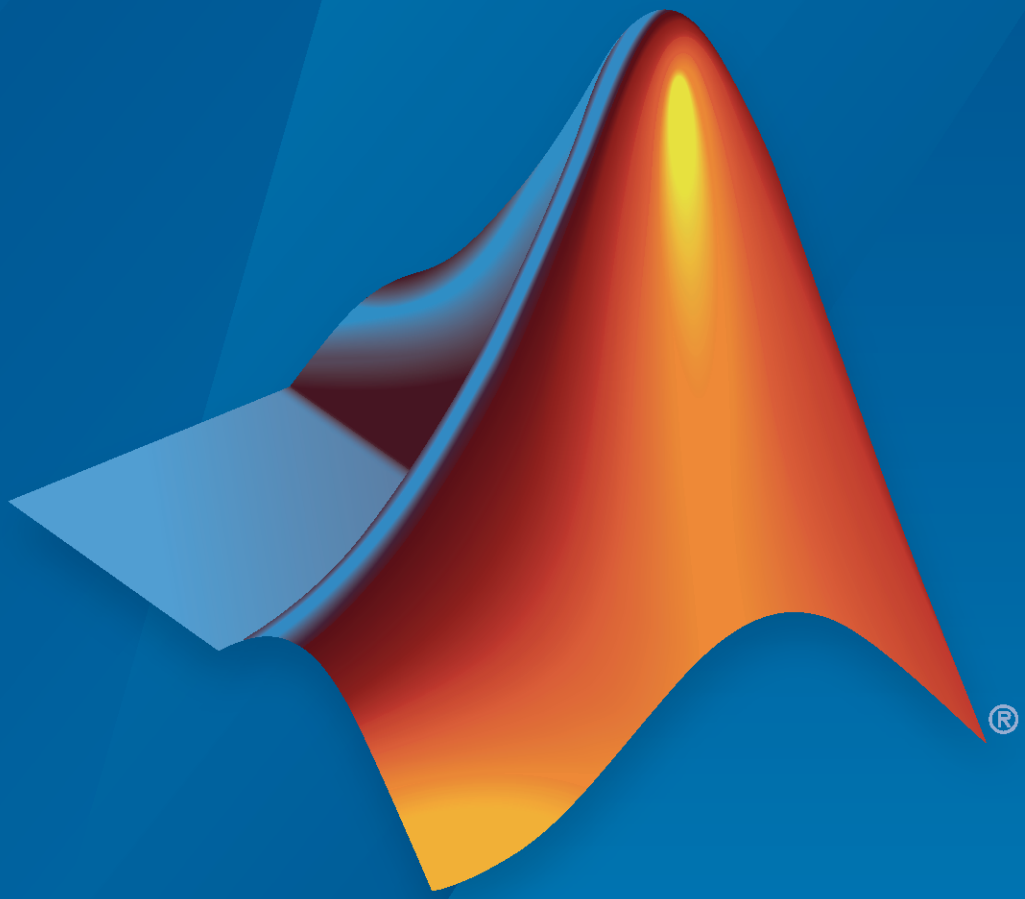
Requirements Toolbox™

Reference



MATLAB® & SIMULINK®

R2022a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

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

Phone: 508-647-7000

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

Requirements Toolbox™ 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)
March 2021	Online only	Revised for Version 1.7 (Release 2021a)
September 2021	Online only	Revised for Version 1.8 (Release 2021b)
March 2022	Online only	Revised for Version 2.0 (Release 2022a)

Contents

Functions

addAssumptionRow

Package: `slreq.modeling`

Add assumption to Requirements Table block

Syntax

```
AssumptionRow = addAssumptionRow(reqTable)
AssumptionRow = addAssumptionRow(reqTable,Name=Value)
```

Description

`AssumptionRow = addAssumptionRow(reqTable)` adds an assumption to the Requirements Table block, specified by `reqTable`.

`AssumptionRow = addAssumptionRow(reqTable,Name=Value)` adds an assumption by using one or more name-value arguments.

Examples

Add an Assumption to a Requirement Table Block

Create a Requirements Table block and retrieve the `RequirementsTable` object.

```
table = slreq.modeling.create("myModel");
```

Add an assumption to the block.

```
row = addAssumptionRow(table);
```

Add an Assumption with a Precondition and Postcondition

Create a Requirements Table block and retrieve the `RequirementsTable` object.

```
table = slreq.modeling.create("myModel");
```

Add an assumption to the block with expressions in the **Precondition** and **Postcondition** columns.

```
row = addAssumptionRow(table, Preconditions={'u1 > 1'},...
Postcoditions={'y1 > 0'});
```

Input Arguments

`reqTable` — Requirements Table block

`RequirementsTable` object

Requirements Table block, specified as a `RequirementsTable` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `addAssumptionRow(table,rowType="row",Preconditions={'u1 > 0'})` adds an assumption to a Requirements Table block with a precondition `u1 > 0`.

`Preconditions` — Precondition expression

`{''}` (default) | cell array of character vectors

Precondition expression, specified as a cell array of character vectors. For more information on preconditions in assumptions, see “Add Assumptions to Requirements”.

Data Types: `char` | `cell`

`Postconditions` — Postcondition expression

`{''}` (default) | cell array of character vectors

Postcondition expression, specified as a cell array of character vectors. For more information on postconditions in assumptions, see “Add Assumptions to Requirements”.

Data Types: `char` | `cell`

`rowType` — Assumption type

`"normal"` (default) | `"default"` | `"anyChildActive"` | `"allChildrenActive"`

Assumption type, specified by one of these values:

Value	Description
`"normal"`	Creates a normal assumption with all of the available properties.
`"default"`	Creates a default semantic assumption. Default assumptions cannot have a precondition.
`"anyChildActive"`	Creates an Any Child Active semantic assumption. The parent assumption cannot have a precondition, and the children cannot have a postcondition.
`"allChildrenActive"`	Creates an All Child Active semantic assumption. The parent assumption cannot have a precondition, and the children cannot have a postcondition.

You can create normal assumptions or semantic assumptions. For more information on semantic requirements and assumptions, see “Add Semantic Rows”. If you do not include this name-value pair, the function creates a normal assumption.

Data Types: `enumerated`

`Summary` — Assumption summary text

`""` (default) | string scalar | character vector

Assumption summary text, specified as a string scalar or character vector. Use this name-value argument to add text to the **Summary** column in the **Assumptions** tab of the Requirements Table block.

Data Types: `char` | `string`

Output Arguments

`AssumptionRow` — Assumption
`AssumptionRow` object

Assumption, returned as an `AssumptionRow` object.

See Also

Blocks
Requirements Table

Functions
`addRequirementRow` | `getAssumptionRows`

Objects
`RequirementsTable` | `AssumptionRow`

Topics
“Establish Hierarchy in Requirements Table Blocks”

Introduced in R2022a

addChild

Package: `slreq.modeling`

Add child requirement or assumption to Requirements Table block

Syntax

```
newChild = addChild(row)
newChild = addChild(row,Name=Value)
```

Description

`newChild = addChild(row)` adds a child row to the requirement or assumption specified by `row`.

`newChild = addChild(row,Name=Value)` adds a child row using one or more name-value arguments. The available name-value arguments depend on whether `row` is a requirement or assumption.

Examples

Add a Child Requirement to a Requirement Table Block

Create a Requirements Table block and retrieve the `RequirementsTable` object.

```
table = slreq.modeling.create("myModel");
```

New Requirements Table blocks start with one requirement. Find the `RequirementRow` object that corresponds to the requirement by using the `getRequirementRows` function.

```
row = getRequirementRows(table);
```

Add a child to the requirement.

```
childReq = addChild(row);
```

Add a Child Assumption with a Precondition and Postcondition

Create a Requirements Table block and retrieve the `RequirementsTable` object.

```
table = slreq.modeling.create("myModel");
```

Add an assumption to the block by using the `addAssumptionRow` function.

```
row = addAssumptionRow(table);
```

Add a child with expressions in the **Precondition** and **Postcondition** columns to the assumption.

```
child = addChild(row,Preconditions={'u1 > 1'},...
Postcoditions={'y1 > 0'});
```

Input Arguments

`row` — Requirement or assumption

`RequirementRow` object | `AssumptionRow` object

Requirement or assumption in a Requirements Table block, specified as a `RequirementRow` or `AssumptionRow` object. To retrieve the row, use `getRequirementRows`, `getAssumptionRows`, or `getChildren`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `newChild = addChild(row,Preconditions={'u1 > 1'},Duration="5")` returns a child requirement from the `RequirementRow` object `row` that has the precondition `u1 > 1` and a duration equal to `5`. This example produces an error if `row` is a `AssumptionRow`, because assumptions do not have a duration property.

`Actions` — Action expressions

`{''}` (default) | cell array of character vectors

Action expressions, specified as a cell array of character vectors. You can only specify this property if `row` is a `RequirementRow` object. For more information on actions, see “Use a Requirements Table Block to Create Formal Requirements”.

Data Types: `cell` | `char`

`Duration` — Duration expression

`""` (default) | string scalar | character vector

Duration expression, specified as a string scalar or character vector. You can only specify this property if `row` is a `RequirementRow` object. For more information on the duration, see “Control Requirement Execution by Using Temporal Logic”.

Data Types: `char` | `string`

`Preconditions` — Precondition expressions

`{''}` (default) | cell array of character vectors

Precondition expressions, specified as a cell array of character vectors. If `row` is an assumption, you can specify only one precondition per child. For more information on preconditions, see “Use a Requirements Table Block to Create Formal Requirements”.

Data Types: `cell` | `char`

`Postconditions` — Postcondition expression

`{''}` (default) | cell array of character vectors

Postcondition expressions, specified as a cell array of character vectors. If `row` is an assumption, you can specify only one postcondition per child. For more information on postconditions, see “Use a Requirements Table Block to Create Formal Requirements”.

Data Types: `cell` | `char`

`rowType` — Row type
`"row"` (default) | `"defaultRow"` | `"anyChildActive"` | `"allChildrenActive"`

Row type, specified as one of these values:

Value	Description
`"row"`	Creates a normal child row with all of the available properties.
`"defaultRow"`	Creates a default semantic child row. Default rows cannot have a precondition.
`"anyChildActive"`	Creates a semantic child row where any of the child rows can be active. The children of the added row cannot have postconditions or actions, and the added row cannot have preconditions. See “Add Semantic Rows”.
`"allChildrenActive"`	Creates a semantic child row where all of the child rows must be active. The children of the added row cannot have postconditions or actions, and the added row cannot have preconditions. See “Add Semantic Rows”.

If you do not include this name-value pair, the function creates a normal row.

Data Types: `enumerated`

`Summary` — Child row summary text
`""` (default) | string scalar | character vector

Child row summary text, specified as a string scalar or character vector. Use this name-value argument to add text to the **Summary** column in the **Requirements** or **Assumptions** tabs of the Requirements Table block.

Data Types: `char` | `string`

Output Arguments

`newChild` — Child requirement or assumption
`RequirementRow` or `AssumptionRow` object

Child requirement or assumption, returned as the same object type specified by the input argument `row`. For example, if `row` is a `RequirementRow`, `newChild` is a `RequirementRow`. For more information on requirement hierarchies in Requirements Table blocks, see “Establish Hierarchy in Requirements Table Blocks”.

See Also

`RequirementRow` | `AssumptionRow` | `RequirementsTable`

Topics
“Define Requirements Hierarchy”
“Establish Hierarchy in Requirements Table Blocks”

Introduced in R2022a

addLink

Add link to local OSLC requirement resource object

Syntax

`addLink(reqResource,resource)`

Description

`addLink(reqResource,resource)` adds an RDF/XML element to the requirement or requirement collection resource specified by `reqResource`. The function sets the element name to `j.0:Link` and the `rdf:resource` attribute to the resource URL associated with `resource`. Use the `commit` function to apply the change to the service provider. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website and QM Resource Definitions on the Open Services for Lifecycle Collaboration (OSLC) website.

Examples

Add and Remove Links from OSLC Resources to Requirement

This example shows how to add and remove links from OSLC resources to an OSLC requirement.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type. Submit a query request to the service provider for the available requirement resources.

```
myQueryCapability = getQueryService(myClient,'Requirement');
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign one of the requirements to a variable called `myReq` and one to `linkReq`. Fetch the full resource properties for the requirements.

```
myReq = reqs(1);
linkReq = reqs(5);
fetch(myReq,myClient);
fetch(linkReq,myClient);
```

Add a link from `linkReq` to `myReq`. Confirm the link creation by getting the links for `myReq`.

```
addLink(myReq,linkReq)
links = getLinks(myReq)
```

```
links =

  1×1 cell array

    {'https://localhost:9443/rm/CA_3d5ba3752e2c489b965a3ecceffb664a'}
```

In the service provider, identify a test case to link to the requirement. Identify the resource URL of the test case and assign it to a variable called `URL`. Add a link from `URL` to `myReq`. Confirm the link creation by getting the links for `myReq`.

```
URL = 'https://localhost:9443/qm/_ibz6tGWYEeuAF8ZpKyQQtg';
addLink(myReq,URL)
links = getLinks(myReq)
```

```
links =

  1×2 cell array

    {'https://localhost:9443/rm...'}    {'https://localhost:9443/qm...'}
```

Commit the changes to the service provider.

```
status = commit(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Fetch the full resource properties for the updated requirement `myReq`.

```
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Get the resource URLs linked to `myReq`.

```
links = getLinks(myReq)
```

```
links =

  1×2 cell array

    {'https://localhost:9443/rm...'}    {'https://localhost:9443/qm...'}
```

Get the URL for the first linked resource and assign it to `URL`.

```
URL = links{1}
```

```
URL =

    'https://localhost:9443/rm/CA_3d5ba3752e2c489b965a3ecceffb664a'
```

Before removing the link from `myReq`, confirm that the resource URL points to the requirement that you want to remove. Create a requirement resource object and set the resource URL. Fetch the full resource properties for the requirement and inspect the requirement.

```
req = oslc.rm.Requirement;
setResourceUrl(req,URL);
status = fetch(req,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
req
```

```
ans =

  Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/CA_3d5ba3752e2c489b965a...'
          Dirty: 0
      IsFetched: 1
          Title: '[SAFe] Lifecycle Scenario Template'
     Identifier: '1165'
```

Remove the link from `myReq` and commit the changes to the service provider.

```
removeLink(myReq,URL)
status = commit(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Fetch the full resource properties for the updated requirement `myReq`.

```
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Verify the link removal by getting the URLs for the resources linked to `myReq`.

```
links = getLinks(myReq)
```

```
links =

  1×1 cell array
```

```
{'https://localhost:9443/qm/_ibz6tGWYEeuAF8ZpKyQQtg'}
```

Input Arguments

`reqResource` — OSLC requirement resource

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object

OSLC requirement or requirement collection resource object, specified as an `oslc.rm.Requirement` or `oslc.rm.RequirementCollection` object.

`resource` — OSLC resource URL or object

character vector | `oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource URL, specified as a character vector or OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

Tips

- You can also add a link with `addResourceProperty` to specify the relationship of the link.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `removeLink` | `getLinks` | `addRequirementLink`

Introduced in R2021a

addRequirementLink

Add requirement traceability link to local OSLC test resource object

Syntax

```
addRequirementLink(testResource,requirementURL)
```

Description

`addRequirementLink(testResource,requirementURL)` adds an RDF/XML element to the test case or test script resource specified by `testResource`. The function sets the element name to `oslc_qm:validatesRequirement` and the `rdf:resource` attribute to `requirementURL`. Use the `commit` function to apply the change to the service provider. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website and QM Resource Definitions on the Open Services for Lifecycle Collaboration (OSLC) website.

Examples

Add, Get, and Remove Traceability Links from a Test Case to a Requirement

This example shows how to add, remove, and get OSLC requirement resources linked to a test case resource with a previously configured OSLC client.

After you have created and configured an OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test case resource type.

```
myQueryCapability = getQueryService(myClient,'TestCase');
```

Submit a query request to the service provider for the available test case resources.

```
testCases = queryTestCases(myQueryCapability)
```

```
testCases =

  1×5 TestCase array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Retrieve the requirement resources linked to one of the test cases. Fetch the resource properties from the service provider for the test case.

```
myTestCase = testCases(1);
fetch(myTestCase,myClient);
reqs = getRequirementLinks(myTestCase)
```

```
reqs =
```

```
    Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/resources/_aQ1gRg8bEeuLWbFe'
          Dirty: 1
      IsFetched: 0
          Title: ''
     Identifier: ''
```

Remove the existing link to the requirement resource from the test case resource. Commit the changes to the service provider.

```
removeRequirementLink(myTestCase,reqs.ResourceUrl);
status = commit(myTestCase,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

To add a link to a requirement, in the OSLC service provider, locate the requirement resource that you want to link to the test case resource. Identify the resource URL. Create a variable `URL` and set the value of the variable to the requirement URL that you found in the service provider.

```
URL = 'https://localhost:9443/rm/resources/_oJNtgWrqEeup0a6t';
```

Create a traceability link between the requirement resource and the test case. Commit the change to the service provider.

```
addRequirementLink(myTestCase,URL);
status = commit(myTestCase,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

View the test case in the system browser.

```
show(myTestCase)
```

Input Arguments

`testResource` — OSLC test resource
`oslc.qm.TestCase` object | `oslc.qm.TestScript` object

OSLC test resource, specified as an `oslc.qm.TestCase` or `oslc.qm.TestScript` object.

`requirementURL` — Requirement resource URL
character vector

Requirement or requirement collection resource URL, specified as a character vector.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.qm.TestCase` | `oslc.qm.TestScript` | `oslc.rm.RequirementCollection` | `getRequirementLinks` | `removeRequirementLink`

Introduced in R2021a

addRequirementRow

Package: `slreq.modeling`

Add requirement to Requirements Table block

Syntax

```
RequirementRow = addRequirementRow(reqTable)
RequirementRow = addRequirementRow(reqTable,Name=Value)
```

Description

`RequirementRow = addRequirementRow(reqTable)` adds a requirement to the Requirements Table block specified by `reqTable`.

`RequirementRow = addRequirementRow(reqTable,Name=Value)` adds a requirement using one or more name-value arguments.

Examples

Add a Requirement to a Requirements Table Block

Create a Requirements Table block and retrieve the `RequirementsTable` object.

```
table = slreq.modeling.create("myModel");
```

Add a requirement to the block.

```
row = addRequirementRow(table);
```

Add a Requirement with Preconditions, Postconditions, and Actions

Create a Requirements Table block and retrieve the `RequirementsTable` object.

```
table = slreq.modeling.create("myModel");
```

Add a requirement to the block with expressions in the **Precondition** and **Postcondition** columns.

```
row = addRequirementRow(table,Preconditions={'u1 > 1'},...
Postcoditions={'y1 > 0'},Actions={'y2 = 1'});
```

Input Arguments

`reqTable` — Requirements Table block

`RequirementsTable` object

Requirements Table block, specified as a `RequirementsTable` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `addRequirementRow(table,Preconditions={'u1 > 0'})` adds a requirement to a Requirements Table block with a precondition `u1 > 0`.

`Actions` — Action expressions

`{''}` (default) | cell array of character vectors

Action expressions, specified as a cell array of character vectors. For more information on actions, see “Use a Requirements Table Block to Create Formal Requirements”.

Data Types: `char` | `cell`

`Duration` — Duration expression

`""` (default) | string scalar | character vector

Duration expression, specified as a string scalar or character vector. For more information on the duration, see “Control Requirement Execution by Using Temporal Logic”.

Data Types: `char` | `string`

`Preconditions` — Precondition expressions

`{''}` (default) | cell array of character vectors

Precondition expressions, specified as a cell array of character vectors. For more information on preconditions, see “Use a Requirements Table Block to Create Formal Requirements”.

Example: `addRequirementRow(table,Preconditions={'u1 > 0','','u3 > 0'})` adds a requirement to a Requirements Table block with `u1 > 0` in the first **Precondition** column, nothing in the second **Precondition** column, and `u3 > 0` in the third **Precondition** column.

Data Types: `char` | `cell`

`Postconditions` — Postcondition expressions

`{''}` (default) | cell array of character vectors

Postcondition expressions, entered as a string array or cell array of character vectors. For more information on postconditions, see “Use a Requirements Table Block to Create Formal Requirements”.

Example: `addRequirementRow(table,Postconditions={'u1 > 0','','u3 > 0'})` adds a requirement to a Requirements Table block with `u1 > 0` in the first **Postcondition** column, nothing in the second **Postcondition** column, and `u3 > 0` in the third **Postcondition** column.

Data Types: `char` | `cell`

`rowType` — Requirement type

`"normal"` (default) | `"default"` | `"anyChildActive"` | `"allChildrenActive"`

Requirement type, specified by one of these values:

Value	Description
`"normal"`	Creates a normal requirement with all of the available properties.
`"default"`	Creates a default semantic requirement. Default requirements cannot have preconditions.
`"anyChildActive"`	Creates an Any Child Active semantic requirement. The parent requirement cannot have preconditions, and the children cannot have postconditions or actions.
`"allChildrenActive"`	Creates an All Child Active semantic requirement. The parent requirement cannot have preconditions, and the children cannot have postconditions or actions.

You can create normal requirements or semantic requirements. For more information on semantic requirements and assumptions, see “Add Semantic Rows”. If you do not include this name-value pair, the function creates a normal requirement.

Data Types: `enumerated`

`Summary` — Requirement summary text
`""` (default) | string scalar | character vector

Requirement summary text, specified as a string scalar or character vector. Use this name-value argument to add text to the **Summary** column in the **Requirements** tab of the Requirements Table block.

Data Types: `char` | `string`

Output Arguments

`RequirementRow` — Requirement
`RequirementRow` object

Requirement, returned as a `RequirementRow` object.

See Also

Blocks
Requirements Table

Functions
`addAssumptionRow` | `getRequirementRows`

Objects
`RequirementsTable` | `RequirementRow`

Topics
“Establish Hierarchy in Requirements Table Blocks”
“Leverage Evaluation Order of Formal Requirements”

Introduced in R2022a

addResourceProperty

Add resource property to local OSLC resource object

Syntax

```
addResourceProperty(resource,propertyName,resourceURL)
```

Description

`addResourceProperty(resource,propertyName,resourceURL)` adds a new element to the locally stored RDF/XML data for the Open Services for Lifecycle Collaboration (OSLC) resource specified by `resource`. The function sets the element name to `propertyName` and sets the `rdf:resource` attribute of the element to `resourceURL`. Use the `commit` function to apply the change to the service provider. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website.

Examples

Add, Get, and Remove Properties from OSLC Resources

This example shows how to add, get, and remove properties from an existing OSLC requirement resource.

Create and configure the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3. Then query the service provider for requirements and assign an `oslc.rm.Requirement` object to the variable `myReq` as described in “Submit a Query Request with Query Capability” on page 1-198.

Retrieve the full resource data from the service provider for the requirement resource `myReq`.

```
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

The requirement `myReq` has a linked requirement with an `implementedBy` relationship. Get the `rdf:resource` value for the `oslc_rm:implementedBy` property for the requirement resource `myReq`.

```
linkedReq = getResourceProperty(myReq,'oslc_rm:implementedBy')
```

```
linkedReq =

  1×1 cell array

    {'https://localhost:9443/rm/resources/_72lxMWJREeup0...'}
```

Change the relationship between the linked requirement and `myReq` from `implementedBy` to `decomposedBy`. Remove the `oslc_rm:implementedBy` property and add an `oslc_rm:decomposedBy` property.

```
removeResourceProperty(myReq,'oslc_rm:implementedBy',linkedReq)
addResourceProperty(myReq,'oslc_rm:decomposedBy',linkedReq)
```

Get the text contents for the `dcterms:title` property.

```
title = getProperty(myReq,'dcterms:title')
```

```
title =

    'My New Requirement'
```

Change the title to `My New Requirement (Edited)`. Confirm the changes.

```
setProperty(myReq,'dcterms:title','My New Requirement (Edited)')
title = getProperty(myReq,'dcterms:title')
```

```
title =

    'My New Requirement (Edited)'
```

Add a new text property to the requirement with the tag `dcterms:description`. Confirm the changes.

```
addTextProperty(myReq,'dcterms:description', ...
    'My new requirement edited using the MATLAB OSLC client.');
desc = getProperty(myReq,'dcterms:description')
```

```
desc =

    'My new requirement created using the MATLAB OSLC client.'
```

Commit the changes to the service provider.

```
status = commit(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

View the resource that you edited in the system browser.

```
show(myReq)
```

Input Arguments

`resource` — OSLC resource object

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

`propertyName` — OSLC resource property name
character vector

OSLC resource property name, specified as a character vector.

`resourceURL` — OSLC resource URL
character vector

OSLC resource URL, specified as a character vector.

Tips

- For information about OSLC resource properties, see these pages on the OSLC website:
  - RM Resource Definitions
  - QM Resource Definitions
  - CM Resource Definitions

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `addTextProperty` | `getResourceProperty` | `removeResourceProperty`

External Websites
RDF 1.1 XML Syntax

Introduced in R2021a

addSymbol

Package: `slreq.modeling`

Add data to Requirements Table block

Syntax

```
data = addSymbol(reqTable)
data = addSymbol(reqTable,Name=Value)
```

Description

`data = addSymbol(reqTable)` adds data to the Requirements Table block, specified by `reqTable`.

`data = addSymbol(reqTable,Name=Value)` adds data by using one or more name-value arguments.

Examples

Add Data to a Requirement Table Block

Create a Requirements Table block and retrieve the `RequirementsTable` object.

```
table = slreq.modeling.create("myModel");
```

Add data to the block.

```
data = addSymbol(table);
```

Add Data with Specified Name, Scope, and Type Properties

Create a Requirements Table block and retrieve the `RequirementsTable` object.

```
table = slreq.modeling.create("myModel");
```

Add data to the block and specify the **Name**, **Scope**, and **Type** properties.

```
data = addSymbol(table,Name="u1",Scope="Output",Type="Single");
```

Input Arguments

`reqTable` — Requirements Table block

`RequirementsTable` object

Requirements Table block, specified as a `RequirementsTable` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `addSymbol(table,Complexity="Off")` creates data and sets the complexity of the data to `Off`.

`Complexity` — Whether data accepts complex values

`"Off"` (default) | `"On"` | `"Inherited"`

Whether the data accepts complex values, specified as one of these values:

Complexity	Description
`"Inherited"`	The data inherits complexity based on the `Scope` property. Input and output data inherit complexity from the Simulink® signals connected to the associated input and output ports. Local and parameter data inherit complexity from the parameter to which the data is bound.
`"Off"`	The data is a real number.
`"On"`	The data is a complex number.

Data Types: `enumerated`

`isDesignOutput` — Whether data is design model output

`false` or `0` (default) | `true` or `1`

Whether the data is a design model output, specified as a numeric or logical `1` (`true`) or `0` (`false`). This property applies only when the `Scope` property is `Input`. For more information, see “Treat as design model output for analysis”.

Data Types: `logical`

`Name` — Name of data

`"data"` (default) | string scalar | character vector

Name of the data, specified as a string scalar or character vector.

Data Types: `char` | `string`

`Scope` — Scope of data

`"Input"` (default) | `"Output"` | `"Local"` | `"Constant"` | `"Parameter"`

Scope of the data that specifies where the data resides in memory relative to the block, specified as one of these values:

Scope	Description
`"Input"`	The data is an input signal to a Requirements Table block.
`"Output"`	The data is an output signal of a Requirements Table block.

Scope	Description
`"Local"`	The data is defined in the current block only.
`"Constant"`	The data is a read-only constant value that is visible to the block.
`"Parameter"`	The data resides in a variable of the same name in the MATLAB® workspace, the model workspace, or in the workspace of a masked subsystem that contains this block.

Data Types: `enumerated`

`Size` — Size of data
`"-1"` (default) | string scalar | character vector

Size of the data, specified as a string scalar or character vector. This property must resolve to a scalar value or a MATLAB vector of values. The default value is `"-1"`, which means that the size is inherited. For more information, see “Inherit Size from Simulink” (Simulink).

Data Types: `char` | `string`

`Type` — Data type
`"Inherit: Same as Simulink"` (default) | `"double"` | `"single"` | `"int8"` | ...

Data type, specified as:

- `"Inherit: Same as Simulink"`
- `"double"`
- `"single"`
- `"half"`
- `"int64"`
- `"int32"`
- `"int16"`
- `"int8"`
- `"uint64"`
- `"uint32"`
- `"uint16"`
- `"uint8"`
- `"boolean"`
- `"fixdt(1,16,0)"`
- `"fixdt(1,16,2^0,0)"`
- `"Enum: <class name>"`
- `"Bus: <object name>"`

To modify the data type properties, use the **Symbols** pane and Property Inspector. For more information, see “Set Data Types in Requirements Table Blocks”.

Data Types: `enumerated`

Output Arguments

data — Data
`Symbol` object

Data, returned as a `Symbol` object.

See Also

Objects
`Symbol` | `RequirementsTable`

Functions
`findSymbol`

Topics
“Use a Requirements Table Block to Create Formal Requirements”
“Define Data in Requirements Table Blocks”

Introduced in R2022a

addTextProperty

Add text property to local OSLC resource object

Syntax

```
addTextProperty(resource,propertyName,textContents)
```

Description

`addTextProperty(resource,propertyName,textContents)` adds a new element to the locally stored RDF/XML data for the Open Services for Lifecycle Collaboration (OSLC) resource specified by `resource`. The function sets the element name to `propertyName` and sets the text contents of the element to `textContents`. Use the `commit` function to apply the change to the service provider. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website.

Examples

Add, Get, and Remove Properties from OSLC Resources

This example shows how to add, get, and remove properties from an existing OSLC requirement resource.

Create and configure the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3. Then query the service provider for requirements and assign an `oslc.rm.Requirement` object to the variable `myReq` as described in “Submit a Query Request with Query Capability” on page 1-198.

Retrieve the full resource data from the service provider for the requirement resource `myReq`.

```
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

The requirement `myReq` has a linked requirement with an `implementedBy` relationship. Get the `rdf:resource` value for the `oslc_rm:implementedBy` property for the requirement resource `myReq`.

```
linkedReq = getResourceProperty(myReq,'oslc_rm:implementedBy')
```

```
linkedReq =

  1×1 cell array

    {'https://localhost:9443/rm/resources/_72lxMWJREeup0...'}
```

Change the relationship between the linked requirement and `myReq` from `implementedBy` to `decomposedBy`. Remove the `oslc_rm:implementedBy` property and add an `oslc_rm:decomposedBy` property.

```
removeResourceProperty(myReq,'oslc_rm:implementedBy',linkedReq)
addResourceProperty(myReq,'oslc_rm:decomposedBy',linkedReq)
```

Get the text contents for the `dcterms:title` property.

```
title = getProperty(myReq,'dcterms:title')
```

```
title =

    'My New Requirement'
```

Change the title to `My New Requirement (Edited)`. Confirm the changes.

```
setProperty(myReq,'dcterms:title','My New Requirement (Edited)')
title = getProperty(myReq,'dcterms:title')
```

```
title =

    'My New Requirement (Edited)'
```

Add a new text property to the requirement with the tag `dcterms:description`. Confirm the changes.

```
addTextProperty(myReq,'dcterms:description', ...
    'My new requirement edited using the MATLAB OSLC client.');
desc = getProperty(myReq,'dcterms:description')
```

```
desc =

    'My new requirement created using the MATLAB OSLC client.'
```

Commit the changes to the service provider.

```
status = commit(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

View the resource that you edited in the system browser.

```
show(myReq)
```

Input Arguments

`resource` — OSLC resource object

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

`propertyName` — OSLC resource property name
character vector

OSLC resource property name, specified as a character vector.

`textContents` — OSLC resource text contents
character vector

OSLC resource text content, specified as a character vector.

Tips

- For information about OSLC resource properties, see these pages on the OSLC website:
  - RM Resource Definitions
  - QM Resource Definitions
  - CM Resource Definitions

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `addResourceProperty` | `getProperty` | `setProperty`

External Websites
RDF 1.1 XML Syntax

Introduced in R2021a

clear

Package: `slreq.modeling`

Clear row in Requirements Table block

Syntax

```
clear(row)
clear(row,column)
```

Description

`clear(row)` clears the row content in the requirement or assumption, `row`.

`clear(row,column)` clears the specified column of the row.

Examples

Clear Contents of Requirement in Requirements Table Block

Retrieve the requirements in a Requirements Table block and clear the first requirement.

```
requirements = getRequirementRows(reqTable);
clear(requirements(1));
```

Clear Contents of Assumption in Requirements Table Block

Retrieve the assumptions in a Requirements Table block and clear the first assumption.

```
assumptions = getAssumptionRows(reqTable);
clear(assumptions(1));
```

Clear Preconditions of Requirement

Retrieve the requirements in a Requirements Table block and clear the preconditions of the first requirement.

```
requirements = getRequirementRows(reqTable);
clear(requirements(1),"Preconditions");
```

Input Arguments

`row` — Requirement or assumption

`RequirementRow` object | `AssumptionRow` object

Requirement or assumption in a Requirements Table block, specified as a `RequirementRow` or `AssumptionRow` object. To retrieve the row, use `getRequirementRows`, `getAssumptionRows`, or `getChildren`.

`column` — Column type
`"Summary"` | `"Preconditions"` | `"Duration"` | `"Postconditions"` | `"Actions"` | `""`

Column type to clear, specified as either `"Summary"`, `"Preconditions"`, `"Duration"`, `"Postconditions"`, `"Actions"`, or an empty string scalar or character vector. If `row` is an action, you can only clear the summary, preconditions, or postconditions. If you specify `column` as an empty string scalar or character vector, the function clears the entire row.

Data Types: `enumerated`

See Also

`RequirementsTable` | `RequirementRow` | `AssumptionRow`

Introduced in R2022a

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.

Limitations

If at least one of the requirement sets comes from a model containing a Requirements Table block, you cannot use `slreq.clear()`. To use `slreq.clear()`, close the model first.

See Also

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

Introduced in R2018a

slreq.closeRequirementsManager

Close Requirements Manager app in model

Syntax

```
slreq.closeRequirementsManager(model)
slreq.closeRequirementsManager("all")
```

Description

`slreq.closeRequirementsManager(model)` closes the **Requirements Manager** app in the Simulink model `model` and brings the model to the front.

`slreq.closeRequirementsManager("all")` closes the **Requirements Manager** app in all open models.

Examples

Open and Close the Requirements Manager App Programmatically

This example shows how to open and close the **Requirements Manager** app programmatically.

Open the "Requirements Definition for a Cruise Control Model" project and open the `crs_plant` model.

```
slreqCCProjectStart;
open_system("crs_plant");
```

Open the **Requirements Manager** app in the `crs_plant` model.

```
slreq.openRequirementsManager("crs_plant");
```

Close the **Requirements Manager** app in the `crs_plant` model.

```
slreq.closeRequirementsManager("crs_plant");
```

Cleanup

Clear the open requirement sets and link sets. Close the open models and the current project.

```
slreq.clear;
bdclose all;
close(currentProject);
```

Close the Requirements Manager App in All Open Models

This example shows how to close the **Requirements Manager** app in all open models.

Open the “Requirements Definition for a Cruise Control Model” project. Open the `crs_plant` and `crs_controller` models.

```
slreqCCProjectStart;
open_system("crs_plant");
open_system("crs_controller");
```

Open the **Requirements Manager** app in the `crs_plant` and `crs_controller` models.

```
slreq.openRequirementsManager("crs_plant");
slreq.openRequirementsManager("crs_controller");
```

Close the **Requirements Manager** app in all open models.

```
slreq.closeRequirementsManager("all");
```

Cleanup

Clear the open requirement sets and link sets. Close the open models and the current project.

```
slreq.clear;
bdclose all;
close(currentProject);
```

Input Arguments

`model` — Simulink model

string scalar | character vector | model handle

Simulink model to close the **Requirements Manager** app in, specified as a string scalar or character vector that contains the name of the model, or a model handle.

Tips

- Use `bdroot` to get the top-level model of the current system.
- Use `get_param` and `bdroot` to get the handle for the top-level model of the current system:

  ```
  model = get_param(bdroot,"Handle");
  ```

See Also

`slreq.openRequirementsManager` | `bdroot` | `slreq.editor` | **Requirements Editor**

Introduced in R2021a

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

commit

Send local changes to OSLC service provider

Syntax

```
status = commit(resource,myClient)
```

Description

`status = commit(resource,myClient)` sends the local changes for the resource object `resource` to the Open Services for Lifecycle Collaboration (OSLC) service provider associated with `myClient` and returns the commit success status.

Examples

Edit a Requirement and Commit Changes

This example shows how to submit a query request for requirement resources with a configured OSLC client, edit an existing requirement resource, and commit the changes to the service provider.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type.

```
myQueryCapability = getQueryService(myClient,'Requirement');
```

Submit a query request to the service provider for the available requirement resources.

```
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign a requirement resource to the variable `myReq`. Retrieve the full resource data from the service provider for the requirement resource. Examine the `Title` property.

```
myReq = reqs(1);
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
title = myReq.Title

title =

    'Requirement 1'
```

Edit the requirement title and commit the change to the service provider.

```
myReq.Title = 'My New Requirement Title';
status = commit(myReq,myClient)

status =

  StatusCode enumeration

    OK
```

Open the requirement resource in the system browser by using the `show` function.

```
show(myReq)
```

Input Arguments

`resource` — OSLC resource object

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

Output Arguments

`status` — Commit success status

`matlab.net.http.StatusCode`

Commit success status, returned as a `matlab.net.http.StatusCode` object.

Tips

- When you use `commit`, there are two common causes of error:

  1. You do not have the required permissions from the system administrator to commit.
  2. The RDF/XML data for a locally cached resource object is either missing elements required by the service provider or is otherwise incorrectly configured.

  The returned error message contains information about why the `commit` operation failed. If the error is due to incorrectly configured RDF/XML data, use `getRDF` to see if the locally cached resource object contains the elements and attributes that are required by the service provider.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `view` | `fetch` | `remove`

External Websites
RDF 1.1 XML Syntax

Introduced in R2021a

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 requirement 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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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.Requirement` | `slreq.ReqSet`

Introduced in R2018a

create

Create resource in OSLC service provider

Syntax

```
myResource = create(myCreationFactory,resource)
```

Description

`myResource = create(myCreationFactory,resource)` submits a creation request to the Open Services for Lifecycle Collaboration (OSLC) service provider associated with the creation factory `myCreationFactory` for the resource object `resource`.

Examples

Submit a Creation Request for a User-Created Resource

This example shows how to submit a creation request for a user-created resource with a configured OSLC client.

After you have created and configured an OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a creation factory for the requirement resource type.

```
myCreationFactory = getCreationFactory(myClient,'Requirement');
```

Create a new requirement resource by creating an instance of the `oslc.rm.Requirement` class.

```
myReq = oslc.rm.Requirement
```

```
myReq =
  Requirement with properties:

    ResourceUrl: ''
          Dirty: 0
      IsFetched: 0
          Title: ''
     Identifier: ''
```

Add the `dcterms:title` property to the requirement and set the value.

```
addTextProperty(myReq,'dcterms:title','My New Requirement');
```

Submit a creation request to the service provider for the requirement object.

```
newReq = create(myCreationFactory,myReq)
```

```
newReq =
  Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/resources/_oJNtgWrqEeup0...'
```

```
        Dirty: 1
    IsFetched: 0
        Title: ''
   Identifier: ''
```

Retrieve the full resource data for the requirement resource from the service provider. Open the requirement resource in the system browser with the `show` function..

```
status = fetch(newReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
show(newReq)
```

Input Arguments

`myCreationFactory` — Resource creation factory
`oslc.core.CreationFactory` object

OSLC resource creation factory, specified as an `oslc.core.CreationFactory` object.

`resource` — OSLC resource object
`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

Output Arguments

`myResource` — New OSLC resource
`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

New OSLC resource object, returned as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`

- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

Tips

- For information about OSLC resource properties, see these pages on the OSLC website:
  - RM Resource Definitions
  - QM Resource Definitions
  - CM Resource Definitions

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `createChangeRequest` | `createRequirement` | `createTestCase` | `addResourceProperty` | `addTextProperty` | `getResourceProperty` | `removeResourceProperty` | `getProperty` | `setResourceUrl` | `setProperty`

Introduced in R2021a

slreq.modeling.create

Create new model with Requirements Table block

Syntax

```
reqTable = slreq.modeling.create
reqTable = slreq.modeling.create(model)
```

Description

`reqTable = slreq.modeling.create` creates an untitled model that contains a Requirements Table block.

`reqTable = slreq.modeling.create(model)` creates a model with the name specified by `model`.

Examples

Create a New Model that Contains a Requirements Table Block

Create a new model that contains a Requirements Table block.

```
reqTable = slreq.modeling.create;
```

The function returns `reqTable` as a `RequirementsTable` object.

Create a Model With a Custom Name

Create a new model named `myModel` that contains a Requirements Table block.

```
reqTable = slreq.modeling.create("myModel");
```

The function returns `reqTable` as a `RequirementsTable` object.

Input Arguments

`model` — Model name

string scalar | character vector

Model name, specified as a string scalar or character vector.

Data Types: `char` | `string`

Output Arguments

`reqTable` — Requirements Table block

`RequirementsTable` object

Requirements Table block, returned as a `RequirementsTable` object.

See Also

Objects
`RequirementsTable`

Functions
`slreq.modeling.find`

Topics
“Use a Requirements Table Block to Create Formal Requirements”

Introduced in R2022a

createChangeRequest

Create change request in OSLC service provider

Syntax

```
myChangeRequest = createChangeRequest(myCreationFactory,title)
```

Description

`myChangeRequest = createChangeRequest(myCreationFactory,title)` creates a change request with the specified title by using the creation factory `myCreationFactory` in the Open Services for Lifecycle Collaboration (OSLC) service provider.

Examples

Create a New Change Request

This example shows how to submit a creation request for a new change request resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Change Management Domain” on page 2-5, create a creation factory for the change request resource type.

```
myCreationFactory = getCreationFactory(myClient,'ChangeRequest');
```

Use the creation factory to create a new change request resource with the title `My New Change Request`. Retrieve the full resource data from the service provider for the change request resource and inspect the resource.

```
newCR = createChangeRequest(myCreationFactory,'My New Change Request');
fetch(newCR,myClient);
newCR
```

```
newCR =

  ChangeRequest with properties:

    ResourceUrl: 'https://localhost:9443/ccm/resource/itemName/...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Change Request'
     Identifier: '204'
```

Open the change request resource in the system browser by using the `show` function.

```
show(newCR)
```

Input Arguments

`myCreationFactory` — Resource creation factory
`oslc.core.CreationFactory` object

OSLC resource creation factory, specified as an `oslc.core.CreationFactory` object.

`title` — Change request title
character array

Change request title, specified as a character array.

Output Arguments

`myChangeRequest` — Change request resource
`oslc.cm.ChangeRequest` object

OSLC change request resource, returned as an `oslc.cm.ChangeRequest` object.

See Also

`oslc.Client` | `oslc.cm.ChangeRequest` | `oslc.core.CreationFactory` | `createRequirement` | `createTestCase`

Introduced in R2021a

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 a Link

This example shows how to create a link.

Create a link between the currently selected Simulink block and a requirement `req`.

```
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]
```

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

createRequirement

Create requirement in OSLC service provider

Syntax

```
myRequirement = createRequirement(myCreationFactory,title)
```

Description

`myRequirement = createRequirement(myCreationFactory,title)` creates a requirement with the specified title by using the creation factory `myCreationFactory` in the Open Services for Lifecycle Collaboration (OSLC) service provider.

Examples

Create a New Requirement

This example shows how to submit a creation request for a new requirement resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a creation factory for the requirement resource type.

```
myCreationFactory = getCreationFactory(myClient,'Requirement');
```

Use the creation factory to create a new requirement resource with the title `My New Requirement`. Retrieve the full resource data from the service provider for the requirement resource and inspect the resource.

```
newReq = createRequirement(myCreationFactory,'My New Requirement');
fetch(newReq,myClient);
newReq
```

```
newReq =

  Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/resources/_72lxMWJREeup0...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Requirement'
     Identifier: '1806'
```

Open the requirement resource in the system browser by using the `show` function.

```
show(newReq)
```

Input Arguments

`myCreationFactory` — Resource creation factory
`oslc.core.CreationFactory` object

OSLC resource creation factory, specified as an `oslc.core.CreationFactory` object.

`title` — Requirement title
character array

Requirement title, specified as a character array.

Output Arguments

`myRequirement` — Requirement resource
`oslc.rm.Requirement` object

OSLC requirement resource, returned as an `oslc.rm.Requirement` object.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.core.CreationFactory` | `createChangeRequest` | `createTestCase` | `createRequirementCollection`

Introduced in R2021a

createRequirementCollection

Create requirement collection in OSLC service provider

Syntax

```
myReqCol = createRequirementCollection(myCreationFactory,title)
```

Description

`myReqCol = createRequirementCollection(myCreationFactory,title)` creates a requirement collection with the specified title by using the creation factory `myCreationFactory` in the Open Services for Lifecycle Collaboration (OSLC) service provider.

Examples

Create a New Requirement Collection

This example shows how to submit a creation request for a new requirement collection resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a creation factory for the requirement collection resource type.

```
myCreationFactory = getCreationFactory(myClient,...
'RequirementCollection');
```

Use the creation factory to create a requirement collection resource with the title `My New Requirement Collection`. Retrieve the full resource data from the service provider for the requirement collection resource and inspect the resource.

```
newReqCollection = createRequirementCollection(myCreationFactory,...
'My New Requirement Collection')
fetch(newReqCollection,myClient);
newReqCollection
```

```
newReqCollection =

    RequirementCollection with properties:
    ResourceUrl: 'https://localhost:9443/rm/resources/_72lxMWJREeup0r..'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Requirement Collection'
     Identifier: '1808'
```

Open the requirement collection resource in the system browser by using the `show` function.

```
show(newReqCollection)
```

Input Arguments

`myCreationFactory` — Resource creation factory
`oslc.core.CreationFactory` object

OSLC resource creation factory, specified as an `oslc.core.CreationFactory` object.

`title` — Requirement collection title
character array

Requirement collection title, specified as a character array.

Output Arguments

`myReqCol` — Requirement collection resource
`oslc.rm.RequirementCollection` object

OSLC requirement collection resource, returned as an `oslc.rm.RequirementCollection` object.

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `oslc.rm.RequirementCollection` | `createChangeRequest` | `createRequirement` | `createTestCase`

Introduced in R2021a

createTestCase

Create test case in OSLC service provider

Syntax

```
myTestCase = createTestCase(myCreationFactory,title)
```

Description

`myTestCase = createTestCase(myCreationFactory,title)` creates a test case with the specified title created using the creation factory `myCreationFactory` in the Open Services for Lifecycle Collaboration (OSLC) service provider.

Examples

Create a New Test Case

This example shows how to submit a creation request for a new test case resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a creation factory for the test case resource type.

```
myCreationFactory = getCreationFactory(myClient,'TestCase');
```

Use the creation factory to create a test case resource with the title `My New Test Case`. Retrieve the full resource data from the service provider for the test case resource and inspect the resource.

```
newTestCase = createTestCase(myCreationFactory,'My New Test Case');
fetch(newTestCase,myClient);
newTestCase
```

```
newTestCase =
  TestCase with properties:

    ResourceUrl: 'https://localhost:9443/qm/resource/itemName/_a9aS...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Test Case'
     Identifier: '301'
```

Open the test case resource in the system browser by using the `show` function.

```
show(newTestCase)
```

Input Arguments

`myCreationFactory` — Resource creation factory
`oslc.core.CreationFactory` object

OSLC resource creation factory, specified as an `oslc.core.CreationFactory` object.

`title` — Test case title
character array

Test case title, specified as a character array.

Output Arguments

`myTestCase` — Test case resource
`oslc.qm.TestCase` object

OSLC test case resource, returned as an `oslc.qm.TestCase` object.

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `oslc.qm.TestCase` | `createChangeRequest` | `createRequirement` | `createTestExecutionRecord` | `createTestPlan` | `createTestResult` | `createTestScript`

Introduced in R2021a

createTestExecutionRecord

Create test execution record in OSLC service provider

Syntax

```
myTER = createTestExecutionRecord(myCreationFactory,title,testURL)
```

Description

`myTER = createTestExecutionRecord(myCreationFactory,title,testURL)` creates a test execution record with the specified title for the test case specified by the resource URL `testURL`. The resource is created by creation factory `myCreationFactory` in the Open Services for Lifecycle Collaboration (OSLC) service provider..

Examples

Create a New Test Execution Record

This example shows how to submit a creation request for a new test execution record resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a creation factory for the test execution record resource type.

```
myCreationFactory = getCreationFactory(myClient,'TestExecutionRecord');
```

Use the creation factory to create a test execution record resource with the title `My New Test Execution Record` and associate it with the test case resource URL `testURL` from a test case. For more information about querying the service provider for test cases, see “Edit a Test Case and Commit Changes” on page 2-21. Retrieve full resource data from the service provider for the test execution record resource and inspect the resource.

```
newTestER = createTestExecutionRecord(myCreationFactory, ...
    'My New Test Execution Record',testURL);
fetch(newTestCase,myClient);
newTestER
```

```
newTestER =
  TestExecutionRecord with properties:

    ResourceUrl: 'https://localhost:9443/qm/oslc_qm/resources/CfkIoW...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Test Execution Record'
     Identifier: '301'
```

Open the test execution record resource in the system browser by using the `show` function.

```
show(newTestER)
```

Input Arguments

`myCreationFactory` — Resource creation factory
`oslc.core.CreationFactory` object

OSLC resource creation factory, specified as an `oslc.core.CreationFactory` object.

`title` — Test execution record title
character array

Test execution record title, specified as a character array.

`testURL` — Test case URL
character array

Resource URL of the test case to associate with the test execution record, specified as a character array.

Output Arguments

`myTER` — Test execution record resource
`oslc.qm.TestExecutionRecord` object

OSLC test execution record resource, returned as an `oslc.qm.TestExecutionRecord` object.

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `oslc.qm.TestExecutionRecord` | `createChangeRequest` | `createRequirement` | `createTestCase` | `createTestPlan` | `createTestResult` | `createTestScript`

Introduced in R2021a

createTestPlan

Create test plan in OSLC service provider

Syntax

```
myTestPlan = createTestPlan(myCreationFactory,title)
```

Description

`myTestPlan = createTestPlan(myCreationFactory,title)` creates a test plan with the specified title by using the creation factory `myCreationFactory` in the Open Services for Lifecycle Collaboration (OSLC) service provider.

Examples

Create a New Test Plan

This example shows how to submit a creation request for a new test plan resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a creation factory for the test plan resource type.

```
myCreationFactory = getCreationFactory(myClient,'TestPlan');
```

Use the creation factory to create a test plan resource with the title `My New Test Plan`. Retrieve the full resource data from the service provider for the test plan resource and inspect the resource.

```
newTestPlan = createTestPlan(myCreationFactory,'My New Test Plan');
fetch(newTestPlan,myClient);
newTestPlan
```

```
newTestPlan =
  TestPlan with properties:

    ResourceUrl: 'https://localhost:9443/qm/resource/itemName/_f56s...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Test Plan'
     Identifier: '301'
```

Open the test plan resource in the system browser by using the `show` function.

```
show(newTestPlan)
```

Input Arguments

`myCreationFactory` — Resource creation factory
`oslc.core.CreationFactory` object

OSLC resource creation factory, specified as an `oslc.core.CreationFactory` object.

`title` — Test plan title
character array

Test plan title, specified as a character array.

Output Arguments

`myTestPlan` — Test plan resource
`oslc.qm.TestPlan` object

OSLC test plan resource, returned as an `oslc.qm.TestPlan` object.

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `oslc.qm.TestPlan` | `createChangeRequest` | `createRequirement` | `createTestExecutionRecord` | `createTestCase` | `createTestResult` | `createTestScript`

Introduced in R2021a

createTestResult

Create test result in OSLC service provider

Syntax

```
myTR = createTestResult(myCF,title,executionURL,testURL,status)
```

Description

`myTR = createTestResult(myCF,title,executionURL,testURL,status)` creates a test result with the specified title for the test execution record and test case specified by the resource URLs `executionURL` and `testURL`, respectively. The resource result status is specified by `status`. The resource is created by using the creation factory `myCF` in the Open Services for Lifecycle Collaboration (OSLC) service provider.

Examples

Create a New Test Result

This example shows how to submit a creation request for a new test result resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a creation factory for the test result resource type.

```
myCreationFactory = getCreationFactory(myClient,'TestResult');
```

Use the creation factory to create a test result resource with the title `My New Test Result` and associate it with the test case resource URL specified by `testURL` and the test execution record resource URL specified by `executionURL`. Set the test result status to `Unverified`. For more information about querying the service provider for test cases and execution records, see “Edit a Test Case and Commit Changes” on page 2-21 and “Edit a Test Execution Record and Commit Changes” on page 2-25. Retrieve the full resource data from the service provider for the test result resource and inspect the resource.

```
newTestResult = createTestResult(myCreationFactory, ...
    'My New Test Result',testURL,executionURL,'Unverified');
fetch(newTestCase,myClient);
newTestResult
```

```
newTestResult =
  TestResult with properties:

    ResourceUrl: 'https://localhost:9443/qm/oslc_qm/resources/CdffuW...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Test Result'
     Identifier: '1456'
```

Open the test result resource in the system browser by using the `show` function.

```
show(newTestResult)
```

Input Arguments

`myCF` — Resource creation factory
`oslc.core.CreationFactory` object

OSLC resource creation factory, specified as an `oslc.core.CreationFactory` object.

`title` — Test result title
character array

Test result title, specified as a character array.

`executionURL` — Test execution record resource URL
character array

Resource URL of the test execution record to associate with the test result, specified as a character array.

`testURL` — Test case resource URL
character array

Resource URL of the test case to associate with the test result, specified as a character array.

`status` — Test result status
character array

Test result status, specified as a character array.

Output Arguments

`myTR` — Test result resource
`oslc.qm.TestResult` object

OSLC test result resource, returned as an `oslc.qm.TestResult` object.

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `oslc.qm.TestResult` | `createChangeRequest` | `createRequirement` | `createTestExecutionRecord` | `createTestCase` | `createTestPlan` | `createTestScript`

Introduced in R2021a

createTestScript

Create test script in OSLC service provider

Syntax

```
myTestScript = createTestScript(myCreationFactory,title)
```

Description

`myTestScript = createTestScript(myCreationFactory,title)` creates a test script with the specified title by using the creation factory `myCreationFactory` in the Open Services for Lifecycle Collaboration (OSLC) service provider.

Examples

Create a New Test Script

This example shows how to submit a creation request for a new test script resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a creation factory for the test script resource type.

```
myCreationFactory = getCreationFactory(myClient,'TestScript');
```

Use the creation factory to create a test script resource with the creation factory with the title `My New Test Script`. Retrieve the full resource data from the service provider for the test script resource and inspect the resource.

```
newTestScript = createTestScript(myCreationFactory, ...
    'My New Test Script');
fetch(newTestScript,myClient);
newTestScript
```

```
newTestScript =
  TestScript with properties:

    ResourceUrl: 'https://localhost:9443/qm/resource/itemName/_b19w2...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Test Script'
     Identifier: '498'
```

Open the test script resource in the system browser by using the `show` function.

```
show(newTestScript)
```

Input Arguments

`myCreationFactory` — Resource creation factory
`oslc.core.CreationFactory` object

OSLC resource creation factory, specified as an `oslc.core.CreationFactory` object.

`title` — Test script title
character array

Test script title, specified as a character array.

Output Arguments

`myTestScript` — Test script resource
`oslc.qm.TestScript` object

OSLC test script resource, returned as an `oslc.qm.TestScript` object.

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `oslc.qm.TestScript` | `createChangeRequest` | `createRequirement` | `createTestExecutionRecord` | `createTestCase` | `createTestPlan` | `createTestResult`

Introduced in R2021a

slreq.dngConfigure

Configure IBM DOORS Next session in MATLAB

Syntax

```
slreq.dngConfigure
```

Description

`slreq.dngConfigure` establishes a connection between your MATLAB session and an IBM DOORS Next server. The function prompts you to enter your IBM DOORS Next server URL, port number information, and login credentials, and to select a project configuration.

Examples

Configure a MATLAB Session to Work With IBM DOORS Next

This example shows how to establish a connection between MATLAB and IBM DOORS Next.

Enter `slreq.dngConfigure` at the MATLAB command prompt. In the DOORS Server dialog box, provide the DOORS Next server address, port number, and service root. In the Server Login Name and Server Login Password dialog boxes, enter your login credentials. In the DOORS Project dialog box, select the project to work with and, if applicable, select the configuration context. **Select configuration stream or changeset** lists the recently used configurations. If your configuration context does not appear, select `<more>` to query the full list from the server.

