',true)
atrb1 = ls.CustomAttributeNames
```

```
atrb1 =

  0x0 empty cell array
```

#### Only Delete Custom Attribute if the Attribute is Unused

Add an `Edit` custom attribute to the link set. The attribute is unused because the value is not set for any links. Confirm that it was added successfully by accessing the `CustomAttributeNames` property for the link set.

```
addAttribute(ls,'MyEditAttribute','Edit')
atrb2 = ls.CustomAttributeNames
```

```
atrb2 = 1x1 cell array
    {'MyEditAttribute'}
```

If you set `Force` to `false`, you can delete the attribute only if the attribute is unused. If the attribute is used by links, then an error will occur. Confirm the deletion by accessing the `CustomAttributeNames` property for the link set.

```
deleteAttribute(ls,'MyEditAttribute','Force',false)
atrb3 = ls.CustomAttributeNames
```

```
atrb3 =

  0x0 empty cell array
```

#### Cleanup

Clean up commands. Clear the open requirement sets, link sets, and open models without saving changes.

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

## See Also

`addAttribute` | `inspectAttribute` | `slreq.LinkSet` | `updateAttribute`

**Topics**

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

**Introduced in R2020b**

# find

**Class:** `slreq.LinkSet`
**Package:** `slreq`

Find links in link set with matching attribute values

## Syntax

```
myLinks = find(myLinkSet,'PropertyName1',PropertyValue1,...,'PropertyNameN',
PropertyValueN)
```

## Description

`myLinks = find(myLinkSet,'PropertyName1',PropertyValue1,...,'PropertyNameN', PropertyValueN)` finds and returns all `slreq.Link` objects in the link set `myLinkSet` that match the properties specified by `PropertyName` and `PropertyValue`.

## Input Arguments

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

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

**`PropertyName` — Link property**
character vector

Link property name, specified as a character vector. See the valid property names in the properties section of `slreq.Link`.

Example: `'Type','Keywords','SID'`

**`PropertyValue` — Link property value**
character vector | character array | `datetime` value | scalar | `logical` | structure array

Link property value, specified as a character vector, character array, `datetime` value, scalar, `logical`, or structure array. The data type depends on the specified `propertyName`. See the valid property values in the properties section of `slreq.Link`.

Example: `'Type','Keywords','SID'`

## Output Arguments

**`myLinks` — Link**
`slreq.Link` object

Link or link array, specified as an `slreq.Link` object.

## Examples

### Find a Link in a Requirement Set

This example shows how to find a link in a link set that matches the specified property value.

Load the `crs_req` requirement files, which contain links for a cruise control system. Define the link set by assigning it to a variable.

```
slreq.load('crs_req');
ls = slreq.find('Type','LinkSet');
```

Find a link that matches the specified SID.

```
myLink = find(ls,'SID','3')
```

```
myLink = 
  Link with properties:

           Type: 'Derive'
    Description: '#8: Set Switch Detection'
       Keywords: {}
      Rationale: ''
      CreatedOn: 20-May-2017 13:14:40
      CreatedBy: 'itoy'
     ModifiedOn: 09-Jun-2020 14:57:35
     ModifiedBy: 'ahoward'
       Revision: 5
            SID: 3
       Comments: [0x0 struct]
```

Find all links that are modified in the specified revision.

```
myLinks = find(ls,'Revision','7')
```

```
myLinks=1×8 object
  1x8 Link array with properties:

    Type
    Description
    Keywords
    Rationale
    CreatedOn
    CreatedBy
    ModifiedOn
    ModifiedBy
    Revision
    SID
    Comments
```

Find a link that matches the specified SID and revision.

```
myLink2 = find(ls,'SID','8','Revision','7')
```

```
myLink2 = 
  Link with properties:

           Type: 'Derive'
    Description: '#12: Increment Short Switch Detection'
```

```
         Keywords: {}
        Rationale: ''
        CreatedOn: 20-May-2017 13:15:45
        CreatedBy: 'itoy'
       ModifiedOn: 09-Jun-2020 15:14:55
       ModifiedBy: 'ahoward'
         Revision: 7
              SID: 8
         Comments: [0x0 struct]
```

### Cleanup

Clean up commands. Clear the open requirement sets and link sets and close the open models without saving changes.

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

## See Also

slreq.LinkSet | slreq.find

**Introduced in R2018a**

# getLinks

**Class:** slreq.LinkSet
**Package:** slreq

Get links from link set

## Syntax

```
lks = getLinks(lkset)
```

## Description

`lks = getLinks(lkset)` returns an array `lks` of `Links` from `lkset`, a `LinkSet`.

## Input Arguments

**lkset — Link set**
LinkSet

`LinkSet` from which to get links.

Example: `LinkSet with properties:`

## Output Arguments

**lks — Links**
Link | Link array

`Links` in the link set.

## Examples

### Get Links from a Link Set

```
load_system('reqs_validation_property_proving_original_model');
rq = slreq.load('original_thrust_reverser_requirements.slreqx');
lk = slreq.load('reqs_validation_property_proving_original_model.slmx');

sl = getLinks(lk);
```

## See Also

sources

**Introduced in R2020a**

# inspectAttribute

**Class:** `slreq.LinkSet`
**Package:** `slreq`

Get information about link set custom attribute

## Syntax

```
atrb = inspectAttribute(myLinkSet,name)
```

## Description

`atrb = inspectAttribute(myLinkSet,name)` returns a structure with information about the custom attribute name specified by `name` in the link set `myLinkSet`.

## Input Arguments

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

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

**name — Custom attribute name**
character array

Custom attribute name, specified as a character array.

## Output Arguments

**`atrb` — Custom attribute information**
`struct`

Custom attribute information, returned as a `struct`.

## Examples

### Get Link Set Custom Attribute Information

This example shows how to get information about a link set custom attribute.

Load the `crs_req` requirement files, which describes a cruise control system. Find a link set from the files and assign it to a variable.

```
slreq.load('crs_req');
ls = slreq.find('Type','LinkSet');
```

The custom attribute `Target Speed Change` tracks whether linked requirements are related to incrementing or decrementing the speed, or not related at all. Get information about this custom attribute.

```
atrb = inspectAttribute(ls,'Target Speed Change')
```

```
atrb = struct with fields:
           name: 'Target Speed Change'
           type: Combobox
    description: 'Tracks if linked requirements are related to incrementing or decrementing speed
           list: {'Unset'  'Increment'  'Decrement'}
```

### Cleanup

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

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

## See Also

`addAttribute` | `deleteAttribute` | `slreq.LinkSet` | `updateAttribute`

**Topics**

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

**Introduced in R2020b**

# 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**

# updateAttribute

**Class:** `slreq.LinkSet`
**Package:** `slreq`

Update information for link set custom attribute

## Syntax

```
updateAttribute(myLinkSet,atrb,Name,Value)
```

## Description

`updateAttribute(myLinkSet,atrb,Name,Value)` updates the custom attribute specified by `atrb` with properties specified by the name-value pairs `Name` and `Value` in the link set `myLinkSet`.

## Input Arguments

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

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

**`atrb` — Custom attribute name**
character array

Custom attribute name, specified as a character array.

### 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: `'Description','My new description.'`

**`Description` — Custom attribute description**
character array

Custom attribute description, specified as the comma-separated pair consisting of `'Description'` and a character array.

Example: `'Description','My new description.'`

**`List` — Combobox list options**
cell array

`Combobox` list options, specified as the comma-separated pair consisting of `'List'` and a cell array. The list of options is valid only if `'Unset'` is the first entry. `'Unset'` indicates that the user hasn't chosen an option from the combo box. If the list does not start with `'Unset'`, it will be automatically appended as the first entry.

Example: `'List',{'Unset','A','B','C'}`

**Note** You can only use this name-value pair when the Type property of the custom attribute that you're updating is Combobox.

## Examples

### Update Link Set Custom Attribute Information

This example shows how to update custom attribute information for a link set.

Load the crs_req requirement files, which describe a cruise control system. Find a link set in the files and assign it to a variable.

```
slreq.load('crs_req');
ls = slreq.find('Type','LinkSet');
```

#### Update an Edit Custom Attribute

Add an Edit custom attribute that has a description to the link set. Get the attribute information with inspectAttribute.

```
addAttribute(ls,'MyEditAttribute','Edit','Description','Original attribute.');
inspectAttribute(ls,'MyEditAttribute')
```

```
ans = struct with fields:
           name: 'MyEditAttribute'
           type: Edit
    description: 'Original attribute.'
```

Update the custom attribute with a new description. Confirm the change by getting the attribute information with inspectAttribute.

```
updateAttribute(ls,'MyEditAttribute','Description','Updated attribute.');
inspectAttribute(ls,'MyEditAttribute')
```

```
ans = struct with fields:
           name: 'MyEditAttribute'
           type: Edit
    description: 'Updated attribute.'
```

#### Update a Combobox Custom Attribute

Add a Combobox custom attribute with a list of options to the link set. Get the attribute information with inspectAttribute.

```
addAttribute(ls,'MyCombobox','Combobox','List',{'Unset','A','B','C'});
inspectAttribute(ls,'MyCombobox')
```

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

Update the custom attribute with a new list of options. Confirm the change by getting the attribute information with `inspectAttribute`.

```
updateAttribute(ls,'MyCombobox','List',{'Unset','1','2','3'});
inspectAttribute(ls,'MyCombobox')
```

```
ans = struct with fields:
           name: 'MyCombobox'
           type: Combobox
    description: ''
           list: {'Unset'  '1'  '2'  '3'}
```

Update the custom attribute with a new list of options and a new description. Confirm the change by getting the attribute information with `inspectAttribute`.

```
updateAttribute(ls,'MyCombobox','List',{'Unset','A1','B2','B3'},'Description',...
    'Updated attribute with new options.');
inspectAttribute(ls,'MyCombobox')
```

```
ans = struct with fields:
           name: 'MyCombobox'
           type: Combobox
    description: 'Updated attribute with new options.'
           list: {'Unset'  'A1'  'B2'  'B3'}
```

### Cleanup

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

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

## See Also

`addAttribute` | `deleteAttribute` | `inspectAttribute` | `slreq.LinkSet`

**Topics**

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

**Introduced in R2020b**

# 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 children of parent referenced requirements

## Syntax

```
childRefs = find(ref,'PropertyName1',PropertyValue1,...,'PropertyNameN',
PropertyValueN)
```

## Description

`childRefs = find(ref,'PropertyName1',PropertyValue1,...,'PropertyNameN', PropertyValueN)` finds and returns child referenced requirements `childRefs` of the parent referenced requirement `ref` that match the properties specified by `PropertyName` and `PropertyValue`.

## Input Arguments

**`ref` — Referenced requirement**
`slreq.Reference` object

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

**`PropertyName` — Reference property**
character vector

Reference property name, specified as a character vector. See the valid property names in the properties section of `slreq.Reference`.

Example: `'Type'`,`'Keywords'`,`'SID'`

**`PropertyValue` — Reference property value**
character vector | character array | `datetime` value | scalar | `logical` | structure array

Reference property value, specified as a character vector, character array, `datetime` value, scalar, `logical`, or structure array. The data type depends on the specified `propertyName`. See the valid property values in the properties section of `slreq.Reference`

## Output Arguments

**`childRefs` — Child referenced requirements**
`slreq.Reference` object | `slreq.Reference` object array

Child referenced requirements, returned as `slreq.Reference` objects.

## Examples

### Find Child Referenced Requirements

This example shows how to find child referenced requirements that match property values.

Load the `crs_req` requirement file, which describes a cruise control system, and assign it to a variable. Find the referenced requirement with index `3`, as this referenced requirement has child referenced requirements.