```
slreq.dngConfigure;
```

```
Verifying server address...
Verifying server login username...
When prompted, enter your DOORS Next password
Select Project/Stream/Changeset that you will be working with
```

Tips

- If the function returns an error and does not open any dialog boxes, at the MATLAB command prompt, enter:

  ```
  connector.securePort
  ```

  If `connector.securePort` returns a value that is not `31515`, close all open instances of MATLAB and open one instance.
- After you select your DOORS project and click **OK**, MATLAB tests the connection to DOORS Next in your browser. If the connection is successful, the MATLAB Connector Test dialog box appears with a confirmation message. If the dialog does not appear, check that MATLAB is running on the corresponding HTTPS port. At the MATLAB command line, enter:

  ```
  connector.securePort
  ```

If the output is not `31515`, close all open instances of MATLAB and open one instance. If the dialog box still does not appear, check for security issues in your browser. If the browser indicates that the connection is unsecured or not private, and you trust the connection, click **Advanced** > **Proceed to localhost (unsafe)** to complete the connection.

- If you plan to create direct links to requirements in IBM DOORS Next, leave the test connection browser window open, because this instance of the web browser is authenticated to communicate with MATLAB. Use this authenticated instance of the web browser to select requirements in your IBM DOORS Next project and create direct links. You can re-open the test connection browser window by copying and pasting this address in the browser address bar: `https://localhost:31515/matlab/oslc/inboundTest`.
- If your network requires advanced authentication that the default authentication process does not support, you can use `rmipref` with the `'LoginProvider'` name-value argument to register a custom authentication callback function before using `slreq.dngConfigure`.

Note If you configure a session by using a custom authentication callback function, you can only create direct links to requirements in IBM DOORS Next. For more information, see “Directly Linking DOORS Next Requirements”. You cannot import requirements as described in “Import Requirements from IBM DOORS Next”.

See Also

`slreq.dngCountLinks` | `slreq.dngGetProjectConfig` | `slreq.dngGetUsedConfig` | `slreq.dngUpdateConfig`

Topics

“Link and Trace Requirements with IBM DOORS Next”
“Import Requirements from IBM DOORS Next”

Introduced in R2020a

slreq.dngCountLinks

Get number of links to IBM DOORS Next 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 DOORS Next artifacts.

`count = slreq.dngCountLinks(sourceArtifact, config)` returns the total number of links from `sourceArtifact` to the specified IBM DOORS Next 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 DOORS Next 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 DOORS Next Project, returned as a `double`.

See Also

Introduced in R2018b

slreq.dngGetProjectConfig

Query known configurations from IBM DOORS Next 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 DOORS Next Project.

`configs = slreq.dngGetProjectConfig('project', ProjectName)` returns a structure representing the configuration for the IBM DOORS Next 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 DOORS Next Project.

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

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

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

Output Arguments

`configs` — Server configurations
structure | array of structures

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

`id` — Configuration ID
character vector

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

`name` — Configuration name
character vector

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

`type` — Configuration type
character vector

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

`url` — Configuration URL
character vector

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

See Also

Introduced in R2018b

slreq.dngGetUsedConfig

Query used IBM DOORS Next configurations from MATLAB/Simulink artifacts

Syntax

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

Description

`configs = slreq.dngGetUsedConfig()` returns allIBM DOORS Next configurations linked from loaded Simulink artifacts.

`configs = slreq.dngGetUsedConfig(sourceArtifact)` returns all IBM DOORS Next 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 DOORS Next Project configuration, returned as an array of structures containing these fields.

`id` — Configuration ID
character vector

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

`name` — Configuration name
character vector

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

`type` — Configuration type
character vector

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

`url` — Configuration URL
character vector

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

See Also

Introduced in R2018b

slreq.dngUpdateConfig

Update links to IBM DOORS Next 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 DOORS Next 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 DOORS Next Project configuration name or ID, specified as a character vector.

`newConfig` — New project configuration name or ID

character vector

The new IBM DOORS Next 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 DOORS Next 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.

Tips

- Open the **Requirements Manager** app in a Simulink model with `slreq.openRequirementsManager`. You can use the **Requirements Manager** to edit and link requirements without leaving the Simulink model.

See Also

`slreq.ReqSet` | **Requirements Editor** | `slreq.openRequirementsManager`

Introduced in R2018a

slreq.exportViewSettings

Export view settings

Syntax

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

Description

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

Input Arguments

`viewSettingsFile` — View settings file

character vector

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

See Also

`slreq.importViewSettings` | `slreq.resetViewSettings`

Introduced in R2018b

fetch

Retrieve full resource data from OSLC service provider

Syntax

```
status = fetch(resource,myClient)
```

Description

`status = fetch(resource,myClient)` retrieves the XML/RDF data from the `ResourceUrl` associated with `resource` from the service provider associated with `myClient`. The function stores the XML/RDF data in the Open Services for Lifecycle Collaboration (OSLC) resource object `resource` and returns the retrieval success status. For more information about RDF/XML, see RDF 1.1 XML Syntax on the World Wide Web Consortium website.

Examples

Edit a Requirement and Commit Changes

This example shows how to submit a query request for requirement resources with a configured OSLC client, edit an existing requirement resource, and commit the changes to the service provider.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type.

```
myQueryCapability = getQueryService(myClient,'Requirement');
```

Submit a query request to the service provider for the available requirement resources.

```
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign a requirement resource to the variable `myReq`. Retrieve the full resource data from the service provider for the requirement resource. Examine the `Title` property.

```
myReq = reqs(1);
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration
```

```
    OK

title = myReq.Title

title =

    'Requirement 1'
```

Edit the requirement title and commit the change to the service provider.

```
myReq.Title = 'My New Requirement Title';
status = commit(myReq,myClient)

status =

  StatusCode enumeration

    OK
```

Open the requirement resource in the system browser by using the `show` function.

```
show(myReq)
```

Input Arguments

`resource` — OSLC resource object
`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

`myClient` — OSLC client
`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

Output Arguments

`status` — Retrieval success status
`matlab.net.http.StatusCode`

Retrieval success status, returned as a `matlab.net.http.StatusCode` object.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `remove` | `show` | `commit`

External Websites

RDF 1.1 XML Syntax

Introduced in R2021a

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 Requirements Toolbox artifacts `myArtifacts` of the type specified by `ArtifactType`.

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

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

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

Examples

Find Requirement Sets

```
% Find all requirement sets

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

allReqSets =

  1×8 ReqSet array with properties:

    Description
    Name
    Filename
    Revision
    Dirty
    CustomAttributeNames

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

myReqSet =

  ReqSet with properties:
```

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

Find Requirements

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

allReqs =

  1×72 Requirement array with properties:

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

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

myReq =

  Requirement with properties:

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

Find Referenced Requirements

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

allRefs =
```

```
  1×24 Reference array with properties:

    Keywords
    Artifact
    Id
    Summary
    Description
    SID
    Domain
    SynchronizedOn
    ModifiedOn

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

myRef =

  Reference with properties:

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

Find Link Sets

```
% Find all loaded link sets

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

allLinkSets =

  1×2 LinkSet array with properties:

    Description
    Filename
    Artifact
    Domain
    Revision
    Dirty

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

myLinkSet =

  LinkSet with properties:

    Description: ''
       Filename: 'C:\MATLAB\My_Reqs.slmx'
       Artifact: 'C:\MATLAB\My_Reqs.slreqx'
         Domain: 'linktype_rmi_slreq'
```

```
       Revision: 2
          Dirty: 0
```

Find Requirements and Links by Type

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

myFunctionalReqs =

  1×70 Requirement array with properties:

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

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

myImplementLinks =

  1×95 Link array with properties:

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

Input Arguments

`ArtifactType` — Requirements Toolbox artifact type

`'ReqSet'` | `'Requirement'` | `'Reference'` | `'LinkSet'`

The Requirements Toolbox 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` — Requirements Toolbox artifact array
`slreq.ReqSet` array | `slreq.Requirement` array | `slreq.Reference` array | `slreq.LinkSet` array

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

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

Requirements Toolbox 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
`slreq.ReqSet` | `slreq.Requirement` | `slreq.Reference` | `slreq.Justification` | `slreq.LinkSet` | `find` | `find` | `find` | `find`

Introduced in R2018a

slreq.modeling.find

Find Requirements Table blocks

Syntax

```
reqTables = slreq.modeling.find(model)
reqTables = slreq.modeling.find(handle)
```

Description

`reqTables = slreq.modeling.find(model)` returns the Requirements Table blocks in the model or subsystem specified by `model`.

`reqTables = slreq.modeling.find(handle)` returns the Requirements Table blocks in the model or subsystem specified by the model or subsystem handle `handle`.

Examples

Find Requirements Table Blocks in a Model

Find the Requirements Table blocks in a model named `myModel`.

```
reqTables = slreq.modeling.find("myModel");
```

The function returns `reqTables` as an array of `RequirementsTable` objects.

Find Requirements Table Blocks by Using a Model Handle

Get the handle of the current model.

```
modelH = get_param(gcs,"Handle");
```

Find the Requirements Table blocks in the model named `myModel`.

```
reqTables = slreq.modeling.find(modelH);
```

The function returns `reqTables` as an array of `RequirementsTable` objects.

Input Arguments

`model` — Model or subsystem name
string scalar | character vector

Model or subsystem name, specified as a string scalar or character vector.

Data Types: `char` | `string`

`handle` — Model or subsystem handle
double

Model or subsystem handle, specified as a double. To retrieve the handle, you can use the `get_param` function:

```
modelH = get_param(gcs,"Handle");
```

Data Types: `double`

Output Arguments

`reqTables` — Requirements Table blocks
array of `RequirementsTable` objects

Requirements Table blocks, returned as an array of `RequirementsTable` objects.

See Also

Functions
`slreq.modeling.create` | `get_param`

Objects
`RequirementsTable`

Topics
“Use a Requirements Table Block to Create Formal Requirements”

Introduced in R2022a

findSymbol

Package: `slreq.modeling`

Retrieve data in Requirements Table block

Syntax

```
data = findSymbol(reqTable)
data = findSymbol(reqTable,Name=Value)
```

Description

`data = findSymbol(reqTable)` returns the data defined in the Requirements Table block, `reqTable`.

`data = findSymbol(reqTable,Name=Value)` returns the data and refines the results by using one or more name-value arguments.

Examples

Find the Data in a Requirements Table Block

Retrieve the `RequirementsTable` object from a model named `myModel`.

```
table = slreq.modeling.find("myModel");
```

Retrieve the data in the block as a `Symbol` object array.

```
data = findSymbol(table);
```

Find Data with Specified Scope and Type Properties

In an model named `myModel`, retrieve the `RequirementsTable` object.

```
table = slreq.modeling.find("myModel");
```

Retrieve only data of data type `Single` that has a scope of `Output`.

```
data = findSymbol(table,Scope="Output",Type="Single");
```

Input Arguments

`reqTable` — Requirements Table block

`RequirementsTable` object

Requirements Table block, specified as a `RequirementsTable` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `findSymbol(table,Complexity="Off")` finds data where the `Complexity` property is set to `Off`.

`Complexity` — Whether data accepts complex values

`"Off"` (default) | `"On"` | `"Inherited"`

Whether the data accepts complex values, specified as one of these values:

Complexity	Description
`"Inherited"`	The data inherits complexity based on the `Scope` property. Input and output data inherit complexity from the Simulink signals connected to the associated input and output ports. Local and parameter data inherit complexity from the parameter to which the data is bound.
`"Off"`	The data is a real number.
`"On"`	The data is a complex number.

Data Types: `enumerated`

`isDesignOutput` — Whether data is design model output

`false` or `0` (default) | `true` or `1`

Whether the data is a design model output, specified as a numeric or logical `1` (`true`) or `0` (`false`). This property applies only when the `Scope` property is `Input`. For more information, see “Treat as design model output for analysis”.

Data Types: `logical`

`Name` — Name of data

`"data"` (default) | string scalar | character vector

Name of the data, specified as a string scalar or character vector.

Data Types: `char` | `string`

`Scope` — Scope of data

`"Input"` (default) | `"Output"` | `"Local"` | `"Constant"` | `"Parameter"`

Scope of the data that specifies where the data resides in memory relative to the block, specified as one of these values:

Scope	Description
`"Input"`	The data is an input signal to a Requirements Table block.
`"Output"`	The data is an output signal of a Requirements Table block.

Scope	Description
"Local"	The data is defined in the current block only.
"Constant"	The data is a read-only constant value that is visible to the block.
"Parameter"	The data resides in a variable of the same name in the MATLAB workspace, the model workspace, or in the workspace of a masked subsystem that contains this block.

Data Types: enumerated

Size — Size of data
"-1" (default) | string scalar | character vector

Size of the data, specified as a string scalar or character vector. This property must resolve to a scalar value or a MATLAB vector of values. The default value is "–1", which means that the size is inherited. For more information, see “Inherit Size from Simulink” (Simulink).

Data Types: char | string

Type — Data type
"Inherit: Same as Simulink" (default) | "double" | "single" | "int8" | ...

Data type, specified as:

- "Inherit: Same as Simulink"
- "double"
- "single"
- "half"
- "int64"
- "int32"
- "int16"
- "int8"
- "uint64"
- "uint32"
- "uint16"
- "uint8"
- "boolean"
- "fixdt(1,16,0)"
- "fixdt(1,16,2^0,0)"
- "Enum: <class name>"
- "Bus: <object name>"

To modify the data type properties, use the **Symbols** pane and Property Inspector. For more information, see “Set Data Types in Requirements Table Blocks”.

Data Types: enumerated

Output Arguments

data — Requirements Table block data

`Symbol` object array

Requirements Table block data, returned as a `Symbol` object array. The `Symbol` objects are organized by their time of creation via the array index. You cannot reorganize the data order. For more information on data creation, see “Define Data in Requirements Table Blocks”.

See Also

Objects

`Symbol` | `RequirementsTable`

Functions

`addSymbol`

Topics

“Use a Requirements Table Block to Create Formal Requirements”
“Define Data in Requirements Table Blocks”

Introduced in R2022a

slreq.generateReport

Generate report for requirement set

Syntax

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

Description

`myReportPath = slreq.generateReport(reqSetList, reportOpts)` generates a report for the requirement 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 requirement 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 requirement 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` — Requirement set

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

Requirement sets for report generation. You can specify a single requirement set or an array of requirement sets. To generate a report for all the loaded requirement 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 requirement 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.generateTraceabilityDiagram

Create a traceability diagram

Syntax

```
slreq.generateTraceabilityDiagram(startingItem)
```

Description

`slreq.generateTraceabilityDiagram(startingItem)` creates a traceability diagram that originates from `startingItem`. If a traceability diagram is already open for the specified item, the diagram comes to the foreground.

Note If you create a diagram from a link, the link `source` is the starting node. Similarly, if you create a diagram from a link set, the artifact specified by the `Artifact` is the starting node.

Examples

Create a Traceability Diagram from a Requirement

This example shows how to create a traceability diagram from a requirement object.

Open the “Requirements Definition for a Cruise Control Model” project. Load the `crs_req_func_spec` requirment set.

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

Find the `Enable Switch Detection` requirement.

```
req = slreq.find(Type="Requirement",Summary="Enable Switch Detection")
```

```
req =
  Requirement with properties:

            Type: 'Functional'
              Id: '#9'
         Summary: 'Enable Switch Detection'
     Description: '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-htm
        Keywords: {}
       Rationale: ''
       CreatedOn: 27-Feb-2017 10:15:38
       CreatedBy: 'itoy'
      ModifiedBy: 'asriram'
    IndexEnabled: 1
     IndexNumber: []
             SID: 9
    FileRevision: 55
      ModifiedOn: 03-Aug-2017 14:39:35
```

```
          Dirty: 0
       Comments: [0x0 struct]
          Index: '1.6'
```

Create a traceability diagram for the `Enable Switch Detection` requirement.

```
slreq.generateTraceabilityDiagram(req)
```

Cleanup

Clear the open requirement sets and link sets. Close the current project.

```
slreq.clear;
close(currentProject);
```

Create a Traceability Diagram from a Link

This example shows how to create a traceability diagram from a link object.

Open the “Requirements Definition for a Cruise Control Model” project. Load the `crs_req` requirement set, which also loads the `crs_req` link set.

```
slreqCCProjectStart;
slreq.load("crs_req");
```

Find the `crs_req` link set. Then find the link with description `#9: Enable Switch Detection`.

```
myLinkSet = slreq.find(Type="LinkSet",Name="crs_req");
myLink = find(myLinkSet,Type="Link",Description="#9: Enable Switch Detection")
```

```
myLink =
  Link with properties:

          Type: 'Derive'
   Description: '#9: Enable Switch Detection'
      Keywords: {}
     Rationale: ''
     CreatedOn: 20-May-2017 13:14:10
     CreatedBy: 'itoy'
    ModifiedOn: 02-Feb-2018 14:28:04
    ModifiedBy: 'itoy'
      Revision: 4
           SID: 1
      Comments: [0x0 struct]
```

Create a traceability diagram from the link.

```
slreq.generateTraceabilityDiagram(myLink)
```

Cleanup

Clear the open requirement sets and link sets. Close the current project.

```
slreq.clear;
close(currentProject);
```

Create a Traceability Diagram from a Requirement Set

This example shows how to create a traceability diagram from a requirement set.

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

```
slreqCCProjectStart;
rs = slreq.load("crs_req_func_spec");
```

Create a traceability diagram for the `crs_req_func_spec` requirement set by using the relative file path.

```
relpath = fullfile("documents","crs_req_func_spec.slreqx")
```

```
relpath =
"documents\crs_req_func_spec.slreqx"
```

```
slreq.generateTraceabilityDiagram(relpath)
```

Cleanup

Clear the open requirement sets and link sets. Close the current project.

```
slreq.clear;
close(currentProject);
```

Create a Traceability Diagram from a Link Set

This example shows how to create a traceability diagram from a link set.

Open the “Requirements Definition for a Cruise Control Model” project. Load the `crs_req` link set.

```
slreqCCProjectStart;
```

```
ls = slreq.load("crs_req.slmx");
```

Create a traceability diagram for the `crs_req` link set by using the relative file path.

```
relpath = fullfile("documents","crs_req.slmx")
```

```
relpath =
"documents\crs_req.slmx"
```

```
slreq.generateTraceabilityDiagram(relpath)
```

Cleanup

Clear the open requirement sets and link sets. Close the current project.

```
slreq.clear;
close(currentProject);
```

Input Arguments

`startingItem` — Starting item for diagram

`slreq.Requirement` object | `slreq.Reference` object | `slreq.Justification` object | `slreq.Link` object | `slreq.ReqSet` object | `slreq.LinkSet` object | string scalar | character vector

Starting item to create the traceability diagram from, specified as a:

- `slreq.Requirement` object
- `slreq.Reference` object
- `slreq.Justification` object
- `slreq.Link` object
- `slreq.ReqSet` object
- `slreq.LinkSet` object
- String scalar or character vector that contains the short name, relative file path, or full file path for a requirement set or link set

See Also

Topics

“Visualize Links with a Traceability Diagram”
“Assess Allocation and Impact”

Introduced in R2021b

slreq.generateTraceabilityMatrix

Create traceability matrix

Syntax

```
slreq.generateTraceabilityMatrix
slreq.generateTraceabilityMatrix(opts)
```

Description

`slreq.generateTraceabilityMatrix` opens the Traceability Matrix window.

`slreq.generateTraceabilityMatrix(opts)` creates a traceability matrix with the artifacts specified by `opts`.

Examples

Open the Traceability Matrix Window

Open the Traceability Matrix window.

```
slreq.generateTraceabilityMatrix
```

Close the Traceability Matrix window.

```
slreq.clear;
```

Programmatically Generate a Traceability Matrix

This example shows how to create an options structure for a traceability matrix, then generate a matrix using those options.

Open the Requirements Definition for a Cruise Control Model project.

```
slreqCCProjectStart;
```

Create an options structure for a traceability matrix.

```
opts = slreq.getTraceabilityMatrixOptions;
```

Set the `leftArtifacts` and `topArtifacts` fields of `opts`. Enter a cell array containing the name of the artifacts that you want to use in your traceabilty matrix.

```
opts.leftArtifacts = {'crs_req.slreqx','crs_req_func_spec.slreqx'};
opts.topArtifacts = {'crs_plant.slx', 'crs_controller.slx','DriverSwRequest_Tests.mldatx'};
```

Generate the traceability matrix with the artifacts specified by `opts`.

```
slreq.generateTraceabilityMatrix(opts)
```

Cleanup

Clear the open requirement sets and link sets, and close the Traceability Matrix window. Close the Requirements Definition for a Cruise Control Model project.

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

Input Arguments

`opts` — Traceability matrix options

`struct`

Traceability matrix options, specified as a `struct` with these fields:

- `leftArtifacts`
- `topArtifacts`

See Also

`slreq.getTraceabilityMatrixOptions`

Topics

“Track Requirement Links with a Traceability Matrix”

Introduced in R2021a

getAssumptionRows

Package: `slreq.modeling`

Retrieve assumptions in Requirements Table block

Syntax

```
assumptionRows = getAssumptionRows(reqTable)
```

Description

`assumptionRows = getAssumptionRows(reqTable)` returns the assumptions of the Requirements Table block specified by `reqTable`.

Examples

Retrieve Assumptions from a Requirements Table Block

Retrieve the `RequirementsTable` object from a model named `myModel`.

```
table = slreq.modeling.find("myModel");
```

Retrieve the assumptions as an array of `AssumptionRow` objects.

```
row = getAssumptionRows(table);
```

Input Arguments

`reqTable` — Requirements Table block

`RequirementsTable` object

Requirements Table block, specified as a `RequirementsTable` object.

Output Arguments

`assumptionRows` — Assumptions

array of `AssumptionRow` objects

Assumptions in the Requirements Table block, returned as an array of `AssumptionRow` objects.

See Also

Blocks
Requirements Table

Functions
`addAssumptionRow`

Objects
`RequirementsTable` | `AssumptionRow`

Introduced in R2022a

getChildren

Package: `slreq.modeling`

Retrieve child requirements and assumptions in Requirements Table block

Syntax

```
children = getChildren(row)
```

Description

`children = getChildren(row)` returns the child requirements or assumptions of the row specified by `row`.

Examples

Retrieve Child Requirements from a Requirements Table Block

Retrieve the `RequirementsTable` object from a model named `myModel`.

```
table = slreq.modeling.find("myModel");
```

Retrieve the top-level requirements as an array of `RequirementRow` objects.

```
row = getRequirementRows(table);
```

Retrieve the child requirements of the first requirement as an array of `RequirementRow` objects.

```
children = getChildren(row(1));
```

You can find children of the child rows by calling `getChildren` on child rows.

Input Arguments

`row` — Requirement or assumption

`RequirementRow` object | `AssumptionRow` object

Requirement or assumption in a Requirements Table block, specified as a `RequirementRow` or `AssumptionRow` object. To retrieve the row, use `getRequirementRows` or `getAssumptionRows`.

Output Arguments

`children` — Child requirements or assumptions

array of `RequirementRow` objects | array of `AssumptionRow` objects

Child requirements or assumptions, specified as an array of `RequirementRow` or `AssumptionRow` objects. For more information on requirement hierarchies in Requirements Table blocks, see “Establish Hierarchy in Requirements Table Blocks”.

See Also

Blocks
Requirements Table

Functions
`addRequirementRow` | `addAssumptionRow`

Objects
`RequirementsTable` | `AssumptionRow` | `RequirementRow`

Topics
“Establish Hierarchy in Requirements Table Blocks”

Introduced in R2022a

getConfigurationContextNames

Get configuration context names from OSLC service provider

Syntax

```
configs = getConfigurationContextNames(myClient)
```

Description

`configs = getConfigurationContextNames(myClient)` returns the configuration context names for the service provider specified for the OSLC client `myClient`.

Examples

Create and Configure an OSLC Client for the Requirements Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Then set the service root and catalog path for the requirements management domain and the configuration query path.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'/oslc_rm/catalog');
setConfigurationQueryPath(myClient,'gc/oslc-query/configurations');
myClient
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array

    {'OSLC Plugin'                                  }
    {'Model Based Design with OSLC'                 }
    {'OSLC4RM'                                      }
    {'Interactive Testing (Requirements Management)'}
```

```
setServiceProvider(myClient,'OSLC Plugin');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)
```

```
configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }
```

```
setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient
```

```
myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/rm/oslc_rm/catalog'
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

Output Arguments

`configs` — Configuration context names

cell array

Configuration context names for the configured service provider, returned as a cell array.

See Also

`oslc.Client` | `setConfigurationContext` | `login` | `setServiceProvider` | `getServiceProviderNames` | `setConfigurationQueryPath`

Introduced in R2021a

getCreationFactory

Get OSLC creation service object

Syntax

```
myCreationFactory = getCreationFactory(myClient)
myCreationFactory = getCreationFactory(myClient,resourceType)
```

Description

`myCreationFactory = getCreationFactory(myClient)` returns all available creation factories for the OSLC client `myClient`.

`myCreationFactory = getCreationFactory(myClient,resourceType)` returns a creation factory for the resource type specified by `resourceType` for the OSLC client `myClient`.

Examples

Create All Available Creation Factories for an OSLC Client

This example shows how to create all available creation factories for a previously configured OSLC client.

After you have created and configured an OSLC client as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create all available creation factories for the client `myClient`.

```
myCreationFactory = getCreationFactory(myClient)
```

```
myCreationFactory =

  1×8 CreationFactory array with properties:

    client
    creation
    resourceShape
    title
    resourceType
```

Examine the creation factory `resourceType` to determine which creation factory you want to use.

```
myCreationFactory(8).resourceType
```

```
ans =

  1×1 cell array

    {'http://open-services.net/ns/rm#Requirement'}
```

Submit a Creation Request by using a Creation Factory

This example shows how to submit a creation request by using a creation factory with a previously configured OSLC client.

After you have created and configured an OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a creation factory for the requirement resource type.

```
myCreationFactory = getCreationFactory(myClient,'Requirement')
```

```
myCreationFactory =

  CreationFactory with properties:

          client: [1×1 oslc.Client]
        creation: 'https://localhost:9443/rm/requirementFactory?projectURL=https%3A...'
    resourceShape: {1×22 cell}
           title: 'Requirement Creation Factory'
     resourceType: {'http://open-services.net/ns/rm#Requirement'}
```

Create a new requirement resource by using a creation factory and name the resource `My New Requirement`. Fetch the full resource properties for the requirement resource. Then commit the changes to the service provider.

```
newReq = createRequirement(myCreationFactory,'My New Requirement');
status = fetch(newReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
status = commit(newReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

View the resource that you created in the service provider.

```
show(newReq)
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

`resourceType` — OSLC resource type

`'Requirement'` | `'RequirementCollection'` | `'TestCase'` | `'TestExecutionRecord'` | `'TestPlan'` | `'TestResult'` | `'TestScript'` | `'ChangeRequest'`

OSLC resource type, specified as character array with one of these values:

- `'ChangeRequest'`
- `'TestCase'`
- `'TestExecutionRecord'`
- `'TestPlan'`
- `'TestResult'`
- `'TestScript'`
- `'Requirement'`
- `'RequirementCollection'`

The specified resource type must match the domain for the configured `oslc.Client` object.

Output Arguments

`myCreationFactory` — Resource creation factory
`oslc.core.CreationFactory` object

OSLC resource creation factory, specified as an `oslc.core.CreationFactory` object.

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `oslc.rm.Requirement` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase`

Introduced in R2021a

getCustomLoginProvider

Get registered custom authentication callback function name for OSLC client

Syntax

```
authenticationFunction = getCustomLoginProvider(myClient)
```

Description

`authenticationFunction = getCustomLoginProvider(myClient)` returns the custom authentication callback function name registered to the OSLC client `myClient`.

Examples

Get Registered Custom Authentication Callback Function

This example shows how to get the name of the custom authentication callback function that is registered to an OSLC client object.

After you have created and registered a custom authentication callback function to an OSLC client object as described in “Authenticate a Client that Requires an Advanced Authentication” on page 1-278, get the registered authentication callback function name for the OSLC client object `myClient`.

```
authenticationFunction = getCustomLoginProvider(myClient)
```

```
authenticationFunction =

    'myCustomLoginProvider'
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

Output Arguments

`authenticationFunction` — Custom authentication callback function name

character vector

Custom authentication callback function name, returned as a character vector.

See Also

`oslc.Client` | `setCustomLoginProvider`

Introduced in R2021b

slreq.getCurrentImportOptions

Get import options in `PreImportFcn` callback

Syntax

```
importOptions = slreq.getCurrentImportOptions
```

Description

`importOptions = slreq.getCurrentImportOptions` returns the import options for the current import. You can only call this function in the `PreImportFcn` callback.

Examples

Use `PreImportFcn` Callback During Import

This example shows how to assign a script as the `PreImportFcn` callback for an Import node. You get the contents of the `PreImportFcn` callback for an Import node and register a different script as the `PreImportFcn` callback after you import the requirements.

Import the Requirements

Use `slreq.import` to import the ReqIF™ file `mySpec.reqif` into Requirements Toolbox™. Name the imported requirement set `myReqSet` and register the script `myPreImportScript` as the `PreImportFcn` callback to use during import. Return a handle to the requirement set.

```
[~,~,rs] = slreq.import("mySpec.reqif",ReqSet="myReqSet",preImportFcn="myPreImportScript");
```

The script `myPreImportScript` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile.xml";
```

The mapping file `myMappingFile.xml` uses a generic mapping.

Get the custom ID for the requirement with `Index` set to `1`.

```
req1 = find(rs,Index="1");
cID = req1.CustomId
```

```
cID =

  0x0 empty char array
```

The generic mapping does not map the ReqIF attribute `ID` to the Requirement Toolbox attribute `Custom ID`. Instead, `ID` imports as a custom attribute. Get the value for the `ID` custom attribute for `Requirement 1`.

```
cID = getAttribute(req1,"ID")
```

```
cID =
'A1'
```

Get and Set the PreImportFcn Callback Script

Get a handle to the Import node, then register the script `myPreImportScrip2` as the `PreImportFcn` callback. Confirm that the registered callback was changed.

```
topRef = children(rs);
setPreImportFcn(topRef,"myPreImportScript2")
newCallback = getPreImportFcn(topRef)
```

```
newCallback =
'myPreImportScript2'
```

The script `myPreImportScript2` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript2.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile2.xml";
```

The mapping file `myMappingFile2.xml` maps these attributes from the ReqIF™ file to these properties in Requirements Toolbox™:

- `ReqSum` to `Summary`
- `Desc` to `Description`
- `ID` to `Custom ID`

Update the requirement set. The `PreImportFcn` callback script also executes when you update the requirement set.

```
updateReferences(rs,topRef);
```

Get the custom ID for the requirement with `Index` set to `1`.

```
req1 = find(rs,Index="1");
cID = req1.CustomId
```

```
cID =
'A1'
```

Output Arguments

`importOptions` — Import options

`slreq.callback.CustomImportOptions` object | `slreq.callback.DOORSImportOptions` object | ...

Import options, returned as one of these objects:

- `slreq.callback.CustomImportOptions`
- `slreq.callback.DOORSImportOptions`

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

See Also

`slreq.Reference` | `getPreImportFcn` | `setPreImportFcn`

Topics

"Use Callbacks to Customize Requirement Import Behavior"

Introduced in R2022a

slreq.getCurrentObject

Get selected objects in Requirements Editor, Requirements Browser, or Requirements Table block

Syntax

```
myReqObj = slreq.getCurrentObject
```

Description

`myReqObj = slreq.getCurrentObject` returns the currently selected item or items in the **Requirements Editor** or Requirements Browser, or the currently selected requirement in a Requirements Table block.

Note If you select an item and then select an item or group of items in a different window or block, the function returns the most recently selected item or group of items.

Examples

Get API Object for Selection in Requirements Editor

This example shows how to get the object for the most recently selected item or items in the **Requirements Editor** or the Requirements Perspective.

Open the “Requirements Definition for a Cruise Control Model” project. Load the `crs_req_func_spec` requirement set and open it in the **Requirements Editor**.

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

In the **Requirements Editor**, select requirement `#1: Driver Switch Request Handling`. Get the object for the selected requirement, then inspect the incoming links.

```
myReqObj = slreq.getCurrentObject;
lk = slreq.inLinks(myReqObj)
```

```
lk =
  Link with properties:

          Type: 'Implement'
    Description: '#1: Driver Switch Request Handling'
       Keywords: {}
      Rationale: ''
      CreatedOn: 20-May-2017 11:19:44
      CreatedBy: 'itoy'
     ModifiedOn: 17-Aug-2017 14:41:16
     ModifiedBy: 'itoy'
       Revision: 1
            SID: 1
       Comments: [0×0 struct]
```

Cleanup

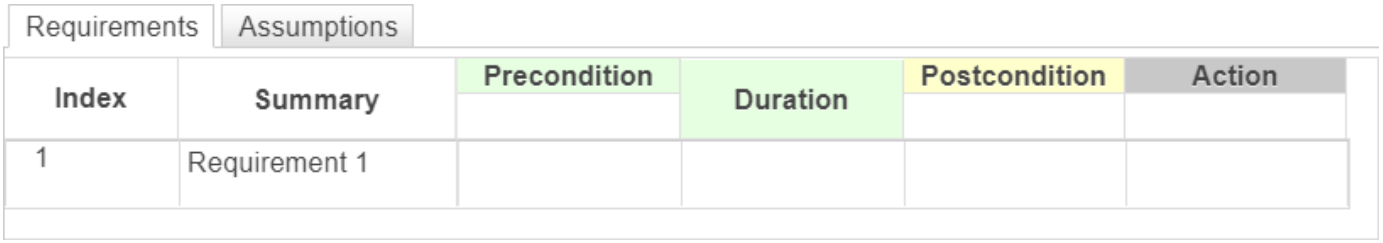
Clear the open requirement sets and link sets, and close the **Requirements Editor**. Close the project.

```
slreq.clear;
close(currentProject);
```

Get `slreq.Requirement` Object for Selected Requirement in Requirements Table Block

Create a new model and add a Requirements Table block to the model.

Open the block to view the empty requirement.



Click the index number to select the requirement.

Get the `slreq.Requirement` object for the selected requirement.

```
myReqObj = slreq.getCurrentObject;
```

Output Arguments

`myReqObj` — Requirements Toolbox object

`slreq.ReqSet` object | `slreq.Requirement` object | `slreq.Reference` object | `slreq.Justification` object | `slreq.LinkSet` object | `slreq.Link` object

Requirements Toolbox object, returned as a:

- `slreq.ReqSet` object
- `slreq.Requirement` object
- `slreq.Reference` object
- `slreq.Justification` object
- `slreq.LinkSet` object
- `slreq.Link` object

Tips

- If you execute this function during Requirements Toolbox callbacks, the function returns the target of the callback:
  - `PreImportFcn` — Returns empty when you are importing requirements. Returns a handle to the Import node when you are updating requirements.

- `PostImportFcn` — Returns a handle to the Import node. If you are importing multiple specifications from a ReqIF™ file, the function returns an array of Import nodes. For more information, see “Import Requirements from ReqIF Files”.
- `PostLoadFcn` — Returns a handle to the requirement set.
- `PreSaveFcn` — Returns a handle to the requirement set.

For more information, see “Use Callbacks to Customize Requirement Import Behavior” and “Execute Code When Loading and Saving Requirement Sets”.

See Also

`slreq.getExternalURL` | `slreq.editor`

Introduced in R2021a

getDialog

Get user interface dialogs from OSLC service provider

Syntax

```
myDialog = getDialog(myClient)
```

Description

`myDialog = getDialog(myClient)` returns the available user interface dialogs for the OSLC client `myClient`.

Examples

Get and View OSLC User Interface Dialogs

This example shows how to get and view an OSLC user interface dialog for a configured OSLC client.

After you have created and configured an OSLC client as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, get the available user interface dialogs in the requirements management domain of the client `myClient`.

```
dialogs = getDialog(myClient)
```

```
dialogs =

  1×4 Dialog array with properties:

    dialog
    hintWidth
    hintHeight
    title
    resourceType
```

Examine the properties of one of the dialogs. From the `title`, determine the resource type and if the dialog is for creating or selecting resources.

```
myDialog = dialogs(1);
title = myDialog.title
```

```
title =

    'Requirement Creation'
```

Open the dialog in a browser.

```
view(myDialog)
```

Input Arguments

myClient — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

Output Arguments

myDialog — OSLC user interface dialog

`oslc.core.Dialog` object

OSLC user interface dialog, returned as an `oslc.core.Dialog` object.

See Also

`oslc.Client` | `oslc.core.Dialog` | `view`

Introduced in R2021a

slreq.getExternalURL

Get navigation URL for link source or destination, requirement, test or Simulink model element

Syntax

```
navURL = slreq.getExternalURL(myDesignItem)
[navURL,navLabel] = slreq.getExternalURL(myDesignItem)
```

Description

`navURL = slreq.getExternalURL(myDesignItem)` returns a navigation URL to a link source or destination, requirement, test or Simulink model element specified by `myDesignItem`.

Note The MATLAB embedded web server must run on HTTP port 31415 to create the navigation URLs. If your MATLAB session is not configured for this HTTP port number, an error occurs when you try to create a link. Use `connector.port` to check the configured port number. If `connector.port` returns `0`, use `rmipref('UnsecureHttpRequests',true)` to enable the embedded HTTP server. If `connector.port` returns a number that is not `31415`, close all instances of MATLAB and reopen one instance.

`[navURL,navLabel] = slreq.getExternalURL(myDesignItem)` also returns an external navigation label, `navLabel`.

Examples

Get a Navigation URL for a Link Source or Destination

Open the “Requirements Definition for a Cruise Control Model” project. Load the `crs_req` requirement set.

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

Find the `crs_req` link set. Find the link with description `#9: Enable Switch Detection`.

```
myLinkSet = slreq.find('Type','LinkSet','Name','crs_req');
myLink = find(myLinkSet,'Description','#9: Enable Switch Detection')
```

```
myLink =
  Link with properties:

          Type: 'Derive'
    Description: '#9: Enable Switch Detection'
       Keywords: {}
      Rationale: ''
      CreatedOn: 20-May-2017 13:14:10
      CreatedBy: 'itoy'
     ModifiedOn: 02-Feb-2018 14:28:04
```

```
     ModifiedBy: 'itoy'
       Revision: 4
            SID: 1
       Comments: [0×0 struct]
```

Get a navigation URL to the link source.

```
navURL1 = slreq.getExternalURL(myLink.source)
```

```
navURL1 =
'http://localhost:31415/matlab/feval/rmi.navigate?arguments=[%22linktype_rmi_slreq%22,%22crs_req
```

Get a navigation URL to the link destination.

```
navURL2 = slreq.getExternalURL(myLink.destination)
```

```
navURL2 =
'http://localhost:31415/matlab/feval/rmi.navigate?arguments=[%22linktype_rmi_slreq%22,%22crs_req_
```

Cleanup

Clear the loaded requirement sets and link sets.

```
slreq.clear;
```

Get a Navigation URL for a Requirement Object

Open the “Requirements Definition for a Cruise Control Model” project. Load the `crs_req_func_spec` requirement set and open it in the **Requirements Editor**.

```
slreqCCProjectStart;
rs = slreq.load("crs_req");
rs2 = slreq.open("crs_req_func_spec");
```

In the **Requirements Editor**, in the `crs_req_func_spec` requirement set, select the requirement with ID `#1`. Get an API object for the requirement using `slreq.getCurrentObject`. Then get an external navigation URL for the requirement and a label for the URL.

```
req = slreq.getCurrentObject
```

```
req =
  Requirement with properties:

           Type: 'Functional'
             Id: '#1'
        Summary: 'Driver Switch Request Handling'
    Description: '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-htm
       Keywords: {}
      Rationale: ''
      CreatedOn: 27-Feb-2017 10:15:38
      CreatedBy: 'itoy'
     ModifiedBy: 'asriram'
            SID: 1
   FileRevision: 46
     ModifiedOn: 02-Aug-2017 13:49:40
```

```
           Dirty: 0
        Comments: [0×0 struct]
           Index: '1'
```

```
[navURL1,navLabel1] = slreq.getExternalURL(req)
```

```
navURL1 =
'http://localhost:31415/matlab/feval/rmi.navigate?arguments=[%22linktype_rmi_slreq%22,%22crs_req
```

```
navLabel1 =
'Driver Switch Request Handling'
```

Find a justification in the requirement set with ID #72. Get an external URL navigation URL for the justification and a label for the URL.

```
jt = find(rs2,"Type","Justification","ID","#72")
```

```
jt =
  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: [0×0 struct]
          Index: '5.1'
```

```
[navURL2,navLabel2] = slreq.getExternalURL(jt)
```

```
navURL2 =
'http://localhost:31415/matlab/feval/rmi.navigate?arguments=[%22linktype_rmi_slreq%22,%22crs_req
```

```
navLabel2 =
'Non-functional requirement'
```

Find all loaded referenced requirements. Get an external navigation URL for the third referenced requirement and a label for the URL.

```
refs = find(rs,"Type","Reference");
ref = refs(3)
```

```
ref =
  Reference with properties:

             Id: 'System overview'
       CustomId: 'System overview'
       Artifact: 'crs_req.docx'
     ArtifactId: '?System overview'
         Domain: 'linktype_rmi_word'
```

```
       UpdatedOn: 02-Feb-2018 13:23:13
       CreatedOn: NaT
       CreatedBy: ''
      ModifiedBy: ''
        IsLocked: 1
         Summary: 'System overview'
     Description: '<div class=WordSection1>↵↵<div style='border:none;border-bottom:solid #595959
       Rationale: ''
        Keywords: {}
            Type: 'Functional'
             SID: 3
    FileRevision: 1
      ModifiedOn: 03-Aug-2017 17:34:56
           Dirty: 0
        Comments: [0×0 struct]
           Index: '2'
```

```
[navURL3,navLabel3] = slreq.getExternalURL(ref)

navURL3 =
'http://localhost:31415/matlab/feval/rmi.navigate?arguments=[%22linktype_rmi_slreq%22,%22crs_req

navLabel3 =
'System overview'
```

Cleanup

Clear the loaded requirement sets and link sets. Close the **Requirements Editor**.

```
slreq.clear;
```

Get a Navigation URL for a Model Element

Open the “Requirements Definition for a Cruise Control Model” project. Open the `crs_plant` model.

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

Select the `Transmission` subsystem and use `gcb` or `gcbh` to get a path or handle to the subsystem. Then get an external navigation URL to the subsystem and a label for the URL.

```
subsys = gcb

subsys =
'crs_plant/Transmission'

[navURL1,navLabel1] = slreq.getExternalURL(subsys)

navURL1 =
'http://localhost:31415/matlab/feval/rmiobjnavigate?arguments=[%22crs_plant.slx%22,%22:414%22]'

navLabel1 =
'Transmission'
```

Look inside the `shift_logic` mask by clicking the icon. Select the `first` Stateflow® state and use `sfgco` to get a handle to the state. Then get an external navigation URL to the state and a label for the URL.

```
firstState = sfgco
```

```
firstState =

                     Path: 'crs_plant/shift_logic/gear_state'
                       Id: 462
                  Machine: [1×1 Stateflow.Machine]
               SSIdNumber: 6
                     Name: 'first'
              Description: ''
              LabelString: 'first↵'
                 FontSize: 10
                ArrowSize: 9.2240
                TestPoint: 0
                    Chart: [1×1 Stateflow.Chart]
          BadIntersection: 0
                Subviewer: [1×1 Stateflow.Chart]
                 Document: ''
                      Tag: []
          RequirementInfo: ''
           ExecutionOrder: 0
    ContentPreviewEnabled: 0
                 Position: [1×4 double]
            Decomposition: 'EXCLUSIVE_OR'
                     Type: 'OR'
               IsSubchart: 0
                IsGrouped: 0
                    Debug: [1×1 Stateflow.StateDebug]
             InlineOption: 'Auto'
              LoggingInfo: [1×1 Stateflow.SigLoggingInfo]
            HasOutputData: 0
               OutputData: []
     OutputMonitoringMode: 'SelfActivity'
    IsExplicitlyCommented: 0
    IsImplicitlyCommented: 0
              CommentText: ''
```

```
[navURL2,navLabel2] = slreq.getExternalURL(firstState)
```

```
navURL2 =
'http://localhost:31415/matlab/feval/rmiobjnavigate?arguments=[%22crs_plant.slx%22,%22:413:6%22]

navLabel2 =
'first'
```

Cleanup

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

```
slreq.clear;
bdclose('all');
```

Get a Navigation URL for a Simulink Test Case

Open the “Requirements Definition for a Cruise Control Model” project. Load the `DriverSwRequest_Tests` test file.

```
slreqCCProjectStart;
tf = sltest.testmanager.load('DriverSwRequest_Tests.mldatx');
```

Get the test suite in the test file.

```
suite = getTestSuites(tf);
```

Get the test cases in the test suite. Get an external navigation URL for the first test case and get a label for the navigation URL.

```
cases = getTestCases(suite)
```

```
cases=1×8 object
  1×8 TestCase array with properties:

    Name
    TestFile
    TestPath
    TestType
    RunOnTarget
    Parent
    Requirements
    Description
    Enabled
    ReasonForDisabling
    Tags
```

```
case1 = cases(1)
```

```
case1 =
  TestCase with properties:

            Name: 'Enable button'
        TestFile: [1×1 sltest.testmanager.TestFile]
        TestPath: 'DriverSwRequest_Tests > Unit test for DriverSwRequest > Enable button'
        TestType: 'simulation'
     RunOnTarget: {[0]}
          Parent: [1×1 sltest.testmanager.TestSuite]
    Requirements: [1×1 struct]
     Description: ''
         Enabled: 1
            Tags: [0×0 string]
```

```
[navURL,navLabel] = slreq.getExternalURL(case1)
```

```
navURL =
'http://localhost:31415/matlab/feval/rmitmnavigate?arguments=[%22DriverSwRequest_Tests.mldatx%22
```

```
navLabel =
'Enable button'
```

Cleanup

Clear the loaded requirement sets and link sets. Clear the loaded test files.

```
slreq.clear;
sltest.testmanager.clear;
```

Input Arguments

`myDesignItem` — Link source or destination, requirement, test, or model element

`slreq.link` source or destination structure | Requirements Toolbox object | path or handle to model element | Simulink Test™ object

Item in MATLAB or Simulink, specified as:

- `slreq.Link` source or destination structure
- Requirements Toolbox object:
  - `slreq.Requirement`
  - `slreq.Reference`
  - `slreq.Justification`
- Path or handle to:
  - Simulink system or block
  - Stateflow chart, subchart, state, or transition
  - System Composer™ model or component
- Simulink Test object:
  - `sltest.testmanager.TestFile`
  - `sltest.testmanager.TestSuite`
  - `sltest.testmanager.TestCase`
  - `sltest.testmanager.TestIteration`

Output Arguments

`navURL` — External navigation URL

character array

External navigation URL, returned as a character array.

`navLabel` — External navigation URL label

character array

External navigation URL label, returned as a character array.

Tips

- You can copy the external navigation URL to your clipboard for a:
  - Requirements Toolbox requirement, referenced requirement, or justification

- Simulink, Stateflow, or System Composer model element
- Simulink data dictionary entry

Right-click one of these items in the **Requirements Editor** or Simulink Editor and select **Copy URL to Clipboard**, or select **Requirements > Copy URL to Clipboard**.

See Also

`slreq.getCurrentObject` | `gcb` | `gcbh` | `sfgco` | `sltest.testmanager.getTestFiles`

Introduced in R2021a

getLinks

Get locally stored traceability links from OSLC requirement resource object

Syntax

```
URLs = getLinks(reqResource)
```

Description

`URLs = getLinks(reqResource)` returns the resource URLs associated with the `rdf:resource` attribute of the RDF/XML element `j.0:Link` for the requirement or requirement collection resource specified by `reqResource`. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website and QM Resource Definitions on the Open Services for Lifecycle Collaboration (OSLC) website.

Examples

Add and Remove Links from OSLC Resources to Requirement

This example shows how to add and remove links from OSLC resources to an OSLC requirement.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type. Submit a query request to the service provider for the available requirement resources.

```
myQueryCapability = getQueryService(myClient,'Requirement');
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign one of the requirements to a variable called `myReq` and one to `linkReq`. Fetch the full resource properties for the requirements.

```
myReq = reqs(1);
linkReq = reqs(5);
fetch(myReq,myClient);
fetch(linkReq,myClient);
```

Add a link from `linkReq` to `myReq`. Confirm the link creation by getting the links for `myReq`.

```
addLink(myReq,linkReq)
links = getLinks(myReq)
```

```
links =

  1×1 cell array

    {'https://localhost:9443/rm/CA_3d5ba3752e2c489b965a3ecceffb664a'}
```

In the service provider, identify a test case to link to the requirement. Identify the resource URL of the test case and assign it to a variable called `URL`. Add a link from `URL` to `myReq`. Confirm the link creation by getting the links for `myReq`.

```
URL = 'https://localhost:9443/qm/_ibz6tGWYEeuAF8ZpKyQQtg';
addLink(myReq,URL)
links = getLinks(myReq)
```

```
links =

  1×2 cell array

    {'https://localhost:9443/rm...'}    {'https://localhost:9443/qm...'}
```

Commit the changes to the service provider.

```
status = commit(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Fetch the full resource properties for the updated requirement `myReq`.

```
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Get the resource URLs linked to `myReq`.

```
links = getLinks(myReq)
```

```
links =

  1×2 cell array

    {'https://localhost:9443/rm...'}    {'https://localhost:9443/qm...'}
```

Get the URL for the first linked resource and assign it to `URL`.

```
URL = links{1}
```

```
URL =

    'https://localhost:9443/rm/CA_3d5ba3752e2c489b965a3ecceffb664a'
```

Before removing the link from `myReq`, confirm that the resource URL points to the requirement that you want to remove. Create a requirement resource object and set the resource URL. Fetch the full resource properties for the requirement and inspect the requirement.

```
req = oslc.rm.Requirement;
setResourceUrl(req,URL);
status = fetch(req,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
req
```

```
ans =

  Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/CA_3d5ba3752e2c489b965a...'
          Dirty: 0
      IsFetched: 1
          Title: '[SAFe] Lifecycle Scenario Template'
     Identifier: '1165'
```

Remove the link from `myReq` and commit the changes to the service provider.

```
removeLink(myReq,URL)
status = commit(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Fetch the full resource properties for the updated requirement `myReq`.

```
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Verify the link removal by getting the URLs for the resources linked to `myReq`.

```
links = getLinks(myReq)
```

```
links =

  1×1 cell array
```

```
{'https://localhost:9443/qm/_ibz6tGWYEeuAF8ZpKyQQtg'}
```

Input Arguments

`reqResource` — OSLC requirement resource

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object

OSLC requirement or requirement collection resource object, specified as an `oslc.rm.Requirement` or `oslc.rm.RequirementCollection` object.

Output Arguments

`URLs` — OSLC resource URLs for linked resources

cell array

OSLC resource URLs for resources linked to the requirement or requirement collection resource, returned as a cell array.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `addLink` | `removeLink` | `getRequirementLinks`

Introduced in R2021a

slreq.getNavigationFcn

Get registered navigation function for referenced requirements

Syntax

```
callbackFunction = slreq.getNavigationFcn(domain)
```

Description

`callbackFunction = slreq.getNavigationFcn(domain)` returns the navigation callback function name registered for imported referenced requirements that have the Domain property value equal to `domain`.

Examples

Register and Get a Navigation Callback Function for Referenced Requirements Imported from ReqIF Files

This example shows how to register and get the registered navigation callback function for referenced requirements imported from ReqIF™ files.

Import the ReqIF file `mySpec.reqif` into Requirements Toolbox™.

```
count = slreq.import("mySpec.reqif");
```

Get the handle for the imported requirement set. Check the domain for the imported referenced requirements.

```
rs = slreq.find("Type","ReqSet","Name","mySpec");
topRef = children(rs);
domain = topRef.Domain
```

```
domain =
'Third-Party Tool'
```

Check if there are any currently registered navigation callback functions for the domain.

```
callback = slreq.getNavigationFcn(domain)
```

```
callback =

  0x0 empty char array
```

Register the custom navigation callback function `myNavigationFcn` for the domain. Confirm that the navigation callback function was registered.

```
slreq.registerNavigationFcn(domain,"myNavigationFunction")
callback = slreq.getNavigationFcn(domain)
```

```
callback =
'myNavigationFunction'
```

Cleanup

Clear the open requirement sets without saving. Unregister the custom navigation callback function.

```
slreq.clear;
slreq.registerNavigationFcn(domain,'');
```

Input Arguments

`domain` — Third-party requirements tool domain

string scalar | character vector

Third-party requirements tool domain for which to get the registered the navigation callback function, specified as a string scalar.

Output Arguments

`callbackFunction` — Registered navigation callback function name

character vector

Registered navigation callback function name, returned as a character vector.

Tips

- You can get the value of the Domain property for a referenced requirement at the MATLAB command prompt by entering:

  ```
  domain = myReferencedRequirement.Domain
  ```

  ```
  domain =

      'Third-Party Tool'
  ```

See Also

`slreq.registerNavigationFcn` | `slreq.Reference` | **Requirements Editor**

Topics

“Navigate from Referenced Requirements to Requirements in Third-Party Applications”

Introduced in R2019a

getProducedTestExecutionRecord

Get locally stored test execution record traceability link from Open Services for Lifecycle Collaboration (OSLC) test result resource object

Syntax

```
executionURL = getProducedTestExecutionRecord(myTR)
```

Description

`executionURL = getProducedTestExecutionRecord(myTR)` returns the `rdf:resource` attribute of the RDF/XML element `oslc_qm:producedByTestExecutionRecord` for the test result `myTR`. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website and QM Resource Definitions on the Open Services for Lifecycle Collaboration (OSLC) website.

Examples

Get Test Resources Associated with Test Result

This example shows how to get the OSLC test execution record resource URL that produced the test result and the test case resource URL that the test result reports on.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test result resource type. Query the service provider for existing test results.

```
myQueryCapability = getQueryService(myClient,'TestResult');
testResults = queryTestResults(myQueryCapability)
```

```
testResults =

  1×9 TestResult array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Retrieve the test execution record resource URL for the test execution record that produced the test result.

```
terURL = getProducedTestExecutionRecord(myTR)
```

```
terURL =

  1×1 cell array

    {'https://localhost:9443/qm/_CfkIoWYpEeuAF8ZpKyQQtg'}
```

Retrieve the test case resource URL for the test case that the test result reports on.

```
testCaseURL = getReportsOnTestCase(myTR)
```

```
testCaseURL =

  1×1 cell array

    {'https://localhost:9443/qm/_ibz6tGWYEeuAF8ZpKyQQtg'}
```

Input Arguments

myTR — Test result resource
`oslc.qm.TestResult` object

OSLC test result resource, specified as an `oslc.qm.TestResult` object.

Output Arguments

executionURL — Test execution record resource URL
cell array

OSLC test execution record resource URL, returned as a cell array.

See Also

`oslc.Client` | `oslc.qm.TestResult` | `createTestResult` | `oslc.qm.TestExecutionRecord`

External Websites
The OSLC Quality Management (QM) Vocabulary

Introduced in R2021a

getProperty

Get local contents of text property from OSLC resource object

Syntax

```
textContents = getProperty(resource,propertyName)
```

Description

`textContents = getProperty(resource,propertyName)` returns the text contents of the RDF/XML element with the name `propertyName` from the locally stored RDF/XML data for the Open Services for Lifecycle Collaboration (OSLC) resource specified by `resource`. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website.

Examples

Add, Get, and Remove Properties from OSLC Resources

This example shows how to add, get, and remove properties from an existing OSLC requirement resource.

Create and configure the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3. Then query the service provider for requirements and assign an `oslc.rm.Requirement` object to the variable `myReq` as described in “Submit a Query Request with Query Capability” on page 1-198.

Retrieve the full resource data from the service provider for the requirement resource `myReq`.