```
rs = slreq.load('crs_req');
parentRef = find(rs,'Type','Reference','Index','3')
```

```
parentRef = 
  Reference with properties:

             Id: 'Functional Requirements'
       CustomId: 'Functional Requirements'
       Artifact: 'crs_req.docx'
     ArtifactId: '?Functional Requirements'
         Domain: 'linktype_rmi_word'
      UpdatedOn: 02-Feb-2018 13:23:13
      CreatedOn: NaT
      CreatedBy: ''
     ModifiedBy: ''
       IsLocked: 1
        Summary: 'Functional Requirements'
    Description: '<div class=WordSection1>...'
      Rationale: ''
       Keywords: {}
           Type: 'Functional'
            SID: 9
   FileRevision: 1
     ModifiedOn: 03-Aug-2017 17:34:56
          Dirty: 0
       Comments: [0x0 struct]
          Index: '3'
```

Find all the child referenced requirements of `parentRef` that were modified in revision `1`.

```
childRefs1 = find(parentRef,'FileRevision',1)
```

```
childRefs1=1×18 object
  1x18 Reference array with properties:

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

Find all the child referenced requirements of `parentRef` that were modified in revision 1 and have an SID equal to 12.

```
childRefs2 = find(parentRef,'FileRevision',1,'SID',12)
```

```
childRefs2 =
  Reference with properties:

            Id: 'Activating cruise control'
      CustomId: 'Activating cruise control'
      Artifact: 'crs_req.docx'
    ArtifactId: '?Activating cruise control'
        Domain: 'linktype_rmi_word'
     UpdatedOn: 02-Feb-2018 13:23:13
     CreatedOn: NaT
     CreatedBy: ''
    ModifiedBy: ''
      IsLocked: 1
       Summary: 'Activating cruise control'
   Description: '<div class=WordSection1>...'
     Rationale: ''
      Keywords: {}
          Type: 'Functional'
           SID: 12
  FileRevision: 1
    ModifiedOn: 03-Aug-2017 17:34:56
         Dirty: 0
      Comments: [0x0 struct]
         Index: '3.3'
```

### Cleanup

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

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

## 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 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 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` as well as the Import node itself. The function returns the number of referenced requirements removed.

### 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 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**

## addAttribute

**Class:** `slreq.ReqSet`
**Package:** `slreq`

Add custom attribute to requirement set

### Syntax

```
addAttribute(rs,name,type)
addAttribute(rs,name,'Checkbox','DefaultValue',value)
addAttribute(rs,name,'Combobox','List',options)
addAttribute(rs,___,'Description',descr)
```

### Description

`addAttribute(rs,name,type)` adds a custom attribute with the name specified by `name` and the custom attribute type specified by `type` to the requirement set `rs`.

`addAttribute(rs,name,'Checkbox','DefaultValue',value)` adds a `Checkbox` custom attribute with the name specified by `name` and the default value specified by `value` to the requirement set `rs`.

`addAttribute(rs,name,'Combobox','List',options)` adds a `Combobox` custom attribute with the name specified by `name`, and the list options specified by `options` to the requirement set `rs`.

`addAttribute(rs,___,'Description',descr)` adds a custom attribute with the name specified by `name`, the type specified by `type`, and the description specified by `descr` to the requirement set `rs`.

### Input Arguments

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

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

**name — Custom attribute name**
character array

Custom attribute name, specified as a character array.

**type — Custom attribute type**
`'Edit'` | `'Checkbox'` | `'Combobox'` | `'DateTime'`

Custom attribute type, specified as a character array. The valid custom attribute types are `Edit`, `Checkbox`, `Combobox`, and `DateTime`.

**`descr` — Custom attribute description**
character array

Custom attribute description, specified as a character array.

**value — Checkbox default value**
`false` (default) | `true`

`Checkbox` default value, specified as a logical `1 (true)` or `0 (false)`.

**options — Combobox list options**
cell array

`Combobox` list options, specified as a cell array. The list of options is valid only if `'Unset'` is the first entry. `'Unset'` indicates that the user hasn't chosen an option from the combo box. If the list does not start with `'Unset'`, it will be automatically appended as the first entry.

Example: `{'Unset','A','B','C'}`

## Examples

### Add Custom Attribute to Requirement Set

This example shows how to add a custom attribute of all four types to a requirement set, `Edit`, `Checkbox`, Combobox, and `DateTime`, and how to add a custom attribute with a description.

#### Add an Edit Custom Attribute

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

```
slreq.load('crs_req_func_spec');
rs = slreq.find('Type','ReqSet');
```

Add an `Edit` custom attribute. Confirm that the attribute was successfully added by using `inspectAttribute`.

```
addAttribute(rs,'MyEditAttribute','Edit');
atrb = inspectAttribute(rs,'MyEditAttribute')
```

```
atrb = struct with fields:
           name: 'MyEditAttribute'
           type: Edit
    description: ''
```

#### Add a Checkbox Custom Attribute

Add a `Checkbox` custom attribute with the default value `true`. Confirm that the attribute was successfully added by using `inspectAttribute`.

```
addAttribute(rs,'MyCheckbox','Checkbox','DefaultValue',true);
atrb2 = inspectAttribute(rs,'MyCheckbox')
```

```
atrb2 = struct with fields:
           name: 'MyCheckbox'
           type: Checkbox
    description: ''
```

```
        default: 1
```

### Add a Combobox Custom Attribute

Add a ComboBox custom attribute with the options Unset, A, B, and C. Confirm that the attribute was successfully added by using inspectAttribute.

```
addAttribute(rs,'MyCombobox','Combobox','List',{'Unset','A','B','C'});
atrb3 = inspectAttribute(rs,'MyCombobox')
```

```
atrb3 = struct with fields:
           name: 'MyCombobox'
           type: Combobox
    description: ''
           list: {'Unset'  'A'  'B'  'C'}
```

### Add a DateTime Custom Attribute

Add a DateTime custom attribute. Confirm that the attribute was successfully added by using inspectAttribute.

```
addAttribute(rs,'MyDateTime','DateTime');
atrb4 = inspectAttribute(rs,'MyDateTime')
```

```
atrb4 = struct with fields:
           name: 'MyDateTime'
           type: DateTime
    description: ''
```

### Add a Custom Attribute with a Description

Add an Edit custom attribute. Add a description to the custom attribute. Confirm that the attribute was successfully added by using inspectAttribute.

```
addAttribute(rs,'MyEditAttribute2','Edit','Description',...
    'You can enter text as the custom attribute value.');
atrb5 = inspectAttribute(rs,'MyEditAttribute2')
```

```
atrb5 = struct with fields:
           name: 'MyEditAttribute2'
           type: Edit
    description: 'You can enter text as the custom attribute value.'
```

Add a ComboBox custom attribute with the options Unset, A, B, and C. Add a description to the custom attribute. Confirm that the attribute was successfully added by using inspectAttribute.

```
addAttribute(rs,'MyCombobox2','Combobox','List',{'Unset','A','B','C'},'Description',...
    'This combobox attribute has 4 options.');
atrb6 = inspectAttribute(rs,'MyCombobox2')
```

```
atrb6 = struct with fields:
           name: 'MyCombobox2'
           type: Combobox
    description: 'This combobox attribute has 4 options.'
```

```
          list: {'Unset'  'A'  'B'  'C'}
```

**Cleanup**

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

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

## See Also
deleteAttribute | inspectAttribute | slreq.ReqSet | updateAttribute

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

**Introduced in R2020b**

## 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**

# deleteAttribute

**Class:** `slreq.ReqSet`
**Package:** `slreq`

Delete custom attribute from requirement set

## Syntax

```
deleteAttribute(rs,name,'Force',true)
deleteAttribute(rs,name,'Force',false)
```

## Description

`deleteAttribute(rs,name,'Force',true)` deletes the custom attribute specified by `name` from the requirement set `rs`, even if the custom attribute is used by requirements in the requirement set.

`deleteAttribute(rs,name,'Force',false)` deletes the custom attribute specified by `name` from the requirement set `rs` only if the custom attribute is not used by requirements in the requirement set.

## Input Arguments

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

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

**name — Custom attribute name**
character array

Custom attribute name, specified as a character array.

## Examples

### Delete Custom Attribute

This example shows how to delete a custom attribute.

Load `crs_req_func_spec`, which is the requirement file for a cruise control system. Find a requirement set in the files.

```
slreq.load('crs_req_func_spec');
rs = slreq.find('Type','ReqSet');
```

Add an `Edit` custom attribute to the requirement set. Confirm that it was successfully added by accessing the `CustomAttributeNames` property for the requirement set.

```
addAttribute(rs,'MyCheckbox','Checkbox')
atrb1 = rs.CustomAttributeNames
```

```
atrb1 = 1x1 cell array
    {'MyCheckbox'}
```

Find a requirement in the requirement set. Set the custom attribute value for the requirement using `setAttribute`.

```
req = find(rs,'ID','#1');
setAttribute(req,'MyCheckbox',true)
```

The custom attribute `MyCheckbox` is now used by a requirement. Delete the requirement by using `deleteAttribute` with `'Force'` set to `true`. Confirm the deletion by accessing the `CustomAttributeNames` property for the requirement set.

```
deleteAttribute(rs,'MyCheckbox','Force',true)
atrb2 = rs.CustomAttributeNames
```

```
atrb2 =

  0x0 empty cell array
```

#### Only Delete Custom Attribute if the Attribute is Unused

Add an `Edit` custom attribute to the requirement set. The attribute is unused because the value is not set for any links. Confirm that it added by accessing the `CustomAttributeNames` property for the requirement set.

```
addAttribute(rs,'MyEditAttribute','Edit')
atrb3 = rs.CustomAttributeNames
```

```
atrb3 = 1x1 cell array
    {'MyEditAttribute'}
```

You can delete the attribute only if the attribute is unused by setting `Force` to `false`. If the attribute is used by links, then an error will occur. Confirm the deletion by accessing the `CustomAttributeNames` property for the requirement set.

```
deleteAttribute(rs,'MyEditAttribute','Force',false)
atrb4 = rs.CustomAttributeNames
```

```
atrb4 =

  0x0 empty cell array
```

#### Cleanup

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

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

## See Also

`addAttribute` | `inspectAttribute` | `slreq.ReqSet` | `updateAttribute`

#### Topics

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

**Introduced in R2020b**

# 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”.

## 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'}`

**`columns` — Range of columns**
`double` array

Range of columns to import from Microsoft Excel spreadsheet, specified as a `double` array.

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

**`rows` — Range of rows**
`double` array

Range of rows to import from Microsoft Excel spreadsheet, 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);
```

For more information on importing requirements from Microsoft Office documents, see “Import Requirements from Microsoft Office Documents”.

## See Also
`createReferences` | `slreq.ReqSet`

**Introduced in R2018a**

# inspectAttribute

**Class:** `slreq.ReqSet`
**Package:** `slreq`

Get information about requirement set custom attribute

## Syntax

`atrb = inspectAttribute(rs,name)`

## Description

`atrb = inspectAttribute(rs,name)` returns a structure with information about the custom attribute name specified by `name` in the requirement set `rs`.

## Input Arguments

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

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

**`name` — Custom attribute name**
character array

Custom attribute name, specified as a character array.

## Output Arguments

**`atrb` — Custom attribute information**
`struct`

Custom attribute information, returned as a `struct`.

## Examples

### Get Requirement Set Custom Attribute Information

This example shows how to get information about a requirement set custom attribute.

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

```
slreq.load('crs_req_func_spec');
rs = slreq.find('Type','ReqSet');
```

Add a `Checkbox` custom attribute to the requirement set with a description. Use `inspectAttribute` to get information about the custom attribute.