```
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

The requirement `myReq` has a linked requirement with an `implementedBy` relationship. Get the `rdf:resource` value for the `oslc_rm:implementedBy` property for the requirement resource `myReq`.

```
linkedReq = getResourceProperty(myReq,'oslc_rm:implementedBy')
```

```
linkedReq =

  1×1 cell array

    {'https://localhost:9443/rm/resources/_72lxMWJREeup0...'}
```

Change the relationship between the linked requirement and `myReq` from `implementedBy` to `decomposedBy`. Remove the `oslc_rm:implementedBy` property and add an `oslc_rm:decomposedBy` property.

```
removeResourceProperty(myReq,'oslc_rm:implementedBy',linkedReq)
addResourceProperty(myReq,'oslc_rm:decomposedBy',linkedReq)
```

Get the text contents for the `dcterms:title` property.

```
title = getProperty(myReq,'dcterms:title')
```

```
title =

    'My New Requirement'
```

Change the title to `My New Requirement (Edited)`. Confirm the changes.

```
setProperty(myReq,'dcterms:title','My New Requirement (Edited)')
title = getProperty(myReq,'dcterms:title')
```

```
title =

    'My New Requirement (Edited)'
```

Add a new text property to the requirement with the tag `dcterms:description`. Confirm the changes.

```
addTextProperty(myReq,'dcterms:description', ...
    'My new requirement edited using the MATLAB OSLC client.');
desc = getProperty(myReq,'dcterms:description')
```

```
desc =

    'My new requirement created using the MATLAB OSLC client.'
```

Commit the changes to the service provider.

```
status = commit(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

View the resource that you edited in the system browser.

```
show(myReq)
```

Input Arguments

`resource` — OSLC resource object

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`

- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

`propertyName` — OSLC resource property name
character vector

OSLC resource property name, specified as a character vector.

Output Arguments

`textContents` — OSLC resource property text contents
character vector

OSLC resource text contents, returned as a character vector.

Tips

- For information about OSLC resource properties, see these pages on the OSLC website:
  - RM Resource Definitions
  - QM Resource Definitions
  - CM Resource Definitions

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `addTextProperty` | `setProperty`

External Websites
RDF 1.1 XML Syntax

Introduced in R2021a

getQueryService

Get OSLC query service object

Syntax

```
myQueryCapability = getQueryService(myClient)
myQueryCapability = getQueryService(myClient,resourceType)
```

Description

`myQueryCapability = getQueryService(myClient)` returns all available query capabilities for the OSLC client `myClient`.

Tip Use this syntax to create query services with resource types that are not defined in the OSLC standard.

`myQueryCapability = getQueryService(myClient,resourceType)` returns a query capability for the resource type specified by `resourceType` for the OSLC client `myClient`.

Examples

Create All Available Query Capabilities for a Given Client

This example shows how to create all available query capabilities for a configured OSLC client.

After you have created and configured an OSLC client as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create all available query capabilities for the client `myClient`.

```
myQueryCapability = getQueryService(myClient)
```

```
myQueryCapability =

  1×4 QueryCapability array with properties:

    queryParameter
    client
    queryBase
    resourceShape
    title
    resourceType
```

Examine the query capability `resourceType` to determine which query capability you want to use.

```
myQueryCapability(3).resourceType(2)
```

```
ans =

  1×1 cell array
```

```
    {'http://open-services.net/ns/rm#Requirement'}
```

Submit a Query Request with Query Capability

This example shows how to submit a query request with a configured OSLC client.

After you have created and configured an OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type.

```
myQueryCapability = getQueryService(myClient,'Requirement')
```

```
myQueryCapability =

  QueryCapability with properties:

    queryParameter: ''
            client: [1×1 oslc.Client]
         queryBase: 'https://localhost:9443/rm/views?oslc.query=true&projectURL=http...'
     resourceShape: {0×1 cell}
             title: 'Query Capability'
      resourceType: {1×2 cell}
```

Submit a query request to the service provider for the available requirement resources.

```
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign the first returned requirement resource to the variable `myReq`, then fetch the full resource properties for `myReq`. Examine the `Title` property.

```
myReq = reqs(1);
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
title = myReq.Title
```

```
title =

    'Requirement 1'
```

Input Arguments

`myClient` — OSLC client
`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

`resourceType` — OSLC resource type
`'Requirement'` | `'RequirementCollection'` | `'TestCase'` | `'TestExecutionRecord'` | `'TestPlan'` | `'TestResult'` | `'TestScript'` | `'ChangeRequest'`

OSLC resource type, specified as character array with one of these values:

- `'ChangeRequest'`
- `'TestCase'`
- `'TestExecutionRecord'`
- `'TestPlan'`
- `'TestResult'`
- `'TestScript'`
- `'Requirement'`
- `'RequirementCollection'`

The specified resource type must match the domain for the configured `oslc.Client` object.

Output Arguments

`myQueryCapability` — Resource query capability
`oslc.core.QueryCapability` object

OSLC resource query capability, specified as an `oslc.core.QueryCapability` object.

See Also

`oslc.Client` | `oslc.core.QueryCapability` | `oslc.rm.Requirement` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase`

Introduced in R2021a

getRDF

Get resource RDF/XML data from OSLC resource object

Syntax

```
rdfContent = getRDF(resource)
```

Description

`rdfContent = getRDF(resource)` returns the locally stored RDF/XML data for the resource specified by `resource`. For more information, see RDF classes and properties in OSLC on the Open Services for Lifecycle Collaboration (OSLC) website.

Examples

Get and Set RDF Content for Requirement Resource

This example shows how to get and set the RDF content of an OSLC requirement resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type.

```
myQueryCapability = getQueryService(myClient);
```

Submit a query request to the service provider for the available requirement resources.

```
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Fetch the full resource properties for a single requirement resource. Inspect the title of the requirement.

```
myReq = reqs(1);
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
title = myReq.Title

title =

    'My New Requirement'
```

Get the locally stored RDF content of the requirement resource.

```
rdfContent = getRDF(myReq)

rdfContent =

    '<?xml version="1.0" encoding="UTF-8" standalone="no" ?><rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:oslc="http://open-services.net/ns/core#"
xmlns:oslc_rm="http://open-services.net/ns/rm#">
         <oslc_rm:Requirement>
        <dcterms:title>My New
Requirement</dcterms:title><oslc:instanceShape
rdf:resource="https://example.com/shapes/oslc-requirement-version1"/>
</oslc_rm:Requirement>
     </rdf:RDF>'
```

Copy and paste the `rdfContent` text into a new variable `newRDF`. Edit the text contents for the `dcterms:title` property to `My New Requirement (Edited)`.

```
newRDF = ['<?xml version="1.0" encoding="UTF-8" ' ...
'standalone="no" ?><rdf:RDF ' ...
'xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" ' ...
'xmlns:dcterms="http://purl.org/dc/terms/" ' ...
'xmlns:oslc="http://open-services.net/ns/core#" ' ...
'xmlns:oslc_rm="http://open-services.net/ns/rm#">' ...
'<oslc_rm:Requirement><dcterms:title>' ...
'My New Requirement (Edited)</dcterms:title>' ...
'<oslc:instanceShape rdf:resource=' ...
'"https://example.com/shapes/oslc-requirement-version1"/>' ...
'</oslc_rm:Requirement></rdf:RDF>']
```

Set the RDF content of the requirement to the variable `newRDF`. Inspect the requirement title.

```
setRDF(myReq,newRDF);
title = myReq.Title

title =

    'My New Requirement (Edited)'
```

Commit the changes to the service provider.

```
status = commit(newReq,myClient)

status =

  StatusCode enumeration
```

```
OK
```

Input Arguments

`resource` — OSLC resource object

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

Output Arguments

`rdfContent` — RDF resource data

character vector

RDF data for the OSLC resource, returned as a character vector.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `setRDF`

External Websites

RDF 1.1 XML Syntax

Introduced in R2021a

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 requirement set custom attributes in your report

See Also

`slreq.generateReport`

Introduced in R2018a

getReportsOnTestCase

Get locally stored test case traceability link from OSLC test result resource object

Syntax

```
testCaseURL = getReportsOnTestCase(myTR)
```

Description

`testCaseURL = getReportsOnTestCase(myTR)` returns the `rdf:resource` attribute of the RDF/XML element `oslc_qm:reportsOnTestCase` for the test result `myTR`. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website and QM Resource Definitions on the Open Services for Lifecycle Collaboration (OSLC) website.

Examples

Get Test Resources Associated with Test Result

This example shows how to get the OSLC test execution record resource URL that produced the test result and the test case resource URL that the test result reports on.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test result resource type. Query the service provider for existing test results.

```
myQueryCapability = getQueryService(myClient,'TestResult');
testResults = queryTestResults(myQueryCapability)
```

```
testResults =

  1×9 TestResult array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Retrieve the test execution record resource URL for the test execution record that produced the test result.

```
terURL = getProducedTestExecutionRecord(myTR)
```

```
terURL =

  1×1 cell array

    {'https://localhost:9443/qm/_CfkIoWYpEeuAF8ZpKyQQtg'}
```

Retrieve the test case resource URL for the test case that the test result reports on.

```
testCaseURL = getReportsOnTestCase(myTR)

testCaseURL =

  1×1 cell array

    {'https://localhost:9443/qm/_ibz6tGWYEeuAF8ZpKyQQtg'}
```

Input Arguments

myTR — Test result resource

`oslc.qm.TestResult` object

OSLC test result resource, specified as an `oslc.qm.TestResult` object.

Output Arguments

testCaseURL — Associated test case resource URL

cell array

Resource URL of the test case that the test result reports on, returned as a cell array.

See Also

`oslc.Client` | `oslc.qm.TestResult` | `createTestResult` | `oslc.qm.TestCase`

External Websites

The OSLC Quality Management (QM) Vocabulary

Introduced in R2021a

getRequirementLinks

Get locally stored requirement traceability links from OSLC test resource object

Syntax

```
reqs = getRequirementLinks(testResource)
```

Description

`reqs = getRequirementLinks(testResource)` returns the requirement resource associated with the `rdf:resource` attribute of the RDF/XML element `oslc_qm:validatesRequirement` for the test case or test script specified by `testResource`. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website and QM Resource Definitions on the Open Services for Lifecycle Collaboration (OSLC) website.

Examples

Add, Get, and Remove Traceability Links from a Test Case to a Requirement

This example shows how to add, remove, and get OSLC requirement resources linked to a test case resource with a previously configured OSLC client.

After you have created and configured an OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test case resource type.

```
myQueryCapability = getQueryService(myClient,'TestCase');
```

Submit a query request to the service provider for the available test case resources.

```
testCases = queryTestCases(myQueryCapability)
```

```
testCases =

  1×5 TestCase array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Retrieve the requirement resources linked to one of the test cases. Fetch the resource properties from the service provider for the test case.

```
myTestCase = testCases(1);
fetch(myTestCase,myClient);
reqs = getRequirementLinks(myTestCase)
```

```
reqs =
```

```
    Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/resources/_aQ1gRg8bEeuLWbFe'
          Dirty: 1
      IsFetched: 0
          Title: ''
     Identifier: ''
```

Remove the existing link to the requirement resource from the test case resource. Commit the changes to the service provider.

```
removeRequirementLink(myTestCase,reqs.ResourceUrl);
status = commit(myTestCase,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

To add a link to a requirement, in the OSLC service provider, locate the requirement resource that you want to link to the test case resource. Identify the resource URL. Create a variable `URL` and set the value of the variable to the requirement URL that you found in the service provider.

```
URL = 'https://localhost:9443/rm/resources/_oJNtgWrqEeup0a6t';
```

Create a traceability link between the requirement resource and the test case. Commit the change to the service provider.

```
addRequirementLink(myTestCase,URL);
status = commit(myTestCase,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

View the test case in the system browser.

```
show(myTestCase)
```

Input Arguments

`testResource` — OSLC test resource

`oslc.qm.TestCase` object | `oslc.qm.TestScript` object

OSLC test resource, specified as an `oslc.qm.TestCase` or `oslc.qm.TestScript` object.

Output Arguments

`reqs` — OSLC requirement resource

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object

OSLC requirement or requirement collection resource object, returned as an `oslc.rm.Requirement` or `oslc.rm.RequirementCollection` object.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.qm.TestCase` | `oslc.qm.TestScript` | `oslc.rm.RequirementCollection` | `addRequirementLink` | `removeRequirementLink`

Introduced in R2021a

getRequirementRows

Package: `slreq.modeling`

Retrieve requirements in Requirements Table block

Syntax

```
RequirementRows = getRequirementRows(reqTable)
```

Description

`RequirementRows = getRequirementRows(reqTable)` returns the requirements of the Requirements Table block specified by `reqTable`.

Examples

Retrieve Requirements from a Requirements Table Block

Retrieve the `RequirementsTable` object from a model named `myModel`.

```
table = slreq.modeling.find("myModel");
```

Retrieve the requirements as an array of `RequirementRow` objects.

```
row = getRequirementRows(table);
```

Input Arguments

`reqTable` — Requirements Table block

`RequirementsTable` object

Requirements Table block, specified as a `RequirementsTable` object.

Output Arguments

`RequirementRows` — Requirements

array of `RequirementRow` objects

Requirements in the Requirements Table block, returned as an array of `RequirementRow` objects.

See Also

Blocks
Requirements Table

Functions
`addRequirementRow`

Objects
`RequirementsTable` | `RequirementRow`

Introduced in R2022a

getResourceProperty

Get local contents of resource property from OSLC resource object

Syntax

```
rdfResource = getResourceProperty(resource,propertyName)
```

Description

`rdfResource = getResourceProperty(resource,propertyName)` returns the `rdf:resource` attribute of the RDF/XML element with name `propertyName` from the locally stored RDF/XML for the Open Services for Lifecycle Collaboration (OSLC) resource specified by `resource`. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website.

Examples

Add, Get, and Remove Properties from OSLC Resources

This example shows how to add, get, and remove properties from an existing OSLC requirement resource.

Create and configure the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3. Then query the service provider for requirements and assign an `oslc.rm.Requirement` object to the variable `myReq` as described in “Submit a Query Request with Query Capability” on page 1-198.

Retrieve the full resource data from the service provider for the requirement resource `myReq`.

```
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

The requirement `myReq` has a linked requirement with an `implementedBy` relationship. Get the `rdf:resource` value for the `oslc_rm:implementedBy` property for the requirement resource `myReq`.

```
linkedReq = getResourceProperty(myReq,'oslc_rm:implementedBy')
```

```
linkedReq =

  1×1 cell array

    {'https://localhost:9443/rm/resources/_72lxMWJREeup0...'}
```

Change the relationship between the linked requirement and `myReq` from `implementedBy` to `decomposedBy`. Remove the `oslc_rm:implementedBy` property and add an `oslc_rm:decomposedBy` property.

```
removeResourceProperty(myReq,'oslc_rm:implementedBy',linkedReq)
addResourceProperty(myReq,'oslc_rm:decomposedBy',linkedReq)
```

Get the text contents for the `dcterms:title` property.

```
title = getProperty(myReq,'dcterms:title')
```

```
title =

    'My New Requirement'
```

Change the title to `My New Requirement (Edited)`. Confirm the changes.

```
setProperty(myReq,'dcterms:title','My New Requirement (Edited)')
title = getProperty(myReq,'dcterms:title')
```

```
title =

    'My New Requirement (Edited)'
```

Add a new text property to the requirement with the tag `dcterms:description`. Confirm the changes.

```
addTextProperty(myReq,'dcterms:description', ...
    'My new requirement edited using the MATLAB OSLC client.');
desc = getProperty(myReq,'dcterms:description')
```

```
desc =

    'My new requirement created using the MATLAB OSLC client.'
```

Commit the changes to the service provider.

```
status = commit(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

View the resource that you edited in the system browser.

```
show(myReq)
```

Input Arguments

`resource` — OSLC resource object

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

`propertyName` — OSLC resource property name
character vector

OSLC resource property name, specified as a character vector.

Output Arguments

`rdfResource` — OSLC resource property `rdf:resource` attribute
cell array

OSLC resource property `rdf:resource` attribute, returned as a cell array.

Tips

- For information about OSLC resource properties see these pages on the OSLC website:
  - RM Resource Definitions
  - QM Resource Definitions
  - CM Resource Definitions

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `addResourceProperty` | `removeResourceProperty`

External Websites
RDF 1.1 XML Syntax

Introduced in R2021a

getRunsTestCase

Get locally stored test case traceability link from OSLC test execution record resource object

Syntax

```
testCaseURL = getRunsTestCase(myTER)
```

Description

`testCaseURL = getRunsTestCase(myTER)` returns the `rdf:resource` attribute of the RDF/XML element `oslc_qm:runsTestCase` for the test execution record `myTER`. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website and QM Resource Definitions on the Open Services for Lifecycle Collaboration (OSLC) website.

Examples

Get Test Case URL Associated with Test Execution Record

This example shows how to get the test case resource URL for the test case run by a test execution resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test execution record resource type. Query the service provide for existing test execution records.

```
myQueryCapability = getQueryService(myClient,'TestExecutionRecord');
TERs = queryTestExecutionRecords(myQueryCapability)
```

```
TERs =

  1×2 TestExecutionRecord array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Retrieve a test case resource URL run by one of the test execution records.

```
myTER = TERs(1);
testCaseURL = getRunsTestCase(myTER)
```

```
testCaseURL =

  1×1 cell array
```

```
{'https://localhost:9443/qm/resources/_NMg4MWJzEeuAF8ZpKyQQtg'}
```

Input Arguments

myTER — Test execution record resource

`oslc.qm.TestExecutionRecord` object

OSLC test execution record resource, specified as an `oslc.qm.TestExecutionRecord` object.

Output Arguments

testCaseURL — Associated test case resource URL

cell array

Resource URL of the test case that the test execution record runs, returned as a cell array.

See Also

`oslc.Client` | `oslc.qm.TestCase` | `createTestExecutionRecord` | `oslc.qm.TestExecutionRecord`

External Websites

The OSLC Quality Management (QM) Vocabulary

Introduced in R2021a

getServer

Get server URL for OSLC client

Syntax

```
myServerURL = getServer(myClient)
```

Description

`myServerURL = getServer(myClient)` returns the server URL for the configured OSLC client `myClient`.

Examples

Get Server URL for an OSLC Client

This example shows how to get the server URL for an OSLC client created in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

After you have created and configured an OSLC client as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, get the server URL for the OSLC client `myClient`.

```
mySeverURL = getServer(myClient)
```

```
myServerURL =

    'https://localhost:9443'
```

Input Arguments

`myClient` — OSLC client
`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

Output Arguments

`myServerURL` — Server URL for OSLC client
character vector

Server URL for OSLC client, returned as a character vector.

Example: `'https://localhost:9443'`

See Also

`oslc.Client` | `setServer`

Introduced in R2021a

getServiceProviderNames

Get service providers for OSLC client

Syntax

```
providerNames = getServiceProviderNames(myClient)
```

Description

`providerNames = getServiceProviderNames(myClient)` returns the service providers for the configured OSLC client `myClient`.

Examples

Create and Configure an OSLC Client for the Requirements Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Then set the service root and catalog path for the requirements management domain and the configuration query path.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'/oslc_rm/catalog');
setConfigurationQueryPath(myClient,'gc/oslc-query/configurations');
myClient
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array

    {'OSLC Plugin'                                  }
    {'Model Based Design with OSLC'                 }
    {'OSLC4RM'                                      }
    {'Interactive Testing (Requirements Management)'}
```

```
setServiceProvider(myClient,'OSLC Plugin');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)
```

```
configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }
```

```
setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient
```

```
myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/rm/oslc_rm/catalog'
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

Output Arguments

`providerNames` — Service providers for OSLC client

cell array

Names of the available service providers for the OSLC client, returned as a cell array.

See Also

`oslc.Client` | `getConfigurationContextNames` | `setConfigurationContext` | `login` | `setServiceProvider` | `setConfigurationQueryPath`

Introduced in R2021a

getSLRequirements

Get imported referenced requirement associated with OSLC requirement resource object

Syntax

```
ref = getSLRequirements(reqResource)
```

Description

`ref = getSLRequirements(reqResource)` returns the imported referenced requirement associated with the OSLC requirement or requirement collection resource `reqResource`.

Examples

Get Imported Referenced Requirement for OSLC Requirement

This example shows how to get the referenced requirement that was imported from IBM DOORS Next that is associated with the OSLC requirement resource in the same project in DOORS Next.

Import requirements from IBM DOORS Next. For more information, see “Import Requirements from IBM DOORS Next”.

Create and configure an OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3. When setting the service provider and configuration context, use the same settings that you used when importing the requirements.

Create a creation factory for the requirement resource type. Query the service provider for requirements. Submit a query request to the service provider for the available requirement resources.

```
myCreationFactory = getCreationFactory(myClient,'Requirement');
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign one of the requirements to the variable `myReq`. Retrieve the full resource data from the service provider for the requirement resource.

```
myReq = reqs(1);
status = fetch(myReq,myClient)
```

```
status =
```

```
  StatusCode enumeration

    OK
```

Get the imported referenced requirement associated with `myReq`.

```
ref = getSLRequirements(myReq)
```

```
ref =

  Reference with properties:

             Id: '431'
       CustomId: '431'
       Artifact: 'https://localhost:9443/rm/_BCoGwgJZEeuFW5Ss3RBk7w'
     ArtifactId: 'https://localhost:9443/rm/_BDSOEwJZEeuFW5Ss3RBk7w'
         Domain: 'OSLC'
      UpdatedOn: 17-Feb-2021 13:54:13
      CreatedOn: 29-Sep-2020 09:38:16
      CreatedBy: ''
     ModifiedBy: ''
       IsLocked: 1
        Summary: 'System Hazards'
    Description: 'System Hazards'
      Rationale: ''
       Keywords: {}
           Type: 'Functional'
            SID: 431
   FileRevision: 1
     ModifiedOn: 29-Sep-2020 09:38:16
          Dirty: 0
       Comments: [0×0 struct]
          Index: '1'
```

Input Arguments

`reqResource` — OSLC requirement resource

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object

OSLC requirement or requirement collection resource object, specified as an `oslc.rm.Requirement` or `oslc.rm.RequirementCollection` object.

Output Arguments

`ref` — Referenced requirement

`slreq.Reference`

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

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `slreq.Reference` | `getLinks` | `getRequirementLinks`

Topics

“Link and Trace Requirements with IBM DOORS Next”

Introduced in R2021a

getStatus

Get locally stored status from OSLC test result resource object

Syntax

```
status = getStatus(myTR)
```

Description

`status = getStatus(myTR)` returns the text contents of the RDF/XML element `oslc_qm:status` for the test result `myTR`. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website and QM Resource Definitions on the Open Services for Lifecycle Collaboration (OSLC) website.

Examples

Get Test Result Status

This example shows how to get the OSLC test result status.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test result resource type. Query the service provide for existing test results.

```
myQueryCapability = getQueryService(myClient,'TestResult');
testResults = queryTestResults(myQueryCapability)
```

```
testResults =

  1×9 TestResult array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Retrieve the test result status for one of the test results.

```
myTR = testResults(1);
status = getStatus(myTR)
```

```
status =

    'example.qm.execution.state.passed'
```

Input Arguments

myTR — Test result resource

`oslc.qm.TestResult` object

OSLC test result resource, specified as an `oslc.qm.TestResult` object.

Output Arguments

`status` — Test result resource status

character vector

OSLC test result resource status, returned as a character vector.

See Also

`oslc.Client` | `oslc.qm.TestResult` | `createTestResult`

External Websites

The OSLC Quality Management (QM) Vocabulary

Introduced in R2021a

slreq.getTraceabilityMatrixOptions

Create options structure for traceability matrix

Syntax

```
opts = slreq.getTraceabilityMatrixOptions
opts = slreq.getTraceabilityMatrixOptions('current')
```

Description

`opts = slreq.getTraceabilityMatrixOptions` creates an empty traceability matrix options structure.

`opts = slreq.getTraceabilityMatrixOptions('current')` creates a traceability matrix options structure containing the artifacts from the selected tab in the Traceability Matrix window.

Examples

Programmatically Generate a Traceability Matrix

This example shows how to create an options structure for a traceability matrix, then generate a matrix using those options.

Open the Requirements Definition for a Cruise Control Model project.

```
slreqCCProjectStart;
```

Create an options structure for a traceability matrix.

```
opts = slreq.getTraceabilityMatrixOptions;
```

Set the `leftArtifacts` and `topArtifacts` fields of `opts`. Enter a cell array containing the name of the artifacts that you want to use in your traceabilty matrix.

```
opts.leftArtifacts = {'crs_req.slreqx','crs_req_func_spec.slreqx'};
opts.topArtifacts = {'crs_plant.slx', 'crs_controller.slx','DriverSwRequest_Tests.mldatx'};
```

Generate the traceability matrix with the artifacts specified by `opts`.

```
slreq.generateTraceabilityMatrix(opts)
```

Cleanup

Clear the open requirement sets and link sets, and close the Traceability Matrix window. Close the Requirements Definition for a Cruise Control Model project.

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

Get Artifacts from the Selected Traceability Matrix

This example shows how to get the artifacts from the selected tab in the Traceability Matrix window, then re-generate the matrix.

Setup

Open the Requirements Definition for a Cruise Control Model project.

```
slreqCCProjectStart;
```

Load the `crs_controller` model, then open the Traceability Matrix window.

```
load_system('crs_controller');
slreq.generateTraceabilityMatrix;
```

Create the Traceability Matrix

1. In the Traceability Matrix window, in the **Select Artifacts** dialog, set **Left** to `crs_req_func.slreqx` and **Top** to `crs_controller.slx`.
2. Click **Generate Matrix**.

Get Artifacts from the Traceability Matrix

Without closing the Traceability Matrix window, get the artifacts that were used to generate the matrix.

```
opts = slreq.getTraceabilityMatrixOptions('current')
```

```
opts = struct with fields:
    leftArtifacts: {'C:\Users\jdoe\MATLAB\Projects\examples\CruiseRequirementsExample\documents\
     topArtifacts: {'C:\Users\jdoe\MATLAB\Projects\examples\CruiseRequirementsExample\models\crs_
```

Close the Traceability Matrix window. Re-generate the matrix with the artifacts specified by `opts`.

```
slreq.generateTraceabilityMatrix(opts)
```

Cleanup

Clear the open requirement sets and link sets, and close the Traceability Matrix window.

```
slreq.clear;
```

Output Arguments

`opts` — Traceability matrix options

`struct`

Traceability matrix options, specified as a `struct` with these fields:

- `leftArtifacts`
- `topArtifacts`

See Also

`slreq.generateTraceabilityMatrix`

Topics
“Track Requirement Links with a Traceability Matrix”

Introduced in R2021a

getUser

Get user for OSLC client

Syntax

```
user = getUser(myClient)
```

Description

`user = getUser(myClient)` returns the configured user for the OSLC client `myClient`.

Examples

Get User for an OSLC Client

This example shows how to get the user for an OSLC client created in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

After you have created and configured an OSLC client as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, get the user for the OSLC client `myClient`.

```
user = getUser(myClient)
```

```
user =

    'jdoe'
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

Output Arguments

`user` — User for OSLC client

character vector

User for the OSLC client, returned as a character vector.

See Also

`oslc.Client` | `setUser` | `login`

Introduced in R2021a

hideAssumptionColumn

Package: `slreq.modeling`

Hide Precondition column in Assumptions tab

Syntax

```
hideAssumptionColumn(reqTable)
```

Description

`hideAssumptionColumn(reqTable)` hides the **Precondition** column in the **Assumptions** tab of the Requirements Table block, `reqTable`. The **Precondition** column must be empty.

Examples

Hide the Precondition Column in a Requirements Table Block

Find the Requirements Table block in a model by using `slreq.modeling.find`.

```
reqTable = slreq.modeling.find("myModel");
```

Hide the **Precondition** column in the **Assumptions** tab.

```
hideAssumptionColumn(reqTable);
```

Input Arguments

`reqTable` — Requirements Table block

`RequirementsTable` object

Requirements Table block, specified as a `RequirementsTable` object.

See Also

Objects
`RequirementsTable`

Functions
`showAssumptionColumn` | `showRequirementColumn` | `hideRequirementColumn`

Introduced in R2022a

hideRequirementColumn

Package: `slreq.modeling`

Hide columns in Requirements tab

Syntax

```
hideRequirementColumn(reqTable,column)
```

Description

`hideRequirementColumn(reqTable,column)` hides the column type specified by `column` in the **Requirements** tab of the Requirements Table block, `reqTable`. The column type must be empty.

Examples

Hide the Postcondition Columns in a Requirements Table Block

Find the Requirements Table block in a model by using `slreq.modeling.find`.

```
reqTable = slreq.modeling.find("myModel");
```

Hide the **Postcondition** columns in the **Requirements** tab.

```
hideRequirementColumn(reqTable,"postconditions");
```

Input Arguments

`reqTable` — Requirements Table block

`RequirementsTable` object

Requirements Table block, specified as a `RequirementsTable` object.

`column` — Column type

`"duration"` | `"actions"` | `"postconditions"`

Column type to be shown, specified as `"duration"`, `"actions"`, or `"postconditions"`. Use this argument to show the **Duration**, **Action**, or **Postcondition** columns, respectively.

Data Types: `enumerated`

See Also

Objects

`RequirementsTable`

Functions

`showRequirementColumn` | `showAssumptionColumn` | `hideAssumptionColumn`

Introduced in R2022a

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 requirement 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 requirement 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 requirement 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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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.

`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);`

`attributeColumn` — Custom Attributes Column
`double` array

Column in the Microsoft Excel® spreadsheet that you want to map as custom attributes of the requirements in your requirement set, specified as a `double` array.

Example: `'attributeColumn',[4 6]`

`attributes` — Attribute names
cell array

Attribute names for custom attribute columns, specified as a cell array of character vectors.

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 `attributeColumn` argument.

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

`bookmarks` — Option to import requirements using bookmarks
`0` (default) | `1`

Option to import requirements content using user-defined bookmarks, specified as a `1` or `0` of data type `logical`.

By default, Requirements Toolbox sets the value to `1` for Microsoft Word documents and `0` for Microsoft Excel spreadsheets.

Example: `'bookmarks',false`

`columns` — Range of columns
`double` array

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

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

`createdByColumn` — Created By Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `CreatedBy` property of the requirements in your requirement set, specified as a `double`.

Example: `'createdByColumn',5`

`descriptionColumn` — Description Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `Description` property of the requirements in your requirement set, specified as a `double`.

Example: `'descriptionColumn',2`

`idColumn` — ID Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `ID` property of the requirements in your requirement set, specified as a `double`.

Example: `'idColumn',1`

`keywords` — Attribute to map to `Keywords`
string scalar | character vector

Name of the attribute from the external document that you want to map to the `Keywords` property for the imported requirements.

Use this argument when you import from IBM Rational DOORS or custom document types.

Example: `"keywords","Requirement Keywords"`

`keywordsColumn` — Keywords Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `Keywords` property of the requirements in your requirement set, specified as a `double`.

Example: `'keywordsColumn',3`

`match` — Regular expression pattern
character vector

Regular expression pattern for ID search in Microsoft Office documents.

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

`modifiedByColumn` — Modified By Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `ModifiedBy` property of the requirements in your requirement set, specified as a `double`.

Example: `'modifiedByColumn',6`

`postImportFcn` — Custom post-import callback
string scalar | character vector

Custom post-import callback script name to use during import, specified as a string scalar or character vector.

The script that you assign to this callback executes after you import or update requirements.

Example: `"postImportFcn","myPostImportScript"`

`preImportFcn` — Custom pre-import callback
string scalar | character vector

Custom pre-import callback script name to use during import, specified as a string scalar or character vector.

The script that you assign to this callback executes before you import or update requirements.

Example: `"preImportFcn","myPreImportScript"`

`rationale` — Attribute to map to `Rationale`
string scalar | character vector

Name of the attribute from the external document that you want to map to the `Rationale` property for the imported requirements.

Use this argument when you import from IBM Rational DOORS or custom document types.

Example: `"rationale","Requirement Rationale"`

`rationaleColumn` — Rationale Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `Rationale` property of the requirements in your requirement set, specified as a `double`.

Example: `'rationaleColumn',5`

`ReqSet` — Requirement Set
character vector

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

If the requirement set exists, the requirements import under a new Import node. If the requirement set does not exist, Requirements Toolbox creates it.

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`

`rows` — Range of rows
`double` array

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

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

`sheet` — Worksheet name
character vector

Worksheet name from Microsoft Excel workbook, specified as a character vector.

Example: `'sheet','Sheet1'`

`summaryColumn` — Summary Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `Summary` property of the requirements in your requirement set, specified as a `double`.

Example: `'summaryColumn',4`

`USDM` — USDM Format Import Option
character vector

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

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

Output Arguments

`refCount` — Imported referenced requirements count
`double`

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

`reqSetFilePath` — Requirement set file path
character vector

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

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

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

See Also
`slreq.Reference` | `createReferences`

Introduced in R2018a

slreq.importViewSettings

Import view settings

Syntax

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

Description

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

`slreq.importViewSettings(viewSettingsFile, overwriteFlag)` imports Requirements Toolbox 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

Requirements Toolbox 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 requirement set or link set

Syntax

```
myReqSet = slreq.load(reqSetArtifact)

myLinkSet = slreq.load(linkSetArtifact)

myLinkSet = slreq.load(model)
[myLinkSet,myReqSet] = slreq.load(model)
```

Description

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

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

`myLinkSet = slreq.load(model)` loads a Simulink model that contains at least one Requirements Table block, specified by `model`, and loads the associated link set into memory.

`[myLinkSet,myReqSet] = slreq.load(model)` loads a Simulink model that contains at least one Requirements Table block and loads the associated requirement set and link set into memory.

Examples

Load a Requirement Set

This example shows how to load a requirement set and return the associated `slreq.ReqSet` object.

```
rs = slreq.load("basicReqSet");
```

Input Arguments

`reqSetArtifact` — Requirement set to load
string scalar | character vector

The requirement set to load, specified as a string scalar or character vector.

`linkSetArtifact` — Link set artifact name
string scalar | character vector

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

`model` — Simulink model
string scalar | character vector

The Simulink model to load, specified as a string scalar or character vector. The model must contain at least one Requirements Table block. Each block is associated with a requirement set. See “Configure Properties of Formal Requirements”. You must include the `.slx` extension.

Output Arguments

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

The requirement 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.ReqSet` | `slreq.LinkSet` | `slreq.open` | Requirements Table

Introduced in R2018a

login

Log in to OSLC client

Syntax

```
login(myClient)
```

Description

`login(myClient)` prompts for login credentials and authenticates `myClient` with the Open Services for Lifecycle Collaboration (OSLC) server.

Note If the `login` function does not work for your service provider, try using `setHttpOptions` and `setHttpHeader` to authenticate an instance of `oslc.Client` with your OSLC service provider. The `login` function might not work with some OSLC service providers.

Examples

Create and Configure an OSLC Client for the Requirements Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Then set the service root and catalog path for the requirements management domain and the configuration query path.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'/oslc_rm/catalog');
setConfigurationQueryPath(myClient,'gc/oslc-query/configurations');
myClient
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array
```

```
    {'OSLC Plugin'                                  }
    {'Model Based Design with OSLC'                 }
    {'OSLC4RM'                                      }
    {'Interactive Testing (Requirements Management)'}

setServiceProvider(myClient,'OSLC Plugin');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)

configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }

setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient

myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/rm/oslc_rm/catalog'
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

See Also

`oslc.Client` | `setCatalogPath` | `setServer` | `setServiceRoot` | `setUser` | `setConfigurationQueryPath`

Introduced in R2021a

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 requirement set

Syntax

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

Description

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

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

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

Examples

Create Requirement Set

```
% Create requirement 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 requirement 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` — Requirement set name

character vector

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

`reqSetPath` — Requirement set path

character vector

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

Output Arguments

`newReqSet` — Created requirement set

`slreq.ReqSet` object

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

See Also

`slreq.ReqSet`

Introduced in R2018a

slreq.open

Open requirement set

Syntax

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

myReqSet = slreq.open(model)
```

Description

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

`myReqSet = slreq.open(ReqSetName)` loads the requirement set named `ReqSetName`.

`myReqSet = slreq.open(model)` loads the specified Simulink model specified by `model` and loads the requirement sets in the **Requirements Editor**. The model must contain at least one Requirements Table block.

Examples

Open a Requirement Set

This example shows how to load and open a requirement set in the **Requirements Editor** and return the associated `slreq.ReqSet` object.

```
rs = slreq.open("basicReqSet");
```

Input Arguments

`ReqSetFilePath` — Requirement set file path
string scalar | character vector

The full file path of the requirement set to be loaded, specified as a string scalar or character vector.

`ReqSetName` — Requirement set name
string scalar | character vector

The name of the requirement set to be loaded, specified as a string scalar or character vector.

`model` — Simulink model
string scalar | character vector

The Simulink model to load, specified as a string scalar or character vector. The model must contain at least one Requirements Table block. Each block is associated with a requirement set. See “Configure Properties of Formal Requirements”. You must include the `.slx` extension.

Output Arguments

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

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

See Also

`slreq.ReqSet` | **Requirements Editor** | Requirements Table

Introduced in R2018a

slreq.openRequirementsManager

Open Requirements Manager app in model

Syntax

```
slreq.openRequirementsManager(model)
```

Description

`slreq.openRequirementsManager(model)` opens the **Requirements Manager** app in the Simulink model `model` and brings the model to the front. The model must be open.

Examples

Open and Close the Requirements Manager App Programmatically

This example shows how to open and close the **Requirements Manager** app programmatically.

Open the "Requirements Definition for a Cruise Control Model" project and open the `crs_plant` model.

```
slreqCCProjectStart;
open_system("crs_plant");
```

Open the **Requirements Manager** app in the `crs_plant` model.

```
slreq.openRequirementsManager("crs_plant");
```

Close the **Requirements Manager** app in the `crs_plant` model.

```
slreq.closeRequirementsManager("crs_plant");
```

Cleanup

Clear the open requirement sets and link sets. Close the open models and the current project.

```
slreq.clear;
bdclose all;
close(currentProject);
```

Input Arguments

`model` — Simulink model

string scalar | character vector | model handle

Simulink model to open the **Requirements Manager** app in, specified as a string scalar or character vector that contains the name of the model, or a model handle.

Tips

- Use `bdroot` to get the top-level model of the current system.
- Use `get_param` and `bdroot` to get the handle for the top-level model of the current system:

  ```
  model = get_param(bdroot,"Handle");
  ```
- Open the **Requirements Editor** with `slreq.editor`.

See Also

`slreq.closeRequirementsManager` | `bdroot` | `slreq.editor` | **Requirements Editor**

Introduced in R2021a

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

queryChangeRequests

Query OSLC service provider for change requests

Syntax

```
changeRequests = queryChangeRequests(myQueryCapability)
```

Description

`changeRequests = queryChangeRequests(myQueryCapability)` returns the available change request resources in the Open Services for Lifecycle Collaboration (OSLC) service provider that is associated with the query capability `myQueryCapability`.

Examples

Query Service Provider for Change Requests

This example shows how to submit a query for change request resources with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Change Management Domain” on page 2-5, create a query capability for the change request resource type.

```
myQueryCapability = getQueryService(myClient,'ChangeRequest')
```

```
myQueryCapability =

  QueryCapability with properties:

    queryParameter: ''
            client: [1×1 oslc.Client]
         queryBase: 'https://localhost:9443/rm/views?oslc.query=true&projectURL=http...'
     resourceShape: {0×1 cell}
               dom: [1×1 matlab.io.xml.dom.Element]
             title: 'Query Capability'
      resourceType: {1×2 cell}
```

Submit a query request to the service provider for the available change request resources.

```
changeRequests = queryChangeRequests(myQueryCapability)
```

```
changeRequests =

  1×7 ChangeRequest array with properties:

    ResourceUrl
    Dirty
    IsFetched
```

```
Title
Identifier
```

Input Arguments

`myQueryCapability` — Resource query capability
`oslc.core.QueryCapability` object

OSLC resource query capability, specified as an `oslc.core.QueryCapability` object.

Output Arguments

`changeRequests` — Change request resource
`oslc.cm.ChangeRequest` object

OSLC change request resource, returned as an `oslc.cm.ChangeRequest` object.

See Also

`oslc.Client` | `oslc.cm.ChangeRequest` | `oslc.core.QueryCapability`

Introduced in R2021a

queryRequirementCollections

Query OSLC service provider for requirement collections

Syntax

```
reqCollections = queryRequirementCollections(myQueryCapability)
```

Description

`reqCollections = queryRequirementCollections(myQueryCapability)` returns the available requirement collection resources in the Open Services for Lifecycle Collaboration (OSLC) service provider that is associated with the query capability `myQueryCapability`.

Examples

Query Service Provider for Requirement Collections

This example shows how to submit a query request for requirement collection resources with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement collection resource type.

```
myQueryCapability = getQueryService(myClient,'RequirementCollection')
```

```
myQueryCapability =

  QueryCapability with properties:

    queryParameter: ''
            client: [1×1 oslc.Client]
         queryBase: 'https://localhost:9443/rm/views?oslc.query=true&projectURL=http...'
     resourceShape: {0×1 cell}
               dom: [1×1 matlab.io.xml.dom.Element]
             title: 'Query Capability'
      resourceType: {1×2 cell}
```

Submit a query request to the service provider for the available requirement collection resources.

```
reqCollections = queryRequirementCollections(myQueryCapability)
```

```
reqCollections =

  1×5 RequirementCollection array with properties:

    ResourceUrl
    Dirty
    IsFetched
```

```
Title
Identifier
```

Input Arguments

`myQueryCapability` — Resource query capability
`oslc.core.QueryCapability` object

OSLC resource query capability, specified as an `oslc.core.QueryCapability` object.

Output Arguments

`reqCollections` — Requirement collection resource
`oslc.rm.RequirementCollection` object

OSLC requirement collection resource, returned as an `oslc.rm.RequirementCollection` object.

See Also

`oslc.Client` | `oslc.cm.ChangeRequest` | `oslc.core.QueryCapability` | `queryRequirements`

Introduced in R2021a

queryRequirements

Query OSLC service provider for requirements

Syntax

```
reqs = queryRequirements(myQueryCapability)
```

Description

`reqs = queryRequirements(myQueryCapability)` returns the available requirement resources in the Open Services for Lifecycle Collaboration (OSLC) service provider that is associated with the query capability `myQueryCapability`.

Examples

Submit a Query Request with Query Capability

This example shows how to submit a query request with a configured OSLC client.

After you have created and configured an OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type.

```
myQueryCapability = getQueryService(myClient,'Requirement')
```

```
myQueryCapability =

  QueryCapability with properties:

    queryParameter: ''
            client: [1×1 oslc.Client]
         queryBase: 'https://localhost:9443/rm/views?oslc.query=true&projectURL=http...'
     resourceShape: {0×1 cell}
             title: 'Query Capability'
      resourceType: {1×2 cell}
```

Submit a query request to the service provider for the available requirement resources.

```
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign the first returned requirement resource to the variable `myReq`, then fetch the full resource properties for `myReq`. Examine the `Title` property.

```
myReq = reqs(1);
status = fetch(myReq,myClient)

status =

  StatusCode enumeration

    OK

title = myReq.Title

title =

    'Requirement 1'
```

Input Arguments

`myQueryCapability` — Resource query capability
`oslc.core.QueryCapability` object

OSLC resource query capability, specified as an `oslc.core.QueryCapability` object.

Output Arguments

`reqs` — Requirement resource
`oslc.rm.Requirement` object

OSLC requirement resource, returned as an `oslc.rm.Requirement` object.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.core.QueryCapability` | `queryRequirementCollections`

Introduced in R2021a

queryTestCases

Query OSLC service provider for test cases

Syntax

```
testCases = queryTestCases(myQueryCapability)
```

Description

`testCases = queryTestCases(myQueryCapability)` returns the available test case resources in the Open Services for Lifecycle Collaboration (OSLC) service provider that is associated with the query capability `myQueryCapability`.

Examples

Query Service Provider for Test Cases

This example shows how to submit a query request for test case resources with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test case resource type.

```
myQueryCapability = getQueryService(myClient,'TestCase')
```

```
myQueryCapability =

  QueryCapability with properties:

    queryParameter: ''
            client: [1×1 oslc.Client]
         queryBase: 'https://localhost:9443/qm/views?oslc.query=true&projectURL=http...'
     resourceShape: {0×1 cell}
               dom: [1×1 matlab.io.xml.dom.Element]
             title: 'Query Capability'
      resourceType: {1×2 cell}
```

Submit a query request to the service provider for the available test case resources.

```
testCases = queryTestCases(myQueryCapability)
```

```
testCases =

  1×5 TestCase array with properties:

    ResourceUrl
    Dirty
    IsFetched
```

```
Title
Identifier
```

Input Arguments

myQueryCapability — Resource query capability

`oslc.core.QueryCapability` object

OSLC resource query capability, specified as an `oslc.core.QueryCapability` object.

Output Arguments

testCases — Test case resource

`oslc.qm.TestCase` object

OSLC test case resource, returned as an `oslc.qm.TestCase` object.

See Also

`oslc.Client` | `oslc.cm.ChangeRequest` | `oslc.core.QueryCapability` | `queryTestExecutionRecords` | `queryTestPlans` | `queryTestResults` | `queryTestScripts`

Introduced in R2021a

queryTestExecutionRecords

Query OSLC service provider for test execution records

Syntax

```
testExecutionRecords = queryTestExecutionRecords(myQueryCapability)
```

Description

`testExecutionRecords = queryTestExecutionRecords(myQueryCapability)` returns the available test execution record resources in the Open Services for Lifecycle Collaboration (OSLC) service provider that is associated with the query capability `myQueryCapability`.

Examples

Query Service Provider for Test Execution Records

This example shows how to submit a query request for test execution record resources with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test execution record resource type.

```
myQueryCapability = getQueryService(myClient,'TestExecutionRecord')
```

```
myQueryCapability =

  QueryCapability with properties:

    queryParameter: ''
            client: [1×1 oslc.Client]
         queryBase: 'https://localhost:9443/rm/views?oslc.query=true&projectURL=http...'
     resourceShape: {0×1 cell}
               dom: [1×1 matlab.io.xml.dom.Element]
             title: 'Query Capability'
      resourceType: {1×2 cell}
```

Submit a query request to the service provider for the available test execution record resources.

```
testExecutionRecords = queryTestExecutionRecords(myQueryCapability)
```

```
testExecutionRecords =

  1×5 TestExecutionRecord array with properties:

    ResourceUrl
    Dirty
    IsFetched
```

```
Title
Identifier
```

Input Arguments

`myQueryCapability` — Resource query capability

`oslc.core.QueryCapability` object

OSLC resource query capability, specified as an `oslc.core.QueryCapability` object.

Output Arguments

`testExecutionRecords` — Test execution record resource

`oslc.qm.TestExecutionRecord` object

OSLC test execution record resource, returned as an `oslc.qm.TestExecutionRecord` object.

See Also

`oslc.Client` | `oslc.cm.ChangeRequest` | `oslc.core.QueryCapability` | `queryTestPlans` | `queryTestResults` | `queryTestCases` | `queryTestScripts`

Introduced in R2021a

queryTestPlans

Query OSLC service provider for test plans

Syntax

```
testPlans = queryTestPlans(myQueryCapability)
```

Description

`testPlans = queryTestPlans(myQueryCapability)` returns the available test plan resources in the Open Services for Lifecycle Collaboration (OSLC) service provider that is associated with the query capability `myQueryCapability`.

Examples

Query Service Provider for Test Plans

This example shows how to submit a query request for test plan resources with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test plan resource type.

```
myQueryCapability = getQueryService(myClient,'TestPlan')
```

```
myQueryCapability =

  QueryCapability with properties:

    queryParameter: ''
            client: [1×1 oslc.Client]
         queryBase: 'https://localhost:9443/rm/views?oslc.query=true&projectURL=http...'
     resourceShape: {0×1 cell}
               dom: [1×1 matlab.io.xml.dom.Element]
             title: 'Query Capability'
      resourceType: {1×2 cell}
```

Submit a query request to the service provider for the available test plan resources.

```
testPlans = queryTestPlans(myQueryCapability)
```

```
testPlans =

  1×5 TestPlan array with properties:

    ResourceUrl
    Dirty
    IsFetched
```

```
Title
Identifier
```

Input Arguments

myQueryCapability — Resource query capability

`oslc.core.QueryCapability` object

OSLC resource query capability, specified as an `oslc.core.QueryCapability` object.

Output Arguments

testPlans — Test plan resource

`oslc.qm.TestPlan` object

OSLC test plan resource, returned as an `oslc.qm.TestPlan` object.

See Also

`oslc.Client` | `oslc.cm.ChangeRequest` | `oslc.core.QueryCapability` | `queryTestExecutionRecords` | `queryTestResults` | `queryTestCases` | `queryTestScripts`

Introduced in R2021a

queryTestResults

Query OSLC service provider for test results

Syntax

```
testResults = queryTestResults(myQueryCapability)
```

Description

`testResults = queryTestResults(myQueryCapability)` returns the available test result resources in the Open Services for Lifecycle Collaboration (OSLC) service provider that is associated with the query capability `myQueryCapability`.

Examples

Query Service Provider for Test Results

This example shows how to submit a query request for test result resources with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test result resource type.

```
myQueryCapability = getQueryService(myClient,'TestResult')
```

```
myQueryCapability =

  QueryCapability with properties:

    queryParameter: ''
            client: [1×1 oslc.Client]
         queryBase: 'https://localhost:9443/rm/views?oslc.query=true&projectURL=http...'
     resourceShape: {0×1 cell}
               dom: [1×1 matlab.io.xml.dom.Element]
             title: 'Query Capability'
      resourceType: {1×2 cell}
```

Submit a query request to the service provider for the available test result resources.

```
testResults = queryTestResults(myQueryCapability)
```

```
testResults =

  1×5 TestResult array with properties:

    ResourceUrl
    Dirty
    IsFetched
```

```
Title
Identifier
```

Input Arguments

`myQueryCapability` — Resource query capability

`oslc.core.QueryCapability` object

OSLC resource query capability, specified as an `oslc.core.QueryCapability` object.

Output Arguments

`testResults` — Test result resource

`oslc.qm.TestResult` object

OSLC test result resource, returned as an `oslc.qm.TestResult` object.

See Also

`oslc.Client` | `oslc.cm.ChangeRequest` | `oslc.core.QueryCapability` | `queryTestExecutionRecords` | `queryTestPlans` | `queryTestCases` | `queryTestScripts`

Introduced in R2021a

queryTestScripts

Query OSLC service provider for test scripts

Syntax

```
testScripts = queryTestScripts(myQueryCapability)
```

Description

`testScripts = queryTestScripts(myQueryCapability)` returns the available test script resources in the Open Services for Lifecycle Collaboration (OSLC) service provider that is associated with the query capability `myQueryCapability`.

Examples

Query Service Provider for Test Scripts

This example shows how to submit a query request for test script resources with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test script resource type.

```
myQueryCapability = getQueryService(myClient,'TestScript')
```

```
myQueryCapability =

  QueryCapability with properties:

    queryParameter: ''
            client: [1×1 oslc.Client]
         queryBase: 'https://localhost:9443/rm/views?oslc.query=true&projectURL=http...'
     resourceShape: {0×1 cell}
               dom: [1×1 matlab.io.xml.dom.Element]
             title: 'Query Capability'
      resourceType: {1×2 cell}
```

Submit a query request to the service provider for the available test script resources.

```
testScripts = queryTestScripts(myQueryCapability)
```

```
testScripts =

  1×5 TestScript array with properties:

    ResourceUrl
    Dirty
    IsFetched
```

```
Title
Identifier
```

Input Arguments

`myQueryCapability` — Resource query capability

`oslc.core.QueryCapability` object

OSLC resource query capability, specified as an `oslc.core.QueryCapability` object.

Output Arguments

`testScripts` — Test script resource

`oslc.qm.TestScript` object

OSLC test script resource, returned as an `oslc.qm.TestScript` object.

See Also

`oslc.Client` | `oslc.cm.ChangeRequest` | `oslc.core.QueryCapability` | `queryTestExecutionRecords` | `queryTestPlans` | `queryTestResults` | `queryTestCases`

Introduced in R2021a

slreq.refreshCustomizations

Register Requirements Toolbox customizations

Syntax

```
slreq.refreshCustomizations
```

Description

`slreq.refreshCustomizations` searches the MATLAB path for `sl_customization.m` files and registers the requirement type and link type customizations defined in the files.

Note If Simulink is installed, this function behaves the same as `sl_refresh_customizations`. If Simulink is not installed, this function only registers Requirements Toolbox customizations and silently ignores other customizations.

Examples

Define and Register Custom Requirement and Link Types

This example shows how to define and register custom requirement types and custom link types in Requirements Toolbox™.

Create an `sl_customization` File

In MATLAB®, select the **Home** tab and click **New Script**. Copy and paste this code in the script.

```
function sl_customization(cm)
    cObj = cm.SimulinkRequirementsCustomizer;
end
```

Select the **Editor** tab and click **Save**. Save the file as `sl_customization.m`.

Define Requirements Toolbox Customizations

Define a custom requirement type called `Stakeholder` by using the `addCustomRequirementType` function. Define the custom requirement type as a subtype of the built-in `Functional` type, then provide a description for the custom requirement type. Copy and paste this code in the `sl_customization` file.

```
addCustomRequirementType(cObj,"Stakeholder",slreq.custom.RequirementType.Functional,...
    "Stakeholder functional requirements");
```

Define a custom link type as a subtype of the built-in `Relate` tpye called `Trace` by using the `addCustomLinkType` function. Define the forward and backward link direction as `Traces` and `Traced from`, respectively, then provide a description for the custom link type. Copy and paste this code in the `sl_customization` file and click **Save**.

```
addCustomLinkType(cObj,"Trace",slreq.custom.LinkType.Relate,"Traces",...
    "Traced from","General purpose link type from requirement to other item.");
```

Register the Requirements Toolbox Customizations

The updated `sl_customization` file defines the requirement type and link type customizations.

```
type sl_customization
```

```
function sl_customization(cm)
    cObj = cm.SimulinkRequirementsCustomizer;
    addCustomRequirementType(cObj,"Stakeholder",slreq.custom.RequirementType.Functional,...
    "Stakeholder functional requirements");
    addCustomLinkType(cObj,"Trace",slreq.custom.LinkType.Relate,"Traces",...
    "Traced from","General purpose link type from requirement to other item.");
end
```

Register the Requirements Toolbox customizations.