```
addAttribute(rs,'MyCheckbox','Checkbox','Description',...
    'This checkbox atrribute can be true or false.');
atrb = inspectAttribute(rs,'MyCheckbox')
```

```
atrb = struct with fields:
           name: 'MyCheckbox'
           type: Checkbox
    description: 'This checkbox atrribute can be true or false.'
        default: 0
```

### Cleanup

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

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

## See Also

addAttribute | deleteAttribute | slreq.ReqSet | updateAttribute

### Topics

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

**Introduced in R2020b**

# 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**

## updateAttribute

**Class:** `slreq.ReqSet`
**Package:** `slreq`

Update information for requirement set custom attribute

### Syntax

`updateAttribute(rs,atrb,Name,Value)`

### Description

`updateAttribute(rs,atrb,Name,Value)` updates the custom attribute specified by `atrb` with properties specified by the name-value pairs `Name` and `Value` in the requirement set `rs`.

### Input Arguments

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

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

**`atrb` — Custom attribute name**
character array

Custom attribute name, specified as a character array.

#### 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: `'Description','My new description.'`

**`Description` — Custom attribute description**
character array

Custom attribute description, specified as the comma-separated pair consisting of `'Description'` and a character array.

Example: `'Description','My new description.'`

**`List` — Combobox list options**
cell array

`Combobox` list options, specified as the comma-separated pair consisting of `'List'` and a cell array. The list of options is valid only if `'Unset'` is the first entry. `'Unset'` indicates that the user hasn't chosen an option from the combo box. If the list does not start with `'Unset'`, it will be automatically appended as the first entry.

Example: `'List',{'Unset','A','B','C'}`

---

**Note** You can only use this name-value pair when the `Type` property of the custom attribute that you're updating is `Combobox`.

---

## Examples

### Update Requirement Set Custom Attribute Information

This example shows how to update custom attribute information for a requirement set.

Load `crs_req_func_spec`, which describes a cruise control system. Find a requirement set in the files and assign it to a variable.

```
slreq.load('crs_req_func_spec');
rs = slreq.find('Type','ReqSet');
```

#### Update an Edit Custom Attribute

Add an `Edit` custom attribute that has a description to the requirement set. Get the attribute information with `inspectAttribute`.

```
addAttribute(rs,'MyEditAttribute','Edit','Description','Original attribute.')
inspectAttribute(rs,'MyEditAttribute')
```

```
ans = struct with fields:
           name: 'MyEditAttribute'
           type: Edit
    description: 'Original attribute.'
```

Update the custom attribute with a new description. Confirm the change by getting the attribute information with `inspectAttribute`.

```
updateAttribute(rs,'MyEditAttribute','Description','Updated attribute.')
inspectAttribute(rs,'MyEditAttribute')
```

```
ans = struct with fields:
           name: 'MyEditAttribute'
           type: Edit
    description: 'Updated attribute.'
```

#### Update a Combobox Custom Attribute

Add a `Combobox` custom attribute that has a list of options to the requirement set. Get the attribute information with `inspectAttribute`.

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

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

Update the custom attribute with a new list of options. Confirm the change by getting the attribute information with `inspectAttribute`.

```
updateAttribute(rs,'MyCombobox','List',{'Unset','1','2','3'})
inspectAttribute(rs,'MyCombobox')
```

```
ans = struct with fields:
           name: 'MyCombobox'
           type: Combobox
    description: ''
           list: {'Unset'  '1'  '2'  '3'}
```

Update the custom attribute with a new list of options and a new description. Confirm the change by getting the attribute information with `inspectAttribute`.

```
updateAttribute(rs,'MyCombobox','List',{'Unset','A1','B2','B3'},'Description',...
    'Updated attribute with new options.')
inspectAttribute(rs,'MyCombobox')
```

```
ans = struct with fields:
           name: 'MyCombobox'
           type: Combobox
    description: 'Updated attribute with new options.'
           list: {'Unset'  'A1'  'B2'  'B3'}
```

### Cleanup

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

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

## See Also

`addAttribute` | `deleteAttribute` | `inspectAttribute` | `slreq.ReqSet`

### Topics

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

**Introduced in R2020b**

# 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

`remove` | `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**

# copy

**Class:** `slreq.Requirement`
**Package:** `slreq`

Copy and paste requirement

## Syntax

```
tf = copy(req1,location,req2)
```

## Description

`tf = copy(req1,location,req2)` copies requirement `req1` and pastes it under, before, or after requirement `req2` depending on the location specified by `location`. The function returns `1` if the copy and paste is performed successfully.

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

## Input Arguments

**`req1` — Requirement to copy**
`slreq.Requirement` object

Requirement to copy, specified as an `slreq.Requirement` object.

**`location` — Requirement paste location**
`'under'` | `'before'` | `'after'`

Paste location, specified as `'under'`, `'before'`, or `'after'`.

**`req2` — Requirement**
`slreq.Requirement` object

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

## Output Arguments

**`tf` — Paste success status**
`0` | `1`

Paste success status, returned as a `1` or `0` of data type `logical`.

## Examples

### Copy and Paste a Requirement

This example shows how to copy a requirement and paste it under, before, or after another requirement.

Load the `crs_req_func_spec` requirement file, which describes a cruise control system, and assign it to a variable. Find two requirements by index. The first requirement will be copied and pasted in relation to the second requirement.

```
rs = slreq.load('crs_req_func_spec');
req1 = find(rs,'Type','Requirement','Index','1');
req2 = find(rs,'Type','Requirement','Index','2');
```

#### Paste Under a Requirement

Copy and paste the first requirement, `req1`, under the second requirement, `req2`. The first requirement becomes the last child requirement of `req2`, which you can verify by finding children of `req2` and comparing the summary of the last child and `req1`.

```
tf = copy(req1,'under',req2);
```

```
Warning: Error occurred while executing the listener callback for event ReqDataChange defined fo
Dot indexing is not supported for variables of this type.

Error in slreq.das.ReqRoot/recAddDasObjctsIfNeeded

Error in slreq.das.ReqRoot/onReqDataChange

Error in slreq.das.ReqRoot>@(varargin)this.onReqDataChange(varargin{:})

Error in slreq.data.ReqData/pasteFromClipboard

Error in slreq.data.ReqData/copyRequirement

Error in slreq.BaseEditableItem/copy

Error in CopyAndPasteARequirementExample (line 4)
tf = copy(req1,'under',req2);

Error in matlab.internal.editor.evaluateRegions

Error in matlab.internal.editor.EvaluationOutputsService.evalRegions

Error in matlab.internal.liveeditor.LiveEditorUtilities.execute (line 43)
matlab.internal.editor.EvaluationOutputsService.evalRegions(editorId, uuid, regionDataList, full

Error in mwtools.liveCodeToDocbook>doRun (line 364)
    LiveEditorUtilities.execute(editorId, source);

Error in mwtools.liveCodeToDocbook>doRunConvert (line 314)
    [exampleTime,exampleWarnings, exampleErrors] = doRun(javaRichDocument, source);

Error in mwtools.liveCodeToDocbook (line 143)
        [exampleTime,exampleRunWarnings,exampleRunErrors] = doRunConvert(...

Error in BML (line 13)
        evalin('base', s);
```

```
childReqs = children(req2);
lastChild = childReqs(numel(childReqs));
lastChild.Summary

ans = 
'Driver Switch Request Handling'
```

```
req1.Summary

ans = 
'Driver Switch Request Handling'
```

### Paste Before a Requirement

Copy and paste the first requirement, `req1`, before the second requirement, `req2`. Confirm that the requirement was pasted before `req2` by checking the index and Summary. The old index of `req2` was 2. The index of the pasted requirement should be 2 and the index of `req2` should be 3.

```
tf = copy(req1,'before',req2);

Warning: Error occurred while executing the listener callback for event ReqDataChange defined fo
Index exceeds the number of array elements (2).

Error in slreq.das.BaseObject/insertChildObject

Error in slreq.das.ReqRoot/recAddDasObjctsIfNeeded

Error in slreq.das.ReqRoot/recAddDasObjctsIfNeeded

Error in slreq.das.ReqRoot/onReqDataChange

Error in slreq.das.ReqRoot>@(varargin)this.onReqDataChange(varargin{:})

Error in slreq.data.ReqData/copyRequirement

Error in slreq.BaseEditableItem/copy

Error in CopyAndPasteARequirementExample (line 9)
tf = copy(req1,'before',req2);

Error in matlab.internal.editor.evaluateRegions

Error in matlab.internal.editor.EvaluationOutputsService.evalRegions

Error in matlab.internal.liveeditor.LiveEditorUtilities.execute (line 43)
matlab.internal.editor.EvaluationOutputsService.evalRegions(editorId, uuid, regionDataList, full

Error in mwtools.liveCodeToDocbook>doRun (line 364)
    LiveEditorUtilities.execute(editorId, source);

Error in mwtools.liveCodeToDocbook>doRunConvert (line 314)
    [exampleTime,exampleWarnings, exampleErrors] = doRun(javaRichDocument, source);

Error in mwtools.liveCodeToDocbook (line 143)
        [exampleTime,exampleRunWarnings,exampleRunErrors] = doRunConvert(...

Error in BML (line 13)
        evalin('base', s);
```

```
pastedReq = find(rs,'Type','Requirement','Index','2');
pastedReq.Summary
```

```
ans = 
'Driver Switch Request Handling'
```

```
req2.Index
```

```
ans = 
'3'
```

### Paste After a Requirement

Copy and paste the first requirement, `req1`, after the second requirement, `req2`. Confirm that the requirement was pasted after `req2` by checking the index. The index of `req2` is `3` and should not change, which means the index of the pasted requirement should be `4`.

```
tf = copy(req1,'after',req2);
```

```
Warning: Error occurred while executing the listener callback for event ReqDataChange defined fo
Index exceeds the number of array elements (2).

Error in slreq.das.BaseObject/insertChildObject

Error in slreq.das.ReqRoot/recAddDasObjctsIfNeeded

Error in slreq.das.ReqRoot/recAddDasObjctsIfNeeded

Error in slreq.das.ReqRoot/onReqDataChange

Error in slreq.das.ReqRoot>@(varargin)this.onReqDataChange(varargin{:})

Error in slreq.data.ReqData/copyRequirement

Error in slreq.BaseEditableItem/copy

Error in CopyAndPasteARequirementExample (line 13)
tf = copy(req1,'after',req2);

Error in matlab.internal.editor.evaluateRegions

Error in matlab.internal.editor.EvaluationOutputsService.evalRegions

Error in matlab.internal.liveeditor.LiveEditorUtilities.execute (line 43)
matlab.internal.editor.EvaluationOutputsService.evalRegions(editorId, uuid, regionDataList, fullI

Error in mwtools.liveCodeToDocbook>doRun (line 364)
    LiveEditorUtilities.execute(editorId, source);

Error in mwtools.liveCodeToDocbook>doRunConvert (line 314)
    [exampleTime,exampleWarnings, exampleErrors] = doRun(javaRichDocument, source);

Error in mwtools.liveCodeToDocbook (line 143)
        [exampleTime,exampleRunWarnings,exampleRunErrors] = doRunConvert(...

Error in BML (line 13)
        evalin('base', s);
```

```
pastedReq2 = find(rs,'Type','Requirement','Index','4');
pastedReq2.Summary

ans = 
'Driver Switch Request Handling'

req2.Index

ans = 
'3'
```

### Cleanup

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

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

## See Also

move | moveDown | moveUp | slreq.Requirement

**Introduced in R2020b**

## 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 children of parent requirements

### Syntax

```
childReqs = find(req,'PropertyName1',PropertyValue1,...,'PropertyNameN',
PropertyValueN)
```

### Description

`childReqs = find(req,'PropertyName1',PropertyValue1,...,'PropertyNameN', PropertyValueN)` finds and returns child requirements `childReqs` of the parent requirement `req` that match the properties specified by `PropertyName` and `PropertyValue`.

### Input Arguments

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

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

**`PropertyName` — Requirement property**
character vector

Requirement property name, specified as a character vector. See the valid property names in the properties section of `slreq.Requirement`.

Example: `'Type'`,`'Keywords'`,`'SID'`

**`PropertyValue` — Requirement property value**
character vector | character array | `datetime` value | scalar | `logical` | structure array

Requirement property value, specified as a character vector, character array, `datetime` value, scalar, `logical`, or structure array. The data type depends on the specified `propertyName`. See the valid property values in the properties section of `slreq.Requirement`.

### Output Arguments

**`childReqs` — Child requirements**
`slreq.Requirement` object | `slreq.Requirement` object array

Child requirements, returned as `slreq.Requirement` objects.

### Examples

### Find Child Requirements

This example shows how to find child requirements that match property values.

Load the `crs_req_func_spec` requirement file, which describes a cruise control system, and assign it to a variable. Find the requirement with index `4`, as this requirement has child requirements.