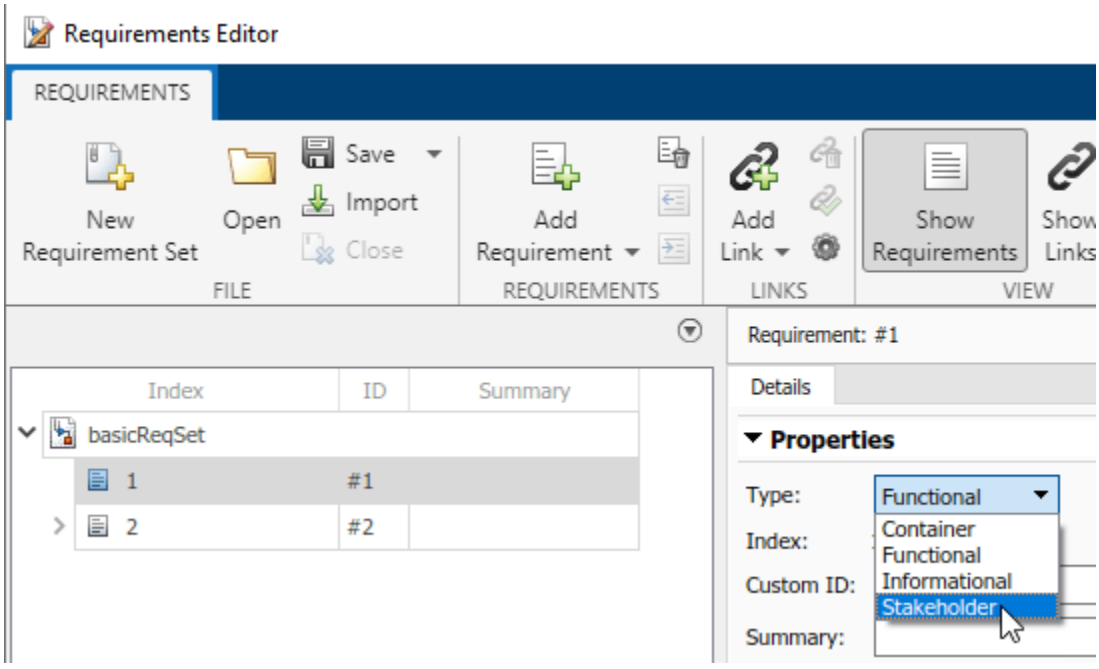
```
slreq.refreshCustomizations
```

View Customizations in the Requirements Editor

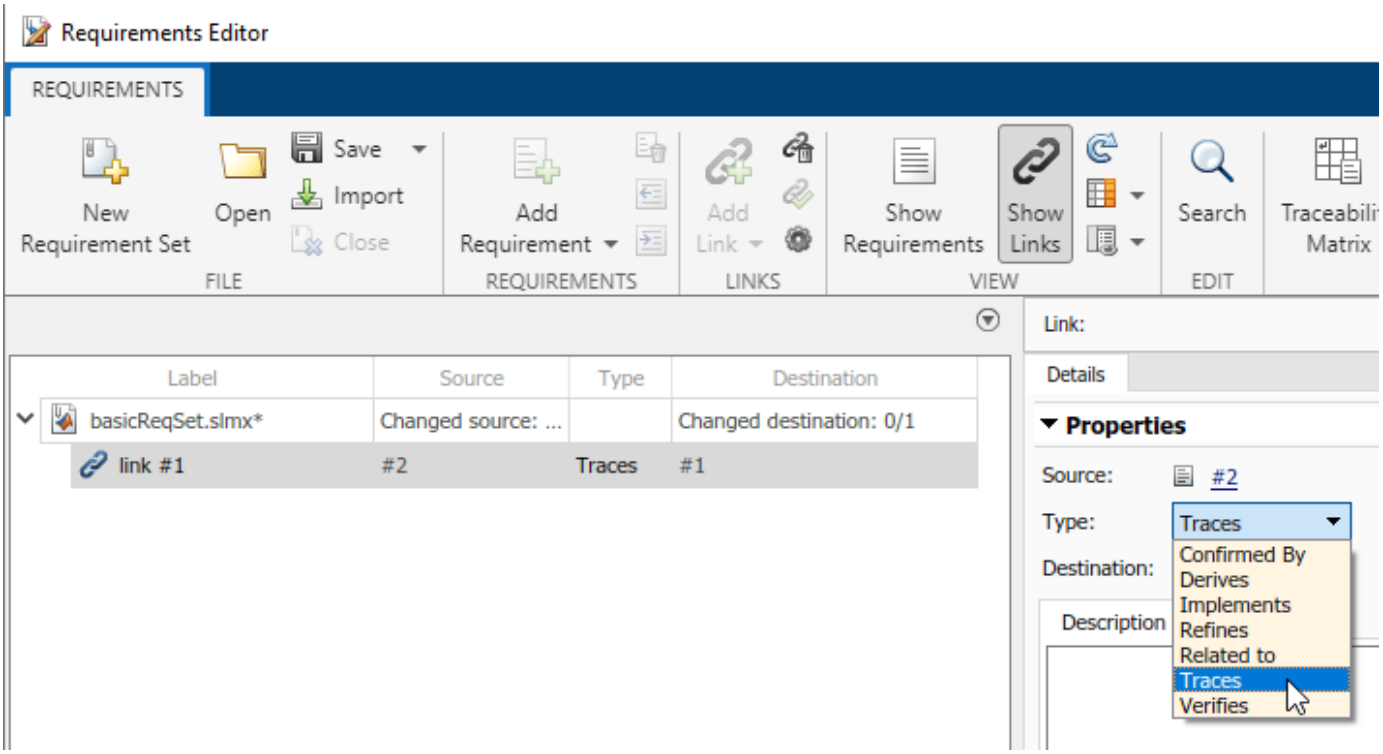
Open the `basicReqSet` requirement set in the **Requirements Editor**.

```
slreq.open("basicReqSet");
```

In the **Requirements Editor**, click **Show Requirements** and then select the requirement with index `1`. In the **Details** pane, under **Properties**, in the **Type** menu, select `Stakeholder` from the list.



Click **Show Links** and select `link #1`. In the **Details** pane, under **Properties**, in the **Type** menu, select `Traces` from the list.



See Also

Topics

`sl_refresh_customizations`

“Define Custom Requirement and Link Types”

“Register Customizations with Simulink” (Simulink)

Introduced in R2022a

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.registerNavigationFcn

Register navigation function for referenced requirements

Syntax

```
slreq.registerNavigationFcn(domain,callbackFunction)
```

Description

`slreq.registerNavigationFcn(domain,callbackFunction)` registers a navigation callback function, `callbackFunction`, for referenced requirements imported from ReqIF files that have the Domain property value equal to `domain`. Use this function to enable navigation from the **Requirements Editor** to the original requirement in a third-party requirements management tool.

Note The navigation callback function should take this form:

```
function myCustomNavigationFunction(ref)
% Enter your implementation here
end
```

The function should take the `slreq.Reference` object as an input.

Examples

Register and Get a Navigation Callback Function for Referenced Requirements Imported from ReqIF Files

This example shows how to register and get the registered navigation callback function for referenced requirements imported from ReqIF™ files.

Import the ReqIF file `mySpec.reqif` into Requirements Toolbox™.

```
count = slreq.import("mySpec.reqif");
```

Get the handle for the imported requirement set. Check the domain for the imported referenced requirements.

```
rs = slreq.find("Type","ReqSet","Name","mySpec");
topRef = children(rs);
domain = topRef.Domain
```

```
domain =
'Third-Party Tool'
```

Check if there are any currently registered navigation callback functions for the domain.

```
callback = slreq.getNavigationFcn(domain)
```

```
callback =

  0x0 empty char array
```

Register the custom navigation callback function `myNavigationFcn` for the domain. Confirm that the navigation callback function was registered.

```
slreq.registerNavigationFcn(domain,"myNavigationFunction")
callback = slreq.getNavigationFcn(domain)
```

```
callback =
'myNavigationFunction'
```

Cleanup

Clear the open requirement sets without saving. Unregister the custom navigation callback function.

```
slreq.clear;
slreq.registerNavigationFcn(domain,'');
```

Input Arguments

`domain` — Third-party requirements tool domain

string scalar | character vector

Third-party requirements tool domain for which to register the navigation callback function, specified as a string scalar or character vector.

This argument should match the Domain property value of the referenced requirement.

`callbackFunction` — Navigation callback function name

string scalar | character vector

Navigation callback function name to register, specified as a string scalar or a character vector.

Tips

- You can clear the registered navigation callback function for a domain by entering:

  ```
  slreq.registerNavigationFcn(domain,"")
  ```

- You can get the value of the Domain property for a referenced requirement at the MATLAB command prompt by entering:

  ```
  domain = myReferencedRequirement.Domain
  ```

  ```
  domain =

      'Third-Party Tool'
  ```

- You can use the template generated by Requirements Toolbox to create your navigation callback function. For more information, see “Navigate from Referenced Requirements to Requirements in Third-Party Applications”.

See Also

`slreq.getNavigationFcn` | `slreq.Reference` | `slreq.import` | **Requirements Editor**

Topics
“Navigate from Referenced Requirements to Requirements in Third-Party Applications”

Introduced in R2019a

remove

Remove resource from OSLC service provider

Syntax

```
status = remove(resource,myClient)
```

Description

`status = remove(resource,myClient)` removes the resource `resource` from the Open Services for Lifecycle Collaboration (OSLC) service provider associated with `myClient` and returns the remove success status.

Examples

Remove an Existing Requirement

This example shows how to submit a query request for requirement resources with a configured OSLC client and remove a requirement resource.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type.

```
myQueryCapability = getQueryService(myClient,'Requirement');
```

Submit a query request to the service provider for the available requirement resources.

```
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Retrieve the full resource data from the service provider for a requirement resource. Inspect the requirement resource.

```
myReq = reqs(1);
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
myReq
```

```
myReq =

  Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/resources/_72lxMWJREeup0...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Requirement'
     Identifier: '1806'
```

Remove the requirement from the service provider.

```
status = remove(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Input Arguments

`resource` — OSLC resource object

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

Output Arguments

`status` — Removal success status

`matlab.net.http.StatusCode` object

Removal success status, returned as a `matlab.net.http.StatusCode` object.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `commit` | `show` | `fetch`

Introduced in R2021a

removeLink

Remove link from local OSLC requirement resource object

Syntax

```
removeLink(reqResource,resourceURL)
```

Description

`removeLink(reqResource,resourceURL)` removes the RDF/XML element `j.0:Link` that has the `rdf:resource` attribute set to `resourceURL` from the requirement or requirement collection resource specified by `reqResource`. Use the `commit` function to apply the change to the service provider. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website and QM Resource Definitions on the Open Services for Lifecycle Collaboration (OSLC) website.

Examples

Add and Remove Links from OSLC Resources to Requirement

This example shows how to add and remove links from OSLC resources to an OSLC requirement.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type. Submit a query request to the service provider for the available requirement resources.

```
myQueryCapability = getQueryService(myClient,'Requirement');
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign one of the requirements to a variable called `myReq` and one to `linkReq`. Fetch the full resource properties for the requirements.

```
myReq = reqs(1);
linkReq = reqs(5);
fetch(myReq,myClient);
fetch(linkReq,myClient);
```

Add a link from `linkReq` to `myReq`. Confirm the link creation by getting the links for `myReq`.

```
addLink(myReq,linkReq)
links = getLinks(myReq)
```

```
links =

  1×1 cell array

    {'https://localhost:9443/rm/CA_3d5ba3752e2c489b965a3ecceffb664a'}
```

In the service provider, identify a test case to link to the requirement. Identify the resource URL of the test case and assign it to a variable called `URL`. Add a link from `URL` to `myReq`. Confirm the link creation by getting the links for `myReq`.

```
URL = 'https://localhost:9443/qm/_ibz6tGWYEeuAF8ZpKyQQtg';
addLink(myReq,URL)
links = getLinks(myReq)
```

```
links =

  1×2 cell array

    {'https://localhost:9443/rm...'}    {'https://localhost:9443/qm...'}
```

Commit the changes to the service provider.

```
status = commit(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Fetch the full resource properties for the updated requirement `myReq`.

```
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Get the resource URLs linked to `myReq`.

```
links = getLinks(myReq)
```

```
links =

  1×2 cell array

    {'https://localhost:9443/rm...'}    {'https://localhost:9443/qm...'}
```

Get the URL for the first linked resource and assign it to `URL`.

```
URL = links{1}
```

```
URL =

    'https://localhost:9443/rm/CA_3d5ba3752e2c489b965a3ecceffb664a'
```

Before removing the link from myReq, confirm that the resource URL points to the requirement that you want to remove. Create a requirement resource object and set the resource URL. Fetch the full resource properties for the requirement and inspect the requirement.

```
req = oslc.rm.Requirement;
setResourceUrl(req,URL);
status = fetch(req,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
req
```

```
ans =

  Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/CA_3d5ba3752e2c489b965a...'
          Dirty: 0
      IsFetched: 1
          Title: '[SAFe] Lifecycle Scenario Template'
     Identifier: '1165'
```

Remove the link from myReq and commit the changes to the service provider.

```
removeLink(myReq,URL)
status = commit(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Fetch the full resource properties for the updated requirement myReq.

```
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Verify the link removal by getting the URLs for the resources linked to myReq.

```
links = getLinks(myReq)
```

```
links =

  1×1 cell array
```

```
{'https://localhost:9443/qm/_ibz6tGWYEeuAF8ZpKyQQtg'}
```

Input Arguments

`reqResource` — OSLC requirement resource
`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object

OSLC requirement or requirement collection resource object, specified as an `oslc.rm.Requirement` or `oslc.rm.RequirementCollection` object.

`resourceURL` — OSLC resource URL
character vector

OSLC resource URL, specified as a character vector.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `addLink` | `removeRequirementLink` | `getLinks`

Introduced in R2021a

removeRequirementLink

Remove requirement traceability link from local OSLC test resource object

Syntax

```
removeRequirementLink(testResource,requirementURL)
```

Description

`removeRequirementLink(testResource,requirementURL)` removes the RDF/XML element `oslc_qm:validatesRequirement` that has the `rdf:resource` attribute set to `requirementURL` from the test case or test script specified by `testResource`. Use the `commit` function to apply the change to the service provider. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website and QM Resource Definitions on the Open Services for Lifecycle Collaboration (OSLC) website.

Examples

Add, Get, and Remove Traceability Links from a Test Case to a Requirement

This example shows how to add, remove, and get OSLC requirement resources linked to a test case resource with a previously configured OSLC client.

After you have created and configured an OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test case resource type.

```
myQueryCapability = getQueryService(myClient,'TestCase');
```

Submit a query request to the service provider for the available test case resources.

```
testCases = queryTestCases(myQueryCapability)
```

```
testCases =

  1×5 TestCase array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Retrieve the requirement resources linked to one of the test cases. Fetch the resource properties from the service provider for the test case.

```
myTestCase = testCases(1);
fetch(myTestCase,myClient);
reqs = getRequirementLinks(myTestCase)
```

```
reqs =
```

```
    Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/resources/_aQ1gRg8bEeuLWbFe'
          Dirty: 1
      IsFetched: 0
          Title: ''
     Identifier: ''
```

Remove the existing link to the requirement resource from the test case resource. Commit the changes to the service provider.

```
removeRequirementLink(myTestCase,reqs.ResourceUrl);
status = commit(myTestCase,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

To add a link to a requirement, in the OSLC service provider, locate the requirement resource that you want to link to the test case resource. Identify the resource URL. Create a variable `URL` and set the value of the variable to the requirement URL that you found in the service provider.

```
URL = 'https://localhost:9443/rm/resources/_oJNtgWrqEeup0a6t';
```

Create a traceability link between the requirement resource and the test case. Commit the change to the service provider.

```
addRequirementLink(myTestCase,URL);
status = commit(myTestCase,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

View the test case in the system browser.

```
show(myTestCase)
```

Input Arguments

`testResource` — OSLC test resource

`oslc.qm.TestCase` object | `oslc.qm.TestScript` object

OSLC test resource, specified as an `oslc.qm.TestCase` or `oslc.qm.TestScript` object.

`requirementURL` — Requirement resource URL

character vector

Requirement or requirement collection resource URL, specified as a character vector.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.qm.TestCase` | `oslc.qm.TestScript` | `oslc.rm.RequirementCollection` | `addRequirementLink` | `getRequirementLinks`

Introduced in R2021a

removeResourceProperty

Remove resource property from local OSLC resource object

Syntax

```
removeResourceProperty(resource,propertyName,rdfResource)
```

Description

`removeResourceProperty(resource,propertyName,rdfResource)` removes the RDF/XML element with the name `propertyName` and `rdf:resource` attribute set to `rdfResource` from the locally stored RDF/XML data for the Open Services for Lifecycle Collaboration (OSLC) resource specified by `resource`. Use the `commit` function to apply the change to the service provider. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website.

Examples

Add, Get, and Remove Properties from OSLC Resources

This example shows how to add, get, and remove properties from an existing OSLC requirement resource.

Create and configure the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3. Then query the service provider for requirements and assign an `oslc.rm.Requirement` object to the variable `myReq` as described in “Submit a Query Request with Query Capability” on page 1-198.

Retrieve the full resource data from the service provider for the requirement resource `myReq`.

```
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

The requirement `myReq` has a linked requirement with an `implementedBy` relationship. Get the `rdf:resource` value for the `oslc_rm:implementedBy` property for the requirement resource `myReq`.

```
linkedReq = getResourceProperty(myReq,'oslc_rm:implementedBy')
```

```
linkedReq =

  1×1 cell array

    {'https://localhost:9443/rm/resources/_72lxMWJREeup0...'}
```

Change the relationship between the linked requirement and `myReq` from `implementedBy` to `decomposedBy`. Remove the `oslc_rm:implementedBy` property and add an `oslc_rm:decomposedBy` property.

```
removeResourceProperty(myReq,'oslc_rm:implementedBy',linkedReq)
addResourceProperty(myReq,'oslc_rm:decomposedBy',linkedReq)
```

Get the text contents for the `dcterms:title` property.

```
title = getProperty(myReq,'dcterms:title')
```

```
title =

    'My New Requirement'
```

Change the title to `My New Requirement (Edited)`. Confirm the changes.

```
setProperty(myReq,'dcterms:title','My New Requirement (Edited)')
title = getProperty(myReq,'dcterms:title')
```

```
title =

    'My New Requirement (Edited)'
```

Add a new text property to the requirement with the tag `dcterms:description`. Confirm the changes.

```
addTextProperty(myReq,'dcterms:description', ...
    'My new requirement edited using the MATLAB OSLC client.');
desc = getProperty(myReq,'dcterms:description')
```

```
desc =

    'My new requirement created using the MATLAB OSLC client.'
```

Commit the changes to the service provider.

```
status = commit(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

View the resource that you edited in the system browser.

```
show(myReq)
```

Input Arguments

`resource` — OSLC resource object

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

`propertyName` — OSLC resource property name
character vector

OSLC resource property name, specified as a character vector.

`rdfResource` — OSLC resource property `rdf:resource` attribute
character array

OSLC resource property `rdf:resource` attribute, specified as a character array.

Tips

- For information about OSLC resource properties, see these pages on the OSLC website:
  - RM Resource Definitions
  - QM Resource Definitions
  - CM Resource Definitions

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `getResourceProperty` | `addResourceProperty`

External Websites
RDF 1.1 XML Syntax

Introduced in R2021a

removeRow

Package: `slreq.modeling`

Remove Requirements Table block row

Syntax

```
removeRow(reqTable,row)
```

Description

`removeRow(reqTable,row)` removes the row specified by `row` in the Requirements Table block, specified by `reqTable`.

Examples

Remove Requirement from Requirements Table Block

Retrieve the requirements in a Requirements Table block and remove the first requirement.

```
requirements = getRequirementRows(reqTable);
removeRow(reqTable,requirements(1));
```

Remove Assumption from Requirements Table Block

Retrieve the assumptions in a Requirements Table block and remove the first assumption.

```
assumptions = getAssumptionRows(reqTable);
removeRow(reqTable,assumptions(1));
```

Input Arguments

`reqTable` — Requirements Table block

`RequirementsTable` object

Requirements Table block, specified as a `RequirementsTable` object.

`row` — Requirement or assumption

`RequirementRow` object | `AssumptionRow` object

Requirement or assumption in a Requirements Table block, specified as a `RequirementRow` or `AssumptionRow` object. To retrieve the row, use `getRequirementRows`, `getAssumptionRows`, or `getChildren`.

Tips

- If you remove a row that has children, the child rows are also removed.

See Also

Functions
`getRequirementRows` | `getAssumptionRows` | `addAssumptionRow` | `addRequirementRow`

Objects
`RequirementsTable`

Introduced in R2022a

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

Requirements Editor

Introduced in R2018b

rmi

Interact programmatically with Requirements Management Interface

Syntax

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

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

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

rmi setup
rmi register linktypename
rmi unregister linktypename
rmi linktypelist

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

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

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

Description

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`rmi('projectreport')` generates a Requirements Traceability report in HTML format for the current project. The top-level 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 ...\slrequirements-e
```

```
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 `.slreqx` 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

This example shows how to add a requirement link to an existing example model, and save the model requirements traceability data in an external file.

Open the `slvnvdemo_powerwindowController` model.

```
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 = '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 `.slmx` 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 model.

```
close_system('slvnvdemo_powerwindowController');
```

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.
```

Tips

- You can also use `updateDocUri` to update the link destinations in a link set from an old external document to a new external document.

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` | `updateDocUri`

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 a list of the `Name,Value` pairs that correspond to the Requirements Management Interface (RMI) preference names and accepted values.

`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 Keywords to Selection-Based Requirements Links

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

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

```
previousVal = rmipref('SelectionLinkKeyword','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 `'SelectionLinkKeyword'` preference.

```
previousVal =

     ''
```

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

```
currentVal = rmipref('SelectionLinkKeyword')
```

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

```
currentVal =

design,reqts
```

These user keywords 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'` | `'FilterRequireKeywords'` | `'CustomSettings'` | ...

RMI preference name, specified as the corresponding `Name` character vector listed in “Name-Value Pair Arguments” on page 1-254.

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`

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`

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`

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

Preference to enable filtering by user keyword keywords, specified as a numeric or logical `1` (`true`) or `0` (`false`). When you filter by user keyword keywords, you can include or exclude subsets of requirements links in highlighting or reports. You can specify user keyword keywords for requirements links filtering in the `'FilterRequireKeywords'` and `'FilterExcludeKeywords'` preferences. For more information about requirements filtering, see “Filter Requirements with User Keywords”.

Data Types: `logical`

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

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

Data Types: `char`

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

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

Data Types: `char`

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

Preference to disable labels of requirements links with designated user keywords, specified as a numeric or logical `1` (`true`) or `0` (`false`). When set to `true`, if a requirement link has a user keyword designated in `'FilterExcludeKeywords'` or `'FilterRequireKeywords'`, that requirements link

will be disabled in the Requirements context menu. For more information about requirements filtering, see “Filter Requirements with User Keywords”.

Data Types: `logical`

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

Preference to filter Model Advisor requirements consistency checks with designated user keywords, 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 keywords designated in `'FilterRequireKeywords'` and excludes requirements links with user keywords designated in `'FilterExcludeKeywords'`. For more information about requirements filtering, see “Filter Requirements with User Keywords”.

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`

`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`

`LoginProvider` — Custom authentication callback function for integration with web-based services
character vector

Custom authentication callback function for integration with web-based services, specified as a character vector.

If your network requires advanced authentication that the default authentication process does not support, use this argument to register a custom authentication callback function. For example, if you register a custom authentication callback function and then use `slreq.dngConfigure`, the function calls your custom function to authenticate the connection with the IBM DOORS Next server. For more information, see the “Tips” on page 1-67 section of `slreq.dngConfigure`.

Note The custom authentication callback function should take this form:

```
function [success,cookies] = myCustomLoginProvider(server,options)
% Provide your implementation here
end
```

The custom authentication function should return two arguments: success status and cookies received from the server.

Example: `"myCustomLoginProvider"`

`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`

`OslcUseGlobalConfig` — Preference to allow global configuration
`false` or `0` (default) | `true` or `1`

Preference to allow global configurations when configuring a MATLAB session for integration with IBM DOORS Next, specified as a numeric or logical `1 (true)` or `0 (false)`.

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`

`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`

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

Preference to list user keywords 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 keywords specified for each requirement link. For more information about requirements filtering, see "Filter Requirements with User Keywords".

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`

`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`

`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`

`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`

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

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

Data Types: `char`

`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`

`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`

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 1-254.

`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 1-254.

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 backlinks to models from requirements documents

Syntax

```
count = rmiref.removeRefs(doc_type)
```

Description

`count = rmiref.removeRefs(doc_type)` removes all backlinks to models from the currently active external requirements document of type `doc_type`, and returns the number of backlinks removed. For more information about backlinks, see “Manage Navigation Backlinks in External Requirements Documents”.

Note You can only remove backlinks from one external document at a time.

Examples

Remove Backlinks from a Microsoft Word Document

This example shows how to remove backlinks from a Microsoft Word document.

Open the “Redirect Direct Links to Imported Requirements Programmatically” on page 3-76 example.

```
openExample(['slrequirements/' ...
    'RedirectDirectLinksToImportedRequirementsByAPIExample'])
```

Open the `FuelSysWithReqLinks` model.

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

The model contains direct links to these documents:

- `FuelSysDesignDescription.docx`
- `FuelSysRequirementsSpecification.docx`
- `FuelSysTestScenarios.xlsx`

Open the `FuelSysDesignDescription.docx` document.

```
open("FuelSysDesignDescription.docx")
```

Remove the backlinks from the `FuelSysDesignDescription.docx` document.

```
count = rmiref.removeRefs("word")
```

```
Removing Simulink references from the current document ...

Current document: fuelsysdesigndescription.docx
```

```
Total references: 6

Remove all Simulink references? y/n

y

Removing ...

count =

     6
```

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

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

Input Arguments

doc_type — External requirements document type

`"Word"` | `"Excel"` | `"DOORS"`

External requirements document type, specified as `"Word"`, `"Excel"`, or `"DOORS"`.

Note The document type `"DOORS"` refers to IBM Rational DOORS modules. You cannot use this function to remove backlinks from IBM DOORS Next modules.

Output Arguments

count — Number of backlinks removed

double

Number of backlinks removed from the external document, returned as a double.

See Also

`rmiref.insertRefs`

Topics

“Manage Navigation Backlinks in External Requirements Documents”

Introduced in R2011a

rmitag

Manage user keywords for requirements links

Syntax

```
rmitag(model, 'list')
rmitag(model, 'add', keyword)
rmitag(model, 'add', keyword, doc_pattern)
rmitag(model, 'delete', keyword)
rmitag(model, 'delete', keyword, doc_pattern)
rmitag(model, 'replace', keyword, new_keyword)
rmitag(model, 'replace', keyword, new_keyword, doc_pattern)
rmitag(model, 'clear', keyword)
rmitag(model, 'clear', keyword, doc_pattern)
```

Description

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

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

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

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

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

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

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

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

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

keyword

Character vector specifying user keyword for requirements links.

doc_pattern

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

new_keyword

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

Examples

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

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

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

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

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

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

Change all instances of the user keyword `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 Keywords and Requirements Filtering”

Introduced in R2010a

RptgenRMI.doorsAttribs

IBM Rational DOORS attributes in requirements report

Syntax

```
settings = RptgenRMI.doorsAttribs('show')
tf = RptgenRMI.doorsAttribs('default')
tf = RptgenRMI.doorsAttribs(Name,Value)
```

Description

`settings = RptgenRMI.doorsAttribs('show')` returns the DOORS attribute report settings. The listed attributes are included in generated requirements reports.

`tf = RptgenRMI.doorsAttribs('default')` restores the default requirements report settings for DOORS attributes. The function returns `1` if the settings are changed. The default settings are:

- Explicitly include the system attributes `Object Heading` and `Object Text`
- Include all other system attributes and user-defined attributes
- Omit the system attribute `Created Thru`
- Omit system attributes with empty string values
- Omit system attributes that are false

`tf = RptgenRMI.doorsAttribs(Name,Value)` specifies which DOORS attributes to include in generated requirements reports. The function returns `1` if the settings are changed without error.

Note This function sets settings used when generating reports for requirements in IBM Rational DOORS. These settings are not applied for generated reports for requirements in IBM Rational DOORS Next.

Examples

Show the DOORS Attribute Report Settings

```
settings = RptgenRMI.doorsAttribs('show')
```

```
settings = 5x1 cell
    {'Object Heading' }
    {'Object Text'    }
    {'$AllAttributes$'}
    {'$NonEmpty$'     }
    {'-Created Thru'  }
```

Restore Default DOORS Attributes Report Settings

If you change the settings for which DOORS attributes to include in the requirements report, you can restore the default settings.

Change the settings by omitting all attributes other than those that are explicitly included in the report. Show the changed settings.

```
tf = RptgenRMI.doorsAttribs('type','none');

Excluding attributes...

settings = RptgenRMI.doorsAttribs('show')

settings = 4x1 cell
    {'Object Text'      }
    {'-Created Thru'    }
    {'+Last Modified By'}
    {'+Last Modified On'}
```

Restore the settings to default. Show the default settings.

```
tf = RptgenRMI.doorsAttribs('default');
settings = RptgenRMI.doorsAttribs('show')

settings = 5x1 cell
    {'Object Heading' }
    {'Object Text'    }
    {'$AllAttributes$'}
    {'$NonEmpty$'     }
    {'-Created Thru'  }
```

The default settings are:

- Explicitly include the system attributes `Object Heading` and `Object Text`
- Include all other system attributes and user-defined attributes
- Omit the system attribute `Created Thru`
- Omit system attributes with empty string values
- Omit system attributes that are false

Include or Omit DOORS Attributes from the Requirements Report by Specifying Type

Specify that generated requirements reports will include only user-defined attributes.

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

Including user attributes...
```

Show the settings.

```
settings = RptgenRMI.doorsAttribs('show')
```

```
settings = 6x1 cell
    {'Object Text'      }
    {'$NonEmpty$'       }
    {'-Created Thru'    }
    {'+Last Modified By'}
    {'+Last Modified On'}
    {'$UserAttributes$' }
```

Explicitly Include or Omit DOORS Attributes from the Requirements Report

Include the `Last Modified By` and `Last Modified On` attributes.

```
tf = RptgenRMI.doorsAttribs('add','Last Modified By');
```

```
Adding Last Modified By...
```

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

```
Adding Last Modified On...
```

Omit the `Object Heading` attribute from the requirements report.

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

```
Removing Object Heading...
```

Show the Current Settings

```
settings = RptgenRMI.doorsAttribs('show')
```

```
settings = 6x1 cell
    {'Object Text'      }
    {'$AllAttributes$'  }
    {'$NonEmpty$'       }
    {'-Created Thru'    }
    {'+Last Modified By'}
    {'+Last Modified On'}
```

Include or Omit Empty User-Defined DOORS Attributes from the Requirements Report

Include empty user-defined attributes in the requirements report.

```
tf = RptgenRMI.doorsAttribs('nonempty','off')
```

```
NonEmpty filter off...
```

```
tf = logical
   1
```

Show the current settings.

```
settings = RptgenRMI.doorsAttribs('show')
```

```
settings = 5x1 cell
    {'Object Text'      }
    {'-Created Thru'    }
    {'+Last Modified By'}
    {'+Last Modified On'}
    {'$UserAttributes$' }
```

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'type','all'`

`type` — Types of attributes to include or omit in report

`'all'` | `'user'` | `'none'`

Types of DOORS attributes to include or omit from the report, specified as `'all'`, `'user'`, or `'none'`.

Example: `'type','all'`

`add` — Attributes to add to report

character array

Attributes to add to the generated report, specified as a character array.

Example: `'add','Last Modified By'`

Note The entered character array should be the same as a DOORS predefined system attribute or user-defined attribute.

`remove` — Attributes to remove from report

character array

Attributes to omit from the generated report, specified as a character array.

Example: `'remove','Object Heading'`

Note The entered character array should be the same as a DOORS predefined system attribute or user-defined attribute.

`nonempty` — Include or omit empty attributes

`'on'` | `'off'`

Whether to include or omit empty user-defined attributes in the report, specified as `'on'` or `'off'`. Empty system-defined attributes are always excluded.

Example: `'nonempty','on'`

Output Arguments

`settings` — Current DOORS attribute report settings

cell array

Current DOORS attribute report settings, returned as a cell array.

`tf` — Changed settings success status

`1` | `0`

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

See Also

`rmi`

Introduced in R2011b

setCatalogPath

Set catalog path for OSLC client

Syntax

```
setCatalogPath(myClient,path)
```

Description

`setCatalogPath(myClient,path)` sets the OSLC client `myClient` to the catalog path specified by `path`.

Examples

Create and Configure an OSLC Client for the Requirements Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Then set the service root and catalog path for the requirements management domain and the configuration query path.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'/oslc_rm/catalog');
setConfigurationQueryPath(myClient,'gc/oslc-query/configurations');
myClient
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array

    {'OSLC Plugin'                                  }
    {'Model Based Design with OSLC'                 }
    {'OSLC4RM'                                      }
    {'Interactive Testing (Requirements Management)'}
```

```
setServiceProvider(myClient,'OSLC Plugin');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)
```

```
configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }
```

```
setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient
```

```
myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/rm/oslc_rm/catalog'
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

`path` — OSLC catalog path

character vector

OSLC catalog path in the specified server and domain, specified as a character vector.

Example: `'/oslc_rm/catalog'`

See Also

`oslc.Client` | `setServer` | `setServiceRoot` | `login` | `setUser`

Introduced in R2021a

setConfigurationContext

Set configuration context for OSLC client

Syntax

```
setConfigurationContext(myClient,configName)
```

Description

`setConfigurationContext(myClient,configName)` sets the OSLC client `myClient` to the configuration context specified by `configName`.

Examples

Create and Configure an OSLC Client for the Requirements Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Then set the service root and catalog path for the requirements management domain and the configuration query path.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'/oslc_rm/catalog');
setConfigurationQueryPath(myClient,'gc/oslc-query/configurations');
myClient
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array

    {'OSLC Plugin'                                  }
    {'Model Based Design with OSLC'                 }
    {'OSLC4RM'                                      }
    {'Interactive Testing (Requirements Management)'}
```

```
setServiceProvider(myClient,'OSLC Plugin');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)
```

```
configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }
```

```
setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient
```

```
myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/rm/oslc_rm/catalog'
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

`configName` — Configuration context name

character vector

Configuration context name to set the OSLC client to, specified as a character vector.

See Also

`oslc.Client` | `getConfigurationContextNames` | `login` | `setServiceProvider` | `getServiceProviderNames` | `setConfigurationQueryPath`

Introduced in R2021a

setConfigurationQueryPath

Set configuration query path for OSLC client

Syntax

```
setConfigurationQueryPath(myClient,path)
```

Description

`setConfigurationQueryPath(myClient,path)` sets the OSLC client `myClient` to the configuration context query path specified by `path`.

Examples

Create and Configure an OSLC Client for the Requirements Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Then set the service root and catalog path for the requirements management domain and the configuration query path.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'/oslc_rm/catalog');
setConfigurationQueryPath(myClient,'gc/oslc-query/configurations');
myClient
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array

    {'OSLC Plugin'                                  }
    {'Model Based Design with OSLC'                 }
    {'OSLC4RM'                                      }
    {'Interactive Testing (Requirements Management)'}
```

```
setServiceProvider(myClient,'OSLC Plugin');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)
```

```
configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }
```

```
setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient
```

```
myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/rm/oslc_rm/catalog'
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

`path` — OSLC configuration query path

character vector

OSLC configuration query path in the specified server and domain, specified as a character vector.

Example: `'gc/oslc-query/configurations'`

See Also

`oslc.Client` | `setConfigurationContext` | `login` | `setServiceProvider` | `getServiceProviderNames`

Introduced in R2021a

setCustomLoginProvider

Register custom authentication callback function to OSLC client

Syntax

```
setCustomLoginProvider(myClient,authenticationFunction)
```

Description

`setCustomLoginProvider(myClient,authenticationFunction)` registers a custom authentication callback function, `authenticationFunction`, for the OSLC client object `myClient`. You can use this function to authenticate an OSLC client object on networks that require advanced authentication that the default authentication process does not support.

Note The custom authentication callback function should take this form:

```
function [success,cookies] = myCustomLoginProvider(server,options)
% Provide your implementation here
end
```

The custom authentication function should return two arguments: success status and cookies received from the server.

Examples

Authenticate a Client that Requires an Advanced Authentication

This example shows how to authenticate an OSLC client by using a custom authentication function and custom HTTP options.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the server URL, service root, and catalog path for your service provider.

```
setServer(myClient,'http://example.com');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'oslc/services/catalog');
```

Create and enter the user credentials by using the `matlab.net.http.Credentials` class with a basic `matlab.net.http.AuthenticationScheme` object.

```
creds = matlab.net.http.Credentials('Username','jdoe','Password', ...
'Password1234','scheme',matlab.net.http.AuthenticationScheme.Basic);
```

Create custom HTTP options by using the `matlab.net.http.HTTPOptions` class constructor. Set the `Credentials` property and certificate information for the custom HTTP options.

```
opts = matlab.net.http.HTTPOptions('Credentials',creds, ...
    'VerifyServerName', false, 'CertificateFilename', '')

opts =

  HTTPOptions with properties:

          MaxRedirects: 20
        ConnectTimeout: 10
              UseProxy: 1
              ProxyURI: []
          Authenticate: 1
           Credentials: [1×1 matlab.net.http.Credentials]
    UseProgressMonitor: 0
           SavePayload: 0
       ConvertResponse: 1
        DecodeResponse: 1
    ProgressMonitorFcn: []
   CertificateFilename: ""
      VerifyServerName: 0
           DataTimeout: Inf
       ResponseTimeout: Inf
      KeepAliveTimeout: Inf
```

Specify the custom HTTP options to authenticate the OSLC client `myClient`.

```
setHttpOptions(myClient,opts);
```

Create a custom authentication callback function called `myCustomLoginProvider`.

```
function [success,cookies] = myCustomLoginProvider(server,options)

end
```

Register the custom authentication callback function with the OSLC client object.

```
setCustomLoginProvider(myClient,myCustomLoginProvider);
```

Authenticate the OSLC client object.

```
login(myClient);
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

`authenticationFunction` — Custom authentication callback function name

character vector

Custom authentication callback function name, specified as a character vector.

Example: `'myCustomLoginProvider'`

Tips

- If your authentication process requires a particular set of HTTP options, you can either:
  - Construct a `matlab.net.http.HTTPOptions` object and assign it to your OSLC client by using `setHttpOptions`, which passes the HTTP options to your custom authentication callback function.
  - Construct the HTTP options internally in your custom authentication callback function.
- If you want to preconfigure the login process with credentials or use a particular authentication scheme, you can create a `matlab.net.http.Credentials` object and include it with a `matlab.net.http.HTTPOptions` object that you assign to the OSLC client object. For more information, see “Server Authentication”.

  Note Depending on the authentication method used by your server, your custom authentication callback function might also have to satisfy authentication requirements. For example, you might have to mimic the form-based authentication required by your authentication server.

- You can unregister all callbacks from an OSLC client object `myClient` by entering:

```
setCustomLoginProvider(myClient,'');
```

See Also

`oslc.Client` | `setHttpHeader` | `setHttpOptions` | `login` | `getCustomLoginProvider`

Topics

“Server Authentication”

Introduced in R2021b

setHttpHeader

Set HTTP header for OSLC client

Syntax

```
setHttpHeader(myClient,header)
```

Description

`setHttpHeader(myClient,header)` assigns the custom HTTP header `header` to the OSLC client `myClient`. The custom header allows for HTTP methods. For more information, see `matlab.net.http.HeaderField` methods.

Examples

Set Custom HTTP Header

This example shows how to set a custom HTTP header for a configured OSLC client.

Create a custom HTTP header by using the `matlab.net.http.HeaderField` class constructor.

```
header = matlab.net.http.HeaderField('Content-Type','text/plain')
```

```
header =

  HeaderField with properties:

     Name: "Content-Type"
    Value: "text/plain"
```

After you have created and configured an OSLC client as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, assign the header to the OSLC client `myClient`.

```
setHttpHeader(myClient,header);
```

Input Arguments

`myClient` — OSLC client
`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

`header` — Custom HTTP header
`matlab.net.http.HeaderField` object

Custom HTTP header, specified as a `matlab.net.http.HeaderField` object.

Tips

- If your OSLC service provider requires a cookie for repeated requests, you can include an authenticated cookie in your `matlab.net.http.HeaderField` object by using `matlab.net.http.field.CookieField`.

See Also

`matlab.net.http.HeaderField` | `oslc.Client` | `setHttpOptions`

Introduced in R2021a

setHttpOptions

Set HTTP options for OSLC client

Syntax

```
setHttpOptions(myClient,opts)
```

Description

`setHttpOptions(myClient,opts)` assigns the custom HTTP options `opts` to the OSLC client `myClient`.

Examples

Authenticate a Client that Requires Custom HTTP Options

This example shows how to authenticate an OSLC client by using custom HTTP options.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the server URL, service root and catalog path for your service provider.

```
setServer(myClient,'http://example.com');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'oslc/services/catalog');
```

Create and enter the user credentials by using the `matlab.net.http.Credentials` class with a basic `matlab.net.http.AuthenticationScheme` object.

```
creds = matlab.net.http.Credentials('Username','jdoe','Password', ...
'Password1234','scheme',matlab.net.http.AuthenticationScheme.Basic);
```

Create custom HTTP options by using the `matlab.net.http.HTTPOptions` class constructor. Set the `Credentials` property for the custom HTTP options.

```
opts = matlab.net.http.HTTPOptions('Credentials',creds)
```

```
opts =

  HTTPOptions with properties:

          MaxRedirects: 20
        ConnectTimeout: 10
              UseProxy: 1
              ProxyURI: []
          Authenticate: 1
           Credentials: [1×1 matlab.net.http.Credentials]
    UseProgressMonitor: 0
           SavePayload: 0
```

```
        ConvertResponse: 1
         DecodeResponse: 1
     ProgressMonitorFcn: []
    CertificateFilename: "default"
       VerifyServerName: 1
            DataTimeout: Inf
        ResponseTimeout: Inf
       KeepAliveTimeout: Inf
```

Specify the custom HTTP options to authenticate the OSLC client `myClient`.

```
setHttpOptions(myClient,opts);
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

`opts` — Custom HTTP options

`matlab.net.http.HTTPOptions` object

Custom HTTP header, specified as a `matlab.net.http.HTTPOptions` object.

Tips

- You can use a `matlab.net.http.HTTPOptions` object for custom authentication for an `oslc.Client` object. For more information, see “Server Authentication”.

See Also

`matlab.net.http.HTTPOptions` | `oslc.Client` | `setHttpHeader`

Topics

“Use HTTP with MATLAB”

Introduced in R2021a

setProperty

Set local contents of text property for OSLC resource object

Syntax

```
setProperty(resource,propertyName,textContents)
```

Description

`setProperty(resource,propertyName,textContents)` sets the text contents of the RDF/XML element `propertyName` to the value specified by `textContents` in the locally stored RDF/XML data for the Open Services for Lifecycle Collaboration (OSLC) resource specified by `resource`. Use the `commit` function to apply the change to the service provider. For more information about RDF/XML elements, see An XML Syntax for RDF on the World Wide Web Consortium website.

Examples

Add, Get, and Remove Properties from OSLC Resources

This example shows how to add, get, and remove properties from an existing OSLC requirement resource.

Create and configure the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3. Then query the service provider for requirements and assign an `oslc.rm.Requirement` object to the variable `myReq` as described in “Submit a Query Request with Query Capability” on page 1-198.

Retrieve the full resource data from the service provider for the requirement resource `myReq`.

```
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

The requirement `myReq` has a linked requirement with an `implementedBy` relationship. Get the `rdf:resource` value for the `oslc_rm:implementedBy` property for the requirement resource `myReq`.

```
linkedReq = getResourceProperty(myReq,'oslc_rm:implementedBy')
```

```
linkedReq =

  1×1 cell array

    {'https://localhost:9443/rm/resources/_72lxMWJREeup0...'}
```

Change the relationship between the linked requirement and `myReq` from `implementedBy` to `decomposedBy`. Remove the `oslc_rm:implementedBy` property and add an `oslc_rm:decomposedBy` property.

```
removeResourceProperty(myReq,'oslc_rm:implementedBy',linkedReq)
addResourceProperty(myReq,'oslc_rm:decomposedBy',linkedReq)
```

Get the text contents for the `dcterms:title` property.

```
title = getProperty(myReq,'dcterms:title')
```

```
title =

    'My New Requirement'
```

Change the title to `My New Requirement (Edited)`. Confirm the changes.

```
setProperty(myReq,'dcterms:title','My New Requirement (Edited)')
title = getProperty(myReq,'dcterms:title')
```

```
title =

    'My New Requirement (Edited)'
```

Add a new text property to the requirement with the tag `dcterms:description`. Confirm the changes.

```
addTextProperty(myReq,'dcterms:description', ...
    'My new requirement edited using the MATLAB OSLC client.');
desc = getProperty(myReq,'dcterms:description')
```

```
desc =

    'My new requirement created using the MATLAB OSLC client.'
```

Commit the changes to the service provider.

```
status = commit(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

View the resource that you edited in the system browser.

```
show(myReq)
```

Input Arguments

`resource` — OSLC resource object

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

`propertyName` — OSLC resource property name
character vector

OSLC resource property name, specified as a character vector.

`textContents` — OSLC resource text contents
character vector

OSLC resource text content, specified as a character vector.

Tips

- For information about OSLC resource properties, see these pages on the OSLC website:
  - RM Resource Definitions
  - QM Resource Definitions
  - CM Resource Definitions

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `addTextProperty` | `getProperty`

External Websites
RDF 1.1 XML Syntax

Introduced in R2021a

setQueryParameter

Set query parameter for OSLC query service

Syntax

```
setQueryParameter(myQueryCapability,parameter)
```

Description

`setQueryParameter(myQueryCapability,parameter)` sets a query parameter for the query capability `myQueryCapability`.

Note The query parameter is only applied for one query. After you submit a query, the query parameter is automatically cleared from the query capability.

Examples

Set a Query Parameter for a Query Capability

This example shows how to set a query parameter for a query capability.

After you have created and configured an OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type.

```
myQueryCapability = getQueryService(myClient,'Requirement')
```

```
myQueryCapability =

  QueryCapability with properties:

    queryParameter: ''
            client: [1×1 oslc.Client]
         queryBase: 'https://localhost:9443/rm/views?oslc.query=true&projectURL=http...'
     resourceShape: {0×1 cell}
               dom: [1×1 matlab.io.xml.dom.Element]
             title: 'Query Capability'
      resourceType: {1×2 cell}
```

Set a query parameter for the query capability. Inspect the query capability `queryParameter` property.

```
setQueryParameter(myQueryCapability,'?oslc.select=oslc_rm:requirement');
param = myQueryCapability.queryParameter
```

```
param =

    '?oslc.select=oslc_rm:requirement'
```

Input Arguments

`myQueryCapability` — Resource query capability

`oslc.core.QueryCapability` object

OSLC resource query capability, specified as an `oslc.core.QueryCapability` object.

`parameter` — Query condition search parameter

character vector

OSLC query condition search parameter, specified as a character vector.

For more information, see Query Parameters on the OSLC website.

Tips

- For information about query syntaxes, see Open Services for Lifecycle Collaboration Core Specification Version 2.0 Query Syntax on the OSLC website.

See Also

`oslc.Client` | `oslc.core.QueryCapability`

External Websites

OSLC Query Parameters

Introduced in R2021a

setRDF

Set RDF content for local OSLC resource object

Syntax

```
setRDF(resource,rdfContent)
```

Description

`setRDF(resource,rdfContent)` sets the XML/RDF data to the content specified by `rdfContent` for the resource specified by `resource`. Use the `commit` function to apply the change to the service provider. For more information, see RDF classes and properties in OSLC on the Open Services for Lifecycle Collaboration (OSLC) website.

Examples

Get and Set RDF Content for Requirement Resource

This example shows how to get and set the RDF content of an OSLC requirement resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type.

```
myQueryCapability = getQueryService(myClient);
```

Submit a query request to the service provider for the available requirement resources.

```
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Fetch the full resource properties for a single requirement resource. Inspect the title of the requirement.

```
myReq = reqs(1);
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration
```

```
    OK
```

```
title = myReq.Title
```

```
title =

    'My New Requirement'
```

Get the locally stored RDF content of the requirement resource.

```
rdfContent = getRDF(myReq)
```

```
rdfContent =

    '<?xml version="1.0" encoding="UTF-8" standalone="no" ?><rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:oslc="http://open-services.net/ns/core#"
xmlns:oslc_rm="http://open-services.net/ns/rm#">
          <oslc_rm:Requirement>
         <dcterms:title>My New
Requirement</dcterms:title><oslc:instanceShape
rdf:resource="https://example.com/shapes/oslc-requirement-version1"/>
</oslc_rm:Requirement>
     </rdf:RDF>'
```

Copy and paste the `rdfContent` text into a new variable `newRDF`. Edit the text contents for the `dcterms:title` property to `My New Requirement (Edited)`.

```
newRDF = ['<?xml version="1.0" encoding="UTF-8" ' ...
'standalone="no" ?><rdf:RDF ' ...
'xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" ' ...
'xmlns:dcterms="http://purl.org/dc/terms/" ' ...
'xmlns:oslc="http://open-services.net/ns/core#" ' ...
'xmlns:oslc_rm="http://open-services.net/ns/rm#">' ...
'<oslc_rm:Requirement><dcterms:title>' ...
'My New Requirement (Edited)</dcterms:title>' ...
'<oslc:instanceShape rdf:resource=' ...
'"https://example.com/shapes/oslc-requirement-version1"/>' ...
'</oslc_rm:Requirement></rdf:RDF>']
```

Set the RDF content of the requirement to the variable `newRDF`. Inspect the requirement title.

```
setRDF(myReq,newRDF);
title = myReq.Title
```

```
title =

    'My New Requirement (Edited)'
```

Commit the changes to the service provider.

```
status = commit(newReq,myClient)
```

```
status =

  StatusCode enumeration
```

```
OK
```

Input Arguments

`resource` — OSLC resource object

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

`rdfContent` — RDF resource data

character vector

RDF data for OSLC resource, specified as a character vector.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `getRDF`

External Websites

RDF 1.1 XML Syntax

Introduced in R2021a

setResourceUrl

Set resource URL for local OSLC resource object

Syntax

```
setResourceUrl(resource,URL)
```

Description

`setResourceUrl(resource,URL)` sets the `ResourceUrl` property of the resource specified by `resource` to the existing resource URL specified by `URL`.

Examples

Set OSLC Requirement Resource URL

This example shows how to associate an Open Services for Lifecycle Collaboration (OSLC) requirement resource object in MATLAB with an existing OSLC requirement resource in the service provider.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a new requirement resource by creating an instance of the `oslc.rm.Requirement` class.

```
myReq = oslc.rm.Requirement
```

```
myReq =
  Requirement with properties:

    ResourceUrl: ''
          Dirty: 0
      IsFetched: 0
          Title: ''
     Identifier: ''
```

In the OSLC service provider, locate the requirement resource that you want to associate with the object in MATLAB. Identify the resource URL, then create a variable `URL` and set the value of the variable to the resource URL.

```
URL = 'https://localhost:9443/rm/resources/_oJNtgWrqEeup0a6t';
```

Set the resource URL for the requirement object `myReq`. Inspect the requirement.

```
setResourceUrl(myReq,URL);
myReq
```

```
myReq =

  Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/resources/_oJNtgWrqEeup0a6t'
```

```
          Dirty: 1
       IsFetched: 0
           Title: ''
      Identifier: ''
```

Retrieve the full resource data from the service provider for the requirement resource and inspect the resource.

```
fetch(myReq,myClient);
myReq
```

```
myReq =

  Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/resources/_oJNtgWrqEeup0a6t'
          Dirty: 0
       IsFetched: 1
           Title: '[SAFe] Lightweight Business Case Template'
      Identifier: '1172'
```

Open the requirement resource in the system browser.

```
show(newReq)
```

Input Arguments

`resource` — OSLC resource object

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

URL — Existing resource URL

character vector

Existing resource URL, specified as a character vector.

Tips

- Use this function when you have the resource URL for an OSLC resource and want to access the properties or links of the resource in MATLAB.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `show` | `fetch`

Introduced in R2021a

setServer

Set server URL for OSLC client

Syntax

```
setServer(myClient,serverURL)
```

Description

`setServer(myClient,serverURL)` sets the OSLC client `myClient` to the server URL specified by `serverURL`.

Examples

Create and Configure an OSLC Client for the Requirements Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Then set the service root and catalog path for the requirements management domain and the configuration query path.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'/oslc_rm/catalog');
setConfigurationQueryPath(myClient,'gc/oslc-query/configurations');
myClient
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array

    {'OSLC Plugin'                                  }
    {'Model Based Design with OSLC'                 }
    {'OSLC4RM'                                      }
    {'Interactive Testing (Requirements Management)'}
```

```
setServiceProvider(myClient,'OSLC Plugin');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)
```

```
configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }
```

```
setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient
```

```
myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/rm/oslc_rm/catalog'
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

`serverURL` — OSLC server URL

character vector

OSLC server URL to set the OSLC client to, specified as a character vector.

See Also

`oslc.Client` | `setCatalogPath` | `setServiceRoot` | `login` | `setUser`

Introduced in R2021a

setServiceProvider

Set service provider for OSLC client

Syntax

```
setServiceProvider(myClient,providerName)
```

Description

`setServiceProvider(myClient,providerName)` sets the OSLC client `myClient` to the service provider specified by `providerName`.

Examples

Create and Configure an OSLC Client for the Requirements Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Then set the service root and catalog path for the requirements management domain and the configuration query path.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'/oslc_rm/catalog');
setConfigurationQueryPath(myClient,'gc/oslc-query/configurations');
myClient
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array

    {'OSLC Plugin'                                }
    {'Model Based Design with OSLC'               }
    {'OSLC4RM'                                    }
    {'Interactive Testing (Requirements Management)'}
```

```
setServiceProvider(myClient,'OSLC Plugin');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)
```

```
configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }
```

```
setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient
```

```
myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/rm/oslc_rm/catalog'
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

`providerName` — OSLC service provider name

character vector

OSLC service provider name to set the client to, specified as a character array.

See Also

`oslc.Client` | `getConfigurationContextNames` | `setConfigurationContext` | `login` | `getServiceProviderNames` | `setConfigurationQueryPath`

Introduced in R2021a

setServiceRoot

Set service root for OSLC client

Syntax

```
setServiceRoot(myClient,root)
```

Description

`setServiceRoot(myClient,root)` sets the OSLC client `myClient` to the service root specified by `root`.

Examples

Create and Configure an OSLC Client for the Requirements Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Then set the service root and catalog path for the requirements management domain and the configuration query path.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'/oslc_rm/catalog');
setConfigurationQueryPath(myClient,'gc/oslc-query/configurations');
myClient
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array

    {'OSLC Plugin'                                  }
    {'Model Based Design with OSLC'                 }
    {'OSLC4RM'                                      }
    {'Interactive Testing (Requirements Management)'}
```

```
setServiceProvider(myClient,'OSLC Plugin');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)
```

```
configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }
```

```
setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient
```

```
myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/rm/oslc_rm/catalog'
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

`root` — OSLC service root

character vector

OSLC service root, specified as a character vector.

See Also

`oslc.Client` | `setCatalogPath` | `setServer` | `login` | `setUser`

Introduced in R2021a

setUser

Set user for OSLC client

Syntax

```
setUser(myClient,userName)
```

Description

`setUser(myClient,userName)` sets the OSLC client `myClient` to the user specified by `userName`.

Examples

Create and Configure an OSLC Client for the Requirements Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Then set the service root and catalog path for the requirements management domain and the configuration query path.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'/oslc_rm/catalog');
setConfigurationQueryPath(myClient,'gc/oslc-query/configurations');
myClient
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array

    {'OSLC Plugin'                                }
    {'Model Based Design with OSLC'               }
    {'OSLC4RM'                                    }
    {'Interactive Testing (Requirements Management)'}
```

```
setServiceProvider(myClient,'OSLC Plugin');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)
```

```
configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }
```

```
setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient
```

```
myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/rm/oslc_rm/catalog'
```

Input Arguments

`myClient` — OSLC client

`oslc.Client` object

OSLC client, specified as an `oslc.Client` object.

`userName` — OSLC user name

character vector

OSLC user name, specified as a character vector.

See Also

`oslc.Client` | `setCatalogPath` | `setServer` | `setServiceRoot` | `login`

Introduced in R2021a

show

View OSLC resource in system browser

Syntax

`show(resource)`

Description

`show(resource)` opens the `ResourceUrl` associated with `resource` in the system browser.

Examples

Create a New Requirement

This example shows how to submit a creation request for a new requirement resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a creation factory for the requirement resource type.

```
myCreationFactory = getCreationFactory(myClient,'Requirement');
```

Use the creation factory to create a new requirement resource with the title `My New Requirement`. Retrieve the full resource data from the service provider for the requirement resource and inspect the resource.

```
newReq = createRequirement(myCreationFactory,'My New Requirement');
fetch(newReq,myClient);
newReq
```

```
newReq =

  Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/resources/_72lxMWJREeup0...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Requirement'
     Identifier: '1806'
```

Open the requirement resource in the system browser by using the `show` function.

```
show(newReq)
```

Input Arguments

`resource` — OSLC resource object

`oslc.rm.Requirement` object | `oslc.rm.RequirementCollection` object | `oslc.cm.ChangeRequest` object | ...

OSLC resource object, specified as one of these objects:

- `oslc.cm.ChangeRequest`
- `oslc.qm.TestCase`
- `oslc.qm.TestExecutionRecord`
- `oslc.qm.TestPlan`
- `oslc.qm.TestResult`
- `oslc.qm.TestScript`
- `oslc.rm.Requirement`
- `oslc.rm.RequirementCollection`

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `fetch` | `commit` | `remove`

Introduced in R2021a

showAssumptionColumn

Package: `slreq.modeling`

Show Precondition column in Assumptions tab

Syntax

```
showAssumptionColumn(reqTable)
```

Description

`showAssumptionColumn(reqTable)` shows the **Precondition** column in the **Assumptions** tab of the Requirements Table block, `reqTable`.

Examples

Show the Precondition Column in a Requirements Table Block

Find the Requirements Table block in a model by using `slreq.modeling.find`.

```
reqTable = slreq.modeling.find("myModel");
```

Show the **Precondition** column in the **Assumptions** tab.

```
showAssumptionColumn(reqTable);
```

Input Arguments

`reqTable` — Requirements Table block

`RequirementsTable` object

Requirements Table block, specified as a `RequirementsTable` object.

See Also

Objects
`RequirementsTable`

Functions
`hideAssumptionColumn` | `showRequirementColumn` | `hideRequirementColumn`

Introduced in R2022a

showRequirementColumn

Package: `slreq.modeling`

Show columns in Requirements tab

Syntax

```
showRequirementColumn(reqTable,column)
```

Description

`showRequirementColumn(reqTable,column)` shows the column type specified by `column` in the **Requirements** tab of the Requirements Table block, `reqTable`.