```
rs = slreq.load('crs_req_func_spec');
parentReq = find(rs,'Type','Requirement','Index','4');
```

Find all the child requirements of `parentReq` that were modified in revision `1`.

```
childReqs1 = find(parentReq,'FileRevision',1)
```

```
childReqs1=1×10 object
  1x10 Requirement array with properties:

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

Find all the child requirements of `parentReq` that were modified in revision `1` and are `Functional` type requirements.

```
childReqs2 = find(parentReq,'FileRevision',1,'Type','Functional')
```

```
childReqs2=1×10 object
  1x10 Requirement array with properties:

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

#### Cleanup

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

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

## See Also

`slreq.ReqSet` | `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`

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

**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`.

#### `unexecued` — 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**

# move

**Class:** `slreq.Requirement`
**Package:** `slreq`

Move requirement in hierarchy

## Syntax

```
tf = move(req1,location,req2)
```

## Description

`tf = move(req1,location,req2)` moves requirement `req1` under, before, or after requirement `req2` depending on the location specified by `location`. The function returns `1` if the move is performed successfully.

## Input Arguments

**`req1` — Requirement**
`slreq.Requirement` object

Requirement to move, specified as an `slreq.Requirement` object.

**`location` — Requirement move location**
`'under'` | `'before'` | `'after'`

Requirement move location, specified as `'under'`, `'before'`, or `'after'`.

**`req2` — Requirement to move**
`slreq.Requirement` object

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

## Output Arguments

**`tf` — Paste success status**
`0` | `1`

Paste success status, returned as a `1` or `0` of data type `logical`.

## Examples

### Move a Requirement

This example shows how to move a requirement under, before, or after another requirement.

Load the `crs_req_func_spec` requirement file, which describes a cruise control system, and assign it to a variable. Find two requirements by index. The first requirement will be moved in relation to the second requirement.

```
rs = slreq.load('crs_req_func_spec');
req1 = find(rs,'Type','Requirement','Index','1');
req2 = find(rs,'Type','Requirement','Index','2');
```

**Move Under a Requirement**

Move the first requirement, `req1`, under the second requirement, `req2`. The first requirement becomes the last child requirement of requirement `req2`, and `req2` moves up one in the hierarchy, which you can verify by checking the index of `req1` and `req2`. The old indices of `req1` and `req2` were `1` and `2`, respectively.

```
tf = move(req1,'under',req2);
req1.Index
```

```
ans =
'1.3'
```

```
req2.Index
```

```
ans =
'1'
```

**Move Before a Requirement**

Move the first requirement, `req1`, before the second requirement, `req2`. Confirm that the requirement was moved correctly by checking the indices of `req1` and `req2`. The indices of `req1` and `req2` are now the same as they were originally: `1` and `2`, respectively.

```
tf = move(req1,'before',req2);
req1.Index
```

```
ans =
'1'
```

```
req2.Index
```

```
ans =
'2'
```

**Move After a Requirement**

Move the first requirement,`req1`, after the second requirement, `req2`. When you move requirement `req1` down in the hierarchy, requirement `req2` also moves up, which you can verify by checking the indices of `req1` and `req2`.

```
tf = move(req1,'after',req2);
req1.Index
```

```
ans =
'2'
```

```
req2.Index
```

```
ans =
'1'
```

### Cleanup

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

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

## See Also
`copy` | `moveDown` | `moveUp` | `slreq.Requirement`

**Introduced in R2020b**

# moveDown

**Class:** `slreq.Requirement`
**Package:** `slreq`

Move requirement down in hierarchy

## Syntax

```
tf = moveDown(req)
```

## Description

`tf = moveDown(req)` moves the requirement `req` down one spot in the hierarchy, and returns `1` if the move is successful. The requirement `req` cannot be moved to a new level in the hierarchy.

## Input Arguments

**req — Requirement**
`slreq.Requirement`

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

## Output Arguments

**tf — Paste success status**
`0 | 1`

Paste success status, returned as a `1` or `0` of data type `logical`.

## Examples

### Move a Requirement Down

This example shows how to move a requirement down in the hierarchy.

Load the `crs_req_func_spec` requirement file, which describes a cruise control system, and assign it to a variable. Find the requirement with index `3.1`.

```
rs = slreq.load('crs_req_func_spec');
req1 = find(rs,'Type','Requirement','Index','3.1');
```

Move the requirement down one spot in the hierarchy. Confirm the move by checking the success status, `tf1`, and the index.

```
tf1 = moveDown(req1)
```

```
tf1 = logical
   1
```

```
req1.Index
```

```
ans =
'3.2'
```

Find the requirement with index `3.4`. This requirement is already at the bottom of its level in the hierarchy and cannot be moved down further, which you can verify by trying to move it down. Confirm that the move failed by checking the success status, `tf2`, and the index.

```
req2 = find(rs,'Type','Requirement','Index','3.4');
tf2 = moveDown(req2)
```

```
tf2 = logical
   0
```

```
req2.Index
```

```
ans =
'3.4'
```

**Cleanup**

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

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

## See Also

`copy` | `move` | `moveUp` | `slreq.Requirement`

**Introduced in R2020b**

# moveUp

**Class:** `slreq.Requirement`
**Package:** `slreq`

Move requirement up in hierarchy

## Syntax

```
tf = moveUp(req)
```

## Description

`tf = moveUp(req)` moves the requirement `req` up one spot in the hierarchy, and returns `1` if the move is successful. The requirement `req` cannot be moved to a new level in the hierarchy.

## Input Arguments

**`req` — Requirement**
`slreq.Requirement`

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

## Output Arguments

**`tf` — Paste success status**
`0 | 1`

Paste success status, returned as a `1` or `0` of data type `logical`.

## Examples

### Move a Requirement Up

This example shows how to move a requirement up in the hierarchy.

Load the `crs_req_func_spec` requirement file, which describes a cruise control system, and assign it to a variable. Find the requirement with index `3.4`.