Examples

Show the Postcondition Columns in a Requirements Table Block

Find the Requirements Table block in a model by using `slreq.modeling.find`.

```
reqTable = slreq.modeling.find("myModel");
```

Show the **Postcondition** columns in the **Requirements** tab.

```
showRequirementColumn(reqTable,"postconditions");
```

Input Arguments

`reqTable` — Requirements Table block

`RequirementsTable` object

Requirements Table block, specified as a `RequirementsTable` object.

`column` — Column type

`"duration"` | `"actions"` | `"postconditions"`

Column type to be shown, specified as `"duration"`, `"actions"`, or `"postconditions"`. Use this argument to show the **Duration**, **Action**, or **Postcondition** columns, respectively.

Data Types: `enumerated`

See Also

Objects

`RequirementsTable`

Functions

`hideRequirementColumn` | `showAssumptionColumn` | `hideAssumptionColumn`

Introduced in R2022a

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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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

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

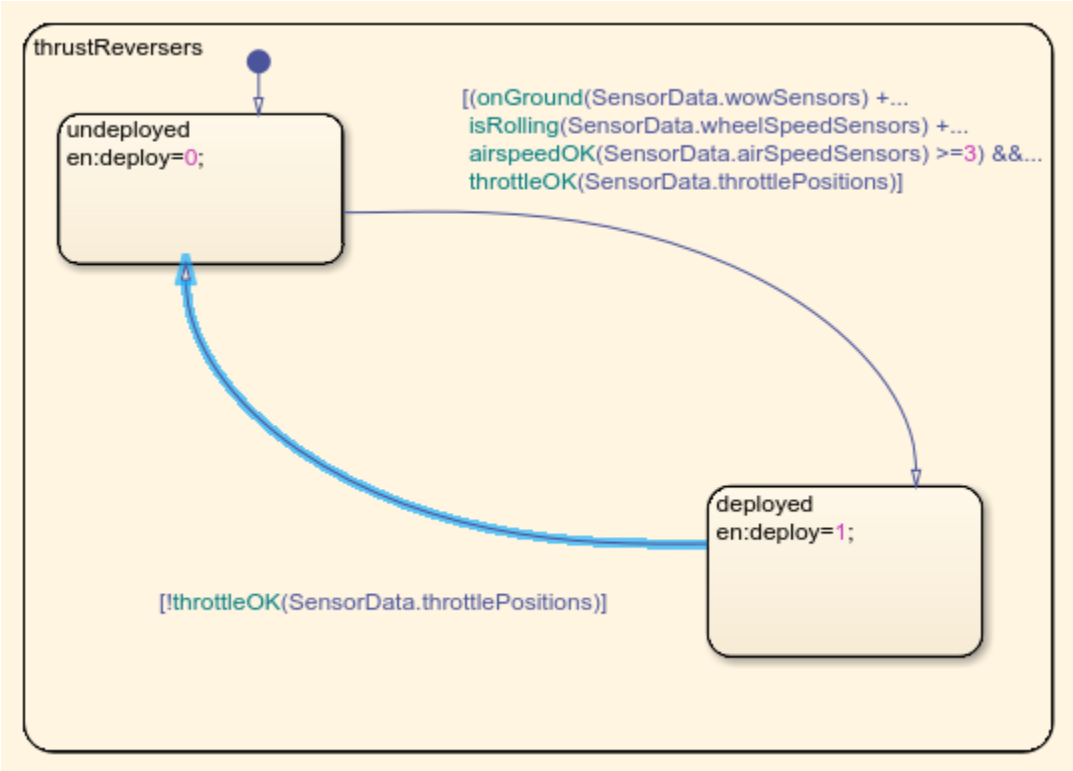
This example shows how to navigate to a link source.

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))
```



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 Requirements Toolbox.

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.LinkSet` | `slreq.Link`

Topics

“Use Command-Line API to Update or Repair Requirements Links”

Introduced in R2018a

view

View OSLC dialog in system browser

Syntax

```
view(myDialog)
```

Description

`view(myDialog)` opens the Open Services for Lifecycle Collaboration dialog `myDialog` in the system browser.

Examples

Get and View OSLC User Interface Dialogs

This example shows how to get and view an OSLC user interface dialog for a configured OSLC client.

After you have created and configured an OSLC client as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, get the available user interface dialogs in the requirements management domain of the client `myClient`.

```
dialogs = getDialog(myClient)
```

```
dialogs =

  1×4 Dialog array with properties:

    dialog
    hintWidth
    hintHeight
    title
    resourceType
```

Examine the properties of one of the dialogs. From the `title`, determine the resource type and if the dialog is for creating or selecting resources.

```
myDialog = dialogs(1);
title = myDialog.title
```

```
title =

    'Requirement Creation'
```

Open the dialog in a browser.

```
view(myDialog)
```

Input Arguments

myDialog — OSLC user interface dialog

`oslc.core.Dialog` object

OSLC user interface dialog, specified as an `oslc.core.Dialog` object.

See Also

`oslc.core.Dialog` | `oslc.Client` | `getDialog`

Introduced in R2021a

Classes

oslc.Client

Client to integrate with OSLC providers

Description

Use an `oslc.Client` object to integrate with an Open Services for Lifecycle Collaboration (OSLC) service provider. Specify the service provider properties on the object, then use the object functions to set your user name and log in to the server. You can then use `oslc.core.CreationFactory` and `oslc.core.QueryCapability` objects to create and query resources in the OSLC service provider.

Creation

Syntax

```
myClient = oslc.Client
```

Description

`myClient = oslc.Client` returns an OSLC client object.

Properties

`ServiceProvider` — OSLC service provider name
character array

OSLC service provider name, specified as a character array.

`ConfigurationContext` — Service provider configuration context name
character array

Service provider configuration context name, specified as a character array.

`CatalogUrl` — Service provider catalog URL
character array

Service provider catalog URL, specified as a character array.

Example: `'https://localhost:9443/qm/oslc_qm/catalog'`

Object Functions

getConfigurationContextNames	Get configuration context names from OSLC service provider
getCreationFactory	Get OSLC creation service object
getCustomLoginProvider	Get registered custom authentication callback function name for OSLC client
getDialog	Get user interface dialogs from OSLC service provider
getQueryService	Get OSLC query service object
getServer	Get server URL for OSLC client

getServiceProviderNames	Get service providers for OSLC client
getUser	Get user for OSLC client
login	Log in to OSLC client
setCatalogPath	Set catalog path for OSLC client
setConfigurationContext	Set configuration context for OSLC client
setConfigurationQueryPath	Set configuration query path for OSLC client
setCustomLoginProvider	Register custom authentication callback function to OSLC client
setHttpHeader	Set HTTP header for OSLC client
setHttpOptions	Set HTTP options for OSLC client
setServer	Set server URL for OSLC client
setServiceProvider	Set service provider for OSLC client
setServiceRoot	Set service root for OSLC client
setUser	Set user for OSLC client

Examples

Create and Configure an OSLC Client for the Requirements Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the requirements management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Then set the service root and catalog path for the requirements management domain and the configuration query path.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'rm');
setCatalogPath(myClient,'/oslc_rm/catalog');
setConfigurationQueryPath(myClient,'gc/oslc-query/configurations');
myClient
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array

    {'OSLC Plugin'                                  }
    {'Model Based Design with OSLC'                 }
    {'OSLC4RM'                                      }
    {'Interactive Testing (Requirements Management)'}
```

```
setServiceProvider(myClient,'OSLC Plugin');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)
```

```
configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }
```

```
setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient
```

```
myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/rm/oslc_rm/catalog'
```

Create and Configure an OSLC Client for the Quality Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the quality management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Set the service root and catalog path for the quality management domain.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'qm');
setCatalogPath(myClient,'/oslc_qm/catalog');
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array

    {'OSLC Plugin (Quality Management)'                }
```

```
    {'Model Based Design with OSLC (Quality Management)'}
    {'OSLC4RM (Quality Management)'                      }
    {'Interactive Testing (Quality Management)'          }

setServiceProvider(myClient,'OSLC Plugin (Quality Management)');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)

configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }

setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient

myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin (Quality Management)'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/qm/oslc_qm/catalog'
```

Create and Configure an OSLC Client for the Change Management Domain

This example shows how to create an OSLC client in MATLAB and configure the client to connect to an OSLC service provider for the change management domain.

Create the OSLC client.

```
myClient = oslc.Client;
```

Set the user and server URL for your service provider. Set the service root and catalog path for the change management domain.

```
setUser(myClient,'jdoe');
setServer(myClient,'https://localhost:9443');
setServiceRoot(myClient,'ccm');
setCatalogPath(myClient,'/oslc/workitems/catalog');
```

Log in to the client and enter your credentials when prompted.

```
login(myClient);
```

Get the available service providers in the specified catalog path and service root. Set the OSLC client to the desired service provider.

```
providers = getServiceProviderNames(myClient)
```

```
providers =

  4×1 cell array

    {'OSLC Plugin (Change Management)'                 }
    {'Model Based Design with OSLC (Change Management)'}
    {'OSLC4RM (Change Management)'                     }
    {'Interactive Testing (Change Management)'         }
```

```
setServiceProvider(myClient,'OSLC Plugin (Change Management)');
```

If applicable, get the available configuration contexts. Set the OSLC client to the desired configuration context.

```
configurations = getConfigurationContextNames(myClient)
```

```
configurations =

  2×1 cell array

    {'Initial Development'}
    {'Initial Baseline'   }
```

```
setConfigurationContext(myClient,'Initial Development');
```

Inspect the client properties.

```
myClient
```

```
myClient =

  Client with properties:

         ServiceProvider: 'OSLC Plugin (Change Management)'
    ConfigurationContext: 'Initial Development'
              CatalogUrl: 'https://localhost:9443/cm/oslc_cm/catalog'
```

See Also

`oslc.core.CreationFactory` | `oslc.core.QueryCapability` | `oslc.core.Dialog` | `oslc.rm.Requirement` | `oslc.qm.TestCase` | `oslc.cm.ChangeRequest`

External Websites

Open Services for Lifecycle Collaboration

Introduced in R2021a

oslc.cm.ChangeRequest

Change request resource for OSLC change management domain

Description

The `oslc.cm.ChangeRequest` object represents change request resources in the change management domain of the Open Services for Lifecycle Collaboration (OSLC) service provider. After creating and configuring `oslc.Client` and `oslc.core.QueryCapability` objects, query the service provider for available change request resources by using the `queryChangeRequests` function.

Creation

Create an `oslc.cm.ChangeRequest` object by using the `createChangeRequest` function.

Properties

`ResourceUrl` — Resource navigation URL
character array

Navigation URL for the change request resource, specified as a character array.

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

Indicator for uncommitted changes to the change request resource, specified as a logical `1`or `0` where:

- `1` indicates the change request resource has uncommitted changes.
- `0` indicates the change request resource has no uncommitted changes.

Data Types: `logical`

`IsFetched` — Resource fetch status
`0` | `1`

Change request resource fetch status, specified as a logical `1` or `0` where:

- `1` indicates the change request resource is fetched.
- `0` indicates the change request resource is not fetched.

Data Types: `logical`

`Title` — Change request title
character array

Change request title, specified as a character array.

`Identifier` — Change request resource identifier
character array

OSLC change request resource identifier, specified as a character array.

Object Functions

addResourceProperty	Add resource property to local OSLC resource object
addTextProperty	Add text property to local OSLC resource object
commit	Send local changes to OSLC service provider
fetch	Retrieve full resource data from OSLC service provider
getProperty	Get local contents of text property from OSLC resource object
getRDF	Get resource RDF/XML data from OSLC resource object
getResourceProperty	Get local contents of resource property from OSLC resource object
remove	Remove resource from OSLC service provider
removeResourceProperty	Remove resource property from local OSLC resource object
setProperty	Set local contents of text property for OSLC resource object
setRDF	Set RDF content for local OSLC resource object
setResourceUrl	Set resource URL for local OSLC resource object
show	View OSLC resource in system browser

Examples

Edit a Change Request and Commit Changes

This example shows how to submit a query request for change request resources with a configured OSLC client, edit an existing change request resource, and commit the changes to the service provider.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Change Management Domain” on page 2-5, create a query capability for the change request resource type.

```
myQueryCapability = getQueryService(myClient,'ChangeRequest');
```

Submit a query request to the service provider for the available change request resources.

```
changeRequests = queryChangeRequests(myQueryCapability)
```

```
changeRequests =

  1×6 ChangeRequest array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign a change request resource to the variable `myCR`. Retrieve the full resource data from the service provider for the change request resource. Examine the `Title` property.

```
myCR = changeRequests(1);
status = fetch(myCR,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
title = myCR.Title
```

```
title =

    'Change Request 1'
```

Edit the change request title and commit the change to the service provider.

```
myCR.Title = 'My New Change Request Title';
status = commit(myCR,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Open the change request resource in the system browser by using the `show` function.

```
show(myChangeRequest)
```

Create a New Change Request

This example shows how to submit a creation request for a new change request resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Change Management Domain” on page 2-5, create a creation factory for the change request resource type.

```
myCreationFactory = getCreationFactory(myClient,'ChangeRequest');
```

Use the creation factory to create a new change request resource with the title `My New Change Request`. Retrieve the full resource data from the service provider for the change request resource and inspect the resource.

```
newCR = createChangeRequest(myCreationFactory,'My New Change Request');
fetch(newCR,myClient);
newCR
```

```
newCR =

  ChangeRequest with properties:

    ResourceUrl: 'https://localhost:9443/ccm/resource/itemName/...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Change Request'
     Identifier: '204'
```

Open the change request resource in the system browser by using the `show` function.

```
show(newCR)
```

See Also

`oslc.core.CreationFactory` | `oslc.core.QueryCapability` | `oslc.Client` | `oslc.qm.TestCase` | `oslc.rm.Requirement` | `queryChangeRequests` | `createChangeRequest`

External Websites

Open Services for Lifecycle Collaboration
Resource ChangeRequest

Introduced in R2021a

oslc.core.CreationFactory

OSLC service provider creation factory

Description

Use `oslc.core.CreationFactory` object functions to create resources in an Open Services for Lifecycle Collaboration (OSLC) service provider. After creating and configuring an `oslc.Client`, you can create a creation factory object for the service provider specified in the client object.

Creation

Create an `oslc.core.CreationFactory` object by using `getCreationFactory`.

Properties

`client` — Associated OSLC Client
`oslc.Client` object

OSLC client associated with the creation factory, specified as an `oslc.Client` object.

`creation` — Creation factory resource URI
character vector

Creation factory resource URI, specified as a character vector.

`resourceShape` — Resource URI for RDF representation of specified resource type
cell array

Resource URI for the RDF representation of the expected contents of the specified resource type, specified as a cell array.

Example: `{'https://localhost:9443/rm/types/_4zFVsRL5EeuLWbFL3e4vrw'}`

`title` — Creation factory object title
character array

Creation factory object title, returned as a character array.

`resourceType` — Resource type to create
cell array

Resource type to create in the OSLC service provider, specified as a cell array.

Object Functions

create	Create resource in OSLC service provider
createChangeRequest	Create change request in OSLC service provider
createRequirement	Create requirement in OSLC service provider

createRequirementCollection	Create requirement collection in OSLC service provider
createTestCase	Create test case in OSLC service provider
createTestExecutionRecord	Create test execution record in OSLC service provider
createTestPlan	Create test plan in OSLC service provider
createTestResult	Create test result in OSLC service provider
createTestScript	Create test script in OSLC service provider

Examples

Create All Available Creation Factories for an OSLC Client

This example shows how to create all available creation factories for a previously configured OSLC client.

After you have created and configured an OSLC client as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create all available creation factories for the client `myClient`.

```
myCreationFactory = getCreationFactory(myClient)
```

```
myCreationFactory =

  1×8 CreationFactory array with properties:

    client
    creation
    resourceShape
    title
    resourceType
```

Examine the creation factory `resourceType` to determine which creation factory you want to use.

```
myCreationFactory(8).resourceType
```

```
ans =

  1×1 cell array

    {'http://open-services.net/ns/rm#Requirement'}
```

Submit a Creation Request by using a Creation Factory

This example shows how to submit a creation request by using a creation factory with a previously configured OSLC client.

After you have created and configured an OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a creation factory for the requirement resource type.

```
myCreationFactory = getCreationFactory(myClient,'Requirement')
```

```
myCreationFactory =

  CreationFactory with properties:
```

```
          client: [1×1 oslc.Client]
        creation: 'https://localhost:9443/rm/requirementFactory?projectURL=https%3A...'
    resourceShape: {1×22 cell}
           title: 'Requirement Creation Factory'
     resourceType: {'http://open-services.net/ns/rm#Requirement'}
```

Create a new requirement resource by using a creation factory and name the resource `My New Requirement`. Fetch the full resource properties for the requirement resource. Then commit the changes to the service provider.

```
newReq = createRequirement(myCreationFactory,'My New Requirement');
status = fetch(newReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
status = commit(newReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

View the resource that you created in the service provider.

```
show(newReq)
```

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `getCreationFactory`

External Websites

Open Services for Lifecycle Collaboration
Creation Factories

Introduced in R2021a

oslc.core.Dialog

OSLC service provider user interface dialog

Description

The `oslc.core.Dialog` objects represent user interface dialogs from an Open Services for Lifecycle Collaboration (OSLC) service provider. After creating and configuring an `oslc.Client` object, query the service provider for available user interface dialogs by using the `getDialog` object function.

Properties

`dialog` — Dialog URL
character array

User interface dialog URL, returned as a character array.

`hintWidth` — User interface width
character array

User interface width in pixels, specified as a character array.

`hintHeight` — User interface height
character array

User interface height in pixels, specified as a character array.

`title` — Dialog title
character array

User interface dialog title, returned as a character array.

`resourceType` — OSLC resource type
cell array

Resource type to select or create in user interface dialog, specified as a cell array.

Object Functions

view View OSLC dialog in system browser

Examples

Get and View OSLC User Interface Dialogs

This example shows how to get and view an OSLC user interface dialog for a configured OSLC client.

After you have created and configured an OSLC client as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, get the available user interface dialogs in the requirements management domain of the client `myClient`.

```
dialogs = getDialog(myClient)

dialogs =

  1×4 Dialog array with properties:

    dialog
    hintWidth
    hintHeight
    title
    resourceType
```

Examine the properties of one of the dialogs. From the `title`, determine the resource type and if the dialog is for creating or selecting resources.

```
myDialog = dialogs(1);
title = myDialog.title

title =

    'Requirement Creation'
```

Open the dialog in a browser.

```
view(myDialog)
```

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `oslc.core.QueryCapability` | `getDialog`

External Websites

Open Services for Lifecycle Collaboration
Delegated User Interface Dialogs

Introduced in R2021a

oslc.core.QueryCapability

OSLC service provider query capability

Description

Use `oslc.core.QueryCapability` object functions to query resources in an Open Services for Lifecycle Collaboration (OSLC) service provider. After creating and configuring an `oslc.Client`, you can create a query capability object for the service provider specified in the Client object.

Creation

Create an `oslc.core.QueryCapability` object by using `getQueryService`.

Properties

`queryParameter` — Additional query capability parameters
character array

Additional query parameters defined in query capability object, specified as a character array.

For more information, see Query Parameters in the OSLC Core Specification Version 2.0 Query Syntax.

Example: `'?oslc.select=oslc_qm:testResult'`

`client` — Associated OSLC Client
`oslc.Client` object

OSLC client associated with the query capability, specified as an `oslc.Client` object.

`queryBase` — Query capability resource URI
character vector

Query capability resource URI, specified as a character vector.

`resourceShape` — Resource URI for RDF representation of specified resource type
cell array

Resource URI for the RDF representation of the expected contents of the specified resource type, specified as a cell array.

Example: `{'https://localhost:9443/rm/types/_4zFVsRL5EeuLWbFL3e4vrw'}`

`title` — Query capability object title
character array

Query capability object title, specified as a character array.

`resourceType` — Resource type to query
cell array

Resource type to query the OSLC client for, specified as a cell array.

Object Functions

queryChangeRequests	Query OSLC service provider for change requests
queryRequirementCollections	Query OSLC service provider for requirement collections
queryRequirements	Query OSLC service provider for requirements
queryTestCases	Query OSLC service provider for test cases
queryTestExecutionRecords	Query OSLC service provider for test execution records
queryTestPlans	Query OSLC service provider for test plans
queryTestResults	Query OSLC service provider for test results
queryTestScripts	Query OSLC service provider for test scripts
setQueryParameter	Set query parameter for OSLC query service

Examples

Create All Available Query Capabilities for a Given Client

This example shows how to create all available query capabilities for a configured OSLC client.

After you have created and configured an OSLC client as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create all available query capabilities for the client `myClient`.

```
myQueryCapability = getQueryService(myClient)
```

```
myQueryCapability =

  1×4 QueryCapability array with properties:

    queryParameter
    client
    queryBase
    resourceShape
    title
    resourceType
```

Examine the query capability `resourceType` to determine which query capability you want to use.

```
myQueryCapability(3).resourceType(2)
```

```
ans =

  1×1 cell array

    {'http://open-services.net/ns/rm#Requirement'}
```

Submit a Query Request with Query Capability

This example shows how to submit a query request with a configured OSLC client.

After you have created and configured an OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type.

```
myQueryCapability = getQueryService(myClient,'Requirement')
```

```
myQueryCapability =

  QueryCapability with properties:

    queryParameter: ''
            client: [1×1 oslc.Client]
         queryBase: 'https://localhost:9443/rm/views?oslc.query=true&projectURL=http...'
     resourceShape: {0×1 cell}
             title: 'Query Capability'
      resourceType: {1×2 cell}
```

Submit a query request to the service provider for the available requirement resources.

```
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign the first returned requirement resource to the variable `myReq`, then fetch the full resource properties for `myReq`. Examine the `Title` property.

```
myReq = reqs(1);
status = fetch(myReq,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
title = myReq.Title
```

```
title =

    'Requirement 1'
```

Tips

- For information about query syntaxes, see Open Services for Lifecycle Collaboration Core Specification Version 2.0 Query Syntax on the OSLC website.

See Also

`oslc.Client` | `oslc.rm.Requirement` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `getQueryService`

External Websites

Query Capabilities
Open Services for Lifecycle Collaboration

Introduced in R2021a

oslc.qm.TestCase

Test case resource for OSLC quality management domain

Description

The `oslc.qm.TestCase` object represents test case resources in the quality management domain of the Open Services for Lifecycle Collaboration (OSLC) service provider. After creating and configuring `oslc.Client` and `oslc.core.QueryCapability` objects, query the service provider for available test case resources with the `queryTestCases` function.

Creation

Create an `oslc.qm.TestCase` object by using the `createTestCase` function.

Properties

`ResourceUrl` — Resource navigation URL
character array

Navigation URL for the test case resource, specified as a character array.

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

Indicator for uncommitted changes to the test case resource, specified as a logical `1`or `0` where:

- `1` indicates the test case resource has uncommitted changes.
- `0` indicates the test case resource has no uncommitted changes.

Data Types: `logical`

`IsFetched` — Resource fetch status
`0` | `1`

test case resource fetch status, specified as a logical `1` or `0` where:

- `1` indicates the test case resource is fetched.
- `0` indicates the test case resource is not fetched.

Data Types: `logical`

`Title` — Test case title
character array

Test case title, specified as a character array.

`Identifier` — Test case resource identifier
character array

OSLC test case resource identifier, specified as a character array.

Object Functions

addRequirementLink	Add requirement traceability link to local OSLC test resource object
addResourceProperty	Add resource property to local OSLC resource object
addTextProperty	Add text property to local OSLC resource object
commit	Send local changes to OSLC service provider
fetch	Retrieve full resource data from OSLC service provider
getProperty	Get local contents of text property from OSLC resource object
getRDF	Get resource RDF/XML data from OSLC resource object
getRequirementLinks	Get locally stored requirement traceability links from OSLC test resource object
getResourceProperty	Get local contents of resource property from OSLC resource object
remove	Remove resource from OSLC service provider
removeRequirementLink	Remove requirement traceability link from local OSLC test resource object
removeResourceProperty	Remove resource property from local OSLC resource object
setProperty	Set local contents of text property for OSLC resource object
setRDF	Set RDF content for local OSLC resource object
setResourceUrl	Set resource URL for local OSLC resource object
show	View OSLC resource in system browser

Examples

Edit a Test Case and Commit Changes

This example shows how to submit a query request for test case resources with a configured OSLC client, edit an existing test case resource, and commit the changes to the service provider.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test case resource type.

```
myQueryCapability = getQueryService(myClient,'TestCase');
```

Submit a query request to the service provider for the available test case resources.

```
testCases = queryTestCases(myQueryCapability)
```

```
testCases =

  1×4 TestCase array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign a test case resource to the variable `myTestCase`. Retrieve the full resource data from the service provider for the test case resource. Examine the `Title` property.

```
myTestCase = testCases(1);
status = fetch(myTestCase,myClient)
```

```
status =

  StatusCode enumeration

    OK

title = myTestCase.Title

title =

    'Test Case 1'
```

Edit the test case title and commit the change to the service provider.

```
myTestCase.Title = 'My New Test Case Title';
status = commit(myTestCase,myClient)

status =

  StatusCode enumeration

    OK
```

Open the test case resource in the system browser by using the `show` function.

```
show(myTestCase)
```

Create a New Test Case

This example shows how to submit a creation request for a new test case resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a creation factory for the test case resource type.

```
myCreationFactory = getCreationFactory(myClient,'TestCase');
```

Use the creation factory to create a test case resource with the title `My New Test Case`. Retrieve the full resource data from the service provider for the test case resource and inspect the resource.

```
newTestCase = createTestCase(myCreationFactory,'My New Test Case');
fetch(newTestCase,myClient);
newTestCase

newTestCase =
  TestCase with properties:

    ResourceUrl: 'https://localhost:9443/qm/resource/itemName/_a9aS...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Test Case'
     Identifier: '301'
```

Open the test case resource in the system browser by using the `show` function.

```
show(newTestCase)
```

See Also

`oslc.core.CreationFactory` | `oslc.core.QueryCapability` | `oslc.Client` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `oslc.rm.Requirement` | `createTestCase` | `queryTestCases`

External Websites

Open Services for Lifecycle Collaboration
Resource: TestCase

Introduced in R2021a

oslc.qm.TestExecutionRecord

Test execution record resource for OSLC quality management domain

Description

The `oslc.qm.TestExecutionRecord` object represents test execution record resources in the quality management domain of the Open Services for Lifecycle Collaboration (OSLC) service provider. After creating and configuring `oslc.Client` and `oslc.core.QueryCapability` objects, query the service provider for available test execution record resources by using the `queryTestExecutionRecords` function.

Creation

Create an `oslc.qm.TestExecutionRecord` object by using the `createTestExecutionRecord` function.

Properties

`ResourceUrl` — Resource navigation URL
character array

Navigation URL for the test execution record resource, specified as a character array.

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

Indicator for uncommitted changes to the test execution record resource, specified as a logical `1`or `0` where:

- `1` indicates the test execution record resource has uncommitted changes.
- `0` indicates the test execution record resource has no uncommitted changes.

Data Types: `logical`

`IsFetched` — Resource fetch status
`0` | `1`

test execution record resource fetch status, specified as a logical `1` or `0` where:

- `1` indicates the test execution record resource is fetched.
- `0` indicates the test execution record resource is not fetched.

Data Types: `logical`

`Title` — Test execution record title
character array

Test execution record title, specified as a character array.

`Identifier` — Test execution record resource identifier

character array

OSLC test execution record resource identifier, specified as a character array.

Object Functions

addResourceProperty	Add resource property to local OSLC resource object
addTextProperty	Add text property to local OSLC resource object
commit	Send local changes to OSLC service provider
fetch	Retrieve full resource data from OSLC service provider
getProperty	Get local contents of text property from OSLC resource object
getRDF	Get resource RDF/XML data from OSLC resource object
getResourceProperty	Get local contents of resource property from OSLC resource object
getRunsTestCase	Get locally stored test case traceability link from OSLC test execution record resource object
remove	Remove resource from OSLC service provider
removeResourceProperty	Remove resource property from local OSLC resource object
setProperty	Set local contents of text property for OSLC resource object
setRDF	Set RDF content for local OSLC resource object
setResourceUrl	Set resource URL for local OSLC resource object
show	View OSLC resource in system browser

Examples

Edit a Test Execution Record and Commit Changes

This example shows how to submit a query request for test execution record resources with a configured OSLC client, edit an existing test execution record resource, and commit the changes to the service provider.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test execution record resource type.

```
myQueryCapability = getQueryService(myClient,'TestExecutionRecord');
```

Submit a query request to the service provider for the available test execution record resources.

```
testERs = queryTestExecutionRecords(myQueryCapability)
```

```
testERs =

  1×2 TestExecutionRecord array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign a test execution record resource to the variable `myTestER`. Retrieve the full resource data from the service provider for the test execution record resource. Examine the `Title` property.

```
myTestER = testERs(1);
status = fetch(myTestER,myClient)

status =

  StatusCode enumeration

    OK

title = myTestER.Title

title =

    'Test Case 1'
```

Edit the test execution record title and commit the change to the service provider.

```
myTestER.Title = 'My New Test Execution Record Title';
status = commit(myTestER,myClient)

status =

  StatusCode enumeration

    OK
```

Open the test execution record resource in the system browser by using the `show` function.

```
show(myTestER)
```

Create a New Test Execution Record

This example shows how to submit a creation request for a new test execution record resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a creation factory for the test execution record resource type.

```
myCreationFactory = getCreationFactory(myClient,'TestExecutionRecord');
```

Use the creation factory to create a test execution record resource with the title `My New Test Execution Record` and associate it with the test case resource URL `testURL` from a test case. For more information about querying the service provider for test cases, see “Edit a Test Case and Commit Changes” on page 2-21. Retrieve full resource data from the service provider for the test execution record resource and inspect the resource.

```
newTestER = createTestExecutionRecord(myCreationFactory, ...
    'My New Test Execution Record',testURL);
fetch(newTestCase,myClient);
newTestER

newTestER =
  TestExecutionRecord with properties:

    ResourceUrl: 'https://localhost:9443/qm/oslc_qm/resources/CfkIoW...'
          Dirty: 0
```

```
      IsFetched: 1
          Title: 'My New Test Execution Record'
     Identifier: '301'
```

Open the test execution record resource in the system browser by using the `show` function.

```
show(newTestER)
```

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `oslc.core.QueryCapability` | `oslc.qm.TestCase` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `queryTestExecutionRecords` | `createTestExecutionRecord`

External Websites

Open Services for Lifecycle Collaboration
Resource: TestExecutionRecord

Introduced in R2021a

oslc.qm.TestPlan

Test plan resource for OSLC quality management domain

Description

The `oslc.qm.TestPlan` object represents test plan resources in the quality management domain of the Open Services for Lifecycle Collaboration (OSLC) service provider. After creating and configuring `oslc.Client` and `oslc.core.QueryCapability` objects, query the service provider for available test plan resources by using the `queryTestPlans` function.

Creation

Create an `oslc.qm.TestPlan` object by using the `createTestPlan` function.

Properties

`ResourceUrl` — Resource navigation URL
character array

Navigation URL for the test plan resource, specified as a character array.

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

Indicator for uncommitted changes to the test plan resource, specified as a logical `1`or `0` where:

- `1` indicates the test plan resource has uncommitted changes.
- `0` indicates the test plan resource has no uncommitted changes.

Data Types: `logical`

`IsFetched` — Resource fetch status
`0` | `1`

test plan resource fetch status, specified as a logical `1` or `0` where:

- `1` indicates the test plan resource is fetched.
- `0` indicates the test plan resource is not fetched.

Data Types: `logical`

`Title` — Test plan title
character array

Test plan title, specified as a character array.

`Identifier` — Test plan resource identifier
character array

OSLC test plan resource identifier, specified as a character array.

Object Functions

addResourceProperty	Add resource property to local OSLC resource object
addTextProperty	Add text property to local OSLC resource object
commit	Send local changes to OSLC service provider
fetch	Retrieve full resource data from OSLC service provider
getProperty	Get local contents of text property from OSLC resource object
getRDF	Get resource RDF/XML data from OSLC resource object
getResourceProperty	Get local contents of resource property from OSLC resource object
remove	Remove resource from OSLC service provider
removeResourceProperty	Remove resource property from local OSLC resource object
setProperty	Set local contents of text property for OSLC resource object
setRDF	Set RDF content for local OSLC resource object
setResourceUrl	Set resource URL for local OSLC resource object
show	View OSLC resource in system browser

Examples

Edit a Test Plan and Commit Changes

This example shows how to submit a query request for test plan resources with a configured OSLC client, edit an existing test plan resource, and commit the changes to the service provider.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test plan resource type.

```
myQueryCapability = getQueryService(myClient,'TestPlan');
```

Submit a query request to the service provider for the available test plan resources.

```
testPlans = queryTestPlans(myQueryCapability)
```

```
testPlans =

  1×2 TestPlan array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign a test plan resource to the variable `myTestPlan`. Retrieve the full resource data from the service provider for the test plan resource. Examine the `Title` property.

```
myTestPlan = testPlans(1);
status = fetch(myTestPlan,myClient)
```

```
status =

  StatusCode enumeration
```

```
    OK

title = myTestPlan.Title

title =

    'Test Plan 1'
```

Edit the test plan title and commit the change to the service provider.

```
myTestPlan.Title = 'My New Test Plan Title';
status = commit(myTestPlan,myClient)

status =

  StatusCode enumeration

    OK
```

Open the test plan resource in the system browser by using the `show` function.

```
show(myTestCase)
```

Create a New Test Plan

This example shows how to submit a creation request for a new test plan resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a creation factory for the test plan resource type.

```
myCreationFactory = getCreationFactory(myClient,'TestPlan');
```

Use the creation factory to create a test plan resource with the title `My New Test Plan`. Retrieve the full resource data from the service provider for the test plan resource and inspect the resource.

```
newTestPlan = createTestPlan(myCreationFactory,'My New Test Plan');
fetch(newTestPlan,myClient);
newTestPlan

newTestPlan =
  TestPlan with properties:

    ResourceUrl: 'https://localhost:9443/qm/resource/itemName/_f56s...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Test Plan'
     Identifier: '301'
```

Open the test plan resource in the system browser by using the `show` function.

```
show(newTestPlan)
```

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `oslc.core.QueryCapability` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestResult` | `oslc.qm.TestScript` | `createTestPlan` | `queryTestPlans`

External Websites

Open Services for Lifecycle Collaboration
Resource: TestPlan

Introduced in R2021a

oslc.qm.TestResult

Test result resource for OSLC quality management domain

Description

The `oslc.qm.TestResult` object represents test result resources in the quality management domain of the Open Services for Lifecycle Collaboration (OSLC) service provider. After creating and configuring `oslc.Client` and `oslc.core.QueryCapability` objects, query the service provider for available test result resources by using the `queryTestResults` function.

Creation

Create an `oslc.qm.TestResult` by using the `createTestResult` function.

Properties

`ResourceUrl` — Resource navigation URL
character array

Navigation URL for the test result resource, specified as a character array.

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

Indicator for uncommitted changes to the test result resource, specified as a logical `1`or `0` where:

- `1` indicates the test result resource has uncommitted changes.
- `0` indicates the test result resource has no uncommitted changes.

Data Types: `logical`

`IsFetched` — Resource fetch status
`0` | `1`

test result resource fetch status, specified as a logical `1` or `0` where:

- `1` indicates the test result resource is fetched.
- `0` indicates the test result resource is not fetched.

Data Types: `logical`

`Title` — Test result title
character array

Test result title, specified as a character array.

`Identifier` — Test result resource identifier
character array

OSLC test result resource identifier, specified as a character array.

Object Functions

addResourceProperty	Add resource property to local OSLC resource object
addTextProperty	Add text property to local OSLC resource object
commit	Send local changes to OSLC service provider
fetch	Retrieve full resource data from OSLC service provider
getProducedTestExecutionRecord	Get locally stored test execution record traceability link from Open Services for Lifecycle Collaboration (OSLC) test result resource object
getProperty	Get local contents of text property from OSLC resource object
getRDF	Get resource RDF/XML data from OSLC resource object
getReportsOnTestCase	Get locally stored test case traceability link from OSLC test result resource object
getResourceProperty	Get local contents of resource property from OSLC resource object
getStatus	Get locally stored status from OSLC test result resource object
remove	Remove resource from OSLC service provider
removeResourceProperty	Remove resource property from local OSLC resource object
setProperty	Set local contents of text property for OSLC resource object
setRDF	Set RDF content for local OSLC resource object
setResourceUrl	Set resource URL for local OSLC resource object
show	View OSLC resource in system browser

Examples

Edit a Test Result and Commit Changes

This example shows how to submit a query request for test result resources with a configured OSLC client, edit an existing test result resource, and commit the changes to the service provider.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test result resource type.

```
myQueryCapability = getQueryService(myClient,'TestResult');
```

Submit a query request to the service provider for the available test result resources.

```
testResults = queryTestResults(myQueryCapability)
```

```
testResults =

  1×2 TestResult array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign a test result resource to the variable `myTestResult`. Retrieve the full resource data from the service provider for the test result resource. Examine the `Title` property.

```
myTestResult = testResults(1);
status = fetch(myTestResult,myClient)

status =

  StatusCode enumeration

    OK

title = myTestResult.Title

title =

    'Test Case 1'
```

Edit the test result title and commit the change to the service provider.

```
myTestResult.Title = 'My New Test Result Title';
status = commit(myTestResult,myClient)

status =

  StatusCode enumeration

    OK
```

Open the test result resource in the system browser by using the `show` function.

```
show(myTestResult)
```

Create a New Test Result

This example shows how to submit a creation request for a new test result resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in "Create and Configure an OSLC Client for the Quality Management Domain" on page 2-4, create a creation factory for the test result resource type.

```
myCreationFactory = getCreationFactory(myClient,'TestResult');
```

Use the creation factory to create a test result resource with the title `My New Test Result` and associate it with the test case resource URL specified by `testURL` and the test execution record resource URL specified by `executionURL`. Set the test result status to `Unverified`. For more information about querying the service provider for test cases and execution records, see "Edit a Test Case and Commit Changes" on page 2-21 and "Edit a Test Execution Record and Commit Changes" on page 2-25. Retrieve the full resource data from the service provider for the test result resource and inspect the resource.

```
newTestResult = createTestResult(myCreationFactory, ...
    'My New Test Result',testURL,executionURL,'Unverified');
fetch(newTestCase,myClient);
newTestResult

newTestResult =
  TestResult with properties:
```

```
    ResourceUrl: 'https://localhost:9443/qm/oslc_qm/resources/CdffuW...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Test Result'
     Identifier: '1456'
```

Open the test result resource in the system browser by using the `show` function.

```
show(newTestResult)
```

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `oslc.core.QueryCapability` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestScript` | `queryTestResults` | `createTestResult`

External Websites

Open Services for Lifecycle Collaboration
Resource: TestResult

Introduced in R2021a

oslc.qm.TestScript

Test script resource for OSLC quality management domain

Description

The `oslc.qm.TestScript` object represents test script resources in the quality management domain of the Open Services for Lifecycle Collaboration (OSLC) service provider. After creating and configuring `oslc.Client` and `oslc.core.QueryCapability` objects, query the service provider for available test script resources by using the `queryTestScripts` function.

Creation

Create an `oslc.qm.TestScript` object by using the `createTestScript` function.

Properties

`ResourceUrl` — Resource navigation URL
character array

Navigation URL for the test script resource, specified as a character array.

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

Indicator for uncommitted changes to the test script resource, specified as a logical `1`or `0` where:

- `1` indicates the test script resource has uncommitted changes.
- `0` indicates the test script resource has no uncommitted changes.

Data Types: `logical`

`IsFetched` — Resource fetch status
`0` | `1`

test script resource fetch status, specified as a logical `1` or `0` where:

- `1` indicates the test script resource is fetched.
- `0` indicates the test script resource is not fetched.

Data Types: `logical`

`Title` — Test script title
character array

Test script title, specified as a character array.

`Identifier` — Test script resource identifier
character array

Test script resource identifier, specified as a character array.

Object Functions

addRequirementLink	Add requirement traceability link to local OSLC test resource object
addResourceProperty	Add resource property to local OSLC resource object
addTextProperty	Add text property to local OSLC resource object
commit	Send local changes to OSLC service provider
fetch	Retrieve full resource data from OSLC service provider
getProperty	Get local contents of text property from OSLC resource object
getRDF	Get resource RDF/XML data from OSLC resource object
getRequirementLinks	Get locally stored requirement traceability links from OSLC test resource object
getResourceProperty	Get local contents of resource property from OSLC resource object
remove	Remove resource from OSLC service provider
removeRequirementLink	Remove requirement traceability link from local OSLC test resource object
removeResourceProperty	Remove resource property from local OSLC resource object
setProperty	Set local contents of text property for OSLC resource object
setRDF	Set RDF content for local OSLC resource object
setResourceUrl	Set resource URL for local OSLC resource object
show	View OSLC resource in system browser

Examples

Edit a Test Script and Commit Changes

This example shows how to submit a query request for test script resources with a configured OSLC client, edit an existing test script resource, and commit the changes to the service provider.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a query capability for the test script resource type.

```
myQueryCapability = getQueryService(myClient,'TestScript');
```

Submit a query request to the service provider for the available test script resources.

```
testScripts = queryTestScripts(myQueryCapability)
```

```
testScripts =

  1×7 TestScript array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign a test script resource to the variable `myTestScript`. Retrieve the full resource data from the service provider for the test script resource. Examine the `Title` property.

```
myTestScript = testScripts(1);
status = fetch(myTestScript,myClient)
```

```
status =

  StatusCode enumeration

    OK

title = myTestScript.Title

title =

    'Test Script 1'
```

Edit the test script title and commit the change to the service provider.

```
myTestScript.Title = 'My New Test Script Title';
status = commit(myTestScript,myClient)

status =

  StatusCode enumeration

    OK
```

Open the test script resource in the system browser by using the `show` function.

```
show(myTestScript)
```

Create a New Test Script

This example shows how to submit a creation request for a new test script resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Quality Management Domain” on page 2-4, create a creation factory for the test script resource type.

```
myCreationFactory = getCreationFactory(myClient,'TestScript');
```

Use the creation factory to create a test script resource with the creation factory with the title `My New Test Script`. Retrieve the full resource data from the service provider for the test script resource and inspect the resource.

```
newTestScript = createTestScript(myCreationFactory, ...
    'My New Test Script');
fetch(newTestScript,myClient);
newTestScript

newTestScript =
  TestScript with properties:

    ResourceUrl: 'https://localhost:9443/qm/resource/itemName/_b19w2...'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Test Script'
     Identifier: '498'
```

Open the test script resource in the system browser by using the `show` function.

```
show(newTestScript)
```

See Also

`oslc.Client` | `oslc.core.CreationFactory` | `oslc.core.QueryCapability` | `oslc.rm.Requirement` | `oslc.qm.TestCase` | `oslc.qm.TestExecutionRecord` | `oslc.qm.TestPlan` | `oslc.qm.TestResult` | `createTestScript` | `queryTestScripts`

External Websites

Open Services for Lifecycle Collaboration
Resource: TestScript

Introduced in R2021a

oslc.rm.Requirement

Requirement resource for OSLC requirements management domain

Description

The `oslc.rm.Requirement` object represents requirement resources in the requirements management domain of the Open Services for Lifecycle Collaboration (OSLC) service provider. After creating and configuring `oslc.Client` and `oslc.core.QueryCapability` objects, query the service provider for available requirement resources by using the `queryRequirements` function.

Creation

Create an `oslc.rm.Requirement` object by using the `createRequirement` function.

Properties

`ResourceUrl` — Resource navigation URL
character array

Navigation URL for the requirement resource, specified as a character array.

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

Indicator for uncommitted changes to the requirement resource, specified as a logical `1`or `0` where:

- `1` indicates the requirement resource has uncommitted changes.
- `0` indicates the requirement resource has no uncommitted changes.

Data Types: `logical`

`IsFetched` — Resource fetch status
`0` | `1`

requirement resource fetch status, specified as a logical `1` or `0` where:

- `1` indicates the requirement resource is fetched.
- `0` indicates the requirement resource is not fetched.

Data Types: `logical`

`Title` — Requirement title
character array

Requirement title, specified as a character array.

`Identifier` — Requirement resource identifier
character array

OSLC requirement resource identifier, specified as a character array.

Object Functions

addLink	Add link to local OSLC requirement resource object
addResourceProperty	Add resource property to local OSLC resource object
addTextProperty	Add text property to local OSLC resource object
commit	Send local changes to OSLC service provider
fetch	Retrieve full resource data from OSLC service provider
getLinks	Get locally stored traceability links from OSLC requirement resource object
getProperty	Get local contents of text property from OSLC resource object
getRDF	Get resource RDF/XML data from OSLC resource object
getResourceProperty	Get local contents of resource property from OSLC resource object
getSLRequirements	Get imported referenced requirement associated with OSLC requirement resource object
remove	Remove resource from OSLC service provider
removeLink	Remove link from local OSLC requirement resource object
removeResourceProperty	Remove resource property from local OSLC resource object
setProperty	Set local contents of text property for OSLC resource object
setRDF	Set RDF content for local OSLC resource object
setResourceUrl	Set resource URL for local OSLC resource object
show	View OSLC resource in system browser

Examples

Edit a Requirement and Commit Changes

This example shows how to submit a query request for requirement resources with a configured OSLC client, edit an existing requirement resource, and commit the changes to the service provider.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement resource type.

```
myQueryCapability = getQueryService(myClient,'Requirement');
```

Submit a query request to the service provider for the available requirement resources.

```
reqs = queryRequirements(myQueryCapability)
```

```
reqs =

  1×30 Requirement array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign a requirement resource to the variable `myReq`. Retrieve the full resource data from the service provider for the requirement resource. Examine the `Title` property.

```
myReq = reqs(1);
status = fetch(myReq,myClient)

status =

  StatusCode enumeration

    OK

title = myReq.Title

title =

    'Requirement 1'
```

Edit the requirement title and commit the change to the service provider.

```
myReq.Title = 'My New Requirement Title';
status = commit(myReq,myClient)

status =

  StatusCode enumeration

    OK
```

Open the requirement resource in the system browser by using the `show` function.

```
show(myReq)
```

Create a New Requirement

This example shows how to submit a creation request for a new requirement resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a creation factory for the requirement resource type.

```
myCreationFactory = getCreationFactory(myClient,'Requirement');
```

Use the creation factory to create a new requirement resource with the title `My New Requirement`. Retrieve the full resource data from the service provider for the requirement resource and inspect the resource.

```
newReq = createRequirement(myCreationFactory,'My New Requirement');
fetch(newReq,myClient);
newReq

newReq =

  Requirement with properties:

    ResourceUrl: 'https://localhost:9443/rm/resources/_72lxMWJREeup0...'
          Dirty: 0
      IsFetched: 1
```

```
         Title: 'My New Requirement'
    Identifier: '1806'
```

Open the requirement resource in the system browser by using the `show` function.

```
show(newReq)
```

See Also

`oslc.core.CreationFactory` | `oslc.core.QueryCapability` | `oslc.Client` | `oslc.rm.RequirementCollection` | `oslc.cm.ChangeRequest` | `oslc.qm.TestCase` | `queryRequirements` | `createRequirement`

External Websites

Open Services for Lifecycle Collaboration
Resource Requirement

Introduced in R2021a

oslc.rm.RequirementCollection

Requirement collection resource for OSLC requirements management domain

Description

The `oslc.rm.RequirementCollection` object represents requirement collection resources in the requirements management domain of the Open Services for Lifecycle Collaboration (OSLC) service provider. After creating and configuring `oslc.Client` and `oslc.core.QueryCapability` objects, query the service provider for available requirement collection resources by using the `queryRequirementCollections` function.

Creation

Create an `oslc.rm.RequirementCollection` object by using the `createRequirementCollection` function.

Properties

`ResourceUrl` — Resource navigation URL
character array

Navigation URL for the requirement collection resource, specified as a character array.

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

Indicator for uncommitted changes to the requirement collection resource, specified as a logical `1`or `0` where:

- `1` indicates the requirement collection resource has uncommitted changes.
- `0` indicates the requirement collection resource has no uncommitted changes.

Data Types: `logical`

`IsFetched` — Resource fetch status
`0` | `1`

requirement collection resource fetch status, specified as a logical `1` or `0` where:

- `1` indicates the requirement collection resource is fetched.
- `0` indicates the requirement collection resource is not fetched.

Data Types: `logical`

`Title` — Requirement collection title
character array

Requirement collection title, specified as a character array.

`Identifier` — Requirement collection resource identifier

character array

OSLC requirement collection resource identifier, specified as a character array.

Object Functions

addLink	Add link to local OSLC requirement resource object
addResourceProperty	Add resource property to local OSLC resource object
addTextProperty	Add text property to local OSLC resource object
commit	Send local changes to OSLC service provider
fetch	Retrieve full resource data from OSLC service provider
getLinks	Get locally stored traceability links from OSLC requirement resource object
getProperty	Get local contents of text property from OSLC resource object
getRDF	Get resource RDF/XML data from OSLC resource object
getResourceProperty	Get local contents of resource property from OSLC resource object
getSLRequirements	Get imported referenced requirement associated with OSLC requirement resource object
remove	Remove resource from OSLC service provider
removeLink	Remove link from local OSLC requirement resource object
removeResourceProperty	Remove resource property from local OSLC resource object
setProperty	Set local contents of text property for OSLC resource object
setRDF	Set RDF content for local OSLC resource object
setResourceUrl	Set resource URL for local OSLC resource object
show	View OSLC resource in system browser

Examples

Edit a Requirement Collection and Commit Changes

This example shows how to submit a query request for requirement collection resources with a configured OSLC client, edit an existing requirement collection resource, and commit the changes to the service provider.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a query capability for the requirement collection resource type.

```
myQueryCapability = getQueryService(myClient,'RequirementCollection');
```

Submit a query request to the service provider for the available requirement collection resources.

```
reqCollections = queryRequirementCollections(myQueryCapability)
```

```
reqCollections =

  1×5 RequirementCollection array with properties:

    ResourceUrl
    Dirty
    IsFetched
    Title
    Identifier
```

Assign a requirement collection resource to the variable `myReqCollection`. Retrieve the full resource data from the service provider for the requirement collection resource. Examine the `Title` property.

```
myReqCollection = reqCollections(1);
status = fetch(myReqCollection,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

```
title = myReqCollection.Title
```

```
title =

    'Requirement Collection 1'
```

Edit the requirement title and commit the change to the service provider.

```
myReqCollection.Title = 'My New Requirement Collection Title';
status = commit(myReqCollection,myClient)
```

```
status =

  StatusCode enumeration

    OK
```

Open the requirement collection resource in the system browser by using the `show` function.

```
show(myReqCollection)
```

Create a New Requirement Collection

This example shows how to submit a creation request for a new requirement collection resource with a configured OSLC client.

After you have created and configured the OSLC client `myClient` as described in “Create and Configure an OSLC Client for the Requirements Management Domain” on page 2-3, create a creation factory for the requirement collection resource type.

```
myCreationFactory = getCreationFactory(myClient,...
'RequirementCollection');
```

Use the creation factory to create a requirement collection resource with the title `My New Requirement Collection`. Retrieve the full resource data from the service provider for the requirement collection resource and inspect the resource.

```
newReqCollection = createRequirementCollection(myCreationFactory,...
'My New Requirement Collection')
fetch(newReqCollection,myClient);
newReqCollection
```

```
newReqCollection =
```

```
    RequirementCollection with properties:
    ResourceUrl: 'https://localhost:9443/rm/resources/_72lxMWJREeup0r..'
          Dirty: 0
      IsFetched: 1
          Title: 'My New Requirement Collection'
     Identifier: '1808'
```

Open the requirement collection resource in the system browser by using the `show` function.

```
show(newReqCollection)
```

See Also

`oslc.core.CreationFactory` | `oslc.core.QueryCapability` | `oslc.Client` | `oslc.rm.Requirement` | `queryRequirementCollections` | `createRequirementCollection`

External Websites

Open Services for Lifecycle Collaboration
Resource RequirementCollection

Introduced in R2021a

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**.

Creation

`jst = slreq.find(rs, 'Type', 'Justification', 'PropertyName', PropertyValue)` finds and returns an `slreq.Justification` object `jst` in the requirement 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.

Attributes:

GetAccess	public
SetAccess	public

`Summary` — Justification summary

character vector

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

Attributes:

GetAccess	public
SetAccess	public

`Description` — Justification description

character vector

Justification description text, specified as a multiline character vector.

Attributes:

GetAccess	public
SetAccess	public

`Keywords` — Justification keywords

character array

Justification keywords, specified as a character array.

Attributes:

GetAccess	public
SetAccess	public

`Rationale` — Justification rationale

character vector

Justification rationale text, specified as a multiline character vector.

Attributes:

GetAccess	public
SetAccess	public

`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.

Attributes:

GetAccess	public
SetAccess	private

`CreatedBy` — Justification creator

character vector

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

Attributes:

GetAccess	public
SetAccess	private

`ModifiedBy` — Justification modifier

character vector

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

Attributes:

GetAccess	public
SetAccess	private

`IndexEnabled` — Index enabled indicator

`1` (default) | `0`

Indicates whether the index is enabled (`1`) or disabled (`0`), returned as a `1` or `0` of data type `logical`. If you disable the index, Requirements Toolbox does not count this justification when it creates the numbered hierarchy list. However, the justification remains in the same place in the hierarchy.

Attributes:

GetAccess	public
SetAccess	public

`IndexNumber` — User-specified index value

empty `double` array (default) | `int32` array

User-specified index value, returned as an empty `double` array or an `int32` array. If empty, Requirements Toolbox calculates the `Index` value. Otherwise, Requirements Toolbox sets the `Index` property to the specified integer value.

Attributes:

GetAccess	public
SetAccess	public

`SID` — Justification Session Independent Identifier

character vector

The Session Independent Identifier corresponding to the justification.

Attributes:

GetAccess	public
SetAccess	private

`FileRevision` — Justification revision number

scalar

Justification revision number, specified as a scalar.

Attributes:

GetAccess	public
SetAccess	private

`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.

Attributes:

GetAccess	public
SetAccess	private

`Dirty` — Unsaved changes indicator

`0` | `1`

Indicates if the requirement has unsaved changes (`1`) or does not have unsaved changes (`0`).

Attributes:

GetAccess	public
SetAccess	private

`Comments` — Justification comments

structure array

The comments that are attached with the justification, specified as a structure.

Attributes:

GetAccess	public
SetAccess	private

`Index` — Justification index

character array

The index of the justification, specified as a character array.

Attributes:

GetAccess	public
SetAccess	private

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

Add Child Justifications

This example shows how to add a child justification under a justification.

Load a requirement set called `myReqSet`.

```
rs = slreq.load("myReqSet");
```

Find justification objects in the requirement set.

```
myJustifications = find(rs,"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, Requirements Toolbox creates an `slreq.Link` object to store source and destination data of the link.

Creation

`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

`'Relate'` | `'Implement'` | `'Verify'` | `'Derive'` | `'Refine'` | `'Confirm'` | string scalar | character vector

Link type enumeration, specified as one of the options in the table:

Type	Description
`'Relate'`	• General relationship between items for most use cases • Bi-directional link
`'Implement'`	• Specifies the source item that implements the requirement • Contributes to the implementation status For more information, see “Review Requirements Implementation Status”.

Type	Description
`'Verify'`	• Specifies which source item verifies the requirement • Contributes to the verification status if the source item is one of the accepted item types For more information, see “Review Requirements Verification Status”.
`'Derive'`	Specifies which source item derives the destination item
`'Refine'`	Specifies which source item adds detail for the functionality specified by the destination item
`'Confirm'`	• Specifies relationship between a requirement and an external test result source • Can contribute to the verification status in certain cases For more information, see “Include Results from External Sources in Verification Status”.
string scalar or character vector	String scalar or character vector that specifies a custom link type. For more information, see “Define Custom Requirement and Link Types”.

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
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

Examples

Create a Link

This example shows how to create a link.

Create a link between the currently selected Simulink block and a requirement `req`.

```
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]
```

Get Incoming Links

This example shows how to get the incoming links for a requirement.

Load a requirement set called `myReqSet`.

```
rs = slreq.load("myReqSet");
```

Find a requirement in the requirement with ID `R1.1`.

```
myReq = find(rs,"Type","Requirement","Id","R1.1");
```

Query incoming links to the requirement.

```
inLinks = slreq.inLinks(myReq);
```

Get Outgoing Links

This example shows how to get the outgoing links for a link source.

Load a link set called `c5.slmx`.

```
myLinkSet = slreq.load("c5.slx");
```

Get the link sources from the link set.

```
allSrcs = sources(myLinkSet);
```

Get the outgoing links for the first link source.

```
myLink = slreq.outLinks(allSrcs(1));
```

See Also

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

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.

Creation

`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

File path of the link set, specified as a character vector. By default, the link set is stored in the same folder as the artifact and has the same base file name and an `.slmx` extension.

`Artifact` — Artifact containing link sources
character vector

Artifact that contains the link sources for the link set, specified as a character vector. When you create a link, the link set is associated with the artifact that the link source item belongs to. By default, the link set is stored in the same folder as the artifact and has the same base file name and an `.slmx` extension. For more information, see “Requirements Link Storage”. The artifact can be any file that contains a linkable item, such as a Simulink model or a Simulink Test file.

`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
exportToVersion	Export link set to previous MATLAB version
find	Find links in link set with matching attribute values
getLinks	Get links from link set
getRegisteredReqSets	Get requirement sets registered in link set
inspectAttribute	Get information about link set custom attribute
redirectLinksToImportedReqs	Redirect link destination from external document to imported requirement set
save	Save link set
sources	Get link sources
updateAttribute	Update information for link set custom attribute
updateBacklinks	Synchronize external navigation links
updateDocUri	Update link destination for direct links
updateRegisteredReqSets	Update requirement sets registered to link set

Examples

Find, Load, and Edit a Link Set

This example shows how to find, load, and edit a link set.

Find a loaded link set by using the name.

```
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
```

Load a link set associated with a Simulink model called `fuelsys`.

```
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

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

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 requirement set.

Creation

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

`ref = add(rs, 'Artifact', FileName, 'PropertyName', PropertyValue)` adds a referenced requirement `ref` to a requirement 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.

Attributes:

GetAccess	public
SetAccess	private

`CustomId` — Referenced requirement Custom ID

character vector

Referenced requirement custom ID, returned as a character vector.

Attributes:

GetAccess	public
SetAccess	private

`Artifact` — Container identifier

character vector

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

Attributes:

GetAccess	public
SetAccess	private

`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**.

Attributes:

GetAccess	public
SetAccess	private

`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'`

Attributes:

GetAccess	public
SetAccess	private

`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.

Attributes:

GetAccess	public
SetAccess	private

`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.

Attributes:

GetAccess	public
SetAccess	private

`CreatedBy` — Referenced requirement creator

character vector

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

Attributes:

GetAccess	public
SetAccess	private

`ModifiedBy` — Referenced requirement modifier

character vector

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

Attributes:

GetAccess	public
SetAccess	private

`IsLocked` — Referenced requirement lock indicator

1 (default) | 0

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

Attributes:

GetAccess	public
SetAccess	private

`Summary` — Referenced requirement summary

character vector

Referenced requirement summary text, returned as a character vector.

Attributes:

GetAccess	public
SetAccess	public

`Description` — Referenced requirement description

character vector

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

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`Rationale` — Referenced requirement rationale

character vector

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

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`Keywords` — Referenced requirement keywords

character array

Referenced requirement keywords, specified as a character array.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`Type` — Referenced requirement type enumeration

`'Functional'` | `'Informational'` | `'Container'` | string scalar | character vector

Referenced requirement type enumeration, specified as `'Functional'`, `'Informational'`, `'Container'`, or a string scalar or character vector that specifies a custom requirement type. For more information, see “Requirement Types”.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`IndexEnabled` — Referenced requirement index enabled indicator

`1` (default) | `0`

Indicates if the referenced requirement index is enabled (`1`) or disabled (`0`), returned as a `1` or `0` of data type `logical`. Disabling the index omits the referenced requirement from the numbered hierarchy list.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`IndexNumber` — User-specified referenced requirement index value

empty `double` array (default) | `int32` array

User-specified referenced requirement index value, returned as an empty `double` array or an `int32` array. If empty, Requirements Toolbox calculates the `Index` value. Otherwise, Requirements Toolbox sets the `Index` property to the specified integer value.

Attributes:

GetAccess	public
SetAccess	public

SID — Referenced requirement Session Independent Identifier

character vector

The Session Independent Identifier corresponding to the referenced requirement.

Attributes:

GetAccess	public
SetAccess	private

FileRevision — Referenced requirement revision number

scalar

Referenced requirement revision number, specified as a scalar.

Attributes:

GetAccess	public
SetAccess	private

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.

Attributes:

GetAccess	public
SetAccess	private

Dirty — Unsaved changes indicator

`0` | `1`

Indicates if the requirement has unsaved changes (`1`) or does not have unsaved changes (`0`).

Attributes:

GetAccess	public
SetAccess	private

Comments — Referenced requirement comments

structure array

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

Attributes:

GetAccess	public
SetAccess	private

Index — Referenced requirement index

character array

The index of the referenced requirement, specified as a character array.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`

Methods

add	Add child referenced requirement
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
getPostImportFcn	Get contents of `PostImportFcn` callback
getPreImportFcn	Get registered `PreImportFcn` callback script
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
moveDown	Move referenced requirement down in hierarchy
moveUp	Move referenced requirement up in hierarchy
parent	Find parent item of referenced requirement
remove	Remove referenced requirements
reqSet	Return parent requirement set
setAttribute	Set referenced requirement custom attributes
setParent	Set parent of referenced requirement in `PostImportFcn` callback
setPostImportFcn	Assign `PostImportFcn` callback script
setPreImportFcn	Assign `PreImportFcn` callback script
unlock	Unlock referenced requirements
unlockAll	Unlock all child referenced requirements for editing
updateFromDocument	Update referenced requirements from external requirements document

Examples

Find a Referenced Requirement

This example shows how to find a referenced requirement in a requirement set.

Load a requirement set called `myReqSet`.

```
rs = slreq.load("myReqSet");
```

Find a requirement with ID 9 in the requirement set.

```
req = find(rs,"Type","Reference","ID","9");


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

Topics

“Import and Update Requirements from a Microsoft Word Document”

Introduced in R2018a

slreq.ReqSet class

Package: `slreq`

Work with requirement sets

Description

Instances of `slreq.ReqSet` are requirement set objects.

Creation

`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` — Requirement set
`slreq.ReqSet` object

An instance of the `slreq.ReqSet` object.

Properties

`Name` — Requirement set name
character vector

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

`Filename` — Requirement set file path
character vector

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

`Revision` — Requirement set revision number
scalar

Requirement set revision number, specified as a scalar.

`CreatedBy` — Requirement set creator
character vector

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

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

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

`ModifiedBy` — Requirement set modifier
character vector

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

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

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

`Description` — Requirement set description
character vector

Requirement set description text, specified as a character vector.

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

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

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

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

Methods

add	Add requirements to requirement set
addAttribute	Add custom attribute to requirement set
addJustification	Add justifications to requirement set
children	Get top-level items in requirement set
close	Close a requirement set
createReferences	Create read-only references to requirement items in third-party documents
discard	Close requirement set without saving
deleteAttribute	Delete custom attribute from requirement set
explore	Open requirement set in Requirements Editor
exportToVersion	Export requirement set to previous MATLAB version
find	Find requirements in requirement set that have matching attribute values
getImplementationStatus	Query requirement set implementation status summary
getPostLoadFcn	Get contents of `PostLoadFcn` callback
getPreSaveFcn	Get contents of `PreSaveFcn` callback
getVerificationStatus	Query requirement set verification status summary
importFromDocument	Import editable requirements from external documents
inspectAttribute	Get information about requirement set custom attribute
runTests	Run test cases linked to the requirement set
save	Save a requirement set
setPostLoadFcn	Assign `PostLoadFcn` callback script
setPreSaveFcn	Assign `PreSaveFcn` callback script
updateAttribute	Update information for requirement set custom attribute
updateImplementationStatus	Update requirement set implementation status summary
updateReferences	Update referenced requirements in requirement set
updateSrcArtifactUri	Update document resource identifier of imported requirements
updateSrcFileLocation	Update document location of imported requirements
updateVerificationStatus	Update requirement set verification status summary

Examples

Create, Save, and Open a Requirement Set Object

This example shows how to create, save, and open a requirement set object.

Create a new requirement set called `Design_Requirements`.

```
rs = slreq.new("Design_Requirements");
```

Save and close the requirement set.

```
save(rs);
close(rs);
```

Open the requirement set in the **Requirements Editor**.

```
slreq.open(rs);
```

See Also

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

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 Requirements Toolbox and that do not have a persistent association with artifacts managed by external applications. Requirement objects can exist only within a requirement set.

Creation

`req = find(rs, 'PropertyName', PropertyValue)` finds and returns a requirement `req` in the requirement 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` — Requirement set object
`slreq.ReqSet` object

Requirement 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 enumeration
`'Functional'` | `'Informational'` | `'Container'` | string scalar | character vector

Requirement type enumeration, specified as `'Functional'`, `'Informational'`, `'Container'`, or a string scalar or character vector that specifies a custom requirement type. For more information, see “Requirement Types”.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

`Id` — Requirement custom ID
string scalar | character vector

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

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`Summary` — Requirement summary

string scalar | character vector

Requirement summary text, specified as a string scalar or character vector.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`Description` — Requirement description

string scalar | character vector

Requirement description text, specified as a string scalar or character vector.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`Keywords` — Requirement keywords

string array | cell array

Requirement keywords, specified as a string array or cell array of character vectors.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`Rationale` — Requirement rationale

string scalar | character vector

Requirement rationale text, specified as a string scalar or character vector.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`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.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`

`CreatedBy` — Requirement creator

character vector

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

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`

`ModifiedBy` — Requirement modifier

character vector

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

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`

`IndexEnabled` — Index enabled indicator

`1` (default) | `0`

Indicates whether the index is enabled (`1`) or disabled (`0`), returned as a `1` or `0` of data type `logical`. If you disable the index, Requirements Toolbox does not count this requirement when it creates the numbered hierarchy list. However, the requirement remains in the same place in the hierarchy.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

`IndexNumber` — User-specified index value

empty `double` array (default) | `int32` array

User-specified index value, returned as an empty `double` array or an `int32` array. If empty, Requirements Toolbox calculates the `Index` value. Otherwise, Requirements Toolbox sets the `Index` property to the specified integer value.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

`SID` — Requirement Session Independent Identifier

character vector

The Session Independent Identifier corresponding to the requirement, specified as a character vector.

Attributes:

GetAccess	public
SetAccess	private

`FileRevision` — Requirement revision number

scalar

Requirement revision number, specified as a scalar.

Attributes:

GetAccess	public
SetAccess	private

`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.

Attributes:

GetAccess	public
SetAccess	private

`Dirty` — Unsaved changes indicator

`0` | `1`

Indicates if the requirement has unsaved changes (`1`) or does not have unsaved changes (`0`).

Attributes:

GetAccess	public
SetAccess	private

`Comments` — Requirement comments

structure array

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

Attributes:

GetAccess	public
SetAccess	private

`Index` — Requirement index

character array

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

Attributes:

GetAccess	public
SetAccess	private

Methods

add	Add child requirement
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 requirement set
setAttribute	Set requirement custom attributes

Examples

Find a Requirement in a Requirement Set

This example shows how to find a requirement in a requirement set.

Load a requirement set called `myReqSet`.

```
rs = slreq.load("myReqSet");
```

Find a requirement with ID 77 in the requirement set.

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

Introduced in R2018a

slreq.callback.CustomImportOptions class

Package: `slreq.callback`

Custom import options

Description

Use objects of the `slreq.callback.CustomImportOptions` class to adjust the options to use when import requirements. When you import requirements from a custom third-party document, `slreq.getCurrentImportOptions` generates an `slreq.callback.CustomImportOptions` object that you can use to adjust the options to use during import. You can only access this object in the `PreImportFcn` callback.

The `slreq.callback.CustomImportOptions` class is a `handle` class.

Creation

`options = slreq.getCurrentImportOptions` returns an `slreq.callback.CustomImportOptions` object if you import requirements from a custom third-party document.

Properties

`Rationale` — External attribute mapped to `Rationale`

string scalar | character vector

External attribute mapped to the “`Rationale`” on page 2-0 property, specified as a string scalar or character vector.

Example: `myImportOptions.Rationale = "Requirement rationale";`

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

`Keywords` — External attribute mapped to `Keywords`

string scalar | character vector

External attribute mapped to the “`Keywords`” on page 2-0 property, specified as a string scalar or character vector.

Example: `myImportOptions.Keywords = "Requirement keywords";`

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

`Attributes` — External attributes to import

cell array

External attributes to import as custom attributes, specified as a `cell` array.

Example: `myImportOptions.Attributes = {'Priority','Status'};`

Attributes:

```
GetAccess                                       public
SetAccess                                       public
```

`Filter` — Filter condition to apply during import

string scalar | character vector

Filter condition to apply during import, specified as a string scalar or a character vector.

Example: `myImportOptions.Filter = "AttributeName==Value";`

Attributes:

```
GetAccess                                       public
SetAccess                                       public
```

`AsReference` — Option to import as references

`1` (default) | `0`

Option to import as `slreq.Reference` objects, specified as a `1` or `0` of data type `logical`. If `0`, requirements import as `slreq.Requirement` objects.

Attributes:

```
GetAccess                                       public
SetAccess                                       public
```

`RichText` — Option to import with rich text

`0` (default) | `1`

Option to import requirements with rich text, specified as a `1` or `0` of data type `logical`.

Attributes:

```
GetAccess                                       public
SetAccess                                       public
```

`DocUri` — Resource identifier for requirements document

string scalar | character vector

Resource identifier for external requirements document, specified as a string scalar or character vector.

Attributes:

```
GetAccess                                       public
SetAccess                                       public
```

`DocType` — Requirements document custom link type

string scalar | character vector

Requirements document custom link type, returned as a string scalar or character vector.

Attributes:

GetAccess	public
SetAccess	private

ReqSet — Requirement set name

character vector

Requirement set name, returned as a character vector.

Attributes:

GetAccess	public
SetAccess	private

PreImportFcn — Contents of PreImportFcn callback

string scalar | character vector

Contents of the `PreImportFcn` callback for the current Import node, specified as a string scalar or a character vector.

Attributes:

GetAccess	public
SetAccess	public

PostImportFcn — Contents of PostImportFcn callback

string scalar | character vector

Contents of the `PostImportFcn` callback for the current Import node, specified as a string scalar or a character vector.

Attributes:

GetAccess	public
SetAccess	public

See Also

`slreq.getCurrentImportOptions` | `setPreImportFcn` | `getPreImportFcn`

Topics

“Use Callbacks to Customize Requirement Import Behavior”

Introduced in R2022a

slreq.callback.DOORSImportOptions class

Package: `slreq.callback`

IBM Rational DOORS import options

Description

Use objects of the `slreq.callback.DOORSImportOptions` class to adjust the options to use when import requirements. When you import requirements from IBM Rational DOORS, `slreq.getCurrentImportOptions` generates an `slreq.callback.DOORSImportOptions` object that you can use to adjust the options to use when you import requirements. You can only access this object in the `PreImportFcn` callback.

The `slreq.callback.DOORSImportOptions` class is a `handle` class.

Creation

`options = slreq.getCurrentImportOptions` returns an `slreq.callback.DOORSImportOptions` object if you are importing requirements from IBM Rational DOORS.

Properties

`Rationale` — External attribute mapped to `Rationale`

string scalar | character vector

External attribute mapped to the “Rationale” on page 2-0 property, specified as a string scalar or character vector.

Example: `myImportOptions.Rationale = "Requirement rationale";`

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

`Keywords` — External attribute mapped to `Keywords`

string scalar | character vector

External attribute mapped to the “Keywords” on page 2-0 property, specified as a string scalar or character vector.

Example: `myImportOptions.Keywords = "Requirement keywords";`

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

`Attributes` — External attributes to import

cell array

External attributes to import as custom attributes, specified as a `cell` array.

Example: `myImportOptions.Attributes = {'Priority','Status'};`

Attributes:

```
GetAccess                                           public
SetAccess                                           public
```

`Filter` — Filter condition to apply during import

string scalar | character vector

Filter condition to apply during import, specified as a string scalar or a character vector.

Example: `myImportOptions.Filter = "AttributeName==Value";`

Attributes:

```
GetAccess                                           public
SetAccess                                           public
```

`AsReference` — Option to import as references

`1` (default) | `0`

Option to import as `slreq.Reference` objects, specified as a `1` or `0` of data type `logical`. If `0`, requirements import as `slreq.Requirement` objects.

Attributes:

```
GetAccess                                           public
SetAccess                                           public
```

`RichText` — Option to import with rich text

`0` (default) | `1`

Option to import requirements with rich text, specified as a `1` or `0` of data type `logical`.

Attributes:

```
GetAccess                                           public
SetAccess                                           public
```

`DocUri` — Resource identifier for requirements document

string scalar | character vector

Resource identifier for external requirements document, specified as a string scalar or character vector.

Attributes:

```
GetAccess                                           public
SetAccess                                           public
```

`DocType` — Requirements document custom link type

string scalar | character vector

Requirements document custom link type, returned as a string scalar or character vector.

Attributes:

GetAccess	public
SetAccess	private

ReqSet — Requirement set name

character vector

Requirement set name, returned as a character vector.

Attributes:

GetAccess	public
SetAccess	private

PreImportFcn — Contents of PreImportFcn callback

string scalar | character vector

Contents of the `PreImportFcn` callback for the current Import node, specified as a string scalar or a character vector.

Attributes:

GetAccess	public
SetAccess	public

PostImportFcn — Contents of PostImportFcn callback

string scalar | character vector

Contents of the `PostImportFcn` callback for the current Import node, specified as a string scalar or a character vector.

Attributes:

GetAccess	public
SetAccess	public

See Also

`slreq.getCurrentImportOptions` | `setPreImportFcn` | `getPreImportFcn`

Topics

“Use Callbacks to Customize Requirement Import Behavior”

Introduced in R2022a

slreq.callback.MSExcelImportOptions class

Package: `slreq.callback`

Microsoft Excel import options

Description

Use objects of the `slreq.callback.MSExcelImportOptions` class to adjust the options to use when import requirements. When you import requirements from a Microsoft Excel file, `slreq.getCurrentImportOptions` generates an `slreq.callback.MSExcelImportOptions` object that you can use to adjust the options to use when you import requirements. You can only access this object in the `PreImportFcn` callback.

The `slreq.callback.MSExcelImportOptions` class is a `handle` class.

Creation

`options = slreq.getCurrentImportOptions` returns an `slreq.callback.MSExcelImportOptions` object if you are importing requirements from a Microsoft Excel file.

Properties

`Worksheet` — Worksheet name

string scalar | character vector

Name ofMicrosoft Excel worksheet, specified as a string scalar or a character vector.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

`SubDocPrefix` — Option to prepend sheet name in custom ID

`0` (default) | `1`

Option to prepend the sheet name in the “`CustomId” on page 2-0` property of the imported requirements, specified as a `1` or `0` of data type `logical`.

Tip If requirements from multiple sheets import with the same custom ID, set this property to `1` to generate unique custom IDs.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

Rows — Range of rows

`double` array

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

Example: `myImportOptions.Rows = [3 35];`

Attributes:

GetAccess	public
SetAccess	public

Columns — Range of columns

`double` array

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

Example: `myImportOptions.Columns = [1 6];`

Attributes:

GetAccess	public
SetAccess	public

Attributes — External attributes to import

cell array

External attributes to import as custom attributes, specified as a `cell` array.

The length of this cell array must match the number of columns specified by the "AttributeColumn" on page 2-0 property.

Example: `myImportOptions.Attributes = {'Test Status','Test Procedure'};`

Attributes:

GetAccess	public
SetAccess	public

IdColumn — Column to map to the Id property

`double`

Column in the Microsoft Excel spreadsheet to map to the "Id" on page 2-0 property of the requirements in your requirement set, specified as a `double`.

Example: `myImportOptions.IdColumn = 1;`

Attributes:

GetAccess	public
SetAccess	public

SummaryColumn — Column to map to the Summary property

`double`

Column in the Microsoft Excel spreadsheet to map to the "Summary" on page 2-0 property of the requirements in your requirement set, specified as a `double`.

Example: `myImportOptions.SummaryColumn = 2;`

Attributes:

GetAccess	public
SetAccess	public

DescriptionColumn — Column to map to the Description property

double

Column in the Microsoft Excel spreadsheet to map to the "Description" on page 2-0 property of the requirements in your requirement set, specified as a double.

Example: myImportOptions.DescriptionColumn = 3;

Attributes:

GetAccess	public
SetAccess	public

RationaleColumn — Column to map to the Rationale property

double

Column in the Microsoft Excel spreadsheet to map to the "Rationale" on page 2-0 property of the requirements in your requirement set, specified as a double.

Example: myImportOptions.RationaleColumn = 4;

Attributes:

GetAccess	public
SetAccess	public

KeywordsColumn — Column to map to the Keywords property

double

Column in the Microsoft Excel spreadsheet to map to the "Keywords" on page 2-0 property of the requirements in your requirement set, specified as a double.

Example: myImportOptions.KeywordsColumn = 5;

Attributes:

GetAccess	public
SetAccess	public

AttributeColumn — Columns to map to custom attributes

double array

Columns in the Microsoft Excel spreadsheet to map as custom attributes of the requirements in your requirement set, specified as a double array.

Example: myImportOptions.AttributeColumn = [4 6];

Attributes:

GetAccess	public
SetAccess	public

USDM — USDM format

string scalar | character vector

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

Example: `myImportOptions.USDM = "RQ -"` will match entries with IDs similar to `RQ01`, `RQ01-2`, `RQ01-2-1` etc.

Attributes:

GetAccess	public
SetAccess	public

Bookmarks — Option to import requirements using bookmarks

`0` (default) | `1`

Option to import requirements content using user-defined bookmarks, specified as a `1` or `0` of data type `logical`.

By default, Requirements Toolbox sets the value to `1` for Microsoft Word documents and `0` for Microsoft Excel spreadsheets.

Attributes:

GetAccess	public
SetAccess	public

Match — Regular expression pattern

string scalar | character vector

Regular expression pattern, specified as a string scalar or character vector. Use this expression to search for matches in Microsoft Office documents.

Attributes:

GetAccess	public
SetAccess	public

AsReference — Option to import as references

`1` (default) | `0`

Option to import as `slreq.Reference` objects, specified as a `1` or `0` of data type `logical`. If `0`, requirements import as `slreq.Requirement` objects.

Attributes:

GetAccess	public
SetAccess	public

RichText — Option to import with rich text

`0` (default) | `1`

Option to import requirements with rich text, specified as a `1` or `0` of data type `logical`.

Attributes:

GetAccess	public
SetAccess	public

`DocUri` — Resource identifier for requirements document
string scalar | character vector

Resource identifier for external requirements document, specified as a string scalar or character vector.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

`DocType` — Requirements document custom link type
string scalar | character vector

Requirements document custom link type, returned as a string scalar or character vector.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`

`ReqSet` — Requirement set name
character vector

Requirement set name, returned as a character vector.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`

`PreImportFcn` — Contents of `PreImportFcn` callback
string scalar | character vector

Contents of the `PreImportFcn` callback for the current Import node, specified as a string scalar or a character vector.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

`PostImportFcn` — Contents of `PostImportFcn` callback
string scalar | character vector

Contents of the `PostImportFcn` callback for the current Import node, specified as a string scalar or a character vector.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

Examples

Customize Excel Import Options

This example shows how to customize Microsoft® Excel® import options by using the `PreImportFcn` callback.

Use `slreq.import` to import the Excel file `ExampleRequirements.xlsx` into Requirements Toolbox™. Name the imported requirement set `myReqSet` and register the script `excelPreImport` as the `PreImportFcn` callback. Return a handle to the requirement set.

```
[~,~,rs] = slreq.import("ExampleRequirements.xlsx", ...
    ReqSet="myReqSet",preImportFcn="excelPreImport");
```

The script `excelPreImport` uses `slreq.getCurrentImportOptions` to get the import options, then maps columns 2, 4, and 5 to the built-in `slreq.Reference` properties `ID`, `Summary`, and `Description`. The script also maps columns 3, 6, and 7 to custom attributes `orig_Type`, `Remark`, and `Status`.

```
type excelPreImport.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.IdColumn = 2;
importOptions.SummaryColumn = 4;
importOptions.DescriptionColumn = 5;
importOptions.Attributes = {'orig_type','Remark','Status'};
importOptions.AttributeColumn = [3 6 7];
```

Return the `importOptions` object.

```
importOptions
```

```
importOptions =
  MSExcelImportOptions with properties:

           Worksheet: []
         SubDocPrefix: 0
                 Rows: []
              Columns: ''
           Attributes: {'orig_type'  'Remark'  'Status'}
             IdColumn: 2
        SummaryColumn: 4
    DescriptionColumn: 5
      RationaleColumn: []
       KeywordsColumn: []
      AttributeColumn: [3 6 7]
      CreatedByColumn: []
     ModifiedByColumn: []
                 USDM: ''
            Bookmarks: 0
                Match: []
          AsReference: 1
             RichText: 0
               DocUri: 'C:\Users\jdoe\MATLAB\Examples\slrequirements-ex00521778\ExampleRequiremer
              DocType: 'linktype_rmi_excel'
               ReqSet: 'myReqSet'
         PreImportFcn: 'excelPreImport'
        PostImportFcn: ''
```

See Also

`slreq.getCurrentImportOptions` | `setPreImportFcn` | `getPreImportFcn`

Topics

“Use Callbacks to Customize Requirement Import Behavior”

Introduced in R2022a

slreq.callback.MSWordImportOptions class

Package: `slreq.callback`

Microsoft Word import options

Description

Use objects of the `slreq.callback.MSWordImportOptions` class to adjust the options to use when import requirements. When you import requirements from a Microsoft Word file, `slreq.getCurrentImportOptions` generates an `slreq.callback.MSWordImportOptions` object that you can use to adjust the options to use when you import requirements. You can only access this object in the `PreImportFcn` callback.

The `slreq.callback.MSWordImportOptions` class is a `handle` class.

Creation

`options = slreq.getCurrentImportOptions` returns an `slreq.callback.MSWordImportOptions` object if you are importing requirements from a Microsoft Word file.

Properties

`IgnoreOutlineNumbers` — Option to ignore outline numbers

`0` (default) | `1`

Option to ignore outline numbers in section headers, specified as a `1` or `0` of data type `logical`.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`Bookmarks` — Option to import requirements using bookmarks

`0` (default) | `1`

Option to import requirements content using user-defined bookmarks, specified as a `1` or `0` of data type `logical`.

By default, Requirements Toolbox sets the value to `1` for Microsoft Word documents and `0` for Microsoft Excel spreadsheets.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`Match` — Regular expression pattern

string scalar | character vector

Regular expression pattern, specified as a string scalar or character vector. Use this expression to search for matches in Microsoft Office documents.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`AsReference` — Option to import as references

`1` (default) | `0`

Option to import as `slreq.Reference` objects, specified as a `1` or `0` of data type `logical`. If `0`, requirements import as `slreq.Requirement` objects.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`RichText` — Option to import with rich text

`0` (default) | `1`

Option to import requirements with rich text, specified as a `1` or `0` of data type `logical`.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`DocUri` — Resource identifier for requirements document

string scalar | character vector

Resource identifier for external requirements document, specified as a string scalar or character vector.

Attributes:

```
GetAccess                                   public
SetAccess                                   public
```

`DocType` — Requirements document custom link type

string scalar | character vector

Requirements document custom link type, returned as a string scalar or character vector.

Attributes:

```
GetAccess                                 public
SetAccess                                 private
```

`ReqSet` — Requirement set name

character vector

Requirement set name, returned as a character vector.

Attributes:

GetAccess	public
SetAccess	private

`PreImportFcn` — Contents of `PreImportFcn` callback

string scalar | character vector

Contents of the `PreImportFcn` callback for the current Import node, specified as a string scalar or a character vector.

Attributes:

GetAccess	public
SetAccess	public

`PostImportFcn` — Contents of `PostImportFcn` callback

string scalar | character vector

Contents of the `PostImportFcn` callback for the current Import node, specified as a string scalar or a character vector.

Attributes:

GetAccess	public
SetAccess	public

Examples

Customize Word Import Options

This example shows how to customize Microsoft® Word import options by using the `PreImportFcn` callback.

Use `slreq.import` to import the Word document `Reject_Double_Button_Press_Model_Requirements.docx` into Requirements Toolbox™. Name the imported requirement set `myReqSet` and register the script `wordPreImport` as the `PreImportFcn` callback to use during import. Return a handle to the requirement set.

```
[~,~,rs] = slreq.import("Reject_Double_Button_Press_Model_Requirements.docx", ...
    ReqSet="myReqSet",preImportFcn="wordPreImport");
```

The script `wordPreImport` uses `slreq.getCurrentImportOptions` to get the import options, then sets the `Bookmark` property to `1` to use bookmarks to identify items and serve as custom IDs.

```
type wordPreImport.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.Bookmarks = 1;
```

Return the `importOptions` object.

```
importOptions
```

```
importOptions =
  MSWordImportOptions with properties:
```

```
    IgnoreOutlineNumbers: 0
               Bookmarks: 1
                   Match: []
             AsReference: 1
                RichText: 1
                  DocUri: 'C:\Users\jdoe\MATLAB\Examples\slrequirements-ex48179482\Reject_Double_
                 DocType: 'linktype_rmi_word'
                  ReqSet: 'myReqSet'
           PreImportFcn: 'wordPreImport'
          PostImportFcn: ''
```

See Also

`slreq.getCurrentImportOptions` | `setPreImportFcn` | `getPreImportFcn`

Topics

"Use Callbacks to Customize Requirement Import Behavior"

Introduced in R2022a

slreq.callback.ReqIFImportOptions class

Package: `slreq.callback`

ReqIF import options

Description

Use objects of the `slreq.callback.ReqIFImportOptions` class to adjust the options to use when import requirements. When you import requirements from a ReqIF file, `slreq.getCurrentImportOptions` generates an `slreq.callback.ReqIFImportOptions` object that you can use to adjust the options to use during import. You can only access this object in the `PreImportFcn` callback.

The `slreq.callback.ReqIFImportOptions` class is a `handle` class.

Creation

`options = slreq.getCurrentImportOptions` returns an `slreq.callback.ReqIFImportOptions` object if you are importing requirements from a ReqIF file.

Properties

`MappingFile` — Attribute mapping file

string scalar | character vector

Attribute mapping file to use during import, specified as a string scalar or character vector. Specify the full file path for the file.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

`Attr2ReqProp` — Attribute mapping

`containers.Map` object

Attribute mapping from ReqIF attributes to Requirements Toolbox properties, specified as a `containers.Map` object. For example, this code creates a `containers.Map` object that maps:

- `ReqSum` to “Summary” on page 2-0
- `Desc` to “Description” on page 2-0
- `ID` to “CustomId” on page 2-0

```
attrMap = containers.Map(ReqSum="Summary");
attrMap("Desc") = "Description";
attrMap("ID") = "Custom ID";
```

Example: `myImportOptions.Attr2ReqProp = attrMap;`

Attributes:

GetAccess	public
SetAccess	public

`SingleSpec` — Name of single specification to import

string scalar | character vector

Name of the single specification to import from the ReqIF file, specified as a string scalar or character vector. If the ReqIF file has multiple specifications, only this specification is imported.

Attributes:

GetAccess	public
SetAccess	public

`AsMultipleReqSets` — Option to import into separate requirement sets

`0` (default) | `1`

Option to import each specification into separate requirement sets, specified as a `1` or `0` of data type `logical`.

If your ReqIF file has multiple specifications and you set this property to `0`, the specifications are combined into one requirement set.

Attributes:

GetAccess	public
SetAccess	public

`ImportLinks` — Option to import links

`1` (default) | `0`

Option to import the links from the ReqIF file, specified as a `1` or `0` of data type `logical`.

Attributes:

GetAccess	public
SetAccess	public

`AutoDetectMapping` — Option to automatically detect mapping

`1` (default) | `0`

Option to allow Requirements Toolbox to automatically detect the attribute mapping to use based on the contents of the ReqIF file, specified as a `1` or `0` of data type `logical`.

Attributes:

GetAccess	public
SetAccess	public

`AsReference` — Option to import as references

`1` (default) | `0`

Option to import as `slreq.Reference` objects, specified as a `1` or `0` of data type `logical`. If `0`, requirements import as `slreq.Requirement` objects.

Attributes:

GetAccess	public
SetAccess	public

`RichText` — Option to import with rich text

`0` (default) | `1`

Option to import requirements with rich text, specified as a `1` or `0` of data type `logical`.

Attributes:

GetAccess	public
SetAccess	public

`DocUri` — Resource identifier for requirements document

string scalar | character vector

Resource identifier for external requirements document, specified as a string scalar or character vector.

Attributes:

GetAccess	public
SetAccess	public

`DocType` — Requirements document custom link type

string scalar | character vector

Requirements document custom link type, returned as a string scalar or character vector.

Attributes:

GetAccess	public
SetAccess	private

`ReqSet` — Requirement set name

character vector

Requirement set name, returned as a character vector.

Attributes:

GetAccess	public
SetAccess	private

`PreImportFcn` — Contents of `PreImportFcn` callback

string scalar | character vector

Contents of the `PreImportFcn` callback for the current Import node, specified as a string scalar or a character vector.

Attributes:

GetAccess	public
SetAccess	public

PostImportFcn — Contents of PostImportFcn callback

string scalar | character vector

Contents of the `PostImportFcn` callback for the current Import node, specified as a string scalar or a character vector.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

Examples

Customize ReqIF Import Options

This example shows how to customize ReqIF™ import options by using the `PreImportFcn` callback.

Use `slreq.import` to import the ReqIF™ file `mySpec.reqif` into Requirements Toolbox™. Name the imported requirement set `myReqSet` and register the script `myPreImportScript` as the `PreImportFcn` callback to use during import. Return a handle to the requirement set.

```
[~,~,rs] = slreq.import("mySpec.reqif",ReqSet="myReqSet",preImportFcn="myPreImportScript");
```

The script `myPreImportScript` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile.xml";
```

Return the `importOptions` object.

```
importOptions
```

```
importOptions =
  ReqIFImportOptions with properties:

          MappingFile: "myMappingFile.xml"
        Attr2ReqProp: []
          SingleSpec: ''
    AsMultipleReqSets: 0
         ImportLinks: 1
    AutoDetectMapping: 1
         AsReference: 1
            RichText: 0
              DocUri: 'C:\Users\jdoe\MATLAB\Examples\CustomizeReqIFImportOptionsExample\mySpec.
             DocType: 'REQIF'
              ReqSet: 'myReqSet'
       PreImportFcn: 'myPreImportScript'
      PostImportFcn: ''
```

See Also

`slreq.getCurrentImportOptions` | `setPreImportFcn` | `getPreImportFcn`

Topics
“Use Callbacks to Customize Requirement Import Behavior”

Introduced in R2022a

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.

Creation

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

`slreq.Link` | “Link Type Properties”

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.

Creation

`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

`slreq.Link` | “Link Type Properties”

Introduced in R2020a

Methods

add

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

Add child justification

Syntax

```
childJustification = add(jt)
childJustification = add(jt,PropertyName,
PropertyValue,...,PropertyNameN,PropertyValueN)
```

Description

`childJustification = add(jt)` adds a child justification to the justification object `jt`.

`childJustification = add(jt,PropertyName,
PropertyValue,...,PropertyNameN,PropertyValueN)` adds a child justification with the additional properties specified by `PropertyName` and `PropertyValue`.

Input Arguments

`jt` — Justification

`slreq.Justification` object

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

`PropertyName` — Justification property name

string scalar | character vector

Justification property name, specified as an string scalar or a character vector.

For more information, see `slreq.Justification` properties on page 2-48.

`PropertyValue` — Justification property value

string scalar | character vector

Justification property value, specified as an string scalar or a character vector.

Output Arguments

`childJustification` — Requirement justification

`slreq.Justification` object

New child justification, returned as an `slreq.Justification` object.

Examples

Add a Child Justification Under a Justification

This example shows how to add a child justification under another justification.

Load a requirement set file called `My_Requirement_Set_1`.

```
rs = slreq.load('C:\MATLAB\My_Requirement_Set_1.slreqx');
```

Add a justification to the requirement set.

```
jt = addJustification(rs,"Id","J1",...
"Summary","Non-functional requirement justification");
```

Add a child justification to the justification `jt`.

```
childJt = 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]
```

Tips

- To add a top-level requirement to a requirement set, use `slreq.ReqSet.add`. To add a requirement as a child of another requirement, use `slreq.Requirement.add`. To add a referenced requirement as a child of another referenced requirement, use `slreq.Requirement.add`.

See Also

`slreq.Justification` | `slreq.ReqSet.add` | `slreq.Requirement.add` | `slreq.Requirement.add` | `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
```

Tips

- To get the top-level items in a requirement set, use `slreq.ReqSet.children`. To get the child requirements of a requirement use `slreq.Requirement.children`. To get the child referenced requirements of a referenced requirement, use `slreq.Reference.children`.

See Also

`slreq.Justification` | `add` | `slreq.ReqSet.children` | `slreq.Requirement.children` | `slreq.Reference.children` | `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 executed.

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);
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);
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);
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

promote | children | parent

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
    IndexEnabled
    IndexNumber
    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'
   IndexEnabled: 1
    IndexNumber: []
            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.find` | `slreq.ReqSet` | `slreq.Justification`

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

setHierarchical | children

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 executed without error.

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

`moveDown` | `copy` | `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 executed without error. 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

`move` | `copy` | `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 executes without error. 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

`moveDown` | `copy` | `move` | `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

demote | children | 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 one of the justification objects that was created by Jane Doe.

```
count = remove(myJustifications(1), 'CreatedBy', 'Jane Doe')
```

```
count =

  1
```

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 Requirement 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

Syntax

```
dest = destination(myLink)
```

Description

`dest = destination(myLink)` returns the link destination of the link `myLink`.

Input Arguments

`myLink` — Link object

`slreq.Link` object

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

Output Arguments

`dest` — Link destination

`struct`

Link destination, returned as a MATLAB structure that contains these fields:

- `domain`
- `artifact`
- `id`
- `summary`
- `reqSet`
- `sid`

Examples

Get a Link Destination

This example shows how to get a link destination from a link object.

Open the “Requirements Definition for a Cruise Control Model” project. Load the `crs_req` requirement set, which also loads the `crs_req` link set.

```
slreqCCProjectStart;
slreq.load("crs_req");
```

Find the `crs_req` link set.

```
myLinkSet = slreq.find(Type="LinkSet",Name="crs_req");
```

Get the links from the link set.

```
myLinks = getLinks(myLinkSet)
```

```
myLinks=1×12 object
  1x12 Link array with properties:

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

Get the link destination structure for one of the links.

```
dest = destination(myLinks(1))
```

```
dest = struct with fields:
         domain: 'linktype_rmi_slreq'
       artifact: 'C:\TEMP\Bdoc22a_1891349_13144\mlx_to_docbook1\bml.batserve.009704\MATLAB\Proje
             id: '#9'
        summary: 'Enable Switch Detection'
         reqSet: 'crs_req_func_spec.slreqx'
            sid: 9
    embeddedReq: 0
```

Convert the link destination structure to an object.

```
destObj = slreq.structToObj(dest)
```

```
destObj =
  Requirement with properties:

           Type: 'Functional'
             Id: '#9'
        Summary: 'Enable Switch Detection'
    Description: '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-htm
       Keywords: {}
      Rationale: ''
      CreatedOn: 27-Feb-2017 10:15:38
      CreatedBy: 'itoy'
     ModifiedBy: 'asriram'
   IndexEnabled: 1
    IndexNumber: []
            SID: 9
   FileRevision: 55
     ModifiedOn: 03-Aug-2017 14:39:35
```

```
    Dirty: 0
 Comments: [0x0 struct]
    Index: '1.6'
```

Cleanup

Clear the open requirement sets and link sets. Close the current project.

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

Tips

- You can use `slreq.structToObj` to convert the link destination structure to an object.

See Also

`source` | `slreq.Link` | `linkSet`

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

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

Topics

“Manage Custom Attributes for Links by Using the Requirements Toolbox 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.

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

- The design artifact that contains the source or destination item is not loaded. For example, if you load a requirement set that has incoming links from a Simulink model, this also loads the link set that belongs to the model. However, if you do not load the Simulink model, the links are unresolved because the link sources are not available.
- The design artifact loaded, but the specified ID does not exist. For example, if you delete a linked requirement, the link becomes unresolved because the stored ID no longer corresponds to a valid item.

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

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 | setSource | setDestination

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.

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

- The design artifact that contains the source or destination item is not loaded. For example, if you load a requirement set that has incoming links from a Simulink model, this also loads the link set that belongs to the model. However, if you do not load the Simulink model, the links are unresolved because the link sources are not available.
- The design artifact loaded, but the specified ID does not exist. For example, if you delete a linked requirement, the link becomes unresolved because the stored ID no longer corresponds to a valid item.

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

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.

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

- The design artifact that contains the source or destination item is not loaded. For example, if you load a requirement set that has incoming links from a Simulink model, this also loads the link set that belongs to the model. However, if you do not load the Simulink model, the links are unresolved because the link sources are not available.
- The design artifact loaded, but the specified ID does not exist. For example, if you delete a linked requirement, the link becomes unresolved because the stored ID no longer corresponds to a valid item.

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

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

`slreq.Link` | `source` | `destination`

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

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

Topics

“Manage Custom Attributes for Links by Using the Requirements Toolbox 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

Requirements Toolbox linkable item

Artifact to serve as the link destination, specified as a Requirements Toolbox 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

Requirements Toolbox linkable artifact

Artifact to serve as the link source, specified as a Requirements Toolbox 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

Syntax

```
src = source(myLink)
```

Description

`src = source(myLink)` returns a link source of the link `myLink`.

Input Arguments

`myLink` — Link object

`slreq.Link` object

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

Output Arguments

`src` — Link source

`struct`

Link source, returned as a MATLAB structure that contains these fields:

- `domain`
- `artifact`
- `id`

Examples

Get a Link Source

This example shows how to get a link source from a link object.

Open the "Requirements Definition for a Cruise Control Model" project. Load the `crs_req` requirement set, which also loads the `crs_req` link set.

```
slreqCCProjectStart;
slreq.load("crs_req");
```

Find the `crs_req` link set.

```
myLinkSet = slreq.find(Type="LinkSet",Name="crs_req");
```

Get the links from the link set.

```
myLinks = getLinks(myLinkSet)
```

```
myLinks=1×12 object
  1x12 Link array with properties:

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

Get the link source structure for one of the links.

```
src = source(myLinks(1))
```

```
src = struct with fields:
      domain: 'linktype_rmi_slreq'
    artifact: 'C:\TEMP\Bdoc22a_1891349_13144\mlx_to_docbook1\bml.batserve.009704\MATLAB\Projects\
          id: '10'
```

Convert the link source structure to an object.

```
srcObj = slreq.structToObj(src)
```

```
srcObj =
  Reference with properties:

              Id: 'Enabling cruise control'
        CustomId: 'Enabling cruise control'
        Artifact: 'crs_req.docx'
      ArtifactId: '?Enabling cruise control'
          Domain: 'linktype_rmi_word'
       UpdatedOn: 02-Feb-2018 13:23:13
       CreatedOn: NaT
       CreatedBy: ''
      ModifiedBy: ''
        IsLocked: 1
         Summary: 'Enabling cruise control'
     Description: '<div class=WordSection1>...'
       Rationale: ''
        Keywords: {}
            Type: 'Functional'
    IndexEnabled: 1
     IndexNumber: []
             SID: 10
    FileRevision: 1
      ModifiedOn: 03-Aug-2017 17:34:56
           Dirty: 0
        Comments: [0x0 struct]
```

```
            Index: '3.1'
```

Cleanup

Clear the open requirement sets and link sets. Close the current project.

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

Tips

- You can use `slreq.structToObj` to convert the link source structure to an object.

See Also

`slreq.Link` | `destination` | `linkSet` | `slreq.structToObj`

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.

Setup

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

```
slreqCCProjectStart;
rs = slreq.load("crs_req_func_spec");
```

Get a handle for the `crs_controller` link set by finding the referenced requirement with summary `Driver Switch Request Handling`, getting the incoming link for that requirement, and then getting the link set that the link belongs to.

```
req = find(rs,"Summary","Driver Switch Request Handling");
myLink = inLinks(req);
ls = linkSet(myLink);
```

Add an `Edit` Custom Attribute

Add an `Edit` custom attribute to the link set. 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 and link sets without saving changes and close the open models without saving changes. Close the project.

```
slreq.clear;
bdclose all;
close(currentProject);
```

See Also

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

Topics

“Manage Custom Attributes for Links by Using the Requirements Toolbox 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

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

Topics

“Manage Custom Attributes for Links by Using the Requirements Toolbox API”

Introduced in R2020b

exportToVersion

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

Export link set to previous MATLAB version

Syntax

```
tf = exportToVersion(myLinkSet,name,version)
```

Description

`tf = exportToVersion(myLinkSet,name,version)` saves a copy of the link set `myLinkSet` as a new link set file that is compatible with the MATLAB version specified by `version` and with file name specified by `name`. The method returns `1` if the file is exported. The file is saved in the current folder.

Note You can only export link sets to version R2017b or later.

Input Arguments

`myLinkSet` — Link set

`slreq.LinkSet` object

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

`name` — File name for exported link set

string scalar | character vector

File name for exported link set, specified as a string scalar or character vector.

`version` — MATLAB version to export to

string scalar | character vector

MATLAB version to export to, specified as a string scalar or character vector.

You can export to version R2017b or later.

Example: `tf = exportToVersion(myLinkSet,"newLinkSet","R2021a")`

Output Arguments

`tf` — Export success status

`0` | `1`

Export success status, returned as a logical `1` (true) or `0` (false).

Data Types: `logical`

Examples

Export a Link Set to a Previous Version of MATLAB

This example shows how to export a link set to a file that is compatible with a previous version of MATLAB.

Open the “Requirements Definition for a Cruise Control Model” project. Load the `crs_req` requirement set, which also loads the `crs_req` link set.

```
slreqCCProjectStart;
slreq.load("crs_req");
```

Find the `crs_req` link set and assign it to a variable.

```
myLinkSet = slreq.find("Type","LinkSet","Name","crs_req")
```

```
myLinkSet =
  LinkSet with properties:

             Description: ''
                Filename: 'C:\TEMP\Bdoc22a_1891349_13144\mlx_to_docbook1\bml.batserve.009704\MATI
                Artifact: 'C:\TEMP\Bdoc22a_1891349_13144\mlx_to_docbook1\bml.batserve.009704\MATI
                  Domain: 'linktype_rmi_slreq'
                Revision: 5
                   Dirty: 0
    CustomAttributeNames: {}
```

Export the link set to a new file that is compatible with MATLAB R2020a. Name the new file `crs_req_2020a`.

```
tf = exportToVersion(myLinkSet,"crs_req_2020a","R2020a")
```

```
tf = logical
   1
```

Cleanup

Clear the open requirement sets and link sets. Close the current project.

```
slreq.clear;
close(currentProject);
```

Tips

- If the link set contains links to Model-Based Design artifacts, you might also need to export the artifacts to a previous version for the links to be resolved. For more information, see “Export Link Sets”.
- You can export a requirement set to a previous version with `slreq.ReqSet.exportToVersion`.

See Also

`slreq.LinkSet` | `slreq.ReqSet.exportToVersion`

Topics
“Export Requirement Sets and Link Sets to Previous Versions of Requirements Toolbox”

Introduced in R2018a

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 `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.

Open the “Requirements Definition for a Cruise Control Model” project. Load the `crs_req` requirement set, which also loads the `crs_req` link set. Then, find the `crs_req` link set.

```
slreqCCProjectStart;
slreq.load("crs_req");
ls = slreq.find("Type","LinkSet","Name","crs_req")
```

```
ls =
  LinkSet with properties:

            Description: ''
               Filename: 'C:\TEMP\Bdoc22a_1891349_13144\mlx_to_docbook1\bml.batserve.009704\MATL
               Artifact: 'C:\TEMP\Bdoc22a_1891349_13144\mlx_to_docbook1\bml.batserve.009704\MATL
                 Domain: 'linktype_rmi_slreq'
               Revision: 5
                  Dirty: 0
    CustomAttributeNames: {}
```

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: 02-Feb-2018 14:28:04
     ModifiedBy: 'itoy'
       Revision: 4
            SID: 3
       Comments: [0x0 struct]
```

Find all links that are modified in the specified revision.

```
myLinks = find(ls,"Revision","4")
```

```
myLinks=1×12 object
  1x12 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","4")
```

```
myLink2 =
  Link with properties:

          Type: 'Derive'
    Description: '#12: Increment Short Switch Detection'
       Keywords: {}
      Rationale: ''
      CreatedOn: 20-May-2017 13:15:45
      CreatedBy: 'itoy'
     ModifiedOn: 02-Feb-2018 14:28:04
     ModifiedBy: 'itoy'
       Revision: 4
            SID: 8
       Comments: [0x0 struct]
```

Cleanup

Clean up commands. Clear the open requirement sets and link sets. Close the current project.

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

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

getRegisteredReqSets

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

Get requirement sets registered in link set

Syntax

```
registeredReqSets = getRegisteredReqSets(myLinkSet)
```

Description

`registeredReqSets = getRegisteredReqSets(myLinkSet)` returns a cell array of the file names of the requirement sets registered to the link set `myLinkSet`.

Input Arguments

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

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

Output Arguments

`registeredReqSets` — Registered requirement set file names
cell array

File names of requirement sets registered in the link set, returned as a cell array.

Examples

Update Requirement Sets Registered in Link Set

This example shows how to get and update the requirement sets registered in a link set.

Open the Requirements Definition for a Cruise Control Model project.

```
slreqCCProjectStart;
```

Load the `crs_req` requirement set, which describes a cruise control system. This action also loads the `crs_req` link set and the `crs_req_func_spec` requirement set.

```
slreq.load("crs_req");
```

Find the `crs_req` link set and the `crs_req_func_spec` requirement set.

```
myLinkSet = slreq.find("Type","LinkSet","Name","crs_req");
rs = slreq.find("Type","ReqSet","Name","crs_req_func_spec");
```

Get the requirement sets registered in the `crs_req` link set.

```
registeredReqSets = getRegisteredReqSets(myLinkSet)
```

```
registeredReqSets = 1x1 cell array
    {'C:\TEMP\Bdoc22a_1891349_13144\mlx_to_docbook2\bml.batserve.039632\MATLAB\Projects\examples\
```

Get the links from the `crs_req` link set. Remove all of the links from the `crs_req` link set and close the `crs_req_func_spec` requirement set.

```
links = getLinks(myLinkSet);
for i = 1:numel(links)
    remove(links(i));
end
close(rs);
```

Update the requirement sets registered to the link set `crs_req`. Confirm that the requirement set `crs_req_func_spec` is not registered in the link set `crs_req` by getting the currently registered requirement sets.

```
updateRegisteredReqSets(myLinkSet)
registeredReqSets = getRegisteredReqSets(myLinkSet)
```

```
registeredReqSets =

  0x0 empty cell array
```

Cleanup

Clear the open requirement sets and link sets. Close the Requirements Definition for a Cruise Control Model project.

```
slreq.clear;
close(currentProject);
```

Tips

- When you create a link to a requirement, the requirement set of the requirement becomes registered to the link set of the link. If you delete the link to the requirement, you must manually unregister the requirement set from the link set. You can update the registered requirement sets by using `updateRegisteredReqSets`.
- You can register a requirement set without creating a link by opening a requirement set in the Requirements Perspective in the Simulink model editor.

See Also

`slreq.LinkSet` | `updateRegisteredReqSets`

Introduced in R2021b

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

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

Topics

“Manage Custom Attributes for Links by Using the Requirements Toolbox API”

Introduced in R2020b

redirectLinksToImportedReqs

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

Redirect link destination from external document to imported requirement set

Syntax

```
count = redirectLinksToImportedReqs(myLinkSet,rs)
```

Description

`count = redirectLinksToImportedReqs(myLinkSet,rs)` redirects the link destinations for the direct links in the link set `myLinkSet` from the requirements in an external document to the imported referenced requirements in the requirement set `rs`.

Input Arguments

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

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

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

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

Output Arguments

`count` — Number of updated links
character vector

Number of updated `slreq.Link` objects in the link set, returned as a character vector.

Examples

Redirect Direct Links to Imported Requirements Programmatically

This example shows how to programmatically redirect the link destinations for direct links from an external document to a corresponding imported requirement.

Open the `FuelSysWithReqLinks` model. Find the link set associated with the model.

```
open_system("FuelSysWithReqLinks.slx")
myLinkSet = slreq.find("Type","LinkSet","Name","FuelSysWithReqLinks");
```

The model contains direct links to these documents:

- `FuelSysDesignDescription.docx`
- `FuelSysRequirementsSpecification.docx`
- `FuelSysTestScenarios.xlsx`

Redirect Links to Imported References

Load the requirement set `FuelSysRequirements`. The requirement set contains imported referenced requirements from the documents listed above. The import process is described in "Migrating Requirements Management Interface Data to Requirements Toolbox".

```
rs = slreq.load("FuelSysRequirements.slreqx");
```

Redirect the link destination for the direct links in the link set `myLinkSet` to the imported referenced requirements.

```
count = redirectLinksToImportedReqs(myLinkSet,rs)
```

```
count = 13
```

Cleanup

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

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

Tips

- You can also redirect the links to imported requirements in the **Requirements Editor** or Requirements Perspective. For more information, see **Update Model Link Destinations** in "Migrating Requirements Management Interface Data to Requirements Toolbox".

See Also

Requirements Editor | `slreq.LinkSet`

Topics

"Use Command-Line API to Update or Repair Requirements Links"
"Migrating Requirements Management Interface Data to Requirements Toolbox"

Introduced in R2018a

save

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

Save link set

Syntax

```
save(lks)
save(lks, filePath)
```

Description

`save(lks)` saves the link set `lks` by using its file name.

`save(lks, filePath)` saves the link set `lks` and updates its Name and Filename properties.

Input Arguments

`lks` — Link set file

`slreq.LinkSet` object

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

`filePath` — File name and path

character vector

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

Example: `'C:\MATLAB\myLinkSet.slmx'`

Examples

Save Link Set File

Load a link set associated with a Simulink model called `fuelsys`. Save the link set.

```
myLinkSet = slreq.load('fuelsys.slx');
save(myLinkSet);
```

Save the link set to a new file.

```
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 associated with a Simulink model called `fuelsys`. Get the sources for the link set.

```
myLinkSet = slreq.load('fuelsys.slx');
mySources = sources(myLinkSet)
```

```
mySources =

  1×16 struct array with fields:

    domain
    artifact
    id
```

See Also

`slreq.LinkSet` | `save`

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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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

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

Topics

“Manage Custom Attributes for Links by Using the Requirements Toolbox API”

Introduced in R2020b

updateBacklinks

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

Synchronize external navigation links

Syntax

```
[checked,added] = updateBacklinks(myLinkSet)
[checked,added,removed] = updateBacklinks(myLinkSet,removeUnmatched)
```

Description

`[checked,added] = updateBacklinks(myLinkSet)` synchronizes backlinks in external documents to match links in the link set `myLinkSet`. The method returns the number of links in the link set that the method checked and the number of backlinks it added to the external document.

`[checked,added,removed] = updateBacklinks(myLinkSet,removeUnmatched)` removes backlinks from the external document that do not have a corresponding link in the link set when `removeUnmatched` is `true`. The method returns the number of backlinks removed from the external document.

Input Arguments

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

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

`removeUnmatched` — Option to remove unmatched backlinks
`false` (default) | `true`

Option to remove the unmatched backlinks from the external document, specified as a `1` or `0` of data type `logical`.

Output Arguments

`checked` — Links checked in link set
`double`

Number of links in the link set that the method checked, returned as a `double`.

`added` — Backlinks added in external document
`double`

Number of backlinks the method added to the external document, returned as a `double`.

`removed` — Backlinks removed in external document
`double`

Number of backlinks the method removed from the external document, returned as a `double`.

Examples

Update Backlinks for a Microsoft Word Document

This example shows how to update backlinks for a Microsoft® Word document by using `updateBacklinks`.

Open the `crs_req_func_spec` requirement set. The requirement set has outgoing links to the `crs_req.docx` document.

```
rs = slreq.open("crs_req_func_spec");
myLinkSet = slreq.find(Type="LinkSet",Name="crs_req_func_spec");
```

Update the backlinks for the external documents associated with the link set. Remove the unmatched backlinks from the external documents.