```
rs = slreq.load('crs_req_func_spec');
req1 = find(rs,'Type','Requirement','Index','3.4');
```

Move the requirement up one spot in the hierarchy. Confirm the move by checking the success status, `tf1`, and the index.

```
tf1 = moveUp(req1)
```

```
tf1 = logical
   1
```

```
req1.Index
```

```
ans = 
'3.3'
```

Find the requirement with index `3.1`. This requirement is already at the top of its level in the hierarchy and cannot be moved up further, which you can verify by trying to move it up. Confirm that the move failed by checking the success status, `tf2`, and the index.

```
req2 = find(rs,'Type','Requirement','Index','3.1');
tf2 = moveUp(req2)
```

```
tf2 = logical
   0
```

```
req2.Index
```

```
ans = 
'3.1'
```

### Cleanup

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

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

## See Also

`copy` | `move` | `moveDown` | `slreq.Requirement`

**Introduced in R2020b**

# 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**

# remove

**Class:** `slreq.Requirement`
**Package:** `slreq`

Remove requirement from requirement set

## Syntax

```
count = remove(req)
count = remove(parentReq,'PropertyName1',PropertyValue1,...,'PropertyNameN',
PropertyValueN)
```

## Description

`count = remove(req)` removes the requirement `req` and returns the number of requirements deleted. If `req` has child requirements, they are also deleted.

`count = remove(parentReq,'PropertyName1',PropertyValue1,...,'PropertyNameN',
PropertyValueN)` removes child requirements of `parentReq` that match the properties specified by `PropertyName` and `PropertyValue`. The function returns the number of requirements deleted. The parent requirement is not removed.

**Note** When you remove a requirement, the variable corresponding to the removed `slreq.Requirement` object remains in the workspace but is no longer a valid `slreq.Requirement` object.

## Input Arguments

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

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

**parentReq — Parent requirement**
`slreq.Requirement` object

Parent requirement, specified as an `slreq.Requirement` object.

**PropertyName — Requirement property**
character vector

Requirement property name, specified as a character vector. See the valid property names in the properties section of `slreq.Requirement`.

Example: `'Type', 'Id', 'Keywords'`

**PropertyValue — Requirement property value**
character vector | character array | `datetime` value | scalar | `logical` | structure array

Requirement property value, specified as a character vector, character array, `datetime` value, scalar, `logical`, or structure array. The value depends on the specified `propertyName`. See the valid property values in the properties section of `slreq.Requirement`.

Example: `'Functional', '1.1.1', 'Design'`

## Output Arguments

**`count` — Removed requirements count**
`double`

Total number of requirements that were removed, returned as a `double`.

## Examples

### Remove a Single Requirement

This example shows how to find and remove a single requirement.

Load a requirement set file. Find a requirement in the requirement set by using the ID number, then remove it.

```
rs = slreq.load('crs_req_func_spec.slreqx');
req = find(rs,'Type','Requirement','ID','#2');
count = remove(req)
```

```
count = 1
```

### Cleanup

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

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

### Remove a Parent Requirement

This example shows how to remove a parent requirement and its children.

Load a requirement set and find a parent requirement by using the ID number. Confirm that it is a parent requirement by checking if it has children, then remove the requirement. When you remove a parent requirement, the children are also removed.

```
rs = slreq.load('crs_req_func_spec.slreqx');
parentReq1 = find(rs,'Type','Requirement','ID','#24');
childReqs1 = children(parentReq1)
```

```
childReqs1=1×12 object
  1x12 Requirement array with properties:

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

```
count2 = remove(parentReq1)
```

```
count2 = 13
```

#### Cleanup

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

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

### Remove Requirements that Match Property Types

This example shows how to remove child requirements that match a property type, and how to automate the process of removing all requirements with a matching property type.

#### Remove Child Requirements that Match Property Types

Load a requirement set file and find a parent requirement by using the ID number.

```
rs = slreq.load('crs_req_func_spec.slreqx');
parentReq = find(rs,'Type','Requirement','ID','#63');
```

Confirm that the requirement is a parent requirement by checking if it has children, and remove child requirements that match that revision number.

```
childReqs = children(parentReq)
```

```
childReqs=1×7 object
  1x7 Requirement array with properties:

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

```
count1 = remove(parentReq,'FileRevision',54)
```

```
count1 = 4
```

**Remove Multiple Requirements that Match Property Types**

Create a requirements array by finding all requirements in the requirement set that were modified in revision 18.

```
reqs = find(rs,'Type','Requirement','FileRevision',18);
```

Initialize the count variable, then loop through the requirements array and delete all of the requirements. Increment the count variable each time a requirement is deleted, then display the total number of requirements deleted.

```
count2 = 0;
for i = 1:numel(reqs)
    count2 = count2 + remove(reqs(i));
end
count2
```

```
count2 = 4
```

**Cleanup**

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

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

## See Also

add | slreq.Requirement | slreq.find

**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`

**Topics**

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

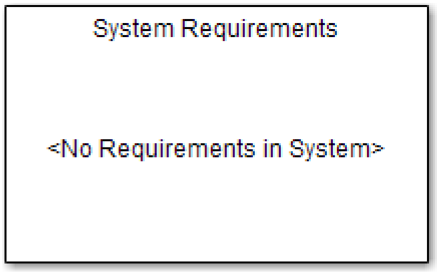
**Introduced in R2018a**

4

# Blocks

## System Requirements

List system requirements in Simulink models



## Library

Simulink Requirements

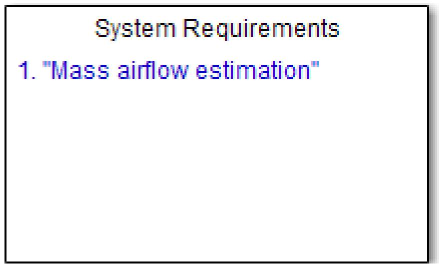
### Description

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 list desired requirement links 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**