```
[checked,added,removed] = updateBacklinks(myLinkSet,true)
```

```
checked = 14
```

```
added = 4
```

```
removed = 1
```

Alternatives

App

You can also update backlinks by using the **Requirements Editor**. For more information, see “Manage Navigation Backlinks in External Requirements Documents”.

See Also

`slreq.LinkSet`

Introduced in R2022a

updateDocUri

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

Update link destination for direct links

Syntax

```
count = updateDocUri(myLinkSet,oldID,newID)
```

Description

`count = updateDocUri(myLinkSet,oldID,newID)` updates the link destinations for the direct links in the link set specified by `myLinkSet` from the external document specified by the resource identifier (such as a file path or IBM Rational DOORS module ID) `oldID` to the external document specified by the resource identifier `newID`. The method returns the number of links updated.

Note You might have to save the link set, close it, and reopen it for the changes to take effect.

Input Arguments

`myLinkSet` — Link set

`slreq.LinkSet` object

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

`oldID` — Resource identifier for original external document

string scalar | character vector

Resource identifier for the original external document, specified as a string scalar or character vector.

`newID` — Resource identifier for new external document

string scalar | character vector

Resource identifier for the new external document to use as the link destinations, specified as a string scalar or character vector.

Output Arguments

`count` — Number of updated links

character vector

Number of updated `slreq.Link` objects in the link set, returned as a character vector.

Examples

Update Direct Links to the URI of a Different External Document

This example shows how to update the link destinations for direct links to the URI of a new document.

Open the “Link to Requirements in Microsoft Word Documents” example, which uses a model that has direct links to external documents.

```
openExample('slrequirements/LinkToRequirementsInMicrosoftWordDocumentsExample');
```

Open the `slvnvdemo_fuelsys_officereq` model. Find the associated link set.

```
open_system("slvnvdemo_fuelsys_officereq.slx")
myLinkSet = slreq.find("Type","LinkSet","Name","slvnvdemo_fuelsys_officereq");
```

Update Direct Link Destinations

Some of the links in `myLinkSet` point to `slvnvdemo_FuelSys_DesignDescription.docx`. Update the link destinations to point to `slvnvdemo_FuelSys_DesignDescription_new.docx`.

```
count = updateDocUri(myLinkSet,"slvnvdemo_FuelSys_DesignDescription.docx","slvnvdemo_FuelSys_Desi
```

```
count = 8
```

Save the link set. Then close the link set and re-open it for the changes to take effect.

```
tf = save(myLinkSet)
```

```
tf = logical
   1
```

```
slreq.clear;
myLinkSet = slreq.load("slvnvdemo_fuelsys_officereq.slmx");
```

Tips

- If you rename or move an external requirements document file, use `updateSrcFileLocation` to update the file name or path of the referenced requirements in the requirement set.
- To update the external requirements document resource identifier for referenced requirements imported from non-file-based domains, use `updateSrcArtifactUri`.

See Also

`slreq.LinkSet` | `setDestination` | `setSource`

Topics

“Use Command-Line API to Update or Repair Requirements Links”

Introduced in R2018a

updateRegisteredReqSets

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

Update requirement sets registered to link set

Syntax

```
updateRegisteredReqSets(myLinkSet)
```

Description

`updateRegisteredReqSets(myLinkSet)` updates the requirement sets registered in the link set `myLinkSet`. If a currently registered requirement set has incoming links from the link set `myLinkSet` or if the requirement set is loaded, then it remains registered. Otherwise, the software unregisters the requirement sets from the link set `myLinkSet`.

Input Arguments

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

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

Examples

Update Requirement Sets Registered in Link Set

This example shows how to get and update the requirement sets registered in a link set.

Open the Requirements Definition for a Cruise Control Model project.

```
slreqCCProjectStart;
```

Load the `crs_req` requirement set, which describes a cruise control system. This action also loads the `crs_req` link set and the `crs_req_func_spec` requirement set.

```
slreq.load("crs_req");
```

Find the `crs_req` link set and the `crs_req_func_spec` requirement set.

```
myLinkSet = slreq.find("Type","LinkSet","Name","crs_req");
rs = slreq.find("Type","ReqSet","Name","crs_req_func_spec");
```

Get the requirement sets registered in the `crs_req` link set.

```
registeredReqSets = getRegisteredReqSets(myLinkSet)
```

```
registeredReqSets = 1x1 cell array
    {'C:\TEMP\Bdoc22a_1891349_13144\mlx_to_docbook2\bml.batserve.039632\MATLAB\Projects\examples\
```

Get the links from the `crs_req` link set. Remove all of the links from the `crs_req` link set and close the `crs_req_func_spec` requirement set.

```
links = getLinks(myLinkSet);
for i = 1:numel(links)
    remove(links(i));
end
close(rs);
```

Update the requirement sets registered to the link set `crs_req`. Confirm that the requirement set `crs_req_func_spec` is not registered in the link set `crs_req` by getting the currently registered requirement sets.

```
updateRegisteredReqSets(myLinkSet)
registeredReqSets = getRegisteredReqSets(myLinkSet)
```

```
registeredReqSets =

  0x0 empty cell array
```

Cleanup

Clear the open requirement sets and link sets. Close the Requirements Definition for a Cruise Control Model project.

```
slreq.clear;
close(currentProject);
```

Tips

- When you create a link to a requirement, the requirement set of the requirement becomes registered to the link set of the link. If you delete the link to the requirement, you must manually unregister the requirement set from the link set. You can get the currently registered requirement sets for the link set by using `getRegisteredReqSets`.
- You can only unregister a requirement set that is not loaded.
- Loading the link set loads the requirement sets registered to that link set. For more information, see “Load Link Information”.

See Also

`slreq.LinkSet` | `getRegisteredReqSets`

Introduced in R2018a

add

Class: `slreq.Reference`
Package: `slreq`

Add child referenced requirement

Syntax

```
refChild = add(ref,"Artifact",FileName)
refChild = add(ref,"Artifact",FileName,PropertyName,
PropertyValue,...,PropertyNameN,PropertyValueN)
```

Description

`refChild = add(ref,"Artifact",FileName)` adds a child referenced requirement under the referenced requirement `ref` that references requirements in the external document, `FileName`.

`refChild = add(ref,"Artifact",FileName,PropertyName,
PropertyValue,...,PropertyNameN,PropertyValueN)` adds a child referenced requirement with properties and property values specified by `PropertyName` and `PropertyValue`.

Input Arguments

`ref` — Referenced requirement

`slreq.Reference` object

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

`FileName` — External requirements document identifier

string scalar | character vector

External requirements document identifier, specified as a string scalar or character vector. Examples of a document identifier are a Microsoft Office document name or an IBM Rational DOORS Module unique ID.

`PropertyName` — Referenced requirement property name

string scalar | character vector

Referenced requirement property name, specified as an string scalar or a character vector.

You can only enter an `slreq.Reference` property on page 2-63 where the `SetAccess` attribute is `public`.

Example: `"Summary"`

`PropertyValue` — Referenced requirement property value

string scalar | character vector

Referenced requirement property value, specified as an string scalar or a character vector.

Output Arguments

`refChild` — Referenced child requirement
`slreq.Reference` object

New referenced child requirement, returned as an `slreq.Reference` object.

Examples

Add a Child Referenced Requirement under a Referenced Requirement

This example shows how to add a child referenced requirement under a referenced requirement.

Open the “Requirements Definition for a Cruise Control Model” project and load the `crs_req` requirement set

```
slreqCCProjectStart;
rs = slreq.load("crs_req");
```

Find the top-level referenced requirement with the summary `Functional Requirements`. Add a child referenced requirement under that referenced requirement that uses the same external document as the top-level referenced requirement.

```
topRef = find(rs,"Summary","Functional Requirements");
childRef = add(topRef,"Artifact",topRef.Artifact)
```

```
childRef =
  Reference with properties:

              Id: ''
        CustomId: ''
        Artifact: 'crs_req.docx'
      ArtifactId: ''
          Domain: 'linktype_rmi_word'
       UpdatedOn: 16-Sep-2021 14:56:35
       CreatedOn: 16-Sep-2021 14:56:35
       CreatedBy: ''
      ModifiedBy: ''
        IsLocked: 1
         Summary: ''
     Description: ''
       Rationale: ''
        Keywords: {}
            Type: 'Functional'
             SID: 32
    FileRevision: 1
      ModifiedOn: 16-Sep-2021 14:56:35
           Dirty: 0
        Comments: [0×0 struct]
           Index: '3.13'
```

Cleanup

Discard the requirement set without saving the changes. Clear the remaining requirement sets and link sets. Close the project.

```
discard(rs);
slreq.clear;
close(currentProject);
```

Tips

- To add a top-level requirement to a requirement set, use `slreq.ReqSet.add`. To add a requirement as a child of another requirement, use `slreq.Requirement.add`. To add a justification as a child of another justification, use `slreq.Justification.add`.

See Also

`slreq.Reference` | `slreq.ReqSet.add` | `slreq.Requirement.add` | `slreq.Justification.add`

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 requirement 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
```

Tips

- To get the top-level items in a requirement set, use `slreq.ReqSet.children`. To get the child requirements of a requirement use `slreq.Requirement.children`. To get the child justifications of a justification, use `slreq.Justification.children`.

See Also

`slreq.Reference` | `slreq.ReqSet` | `slreq.ReqSet.children` | `slreq.Requirement.children` | `slreq.Justification.children` | `parent`

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'
    IndexEnabled: 1
     IndexNumber: []
             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
    IndexEnabled
    IndexNumber
    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'
   IndexEnabled: 1
    IndexNumber: []
            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

`slreq.Reference` | `slreq.ReqSet` | `setAttribute`

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

getPostImportFcn

Class: `slreq.Reference`
Package: `slreq`

Get contents of `PostImportFcn` callback

Syntax

```
callback = getPostImportFcn(topRef)
```

Description

`callback = getPostImportFcn(topRef)` returns the contents of the `PostImportFcn` callback for the Import node `topRef`.

Input Arguments

`topRef` — Import node

`slreq.Reference` object

Import node, specified as an `slreq.Reference` object.

Output Arguments

`callback` — Contents of `PostImportFcn` callback

character vector

Contents of the `PostImportFcn` callback for the Import node, returned as a character vector.

Examples

Use `PostImportFcn` Callback During Import

This example shows how to assign a script as the `PostImportFcn` callback for an Import node. You get the contents of the `PostImportFcn` callback for an Import node and register a different script after you import the requirements.

Import the Requirements

Use `slreq.import` to import the ReqIF file `mySpec.reqif` into Requirements Toolbox™. Name the imported requirement set `myReqSet`, register the script `myPreImportScript2` as the `PreImportFcn`, and register the script `myPostImportScript` as the `PostImportFcn` callback. Return a handle to the requirement set.

```
[~,~,rs] = slreq.import("mySpec.reqif",ReqSet="myReqSet",preImportFcn="myPreImportScript2",postIm
```

The script `myPreImportScript2` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript2.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile2.xml";
```

The mapping file `myMappingFile2.xml` maps these attributes from the ReqIF™ file to these properties in Requirements Toolbox™:

- `ReqSum` to `Summary`
- `Desc` to `Description`
- `ID` to `Custom ID`

The script `myPostImportScript` uses `slreq.getCurrentObject` to get a handle to the Import node, gets the requirement set that the Import node belongs to, and then finds requirements that have the summary `Requirement 1` and `Requirement 2`. Then, the script moves `Requirement 2` under `Requirement 1`.

```
type myPostImportScript.m
```

```
topRef = slreq.getCurrentObject;
rs = reqSet(topRef);
ref = find(rs,Type="Reference",Summary="Requirement 2");
parentRef = find(rs,Type="Reference",Summary="Requirement 1");
parentID = parentRef.SID;
setParent(ref,parentID);
```

Confirm that `Requirement 2` is a child of `Requirement 1`.

```
req1 = find(rs,Summary="Requirement 1");
req2 = children(req1);
reqSummary = req2.Summary
```

```
reqSummary =
'Requirement 2'
```

Get and Set the `PostImportFcn` Callback

Get a handle to the Import node, then register the script `myPostImportScrip2` as the `PostImportFcn` callback. Confirm that the contents of the callback changed.

```
topRef = children(rs);
setPostImportFcn(topRef,"myPostImportScript2")
newCallback = getPostImportFcn(topRef)
```

```
newCallback =
'myPostImportScript2'
```

The `myPostImportScript2` script moves `Requirement 2` under `Requirement 3`.

```
type myPostImportScript2.m
```

```
topRef = slreq.getCurrentObject;
rs = reqSet(topRef);
ref = find(rs,Type="Reference",Summary="Requirement 2");
parentRef = find(rs,Type="Reference",Summary="Requirement 3");
parentID = parentRef.SID;
setParent(ref,parentID);
```

Update the requirement set. The `PostImportFcn` callback executes after you update the requirement set.

```
updateReferences(rs,topRef);
```

Confirm that `Requirement 2` is a child of `Requirement 3`.

```
req3 = find(rs,Summary="Requirement 3");
req2 = children(req3);
reqSummary = req2.Summary
```

```
reqSummary =
'Requirement 2'
```

See Also

`getPreImportFcn` | `setPreImportFcn` | `setPostImportFcn` | `setParent`

Topics

“Use Callbacks to Customize Requirement Import Behavior”

Introduced in R2022a

getPreImportFcn

Class: `slreq.Reference`
Package: `slreq`

Get registered `PreImportFcn` callback script

Syntax

```
callback = getPreImportFcn(topRef)
```

Description

`callback = getPreImportFcn(topRef)` returns the contents of the `PreImportFcn` callback for the Import node `topRef`.

Input Arguments

`topRef` — Import node

`slreq.Reference` object

Import node, specified as an `slreq.Reference` object.

Output Arguments

`callback` — Contents of `PreImportFcn` callback

character vector

Contents of the `PreImportFcn` callback for the Import node, returned as a character vector.

Examples

Use `PreImportFcn` Callback During Import

This example shows how to assign a script as the `PreImportFcn` callback for an Import node. You get the contents of the `PreImportFcn` callback for an Import node and register a different script as the `PreImportFcn` callback after you import the requirements.

Import the Requirements

Use `slreq.import` to import the ReqIF™ file `mySpec.reqif` into Requirements Toolbox™. Name the imported requirement set `myReqSet` and register the script `myPreImportScript` as the `PreImportFcn` callback to use during import. Return a handle to the requirement set.

```
[~,~,rs] = slreq.import("mySpec.reqif",ReqSet="myReqSet",preImportFcn="myPreImportScript");
```

The script `myPreImportScript` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile.xml";
```

The mapping file `myMappingFile.xml` uses a generic mapping.

Get the custom ID for the requirement with `Index` set to `1`.

```
req1 = find(rs,Index="1");
cID = req1.CustomId
```

```
cID =

  0x0 empty char array
```

The generic mapping does not map the ReqIF attribute `ID` to the Requirement Toolbox attribute `Custom ID`. Instead, `ID` imports as a custom attribute. Get the value for the `ID` custom attribute for `Requirement 1`.

```
cID = getAttribute(req1,"ID")
```

```
cID =
'A1'
```

Get and Set the `PreImportFcn` Callback Script

Get a handle to the Import node, then register the script `myPreImportScrip2` as the `PreImportFcn` callback. Confirm that the registered callback was changed.

```
topRef = children(rs);
setPreImportFcn(topRef,"myPreImportScript2")
newCallback = getPreImportFcn(topRef)
```

```
newCallback =
'myPreImportScript2'
```

The script `myPreImportScript2` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript2.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile2.xml";
```

The mapping file `myMappingFile2.xml` maps these attributes from the ReqIF™ file to these properties in Requirements Toolbox™:

- `ReqSum` to `Summary`
- `Desc` to `Description`
- `ID` to `Custom ID`

Update the requirement set. The `PreImportFcn` callback script also executes when you update the requirement set.

```
updateReferences(rs,topRef);
```

Get the custom ID for the requirement with `Index` set to `1`.

```
req1 = find(rs,Index="1");
cID = req1.CustomId
```

```
cID =
'A1'
```

See Also

`getPostImportFcn` | `setPreImportFcn` | `setPostImportFcn`

Topics

“Use Callbacks to Customize Requirement Import Behavior”

Introduced in R2022a

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

`getImplementationStatus` | `addJustification`

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

moveDown

Class: `slreq.Reference`
Package: `slreq`

Move referenced requirement down in hierarchy

Syntax

```
tf = moveDown(ref)
```

Description

`tf = moveDown(ref)` moves the referenced requirement `ref` down one spot in the hierarchy, and returns `1` if the move executes without error. The referenced requirement `ref` cannot be moved to a new level in the hierarchy.

Note You can use this method only in the `PostImportFcn` callback.

Input Arguments

`ref` — Referenced requirement

`slreq.Reference` object

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

Output Arguments

`tf` — Move success status

`0` | `1`

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

Examples

Move Referenced Requirement in `PostImportFcn` Callback

This example shows how to move an imported referenced requirement up and down in the hierarchy in the `PostImportFcn` callback.

Use `slreq.import` to import the ReqIF™ file `mySpec.reqif` into Requirements Toolbox™. Name the imported requirement set `myReqSet`, register the script `myPreImportScript2` as the `PreImportFcn`, and register the script `movePostImport` as the `PostImportFcn` callback to use during import. Return a handle to the requirement set.

```
[~,~,rs] = slreq.import("mySpec.reqif",ReqSet="myReqSet",preImportFcn="myPreImportScript2",postIm
```

The script `myPreImportScript2` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript2.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile2.xml";
```

The mapping file `myMappingFile2.xml` maps these attributes from the ReqIF file to these properties in Requirements Toolbox:

- `ReqSum` to `Summary`
- `Desc` to `Description`
- `ID` to `Custom ID`

The script `myPostImportScript` uses `slreq.getCurrentObject` to get a handle to the import node, gets the requirement set that the import node belongs to. The script then finds the referenced requirement that has `Summary` set to `Requirement 3` and moves it up. It also finds the referenced requirement that has `Summary` set to `Requirement 1` and moves it down.

```
type movePostImport.m
```

```
topRef = slreq.getCurrentObject;
rs = reqSet(topRef);
ref1 = find(rs,Type="Reference",Summary="Requirement 3");
tf1 = moveUp(ref1);
ref2 = find(rs,Type="Reference",Summary="Requirement 1");
tf2 = moveDown(ref2);
```

See Also

`slreq.Reference` | `remove` | `moveUp` | `setParent`

Topics

“Use Callbacks to Customize Requirement Import Behavior”

Introduced in R2022a

moveUp

Class: `slreq.Reference`
Package: `slreq`

Move referenced requirement up in hierarchy

Syntax

```
tf = moveUp(ref)
```

Description

`tf = moveUp(ref)` moves the referenced requirement `ref` up one spot in the hierarchy, and returns `1` if the move executes without error. The referenced requirement `ref` cannot be moved to a new level in the hierarchy.

Note You can use this method only in the `PostImportFcn` callback.

Input Arguments

`ref` — Referenced requirement

`slreq.Reference` object

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

Output Arguments

`tf` — Move success status

`0` | `1`

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

Examples

Move Referenced Requirement in `PostImportFcn` Callback

This example shows how to move an imported referenced requirement up and down in the hierarchy in the `PostImportFcn` callback.

Use `slreq.import` to import the ReqIF™ file `mySpec.reqif` into Requirements Toolbox™. Name the imported requirement set `myReqSet`, register the script `myPreImportScript2` as the `PreImportFcn`, and register the script `movePostImport` as the `PostImportFcn` callback to use during import. Return a handle to the requirement set.

```
[~,~,rs] = slreq.import("mySpec.reqif",ReqSet="myReqSet",preImportFcn="myPreImportScript2",postIm
```

The script `myPreImportScript2` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript2.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile2.xml";
```

The mapping file `myMappingFile2.xml` maps these attributes from the ReqIF file to these properties in Requirements Toolbox:

- `ReqSum` to `Summary`
- `Desc` to `Description`
- `ID` to `Custom ID`

The script `myPostImportScript` uses `slreq.getCurrentObject` to get a handle to the import node, gets the requirement set that the import node belongs to. The script then finds the referenced requirement that has `Summary` set to `Requirement 3` and moves it up. It also finds the referenced requirement that has `Summary` set to `Requirement 1` and moves it down.

```
type movePostImport.m
```

```
topRef = slreq.getCurrentObject;
rs = reqSet(topRef);
ref1 = find(rs,Type="Reference",Summary="Requirement 3");
tf1 = moveUp(ref1);
ref2 = find(rs,Type="Reference",Summary="Requirement 1");
tf2 = moveDown(ref2);
```

See Also

`slreq.Reference` | `remove` | `moveDown` | `setParent`

Topics

“Use Callbacks to Customize Requirement Import Behavior”

Introduced in R2022a

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 requirement 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

`slreq.Reference` | `slreq.ReqSet` | `children`

Introduced in R2018a

remove

Class: `slreq.Reference`
Package: `slreq`

Remove referenced requirements

Syntax

```
count = remove(topRef)
count = remove(ref)
```

Description

`count = remove(topRef)` removes all descendant referenced requirements under the import node `topRef` as well as the import node itself. The function returns the number of referenced requirements removed.

`count = remove(ref)` removes the referenced requirement `ref` and the descendant referenced requirements. The function returns the number of referenced requirements removed. You can use this syntax only in the `PostImportFcn` callback.

Input Arguments

`topRef` — Import node
`slreq.Reference` object

Import node, specified as an `slreq.Reference` object.

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

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

Output Arguments

`count` — Removed referenced requirements count
double

The number of referenced requirements removed, returned as a double.

Examples

Remove Import Node from Requirement Set

Load a requirement set file called `myReqSet`.

```
rs = slreq.load("myReqSet");
```

Get a handle to the import node.

```
topRef = children(rs);
```

Remove the import node and its descendant requirements.

```
count = remove(topRef)
```

```
count =

    46
```

Remove Referenced Requirement in PostImportFcn Callback

This example shows how to remove an imported referenced requirement in the `PostImportFcn` callback.

Use `slreq.import` to import the ReqIF™ file `mySpec.reqif` into Requirements Toolbox™. Name the imported requirement set `myReqSet`, register the script `myPreImportScript2` as the `PreImportFcn`, and register the script `removePostImport` as the `PostImportFcn` callback to use during import. Return a handle to the requirement set.

```
[~,~,rs] = slreq.import("mySpec.reqif",ReqSet="myReqSet", ...
    preImportFcn="myPreImportScript2",postImportFcn="removePostImport");
```

The script `myPreImportScript2` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript2.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile2.xml";
```

The mapping file `myMappingFile2.xml` maps these attributes from the ReqIF file to these properties in Requirements Toolbox:

- `ReqSum` to `Summary`
- `Desc` to `Description`
- `ID` to `Custom ID`

The script `myPostImportScript` uses `slreq.getCurrentObject` to get a handle to the import node, gets the requirement set that the import node belongs to, then finds and removes the referenced requirement that has `Summary` set to `Requirement 3`.

```
type removePostImport.m
```

```
topRef = slreq.getCurrentObject;
rs = reqSet(topRef);
ref = find(rs,Type="Reference",Summary="Requirement 3");
count = remove(ref);
```

See Also

`add` | `slreq.Reference`

Topics
“Use Callbacks to Customize Requirement Import Behavior”

Introduced in R2019a

reqSet

Class: `slreq.Reference`
Package: `slreq`

Return parent requirement set

Syntax

```
rsout = reqSet(ref)
```

Description

`rsout = reqSet(ref)` returns the parent requirement set `rsout` to which the referenced requirement `ref` belongs.

Input Arguments

`ref` — Referenced requirement

`slreq.Reference` object

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

Output Arguments

`rsout` — Parent requirement set

`slreq.ReqSet` object

The parent requirement set of the referenced requirement `ref`, returned as an `slreq.ReqSet` object.

Examples

Query Requirement Set Information

```
% Load a new requirement 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 requirement 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

`slreq.Reference` | `slreq.ReqSet` | `parent`

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 requirement 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 requirement 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

`slreq.Reference` | `slreq.ReqSet` | `getAttribute`

Introduced in R2018a

setParent

Class: `slreq.Reference`
Package: `slreq`

Set parent of referenced requirement in `PostImportFcn` callback

Syntax

```
setParent(ref,parentID)
```

Description

`setParent(ref,parentID)` moves the referenced requirement `ref` under the parent referenced requirement specified by `parentID`. You can only use this method in the `PostImportFcn` callback.

Input Arguments

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

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

`parentID` — SID of parent referenced requirement
`int32` | `double`

`SID on page 2-0` of the parent referenced requirement, specified as an `int32` or a `double`.

Examples

Use `PostImportFcn` Callback During Import

This example shows how to assign a script as the `PostImportFcn` callback for an Import node. You get the contents of the `PostImportFcn` callback for an Import node and register a different script after you import the requirements.

Import the Requirements

Use `slreq.import` to import the ReqIF file `mySpec.reqif` into Requirements Toolbox™. Name the imported requirement set `myReqSet`, register the script `myPreImportScript2` as the `PreImportFcn`, and register the script `myPostImportScript` as the `PostImportFcn` callback. Return a handle to the requirement set.

```
[~,~,rs] = slreq.import("mySpec.reqif",ReqSet="myReqSet",preImportFcn="myPreImportScript2",postIm
```

The script `myPreImportScript2` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript2.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile2.xml";
```

The mapping file `myMappingFile2.xml` maps these attributes from the ReqIF™ file to these properties in Requirements Toolbox™:

- `ReqSum` to `Summary`
- `Desc` to `Description`
- `ID` to `Custom ID`

The script `myPostImportScript` uses `slreq.getCurrentObject` to get a handle to the Import node, gets the requirement set that the Import node belongs to, and then finds requirements that have the summary `Requirement 1` and `Requirement 2`. Then, the script moves `Requirement 2` under `Requirement 1`.

```
type myPostImportScript.m
```

```
topRef = slreq.getCurrentObject;
rs = reqSet(topRef);
ref = find(rs,Type="Reference",Summary="Requirement 2");
parentRef = find(rs,Type="Reference",Summary="Requirement 1");
parentID = parentRef.SID;
setParent(ref,parentID);
```

Confirm that `Requirement 2` is a child of `Requirement 1`.

```
req1 = find(rs,Summary="Requirement 1");
req2 = children(req1);
reqSummary = req2.Summary
```

```
reqSummary =
'Requirement 2'
```

Get and Set the `PostImportFcn` Callback

Get a handle to the Import node, then register the script `myPostImportScrip2` as the `PostImportFcn` callback. Confirm that the contents of the callback changed.

```
topRef = children(rs);
setPostImportFcn(topRef,"myPostImportScript2")
newCallback = getPostImportFcn(topRef)
```

```
newCallback =
'myPostImportScript2'
```

The `myPostImportScript2` script moves `Requirement 2` under `Requirement 3`.

```
type myPostImportScript2.m
```

```
topRef = slreq.getCurrentObject;
rs = reqSet(topRef);
ref = find(rs,Type="Reference",Summary="Requirement 2");
parentRef = find(rs,Type="Reference",Summary="Requirement 3");
parentID = parentRef.SID;
setParent(ref,parentID);
```

Update the requirement set. The `PostImportFcn` callback executes after you update the requirement set.

```
updateReferences(rs,topRef);
```

Confirm that `Requirement 2` is a child of `Requirement 3`.

```
req3 = find(rs,Summary="Requirement 3");
req2 = children(req3);
reqSummary = req2.Summary
```

```
reqSummary =
'Requirement 2'
```

See Also

`slreq.Reference` | `getPostImportFcn` | `setPostImportFcn` | `moveUp` | `moveDown`

Topics

"Use Callbacks to Customize Requirement Import Behavior"

Introduced in R2022a

setPostImportFcn

Class: `slreq.Reference`
Package: `slreq`

Assign `PostImportFcn` callback script

Syntax

```
setPostImportFcn(topRef,callbackScript)
```

Description

`setPostImportFcn(topRef,callbackScript)` assigns the script specified by `callbackScript` as the `PostImportFcn` callback script for the Import node `topRef`.

Input Arguments

`topRef` — Import node

`slreq.Reference` object

Import node, specified as an `slreq.Reference` object.

`callbackScript` — Name of script to register

string scalar | character vector

Name of the script to register as the `PostImportFcn` callback for the Import node, specified as a string scalar or character vector.

Examples

Use `PostImportFcn` Callback During Import

This example shows how to assign a script as the `PostImportFcn` callback for an Import node. You get the contents of the `PostImportFcn` callback for an Import node and register a different script after you import the requirements.

Import the Requirements

Use `slreq.import` to import the ReqIF file `mySpec.reqif` into Requirements Toolbox™. Name the imported requirement set `myReqSet`, register the script `myPreImportScript2` as the `PreImportFcn`, and register the script `myPostImportScript` as the `PostImportFcn` callback. Return a handle to the requirement set.

```
[~,~,rs] = slreq.import("mySpec.reqif",ReqSet="myReqSet",preImportFcn="myPreImportScript2",postIm
```

The script `myPreImportScript2` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript2.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile2.xml";
```

The mapping file `myMappingFile2.xml` maps these attributes from the ReqIF™ file to these properties in Requirements Toolbox™:

- `ReqSum` to `Summary`
- `Desc` to `Description`
- `ID` to `Custom ID`

The script `myPostImportScript` uses `slreq.getCurrentObject` to get a handle to the Import node, gets the requirement set that the Import node belongs to, and then finds requirements that have the summary `Requirement 1` and `Requirement 2`. Then, the script moves `Requirement 2` under `Requirement 1`.

```
type myPostImportScript.m
```

```
topRef = slreq.getCurrentObject;
rs = reqSet(topRef);
ref = find(rs,Type="Reference",Summary="Requirement 2");
parentRef = find(rs,Type="Reference",Summary="Requirement 1");
parentID = parentRef.SID;
setParent(ref,parentID);
```

Confirm that `Requirement 2` is a child of `Requirement 1`.

```
req1 = find(rs,Summary="Requirement 1");
req2 = children(req1);
reqSummary = req2.Summary
```

```
reqSummary =
'Requirement 2'
```

Get and Set the `PostImportFcn` Callback

Get a handle to the Import node, then register the script `myPostImportScrip2` as the `PostImportFcn` callback. Confirm that the contents of the callback changed.

```
topRef = children(rs);
setPostImportFcn(topRef,"myPostImportScript2")
newCallback = getPostImportFcn(topRef)
```

```
newCallback =
'myPostImportScript2'
```

The `myPostImportScript2` script moves `Requirement 2` under `Requirement 3`.

```
type myPostImportScript2.m
```

```
topRef = slreq.getCurrentObject;
rs = reqSet(topRef);
ref = find(rs,Type="Reference",Summary="Requirement 2");
parentRef = find(rs,Type="Reference",Summary="Requirement 3");
parentID = parentRef.SID;
setParent(ref,parentID);
```

Update the requirement set. The `PostImportFcn` callback executes after you update the requirement set.

```
updateReferences(rs,topRef);
```

Confirm that `Requirement 2` is a child of `Requirement 3`.

```
req3 = find(rs,Summary="Requirement 3");
req2 = children(req3);
reqSummary = req2.Summary
```

```
reqSummary =
'Requirement 2'
```

See Also

`getPostImportFcn` | `getPreImportFcn` | `setPreImportFcn` | `setParent`

Topics

“Use Callbacks to Customize Requirement Import Behavior”

Introduced in R2022a

setPreImportFcn

Class: `slreq.Reference`
Package: `slreq`

Assign `PreImportFcn` callback script

Syntax

```
setPreImportFcn(topRef,callbackScript)
```

Description

`setPreImportFcn(topRef,callbackScript)` assigns the script specified by `callbackScript` as the `PreImportFcn` callback script for the Import node `topRef`.

Input Arguments

`topRef` — Import node

`slreq.Reference` object

Import node, specified as an `slreq.Reference` object.

`callbackScript` — Name of script to register

string scalar | character vector

Name of the script to register as the `PreImportFcn` callback for the Import node, specified as a string scalar or character vector.

Examples

Use `PreImportFcn` Callback During Import

This example shows how to assign a script as the `PreImportFcn` callback for an Import node. You get the contents of the `PreImportFcn` callback for an Import node and register a different script as the `PreImportFcn` callback after you import the requirements.

Import the Requirements

Use `slreq.import` to import the ReqIF™ file `mySpec.reqif` into Requirements Toolbox™. Name the imported requirement set `myReqSet` and register the script `myPreImportScript` as the `PreImportFcn` callback to use during import. Return a handle to the requirement set.

```
[~,~,rs] = slreq.import("mySpec.reqif",ReqSet="myReqSet",preImportFcn="myPreImportScript");
```

The script `myPreImportScript` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile.xml";
```

The mapping file `myMappingFile.xml` uses a generic mapping.

Get the custom ID for the requirement with `Index` set to `1`.

```
req1 = find(rs,Index="1");
cID = req1.CustomId
```

```
cID =

  0x0 empty char array
```

The generic mapping does not map the ReqIF attribute `ID` to the Requirement Toolbox attribute `Custom ID`. Instead, `ID` imports as a custom attribute. Get the value for the `ID` custom attribute for `Requirement 1`.

```
cID = getAttribute(req1,"ID")
```

```
cID =
'A1'
```

Get and Set the `PreImportFcn` Callback Script

Get a handle to the Import node, then register the script `myPreImportScrip2` as the `PreImportFcn` callback. Confirm that the registered callback was changed.

```
topRef = children(rs);
setPreImportFcn(topRef,"myPreImportScript2")
newCallback = getPreImportFcn(topRef)
```

```
newCallback =
'myPreImportScript2'
```

The script `myPreImportScript2` uses `slreq.getCurrentImportOptions` to get the import options, then specifies the attribute mapping file to use during import.

```
type myPreImportScript2.m
```

```
importOptions = slreq.getCurrentImportOptions;
importOptions.MappingFile = "myMappingFile2.xml";
```

The mapping file `myMappingFile2.xml` maps these attributes from the ReqIF™ file to these properties in Requirements Toolbox™:

- `ReqSum` to `Summary`
- `Desc` to `Description`
- `ID` to `Custom ID`

Update the requirement set. The `PreImportFcn` callback script also executes when you update the requirement set.

```
updateReferences(rs,topRef);
```

Get the custom ID for the requirement with `Index` set to `1`.

```
req1 = find(rs,Index="1");
cID = req1.CustomId
```

```
cID =
'A1'
```

See Also

`getPostImportFcn` | `getPreImportFcn` | `setPostImportFcn`

Topics

“Use Callbacks to Customize Requirement Import Behavior”

Introduced in R2022a

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

```
[status,changeList] = updateFromDocument(topRef)
```

Description

`[status,changeList] = updateFromDocument(topRef)` updates the referenced requirements under the Import node `topRef`. The function returns the update status and a list of updated requirements.

Input Arguments

`topRef` — Import node

`slreq.Reference` object

Import node, specified as an `slreq.Reference` object.

Output Arguments

`status` — Update status

character vector

Requirement set update status, returned as a character vector.

`changeList` — List of updated referenced requirements

character vector

List of updated referenced requirements, returned as a character vector. The list includes the properties on page 2-63 of each referenced requirement changed by the function.

Examples

Update Referenced Requirements

This example shows how to update referenced requirements from an external requirements document.

Load a requirement set file called `My_Requirement_Set_1`.

```
rs = slreq.load("C:\MATLAB\My_Requirement_Set_1.slreqx");
```

Find the 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
```

The Import node is the first referenced requirement in the array. Update the referenced requirements underneath the Import node.

```
[status,changeList] = updateFromDocument(allRefs(1))
```

```
status =

    'Update completed. Refer to Comments on Import1.'


changeList =

    'Updated: Robot Arm Sensor. Properties: description
     Updated: System Inputs. Properties: description
     Updated: System Outputs. Properties: description'
```

Tips

- You can use `updateReferences` to update the referenced requirements in a requirement set by specifying the external requirements document identifier.

See Also

`slreq.Reference` | `slreq.import` | `updateReferences`

Topics

“Update Imported Requirements”

Introduced in R2019a

add

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

Add requirements to requirement set

Syntax

```
req = add(rs)
req = add(rs,"Artifact",artifactName)
req = add( ___ ,reqProperty,value,...,refPropertyN,valueN)
```

Description

`req = add(rs)` adds a requirement to the requirement set `rs` and returns a handle to the requirement.

`req = add(rs,"Artifact",artifactName)` adds a referenced requirement associated with the external requirements document, `artifactName`.

`req = add( ___ ,reqProperty,value,...,refPropertyN,valueN)` adds a requirement or a referenced requirement to the requirement set with properties and property values specified by `reqProperty` and `value`, respectively.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

`reqProperty` — Requirement property name

string scalar | character vector

Requirement or referenced requirement property name, specified as a string scalar or a character vector.

You can only enter an `slreq.Requirement` property on page 2-74 or `slreq.Reference` property on page 2-63 where the `SetAccess` attribute is `public`.

Example: `"Summary"`

`value` — Requirement property value

string scalar | character vector

Requirement or referenced requirement property value, specified as an string scalar or a character vector.

`artifactName` — External requirements document name

string scalar | character vector

External requirements document name, specified as a string scalar or a character vector.

Output Arguments

`req` — Requirement

`slreq.Requirement` object | `slreq.Reference` object

Requirement or referenced requirement, returned as an `slreq.Requirement` or an `slreq.Reference` object.

Examples

Add a Requirement to a Requirement Set

This example shows how to add a requirement to a requirement set.

Load the requirement set `myReqSet`, which does not contain any requirements.

```
rs = slreq.load("myReqSet");
```

Use the `add` method to add a requirement to the requirement set.

```
req = add(rs)
```

```
req =
  Requirement with properties:

            Type: 'Functional'
              Id: '#2'
         Summary: ''
     Description: ''
        Keywords: {}
       Rationale: ''
       CreatedOn: 27-Feb-2022 04:28:45
       CreatedBy: 'batserve'
      ModifiedBy: 'batserve'
    IndexEnabled: 1
     IndexNumber: []
             SID: 2
    FileRevision: 1
      ModifiedOn: 27-Feb-2022 04:28:45
           Dirty: 1
        Comments: [0x0 struct]
           Index: '1'
```

Cleanup

Discard the requirement set without saving.

```
discard(rs);
```

Add a Referenced Requirement to a Requirement Set

This example shows how to add a referenced requirement to a requirement set.

Open the “Requirements Definition for a Cruise Control Model” project and load the `crs_req` requirement set.

```
slreqCCProjectStart;
rs = slreq.load("crs_req");
```

Use the `add` method to add a referenced requirement to the requirement set as an Import node. Associate the Import node with the `crs_req.docx` file as the external requirements artifact.

```
ref = add(rs,"Artifact","crs_req.docx")
```

```
ref =
  Reference with properties:

             Id: ''
       CustomId: ''
       Artifact: 'crs_req.docx'
     ArtifactId: ''
         Domain: 'linktype_rmi_word'
      UpdatedOn: 16-Sep-2021 09:59:25
      CreatedOn: 16-Sep-2021 09:59:25
      CreatedBy: ''
     ModifiedBy: ''
       IsLocked: 1
        Summary: ''
    Description: ''
      Rationale: ''
       Keywords: {}
           Type: 'Functional'
            SID: 32
   FileRevision: 1
     ModifiedOn: 16-Sep-2021 09:59:25
          Dirty: 0
       Comments: [0×0 struct]
          Index: 'Import2'
```

Cleanup

Close the requirement sets and link sets without saving the changes. Close the project.

```
slreq.clear;
close(currentProject);
```

Specify Properties when Adding Requirements to a Requirement Set

This example shows how to specify properties when adding a requirement to a requirement set.

Load the requirement set `myReqSet`, which does not contain any requirements.

```
rs = slreq.load("myReqSet");
```

Use the `add` method to add a requirement to the requirement set. Set the requirement summary to `New Req` and set the requirement description to `My new requirement`.

```
req = add(rs,"Summary","New Req","Description","My new requirement")
```

```
req =
  Requirement with properties:

            Type: 'Functional'
              Id: '#2'
         Summary: 'New Req'
     Description: 'My new requirement'
        Keywords: {}
       Rationale: ''
       CreatedOn: 27-Feb-2022 04:28:48
       CreatedBy: 'batserve'
      ModifiedBy: 'batserve'
    IndexEnabled: 1
     IndexNumber: []
             SID: 2
    FileRevision: 1
      ModifiedOn: 27-Feb-2022 04:28:48
           Dirty: 1
        Comments: [0x0 struct]
           Index: '1'
```

Cleanup

Discard the requirement set without saving.

```
discard(rs);
```

Specify Properties when Adding Referenced Requirements to a Requirement Set

This example shows how to specify properties when adding a referenced requirement to a requirement set.

Open the “Requirements Definition for a Cruise Control Model” project and load the `crs_req` requirement set.

```
slreqCCProjectStart;
rs = slreq.load("crs_req");
```

Use the `add` method to add a referenced requirement to the requirement set as an Import node. Associate the Import node with the `crs_req.docx` file as the external requirements artifact. Set the requirement summary to `New Import Node` and set the requirement description to `My new Import node`.

```
ref = add(rs,"Artifact","crs_req.docx","Summary","New Import Node","Description","My new Import n
```

```
ref =
  Reference with properties:

              Id: ''
        CustomId: ''
```

```
       Artifact: 'crs_req.docx'
     ArtifactId: ''
         Domain: 'linktype_rmi_word'
      UpdatedOn: 16-Sep-2021 10:08:50
      CreatedOn: 16-Sep-2021 10:08:50
      CreatedBy: ''
     ModifiedBy: ''
       IsLocked: 1
        Summary: 'New Import Node'
    Description: 'My new Import node'
      Rationale: ''
       Keywords: {}
           Type: 'Functional'
            SID: 32
   FileRevision: 1
     ModifiedOn: 16-Sep-2021 10:08:50
          Dirty: 0
       Comments: [0×0 struct]
          Index: 'Import2'
```

Cleanup

Discard the requirement set without saving the changes. Clear the remaining requirement sets and link sets. Close the project.

```
discard(rs);
slreq.clear;
close(currentProject);
```

Tips

- To add a requirement as a child of another requirement, use `slreq.Requirement.add`. To add a referenced requirement as a child of another referenced requirement, use `slreq.Reference.add`. To add a justification as a child of another justification, use `slreq.Justification.add`.

See Also

`slreq.ReqSet` | `slreq.Reference` | `slreq.Requirement` | `slreq.Requirement.add` | `slreq.Reference.add` | `slreq.Justification.add`

Introduced in R2017b

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 and assign it to a variable.

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

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

`slreq.ReqSet` | `deleteAttribute` | `inspectAttribute` | `updateAttribute`

Topics

“Manage Custom Attributes for Requirements by Using the Requirements Toolbox 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 | justifyVerification | justifyImplementation | justifyVerification

Introduced in R2018b

children

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

Get top-level items in requirement set

Syntax

```
reqs = children(rs)
```

Description

`reqs = children(rs)` returns the top-level items in the requirement set `rs`. The items can be requirements, referenced requirements, or justifications.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

Output Arguments

`reqs` — Top-level items in requirement set

`slreq.Requirement` array | `slreq.Reference` array | `slreq.Justification` array

Top-level items in the requirement set, returned as an array of `slreq.Requirement`, `slreq.Reference`, or `slreq.Justification` array.

Examples

Get the Top-Level Items in a Requirement Set

This example shows how to get the top-level items in a requirement set.

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

```
slreqCCProjectStart;
rs = slreq.load("crs_req_func_spec");
```

Get the top-level items in the requirement set.

```
topItems = children(rs)
```

```
topItems=1×5 object
  1x5 heterogeneous BaseEditableItem (Requirement, Justification) array with properties:
```

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

Cleanup

Clear the open requirement sets and link sets. Close the current project.

```
slreq.clear;
close(currentProject);
```

Tips

- To get the child requirements of a requirement, use `slreq.Requirement.children`. To get the child referenced requirements of a referenced requirement, use `slreq.Reference.children`. To get the child justifications of a justification, use `slreq.Justification.children`.

See Also

`slreq.ReqSet` | `slreq.Reference` | `slreq.Requirement` | `slreq.Justification` | `slreq.Requirement.children` | `slreq.Reference.children` | `slreq.Justification.children`

Introduced in R2017b

close

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

Close a requirement set

Syntax

```
close(rs)
```

Description

`close(rs)` closes a requirement set.

Input Arguments

`rs` — Requirement set file

`slreq.ReqSet` object

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

Examples

Close a Requirement Set

```
% Create a new requirement set file
rs1 = slreq.new('C:\MATLAB\My_Requirements_Set_1.slreqx');

% Save the requirement set file
save(rs1);

% Close the requirement 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` — Requirement set file

`slreq.ReqSet` object

Requirement 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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'columns', '[1 8]', 'RichText', true`

`ReqSet` — Requirement Set
`slreq.ReqSet` object

The name of the existing requirement set that you import references to requirements into, specified as the comma-separated pair of `'ReqSet'` and a valid requirement 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 requirement 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 requirement 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 requirement 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.ReqSet` | `slreq.Reference` | `slreq.import`

Introduced in R2018a

discard

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

Close requirement set without saving

Syntax

```
discard(rs)
```

Description

`discard(rs)` closes the requirement set `rs` without saving.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

Examples

Discard Changes to a Requirement Set

This example shows how to discard changes to a requirement set without saving.

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

```
slreqCCProjectStart;
rs = slreq.load("crs_req_func_spec");
```

Set the description of the requirement set to `crs_req_func_spec`.

```
rs.Description = "crs_req_func_spec"
```

```
rs =
  ReqSet with properties:

             Description: 'crs_req_func_spec'
                    Name: 'crs_req_func_spec'
                Filename: 'C:\TEMP\Bdoc22a_1891349_13144\mlx_to_docbook1\bml.batserve.009704\MATL
                Revision: 66
                   Dirty: 1
    CustomAttributeNames: {}
              CreatedBy: 'itoy'
              CreatedOn: 27-Feb-2017 10:20:39
             ModifiedBy: 'Shashidhar'
             ModifiedOn: 13-Jul-2021 10:50:42
```

Discard the changes to the requirement set without saving.

```
discard(rs);
```

Cleanup

Clear any remaining loaded requirement sets and link sets. Close the current project.

```
slreq.clear;
close(currentProject);
```

Tips

- You can also use `close` to close a requirement set, which prompts you to save the requirement set before closing.
- You can use `save` to save the requirement set before discarding.
- You can use `slreq.clear` to close all requirement sets and link sets without saving and close the **Requirements Editor**.

See Also

`slreq.clear` | `close` | `save` | `slreq.ReqSet`

Introduced in R2017b

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

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

Topics

“Manage Custom Attributes for Requirements by Using the Requirements Toolbox API”

Introduced in R2020b

explore

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

Open requirement set in Requirements Editor

Syntax

```
explore(rs)
```

Description

`explore(rs)` opens the requirement set `rs` in the **Requirements Editor**. This function only works if the requirement set is loaded.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

Examples

Open a Requirement Set in the Requirements Editor

This example shows how to open a Requirement Set in the **Requirements Editor**.

Open the "Requirements Definition for a Cruise Control Model" project and load the `crs_req` requirement set.

```
slreqCCProjectStart;
rs = slreq.load('crs_req');
```

Open the requirement set in the **Requirements Editor**.

```
explore(rs)
```

Cleanup

Clear the loaded requirement set and close the project.

```
slreq.clear;
close(currentProject);
```

Tips

- You can also use `slreq.open` to open a Requirement Set in the **Requirements Editor**. This function loads the requirement set if it is not loaded.

See Also

`slreq.ReqSet` | `slreq.load` | `slreq.open`

Introduced in R2017b

exportToVersion

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

Export requirement set to previous MATLAB version

Syntax

```
tf = exportToVersion(rs,name,version)
```

Description

`tf = exportToVersion(rs,name,version)` saves a copy of the requirement set `rs` with the file name `name` that is compatible with the MATLAB version `version`. The function returns `1` if the file exports. The function saves the file in the current folder. If the requirement set has an associated link set, `exportToVersion` also exports the link set and saves it in the current folder.

Note You can export requirement sets only to version R2017b or later.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

`name` — File name for exported requirement set

string scalar | character vector

File name for the exported requirement set, specified as a string scalar or character vector.

`version` — MATLAB version to export to

string scalar | character vector

MATLAB version to export to, specified as a string scalar or character vector.

You can export to version R2017b or later.

Example: `tf = exportToVersion(rs,"newLinkSet","R2021a")`

Output Arguments

`tf` — Export success status

`0` | `1`

Export success status, returned as a logical `1` (true) or `0` (false).

Data Types: `logical`

Examples

Export a Requirement Set to a Previous Version of MATLAB

This example shows how to export a requirement set to a file that is compatible with a previous version of MATLAB.

Open the “Requirements Definition for a Cruise Control Model” project and load the `crs_req` requirement set.

```
slreqCCProjectStart;
rs = slreq.load("crs_req");
```

Export the requirement set to a new file that is compatible with MATLAB R2020a. Name the new file `crs_req_2020a`. The `exportToVersion` function also exports the link set associated with the requirement set using the same file name.

```
tf = exportToVersion(rs,"crs_req_2020a","R2020a")
```

```
tf = logical
   1
```

Cleanup

Clear the open requirement sets and link sets. Close the current project.

```
slreq.clear;
close(currentProject);
```

Tips

- You can export a link set to a previous version with `slreq.LinkSet.exportToVersion`.

See Also

`slreq.ReqSet` | `slreq.LinkSet.exportToVersion`

Topics

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

Introduced in R2018a

find

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

Find requirements in requirement 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 requirement set `rs` specified by the properties matching `PropertyName` and `PropertyValue`. Property name matching is case-insensitive.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

Requirement 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 requirement 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 requirement sets by constructing regular expression search patterns by using the tilde (~) symbol.

```
% Load a requirement 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 requirement 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

getPostLoadFcn

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

Get contents of `PostLoadFcn` callback

Syntax

```
callback = getPostLoadFcn(rs)
```

Description

`callback = getPostLoadFcn(rs)` returns the contents of the `PostLoadFcn` callback for the requirement set `rs`.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

Output Arguments

`callback` — Contents of `PostLoadFcn` callback

character vector

Contents of the `PostLoadFcn` callback script for the requirement set, returned as a character vector.

Examples

Get and Set `PostLoadFcn` Callback

This example shows how to get and set the `PostLoadFcn` callback for a requirement set.

Add the current folder to the path.

```
addpath(pwd)
```

Open a project that contains an algorithm to calculate the shortest path between two nodes on a graph. For more information, see “Verify a MATLAB Algorithm by Using Requirements-Based Tests”.

```
slreqShortestPathProjectStart;
```

Open the `shortest_path_tests_reqs` requirement set. The requirement set contains test requirements that describe the functional behavior that must be tested by a test case in order to verify the `shortest_path` algorithm in the project.

```
testReqs = slreq.open("shortest_path_tests_reqs");
```

Register the `postLoadTestReqs` script as the `PostLoadFcn` callback.

```
setPostLoadFcn(testReqs,"postLoadTestReqs");
```

Confirm that the `postLoadTestReqs` script is the `PostLoadFcn` callback for the `shortest_path_tests_reqs` requirement set.

```
callbackScript = getPostLoadFcn(testReqs)
```

```
callbackScript =
'postLoadTestReqs'
```

Save and close the `shortest_path_tests_reqs` requirement set, then re-open the requirement set. The `PostLoadFcn` callback executes.

```
save(testReqs);
slreq.clear;
testReqs = slreq.load("shortest_path_tests_reqs");
```

The `postLoadTestReqs` script opens the test file associated with the test requirements, `graph_unit_tests.m` and imports the **Requirements Editor** view settings from `myViewSettings.mat`.

```
type postLoadTestReqs.m
```

```
open("graph_unit_tests.m");
slreq.importViewSettings("myViewSettings.mat",1);
```

See Also

`slreq.ReqSet` | `setPostLoadFcn` | `setPreSaveFcn` | `getPreSaveFcn`

Topics

“Execute Code When Loading and Saving Requirement Sets”

Introduced in R2022a

getPreSaveFcn

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

Get contents of `PreSaveFcn` callback

Syntax

```
callback = getPreSaveFcn(rs)
```

Description

`callback = getPreSaveFcn(rs)` returns the contents of the `PreSaveFcn` callback for the requirement set `rs`.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

Output Arguments

`callback` — Contents of `PreSaveFcn` callback

character vector

Contents of the `PreSaveFcn` callback for the requirement set, returned as a character vector.

Examples

Get and Set `PreSaveFcn` Callback

This example shows how to get and set the `PreSaveFcn` callback for a requirement set.

Add the current folder to the path.

```
addpath(pwd)
```

Open a project that contains an algorithm to calculate the shortest path between two nodes on a graph. For more information, see “Verify a MATLAB Algorithm by Using Requirements-Based Tests”.

```
slreqShortestPathProjectStart;
```

Open the `shortest_path_tests_reqs` requirement set. The requirement set contains test requirements that describe the functional behavior that must be tested by a test case in order to verify the `shortest_path` algorithm in the project.

```
testReqs = slreq.open("shortest_path_tests_reqs");
```

Register the `preSaveTestReqs` script as the `PreSaveFcn` callback.

```
setPreSaveFcn(testReqs,"preSaveTestReqs");
```

Confirm that the `preSaveTestReqs` script is the `PreSaveFcn` callback for the `shortest_path_tests_reqs` requirement set.

```
callbackScript = getPreSaveFcn(testReqs)
```

```
callbackScript =
'preSaveTestReqs'
```

Save the `shortest_path_tests_reqs` requirement set to execute the callback.

```
save(testReqs);
```

The `preSaveTestReqs` script saves the current **Requirements Editor** view settings to a MAT-file called `myViewSettings.mat`.

```
type preSaveTestReqs.m
```

```
slreq.exportViewSettings("myViewSettings.mat");
```

See Also

`slreq.ReqSet` | `setPostLoadFcn` | `setPreSaveFcn` | `getPostLoadFcn`

Topics

“Execute Code When Loading and Saving Requirement Sets”

Introduced in R2022a

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 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 requirement 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` — Requirement set file

`slreq.ReqSet` object

Requirement 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'`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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.

`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);`

`attributeColumn` — Custom Attributes Column
`double` array

Column in the Microsoft Excel spreadsheet that you want to map as custom attributes of the requirements in your requirement set, specified as a `double` array.

Example: `'attributeColumn',[4 6]`

`attributes` — Attribute names
cell array

Attribute names for custom attribute columns, specified as a cell array of character vectors.

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 `attributeColumn` argument.

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

`bookmarks` — Option to import requirements using bookmarks
`0` (default) | `1`

Option to import requirements content using user-defined bookmarks, specified as a `1` or `0` of data type `logical`.

By default, Requirements Toolbox sets the value to `1` for Microsoft Word documents and `0` for Microsoft Excel spreadsheets.

Example: `'bookmarks',false`

`columns` — Range of columns
`double` array

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

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

`createdByColumn` — Created By Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `CreatedBy` property of the requirements in your requirement set, specified as a `double`.

Example: `'createdByColumn',5`

`descriptionColumn` — Description Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `Description` property of the requirements in your requirement set, specified as a `double`.

Example: `'descriptionColumn',2`

`idColumn` — ID Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `ID` property of the requirements in your requirement set, specified as a `double`.

Example: `'idColumn',1`

`keywords` — Attribute to map to `Keywords`
string scalar | character vector

Name of the attribute from the external document that you want to map to the `Keywords` property for the imported requirements.

Use this argument when you import from IBM Rational DOORS or custom document types.

Example: `"keywords","Requirement Keywords"`

`keywordsColumn` — Keywords Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `Keywords` property of the requirements in your requirement set, specified as a `double`.

Example: `'keywordsColumn',3`

`match` — Regular expression pattern
character vector

Regular expression pattern for ID search in Microsoft Office documents.

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

`modifiedByColumn` — Modified By Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `ModifiedBy` property of the requirements in your requirement set, specified as a `double`.

Example: `'modifiedByColumn',6`

`postImportFcn` — Custom post-import callback
string scalar | character vector

Custom post-import callback script name to use during import, specified as a string scalar or character vector.

The script that you assign to this callback executes after you import or update requirements.

Example: `"postImportFcn","myPostImportScript"`

`preImportFcn` — Custom pre-import callback
string scalar | character vector

Custom pre-import callback script name to use during import, specified as a string scalar or character vector.

The script that you assign to this callback executes before you import or update requirements.

Example: `"preImportFcn","myPreImportScript"`

`rationale` — Attribute to map to `Rationale`
string scalar | character vector

Name of the attribute from the external document that you want to map to the `Rationale` property for the imported requirements.

Use this argument when you import from IBM Rational DOORS or custom document types.

Example: `"rationale","Requirement Rationale"`

`rationaleColumn` — Rationale Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `Rationale` property of the requirements in your requirement set, specified as a `double`.

Example: `'rationaleColumn',5`

`ReqSet` — Requirement Set
character vector

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

If the requirement set exists, the requirements import under a new Import node. If the requirement set does not exist, Requirements Toolbox creates it.

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`

`rows` — Range of rows
`double` array

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

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

`sheet` — Worksheet name
character vector

Worksheet name from Microsoft Excel workbook, specified as a character vector.

Example: `'sheet','Sheet1'`

`summaryColumn` — Summary Column
`double`

Column in the Microsoft Excel spreadsheet that you want to map to the `Summary` property of the requirements in your requirement set, specified as a `double`.

Example: `'summaryColumn',4`

`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 requirement 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

`slreq.ReqSet` | `createReferences`

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

`slreq.ReqSet` | `addAttribute` | `deleteAttribute` | `updateAttribute`

Topics

“Manage Custom Attributes for Requirements by Using the Requirements Toolbox API”

Introduced in R2020b

runTests

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

Run test cases linked to the requirement set

Syntax

```
status = runTests(rs)
status = runTests(rs,Name,Value)
```

Description

`status = runTests(rs)` runs the tests for the requirement set, `rs` linked with the test cases.

`status = runTests(rs,Name,Value)` selects the instances specified by the name-value pairs `Name` and `Value` in the requirement set `rs`.

You can use `runTests` to run MATLAB unit tests, Simulink tests, and Simulink Design Verifier™ verifiables.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

Name-Value Pair Arguments

`Select` — Options for searching under masks

`'all'` (default) | `'failed'` | `'unexecuted'`

Select one or more criteria for the tests execution, specified as the comma-separated pair consisting of 'select' and one of these options:

- `"all"` — Select and run tests on all tests linked to the requirement set.
- `"failed"` — Select and run tests on failed tests linked to the requirement set.
- `"unexecuted"` — Select and run tests on unexecuted tests linked to the requirement set.

Example: `runTests(rs, 'Select', 'failed')`

Example: `runTests(rs, 'Select', {'unexecuted', 'failed'})`

Output Arguments

`status` — Requirement set verification status summary

structure

The verification status summary for the requirement set after the tests are run, 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

Run Tests on a Requirement Set

```
addpath(fullfile(matlabroot,'examples','slrequirements','main'));
reqSet = slreq.open('counter_req.slreqx');
rmi register linktype_mymljunitresults;
externalSource.id = 'testCounterStartsAtZero';
externalSource.artifact = 'counterTests.m';
externalSource.domain = 'linktype_mymljunitresults';
requirement = reqSet.find('Type', 'Requirement', 'SID', 2);
link = slreq.createLink(requirement, externalSource);
status = runTests(reqSet)
status =

  struct with fields:
```

```
         total: 3
        passed: 0
        failed: 0
    unexecuted: 1
     justified: 0
          none: 2
```

Verify a MATLAB Algorithm by Using Requirements-Based Tests

This example shows how to verify a MATLAB® algorithm by creating verification links from MATLAB code lines in functions and tests to requirements. This example uses a project that contains an algorithm to calculate the shortest path between two nodes on a graph.

Open the project.

```
slreqShortestPathProjectStart
```

Examine the Project Artifacts

The project contains:

- Requirement sets for functional and test requirements, located in the `requirements` folder
- A MATLAB algorithm, located in the `src` folder
- MATLAB unit tests, located in the `tests` folder
- Links from MATLAB code lines to requirements, stored `.slmx` files located in the `src` and `tests` folders
- Scripts to automate project analysis, located in the `scripts` folder

Open the Functional Requirement Set

The `shortest_path_func_reqs` requirement set captures the functional behavior that the `shortest_path` function requires. The requirements describe the nominal behavior and the expected behavior for invalid conditions, such as when the inputs to the function are not valid. Open the requirement set in the **Requirements Editor**.

```
funcReqs = slreq.open("shortest_path_func_reqs");
```

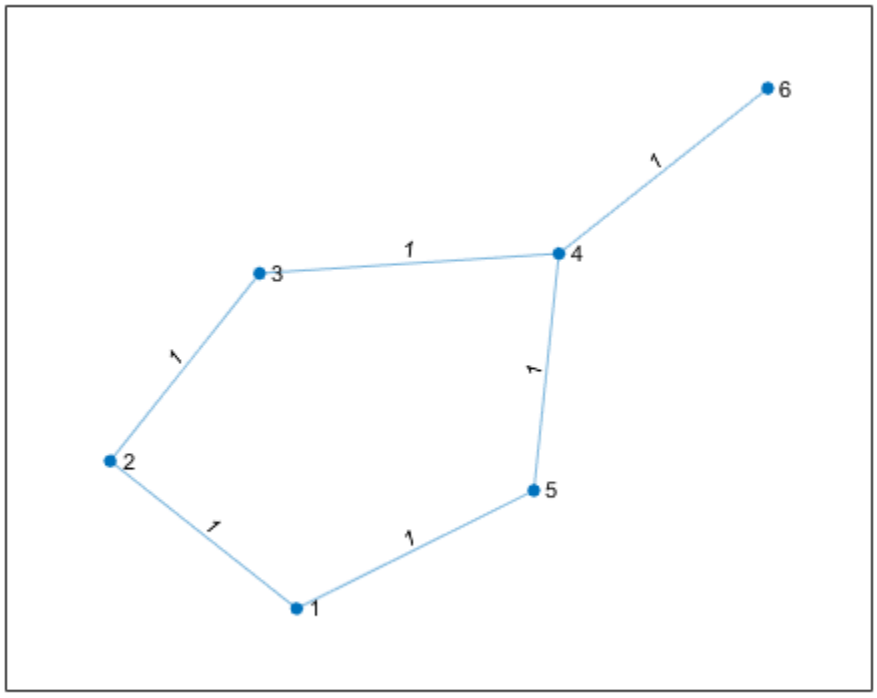
Use the Shortest Path Function

The `shortest_path` function tests the validity of the inputs to the function and then uses the Djikstra algorithm to calculate the number of edges in the shortest path between two nodes on a graph. The inputs to the function are an adjacency matrix that represents a graph, the starting node, and the ending node. For example, consider this adjacency matrix that represents a graph with six nodes.

```
A = [0 1 0 0 1 0;
    1 0 1 0 0 0;
    0 1 0 1 0 0;
    0 0 1 0 1 1;
    1 0 0 1 0 0;
    0 0 0 1 0 0];
```

Create a graph from the matrix and plot it.

```
G = graph(A);
plot(G,EdgeLabel=G.Edges.Weight)
```



Calculate the number of edges in the shortest path between nodes 1 and 6.

```
pathLength = shortest_path(A,1,6)
```

```
pathLength = 3
```

Open the Test Requirement Set

The `shortest_path_tests_reqs` requirement set contains test requirements that describe the functional behavior that must be tested by a test case. The test requirements are derived from the functional requirements. There are test requirements for the nominal behavior and for the invalid conditions. Open the requirement set in the **Requirements Editor**.

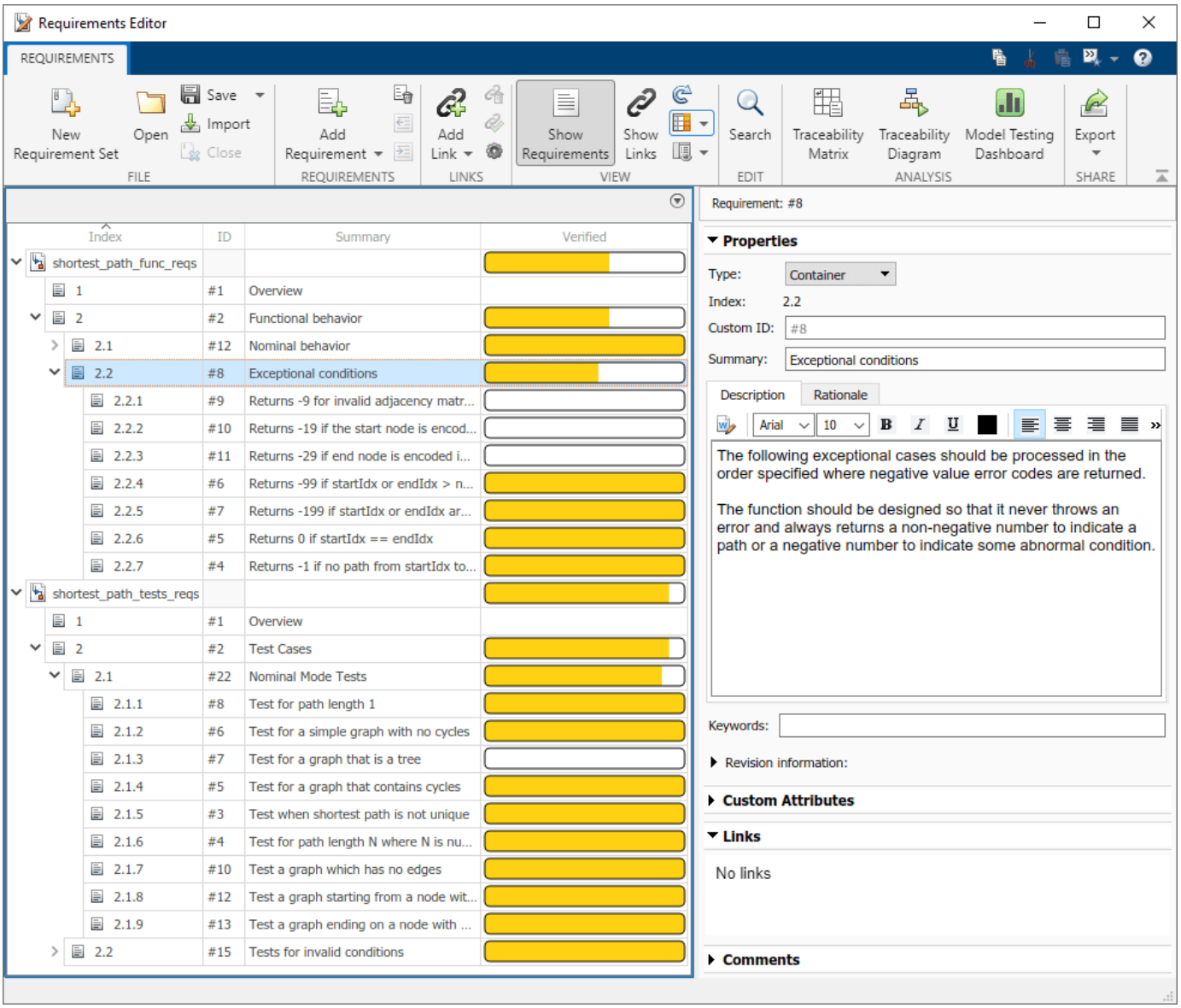
```
testReqs = slreq.open("shortest_path_tests_reqs");
```

The class-based MATLAB unit tests in `graph_unit_tests` implement the test cases described in `shortest_path_tests_reqs`. The class contains test methods based on the test requirements from `shortest_path_tests_reqs`. The class also contains the `verify_path_length` method, which the test cases use as a qualification method to verify that the expected and actual results are equal. The class also contains static methods that create adjacency matrices for the test cases.

View the Verification Status

To view the verification status, in the **Requirements Editor** toolstrip, in the **View** section, click **Columns** and select **Verification Status**. Three of the functional requirements and one test

requirement are missing verification links. The verification status is yellow for each requirement, which indicates that the linked tests have not run.



Run the tests and update the verification status for the requirement sets by using the `runTests` method.

```
status1 = runTests(funcReqs);
```

```
Running graph_unit_tests
.......... ..
Done graph_unit_tests
__________
```

```
status2 = runTests(testReqs);
```

```
Running graph_unit_tests
.......... ...
```

```
Done graph_unit_tests
__________
```

The verification status is green to indicate that the linked tests passed. However, some of the requirements do not have links to tests.

Identify Traceability Gaps in the Project

The functional and test requirements are linked to code lines in the `shortest_path` and `graph_unit_tests` files, but the traceability is not complete. Use a traceability matrix to identify requirements that are not linked to tests and to create links to make the requirements fully traceable.

Find the Missing Links with a Traceability Matrix

Create a traceability matrix for both requirement sets with the requirements on the top and the unit tests on the left. For more information about traceability matrices, see “Track Requirement Links with a Traceability Matrix”

```
mtxOpts = slreq.getTraceabilityMatrixOptions;
mtxOpts.topArtifacts = {'shortest_path_func_reqs.slreqx','shortest_path_tests_reqs.slreqx'};
mtxOpts.leftArtifacts = {'graph_unit_tests'};
slreq.generateTraceabilityMatrix(mtxOpts)
```

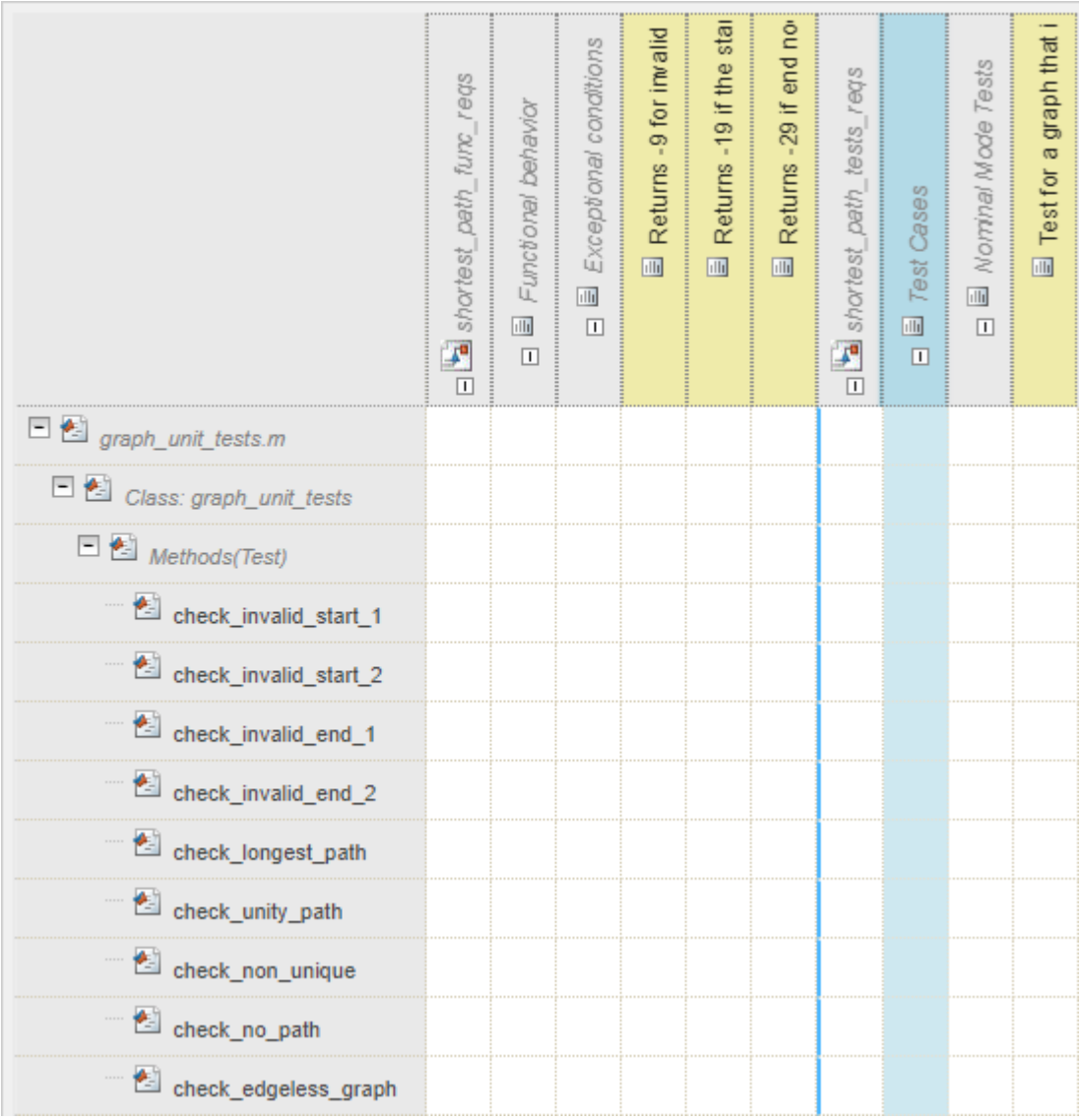
In the **Filter Panel**, in the **Top** section, filter the matrix to show only the functional requirements not linked to tests by clicking:

- **Top > Link > Missing Links**
- **Top > Type > Functional**

In the **Left** section, show only the test functions in the `graph_unit_tests` file by clicking:

- **Left > Type > Function**
- **Left > Attributes > Test**

Click **Highlight Missing Links** in the toolstrip.



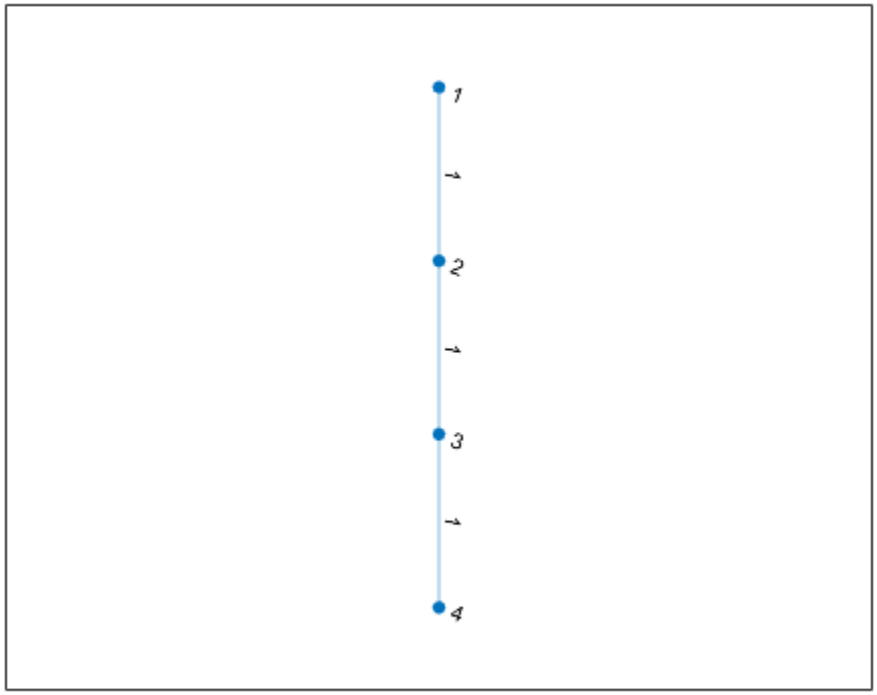
The Traceability Matrix window shows the three functional requirements and one test requirement that are missing verification links.

Create Verification Links for Requirements

The test requirement 2.1.3, `Test for a graph that is a tree`, is not linked to a test. A tree is a graph in which any two nodes are only connected by one path.

The test case `check_invalid_start_1` tests a tree graph by using the `graph_straight_seq` static method to create the adjacency matrix. Use the `graph_straight_seq` method to view the tree graph.

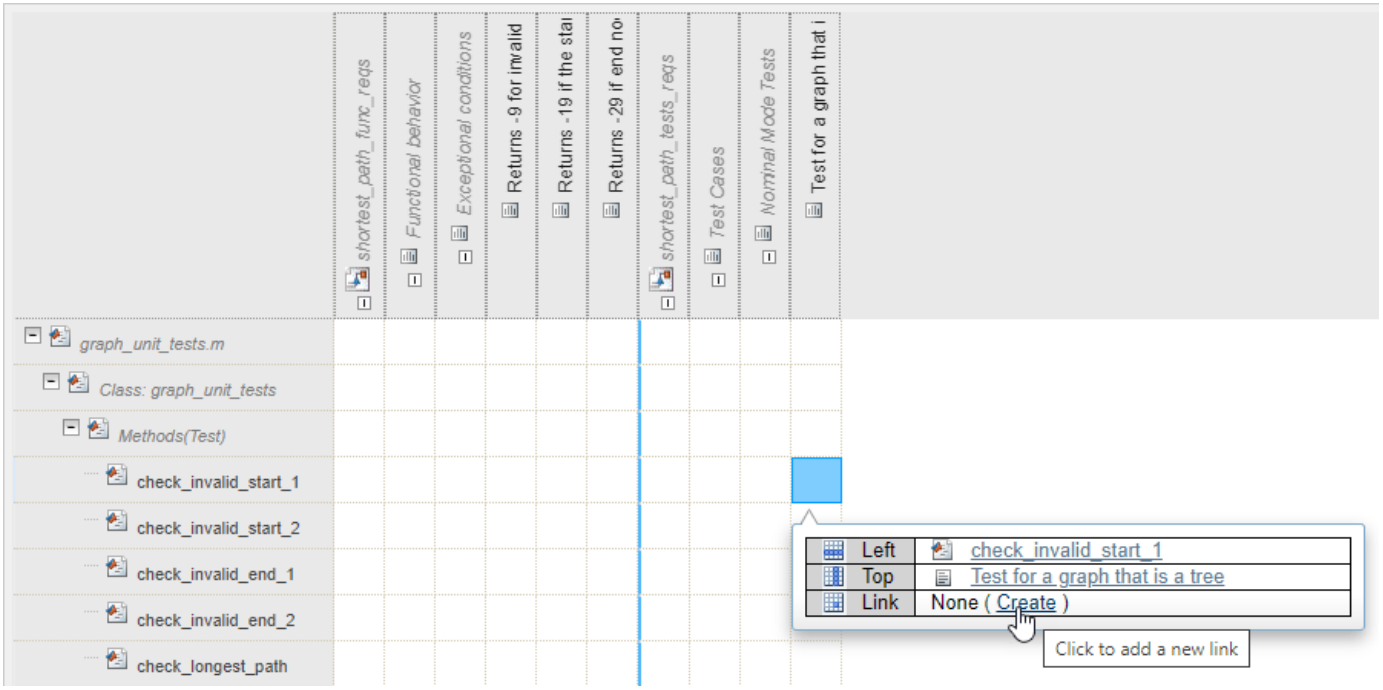
```
A = graph_unit_tests.graph_straight_seq;
G = graph(A);
plot(G,EdgeLabel=G.Edges.Weight)
```



Create a link from the `Test for a graph that is a tree` requirement to the `check_invalid_start_1` test case by using the traceability matrix you previously generated.

```
slreq.generateTraceabilityMatrix(mtxOpts)
```

Click the cell that corresponds to the requirement and the test and select **Create**. In the Create Link dialog box, click **Create**.



Update the verification status in the **Requirements Editor** by running the tests linked to the test requirements. The `check_invalid_start_1` test verifies the `Test for a graph that is a tree` requirement.

```
status3 = runTests(testReqs);
```

```
Running graph_unit_tests
.......... ...
Done graph_unit_tests
__________
```

Additionally, three functional requirements do not have links to tests:

- Requirement 2.2.1: `Returns -9 for invalid adjacency matrices`
- Requirement 2.2.2: `Returns -19 if the start node is encoded incorrectly`
- Requirement 2.2.3: `Returns -29 if end node is encoded incorrectly`

There is a traceability gap for these requirements. You cannot fill this gap by creating links to tests because there are no tests that verify these requirements.

Fix Coverage and Traceability Gaps by Authoring Tests

The three functional requirements that do not have links to tests do have links to lines of code in the `shortest_path` function. Run the tests with coverage to determine if those lines of code in the `shortest_path` function are covered by tests.

Run Tests with Coverage

Use the `RunTestsWithCoverage` script to run the tests with function and statement coverage and view the coverage in a report. For more information, see “Generate Code Coverage Report in HTML Format”.

```
RunTestsWithCoverage

Running graph_unit_tests
.......... .........
Done graph_unit_tests
__________

Code coverage report has been saved to:
 C:\Users\jdoe\MATLAB\Projects\examples\ShortestPath\coverageReport\index.html
```

Open the coverage report. The error code statements on lines 20, 25, and 30 are not covered by tests.

Hit Co...	Line Nun	C:\Users\jdoe\MATLAB\Projects\examples\ShortestPath\src\shortest_path.m
14	1	function pathLength = shortest_path(adjMatrix, startIdx, endIdx) %#codegen
	2	% SHORTEST_PATH - Finds length of shortest path between nodes in a graph
	3	%
	4	% OUT = SHORTEST_PATH(ADJMTX, STARTIDX, ENDIDX) Takes a graph represented by
	5	% its adjacency matrix ADJMTX along with two node STARTIDX, ENDIDX as
	6	% inputs and returns a integer containing the length of the shortest path
	7	% from STARTIDX to ENDIDX in the graph.
	8	
	9	% Copyright 2021 The MathWorks, Inc.
	10	
	11	
	12	%% Validy testing on the inputs
	13	% This code should never throw an error and instead should return
	14	% error codes for invlid inputs.
14	15	ErrorCode = 0;
14	16	pathLength = -1;
	17	
	18	% Check the validity of the adjacency matrix
14	19	if (~isAdjMatrixValid(adjMatrix))
0	20	ErrorCode = -9;
	21	end
	22	
	23	% Check the validity of the startIdx
14	24	if ~isNodeValid(startIdx)
0	25	ErrorCode = -19;
	26	end
	27	
	28	% Check the validity of the endIdx
14	29	if ~isNodeValid(endIdx)
0	30	ErrorCode = -29;
	31	end

Note that the coverage gap for these code lines and the traceability gap for requirements 2.2.1, 2.2.2, and 2.2.3 refer to the same error codes. You can close the coverage and traceability gaps simultaneously by authoring tests for these lines of code and creating links to the requirements.

Improve Coverage by Authoring New Tests

Create tests that improve the coverage for the tests and verify requirements 2.2.1, 2.2.2, and 2.2.2. Open the `graph_unit_tests` test file.

```
open("graph_unit_tests.m");
```

These functions test the three error codes. Copy and paste the code in line 4, in the test methods section of the `graph_unit_tests` file, then save the file.

```
function check_invalid_nonsquare(testCase)
    adjMatrix = zeros(2,3);
    startIdx = 1;
    endIdx = 1;
    expOut = -9;
    verify_path_length(testCase, adjMatrix, startIdx, endIdx, expOut, ...
        'Graph is not square');
end

function check_invalid_entry(testCase)
    adjMatrix = 2*ones(4,4);
    startIdx = 1;
    endIdx = 1;
    expOut = -9;
    verify_path_length(testCase, adjMatrix, startIdx, endIdx, expOut, ...
        'Adjacency matrix is not valid');
end

function check_invalid_noninteger_startnode(testCase)
    adjMatrix = zeros(4,4);
    startIdx = 1.2;
    endIdx = 1;
    expOut = -19;
    verify_path_length(testCase, adjMatrix, startIdx, endIdx, expOut, ...
        'Start node is not an integer');
end

function check_invalid_noninteger_endnode(testCase)
    adjMatrix = zeros(4,4);
    startIdx = 1;
    endIdx = 2.2;
    expOut = -29;
    verify_path_length(testCase, adjMatrix, startIdx, endIdx, expOut, ...
        'End node is not an integer');
end
```

Rerun the tests with coverage and open the coverage report.

```
RunTestsWithCoverage
```

```
Running graph_unit_tests
.......... ........
Done graph_unit_tests
__________

Code coverage report has been saved to:
 C:\Users\jdoe\MATLAB\Projects\examples\ShortestPath\coverageReport\index.html
```

The tests now cover the error code statements.

```
                18          % Check the validity of the adjacency matrix
 18             19          if (~isAdjMatrixValid(adjMatrix))
 2              20              ErrorCode = -9;
                21          end
                22
                23          % Check the validity of the startIdx
 18             24          if ~isNodeValid(startIdx)
 1              25              ErrorCode = -19;
                26          end
                27
                28          % Check the validity of the endIdx
 18             29          if ~isNodeValid(endIdx)
 1              30              ErrorCode = -29;
                31          end
```

However, there is a statement on line 97 that the tests do not cover. The conditions that require the tests to cover the statement on line 97 also cause the `return` on line 87 to execute, which means that the statement on 97 is not reachable and is dead logic.

```
                84          % Stop iterating when the current distance is maximum because
                85          % this indicates no remaining nodes are reachable
 23             86          if (min==max)
 4              87              return;
                88          end
                89
                90          % Mark the current node visited and check if this is end index
 19             91          visited(nodeIdx) = true;
 19             92          if nodeIdx == endIdx
 3              93              pathLength = distance(nodeIdx);
                94
 3              95              if (pathLength==realmax)
                96                  % No path exists so set distance to -1;
 0              97                  pathLength = -1;
                98              end
 3              99              return;
                100         end
```

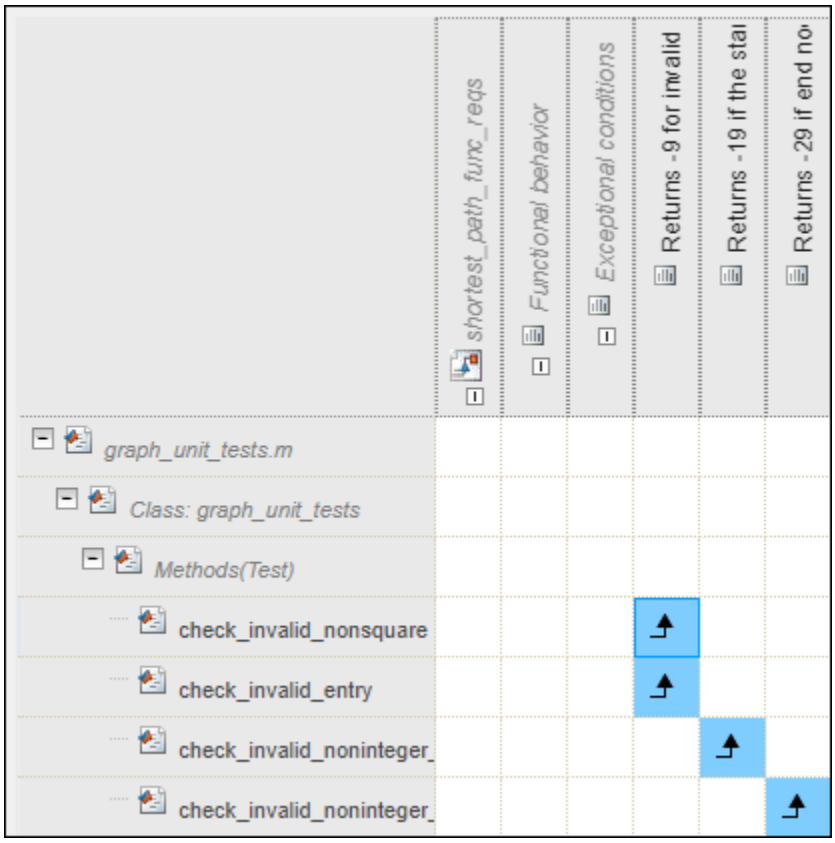
Fix Requirement Traceability Gaps

Regenerate the traceability matrix, apply the same filters from before, then click **Highlight Missing Links** in the toolstrip.

```
slreq.generateTraceabilityMatrix(mtxOpts)
```

- **Top > Link > Missing Links**
- **Top > Type > Functional**
- **Left > Type > Function**
- **Left > Attributes > Test**

Create links between the error code requirements and the new tests.



Update the verification status in the **Requirements Editor** by re-running the tests linked to both requirement sets.

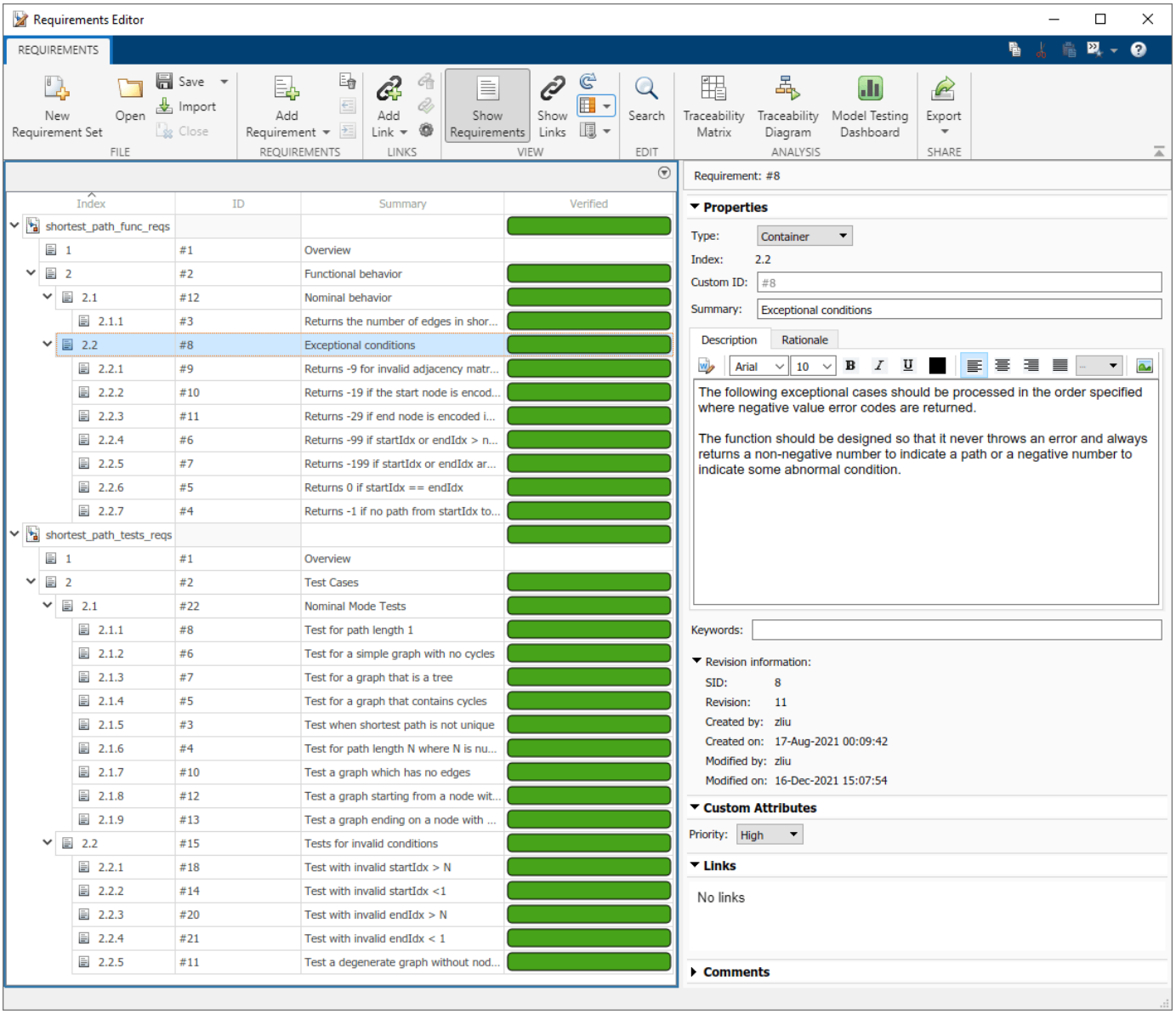
```
status4 = runTests(funcReqs);
```

```
Running graph_unit_tests
.......... ..
Done graph_unit_tests
__________
```

```
status5 = runTests(testReqs);
```

```
Running graph_unit_tests
.......... ...
Done graph_unit_tests
__________
```

All requirements have links to tests and all tests pass.



Generate and Verify Code from the Algorithm

You can generate code from the `shortest_path` algorithm by using MATLAB® Coder™. Use `coder.typeof` to define a variable-sized double array with a maximum size of 100x100, and a scalar double to use as inputs in the generated MEX function.

```
mtxType = coder.typeof(ones(100,100),[],1);
scalarDblType = coder.typeof(1);
```

Generate a MEX function from the `shortest_path` algorithm with the specified input types.

```
codegen shortest_path -args {mtxType, scalarDblType, scalarDblType}
```

```
Code generation successful.
```

Use `coder.runTest` to rerun the tests from the `graph_unit_tests` file by executing the MEX file instead of the `shortest_path` function.

```
coder.runTest("graph_unit_tests","shortest_path")
```

```
Running graph_unit_tests
.......... ........
Done graph_unit_tests
__________

  1×18 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   18 Passed, 0 Failed, 0 Incomplete.
   0.06382 seconds testing time.
```

Name	Passed	Failed	Incomplete
{'graph_unit_tests/check_invalid_nonsquare' }	true	false	false
{'graph_unit_tests/check_invalid_entry' }	true	false	false
{'graph_unit_tests/check_invalid_noninteger_startnode'}	true	false	false
{'graph_unit_tests/check_invalid_noninteger_endnode' }	true	false	false
{'graph_unit_tests/check_invalid_start_1' }	true	false	false
{'graph_unit_tests/check_invalid_start_2' }	true	false	false
{'graph_unit_tests/check_invalid_end_1' }	true	false	false
{'graph_unit_tests/check_invalid_end_2' }	true	false	false
{'graph_unit_tests/check_longest_path' }	true	false	false
{'graph_unit_tests/check_unity_path' }	true	false	false
{'graph_unit_tests/check_non_unique' }	true	false	false
{'graph_unit_tests/check_no_path' }	true	false	false
{'graph_unit_tests/check_edgeless_graph' }	true	false	false
{'graph_unit_tests/check_edgeless_start' }	true	false	false
{'graph_unit_tests/check_edgeless_end' }	true	false	false
{'graph_unit_tests/check_edgeless_graph_self_loop' }	true	false	false
{'graph_unit_tests/check_start_end_same' }	true	false	false
{'graph_unit_tests/check_invalid_idx_empty_adj' }	true	false	false

The tests pass when they execute the generated MEX function. The tests verify the generated code.

See Also

`getVerificationStatus` | “Requirements Traceability for MATLAB Code Lines”

Introduced in R2022a

save

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

Save a requirement set

Syntax

```
save(rs)
save(rs, filePath)
```

Description

`save(rs)` saves a requirement set by using its file name.

`save(rs, filePath)` saves a requirement set and updates its Name and Filename properties.

Input Arguments

`rs` — Requirement set file

`slreq.ReqSet` object

Requirement set file, specified as a `slreq.ReqSet` object.

`filePath` — 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'`

Examples

Save Requirement Set File

```
% Create the requirement set file
rs = slreq.new('C:\MATLAB\My Requirement Set.slreqx');

% Save the requirement set file
save(rs);

% Save the requirement set file as another requirement set file
save(rs, 'C:\MATLAB\Another Requirement Set.slreqx');
```

See Also

`slreq.ReqSet`

Introduced in R2018a

setPostLoadFcn

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

Assign `PostLoadFcn` callback script

Syntax

```
setPostLoadFcn(rs,callbackScript)
```

Description

`setPostLoadFcn(rs,callbackScript)` assigns the script specified by `callbackScript` as the `PostLoadFcn` callback script for the requirement set `rs`.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

`callbackScript` — Name of script to register

string scalar | character vector

Name of the script to register as the `PostLoadFcn` callback for the requirement set, specified as a string scalar or character vector.

Examples

Get and Set `PostLoadFcn` Callback

This example shows how to get and set the `PostLoadFcn` callback for a requirement set.

Add the current folder to the path.

```
addpath(pwd)
```

Open a project that contains an algorithm to calculate the shortest path between two nodes on a graph. For more information, see “Verify a MATLAB Algorithm by Using Requirements-Based Tests”.

```
slreqShortestPathProjectStart;
```

Open the `shortest_path_tests_reqs` requirement set. The requirement set contains test requirements that describe the functional behavior that must be tested by a test case in order to verify the `shortest_path` algorithm in the project.

```
testReqs = slreq.open("shortest_path_tests_reqs");
```

Register the `postLoadTestReqs` script as the `PostLoadFcn` callback.

```
setPostLoadFcn(testReqs,"postLoadTestReqs");
```

Confirm that the `postLoadTestReqs` script is the `PostLoadFcn` callback for the `shortest_path_tests_reqs` requirement set.

```
callbackScript = getPostLoadFcn(testReqs)
```

```
callbackScript =
'postLoadTestReqs'
```

Save and close the `shortest_path_tests_reqs` requirement set, then re-open the requirement set. The `PostLoadFcn` callback executes.

```
save(testReqs);
slreq.clear;
testReqs = slreq.load("shortest_path_tests_reqs");
```

The `postLoadTestReqs` script opens the test file associated with the test requirements, `graph_unit_tests.m` and imports the **Requirements Editor** view settings from `myViewSettings.mat`.

```
type postLoadTestReqs.m
```

```
open("graph_unit_tests.m");
slreq.importViewSettings("myViewSettings.mat",1);
```

See Also

`slreq.getCurrentObject` | `setPreSaveFcn` | `getPostLoadFcn` | `getPreSaveFcn`

Topics

“Execute Code When Loading and Saving Requirement Sets”

Introduced in R2022a

setPreSaveFcn

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

Assign `PreSaveFcn` callback script

Syntax

```
setPreSaveFcn(rs,callbackScript)
```

Description

`setPreSaveFcn(rs,callbackScript)` assigns the script specified by `callbackScript` as the `PreSaveFcn` callback script for the requirement set `rs`.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

`callbackScript` — Name of script to register

string scalar | character vector

Name of the script to register as the `PreSaveFcn` callback for the requirement set, specified as a string scalar or character vector.

Examples

Get and Set `PreSaveFcn` Callback

This example shows how to get and set the `PreSaveFcn` callback for a requirement set.

Add the current folder to the path.

```
addpath(pwd)
```

Open a project that contains an algorithm to calculate the shortest path between two nodes on a graph. For more information, see “Verify a MATLAB Algorithm by Using Requirements-Based Tests”.

```
slreqShortestPathProjectStart;
```

Open the `shortest_path_tests_reqs` requirement set. The requirement set contains test requirements that describe the functional behavior that must be tested by a test case in order to verify the `shortest_path` algorithm in the project.

```
testReqs = slreq.open("shortest_path_tests_reqs");
```

Register the `preSaveTestReqs` script as the `PreSaveFcn` callback.

```
setPreSaveFcn(testReqs,"preSaveTestReqs");
```

Confirm that the `preSaveTestReqs` script is the `PreSaveFcn` callback for the `shortest_path_tests_reqs` requirement set.

```
callbackScript = getPreSaveFcn(testReqs)
```

```
callbackScript =
'preSaveTestReqs'
```

Save the `shortest_path_tests_reqs` requirement set to execute the callback.

```
save(testReqs);
```

The `preSaveTestReqs` script saves the current **Requirements Editor** view settings to a MAT-file called `myViewSettings.mat`.

```
type preSaveTestReqs.m
```

```
slreq.exportViewSettings("myViewSettings.mat");
```

See Also

`slreq.getCurrentObject` | `setPostLoadFcn` | `getPostLoadFcn` | `getPreSaveFcn`

Topics

“Execute Code When Loading and Saving Requirement Sets”

Introduced in R2022a

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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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

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

Topics

“Manage Custom Attributes for Requirements by Using the Requirements Toolbox 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

updateReferences

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

Update referenced requirements in requirement set

Syntax

```
[status,changeList] = updateReferences(rs,docID)
[status,changeList] = updateReferences(rs,topRef)
```

Description

`[status,changeList] = updateReferences(rs,docID)` updates the referenced requirements in the requirement set `rs` by using the external requirements document specified by `docID`. The function returns the update status and a list of changes made to the requirements.

`[status,changeList] = updateReferences(rs,topRef)` updates the referenced requirements under the Import node `topRef` in the requirement set `rs`. The function updates the referenced requirements by using the external document associated with the Import node.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

`docID` — External requirements document identifier

string scalar | character vector

Identifier of the external requirements document associated with the requirement set, specified as a string scalar or a character vector.

Example: `"requirement_spec.docx"`

`topRef` — Import node

`slreq.Reference` object

Import node, specified as an `slreq.Reference` object.

Output Arguments

`status` — Update status

character vector

Requirement set update status, returned as a character vector.

`changeList` — List of updated referenced requirements

character vector

List of updated referenced requirements, returned as a character vector. The list includes the properties on page 2-63 of each referenced requirement changed by the function.

Examples

Update a Requirement Set from an External Requirements Document

This example shows how to update a requirement set from an external requirements document.

Open the “Requirements Definition for a Cruise Control Model” project. Load the `crs_req` requirement set.

```
slreqCCProjectStart;
rs = slreq.load("crs_req");
```

Update the requirement set from the external requirements document `crs_req.docx`.

```
[status,changeList] = updateReferences(rs,"crs_req.docx")
```

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

changeList =
    'Updated: ROM. Properties: description
     Updated: System Inputs. Properties: description
     Updated: System outputs. Properties: description
     Updated: Throttle value calculation. Properties: description
     '
```

Cleanup

Close the requirement sets and link sets without saving the changes. Close the project.

```
slreq.clear;
close(currentProject);
```

Update Referenced Requirements in a Requirement Set from an Import Node

This example shows how to update referenced requirements in a requirement set from an Import node.

Open the “Requirements Definition for a Cruise Control Model” project and load the `crs_req` requirement set.

```
slreqCCProjectStart;
rs = slreq.load("crs_req");
```

Find the Import node in the requirement set. The Import node has an `Index` property set to `Import1`.

```
topRef = find(rs,"Index","Import1");
```

Update the requirement set from the external requirements document associated with the Import node.

```
[status,changeList] = updateReferences(rs,topRef)

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

changeList =
    'Updated: ROM. Properties: description
     Updated: System Inputs. Properties: description
     Updated: System outputs. Properties: description
     Updated: Throttle value calculation. Properties: description
     '
```

Cleanup

Close the requirement sets and link sets without saving the changes. Close the project.

```
slreq.clear;
close(currentProject);
```

Tips

- You can use `updateFromDocument` to update the referenced requirements under an Import node without specifying the requirement set.

See Also

`slreq.ReqSet` | `updateFromDocument`

Introduced in R2017b

updateSrcArtifactUri

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

Update document resource identifier of imported requirements

Syntax

```
updateSrcArtifactUri(rs,oldURI,newURI)
```

Description

`updateSrcArtifactUri(rs,oldURI,newURI)` updates the Artifact property from `oldURI` to `newURI` for the referenced requirements in the requirement set `rs` that have Artifact set to `oldURI`. Use this function to update the external requirements document associated with the imported requirements from non-file-based domains, such as a query URL.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

`oldURI` — Resource identifier for original external document

string scalar | character vector

Resource identifier for the original external document, specified as a string scalar or character vector.

`newURI` — Resource identifier for new external document

string scalar | character vector

Resource identifier for the new external document, specified as a string scalar or character vector.

Examples

Update Source Artifact Resource Identifier

This example shows how to update the stored query for requirements that were previously imported to a requirement set called `myReqSet`.

Get a handle to the requirement set called `myReqSet`.

```
rs = slreq.find(Type="ReqSet",Name="myReqSet");
```

Get a handle to the import node for the requirement set.

```
topRef = children(rs);
```

Update the query stored in the Artifact property of the referenced requirements in the requirement set.

```
oldURI = topRef.Artifact;
newURI = "rm:ofType=%3A9443%2Frm2%2Ftypes%2F_C1KXMwJgEeuFW5Ss3RBk7w%3E";
updateSrcArtifactUri(rs,oldURI,newURI);
```

Tips

- If you rename or move an external requirements document file, use `updateSrcFileLocation` to update the file name or path of the referenced requirements in the requirement set.
- If you rename or move an external requirements document, you can update the link destinations for direct links by using `updateDocUri`.

See Also

`slreq.ReqSet` | `updateDocUri` | `updateSrcFileLocation`

Introduced in R2022a

updateSrcFileLocation

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

Update document location of imported requirements

Syntax

```
updateSrcFileLocation(rs,oldID,newID)
```

Description

`updateSrcFileLocation(rs,oldID,newID)` updates the Artifact property from `oldID` to `newID` for the referenced requirements in the requirement set `rs` that have Artifact set to `oldID`. Use this function to update the external requirements document associated with imported requirements.

Input Arguments

`rs` — Requirement set

`slreq.ReqSet` object

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

`oldID` — Resource identifier for original external document

string scalar | character vector

Resource identifier for the original external document, specified as a string scalar or character vector.

`newID` — Resource identifier for new external document

string scalar | character vector

Resource identifier for the new external document, specified as a string scalar or character vector.

Examples

Update Source File Location for Referenced Requirements in an Imported Requirement Set

This example shows how to update the source file location for referenced requirements in an imported requirement set.

Open the “Requirements Definition for a Cruise Control Model” project and load the `crs_req` requirement set.

```
slreqCCProjectStart;
rs = slreq.load("crs_req");
```

Copy the `crs_req.docx` document and name it `crs_req_v2.docx`. Save the new file in the same folder.

```
oldPath = fullfile(pwd,"documents","crs_req.docx");
newPath = fullfile(pwd,"documents","crs_req_v2.docx");
copyfile(oldPath,newPath);
```

Update the referenced requirements in the requirement set `crs_req` that point to `crs_req.docx` as the source file to point to `crs_req_v2.docx`.

```
updateSrcFileLocation(rs,"crs_req.docx","crs_req_v2.docx")
```

To confirm that the source file updated, get a handle to the Import node for the requirement set and check the `Artifact` property.

```
topRef = children(rs);
srcFile = topRef.Artifact
```

```
srcFile =
'crs_req_v2.docx'
```

Cleanup

Clear the requirement set and close the project.

```
slreq.clear;
close(currentProject);
```

Tips

- If you rename or move an external requirements document, you can update the link destinations for direct links by using `updateDocUri`.
- To update the external requirements document resource identifier for referenced requirements imported from non-file-based domains, use `updateSrcArtifactUri`.

See Also

`slreq.ReqSet` | `updateDocUri`

Topics

“Use Command-Line API to Update or Repair Requirements Links”

Introduced in R2017b

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 child requirement

Syntax

```
reqChild = add(req)
reqChild = add(req,PropertyName,
PropertyValue,...,PropertyNameN,PropertyValueN)
```

Description

`reqChild = add(req)` adds a child requirement to the requirement `req` and returns a handle to the child requirement.

`reqChild = add(req,PropertyName,
PropertyValue,...,PropertyNameN,PropertyValueN)` adds a child requirement with the properties and property values specified by `PropertyName` and `PropertyValue`.

Input Arguments

`req` — Requirement

`slreq.Requirement` object

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

`PropertyName` — Requirement property name

string scalar | character vector

Requirement property name, specified as a string scalar or a character vector.

You can only enter an `slreq.Requirement` property on page 2-74 where the `SetAccess` attribute is `public`.

Example: `"Summary"`

`PropertyValue` — Requirement property value

string scalar | character vector

Requirement property value, specified as an string scalar or a character vector.

Output Arguments

`reqChild` — Child requirement

`slreq.Requirement` object

New child requirement, returned as an `slreq.Requirement` object.

Examples

Add a Child Requirement Under a Requirement

This example shows how to add a child requirement under a requirement.

Load the requirement set `myReqSet`, which does not contain any requirements.

```
rs = slreq.load("myReqSet");
```

Use the `add` method to add a top-level requirement to the requirement set.

```
req = add(rs);
```

Use the `add` method to add a child requirement under the requirement.

```
newReq = add(req)
```

```
newReq =
  Requirement with properties:

            Type: 'Functional'
              Id: '#3'
         Summary: ''
     Description: ''
        Keywords: {}
       Rationale: ''
       CreatedOn: 27-Feb-2022 04:27:45
       CreatedBy: 'batserve'
      ModifiedBy: 'batserve'
    IndexEnabled: 1
     IndexNumber: []
             SID: 3
    FileRevision: 1
      ModifiedOn: 27-Feb-2022 04:27:45
           Dirty: 1
        Comments: [0x0 struct]
           Index: '1.1'
```

Get the value of the `Index` property for the new requirement.

```
idx = newReq.Index
```

```
idx =
'1.1'
```

The value indicates that the new requirement is a child requirement of the original requirement.

Cleanup

Discard the requirement set without saving.

```
discard(rs);
```

Tips

- To add a top-level requirement to a requirement set, use `slreq.ReqSet.add`. To add a referenced requirement as a child of another referenced requirement, use `slreq.Reference.add`. To add a justification as a child of another justification, use `slreq.Justification.add`.

See Also

`slreq.Requirement` | `slreq.ReqSet.add` | `slreq.Reference.add` | `slreq.Justification.add` | `remove`

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 requirement 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
```

Tips

- To get the top-level items in a requirement set, use `slreq.ReqSet.children`. To get the child referenced requirements of a referenced requirement, use `slreq.Reference.children`. To get the child justifications of a justification, use `slreq.Justification.children`.

See Also

`slreq.Requirement` | `slreq.ReqSet` | `slreq.ReqSet.children` | `slreq.Reference.children` | `slreq.Justification.children` | `parent`

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 executed.

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);
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);
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);
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

`slreq.Requirement` | `move` | `moveDown` | `moveUp`

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 requirement 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

`slreq.Requirement` | `slreq.ReqSet` | `promote`

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
    IndexEnabled
    IndexNumber
    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
    IndexEnabled
    IndexNumber
    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.Requirement` | `slreq.ReqSet` | `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 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', 'R1.1');

% Get the Priority (custom attribute) of req1
summaryReq1 = getAttribute(req1, 'Priority')

summaryReq1 =

  'High'
```

See Also

`slreq.Requirement` | `setAttribute`

Topics

“Manage Custom Attributes for Requirements by Using the Requirements Toolbox 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`.

`unexecuted` — Unexecuted requirements
`double`

The total number of requirements (including child requirements) with unexecuted associated tests, returned as a `double`.

`justified` — Justified requirements
`double`

The total number of requirements (including child requirements) that are justified for verification in the requirement set, returned as a `double`.

`none` — Unlinked requirements
`double`

The total number of requirements (including child requirements) without links to verification objects, returned as a `double`.

Examples

Get Verification Status Summary of a Requirement

```
% Get the verification status summary of the requirement req
% and all its child requirements
reqVerifStatus = getVerificationStatus(req)

reqVerifStatus =

  struct with fields:

          total: 34
         passed: 14
         failed: 15
     unexecuted: 4
      justified: 1
           none: 0


% Get the verification status summary of only the requirement myReq
myReqVerifStatus = getVerificationStatus(myReq, 'self')

myReqVerifStatus =

  struct with fields:

        passed: 0
        failed: 1
    unexecuted: 0
     justified: 0
          none: 0
```

See Also

`updateVerificationStatus`

Introduced in R2018b

isJustifiedFor

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

Check if requirement is justified

Syntax

```
tf = isJustifiedFor(req, linkType)
```

Description

`tf = isJustifiedFor(req, linkType)` checks if the requirement `req` is justified for the link type specified by `linkType`.

Input Arguments

`req` — Requirement instance

`slreq.Requirement` object

Requirement to check for justification, specified as an `slreq.Requirement` object.

`linkType` — Justification link type

`'Implement'` | `'Verify'`

Justification link type, specified as a character vector.

Output Arguments

`tf` — Justification status

`0` | `1`

The justification status of the requirement, returned as a Boolean.

Examples

Check if Requirements Are Justified

```
% Check if requirement req1 is justified for Implementation
req1_Status = isJustifiedFor(req1, 'Implement')

req1_Status =

  logical

   1

% Check if requirement req2 is justified for Verification
req2_Status = isJustifiedFor(req2, 'Verify')
```

```
req2_Status =

  logical

   0
```

See Also

`getImplementationStatus` | `getVerificationStatus`

Introduced in R2018b

justifyImplementation

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

Justify requirements for implementation

Syntax

```
implementationJustLink = justifyImplementation(req, jt)
```

Description

`implementationJustLink = justifyImplementation(req, jt)` justifies the requirement `req` for implementation by creating a link `implementationJustLink` from the justification `jt` to `req`.

Input Arguments

`req` — Requirement instance

`slreq.Requirement` object

Requirement to justify for implementation, specified as an `slreq.Requirement` object.

`jt` — Justification object

`slreq.Justification` object

Justification object to justify `req` for implementation, specified as an `slreq.Justification` object.

Output Arguments

`implementationJustLink` — Justification link

`slreq.Link` object

Link to justification object `jt` of type **Implement**, returned as an `slreq.Link` object.

Examples

```
% Justify requirement myReq for implementation by using a justification object myJust

myImplJustification = justifyImplementation(myReq, myJust)

myImplJustification =

  Link with properties:

          Type: 'Implement'
    Description: 'Cruise Control Mode (crs_req_func_spec#1)'
       Keywords: [0×0 char]
      Rationale: ''
      CreatedOn: 13-Jan-2017 13:45:12
      CreatedBy: 'John Doe'
```

```
    ModifiedOn: 24-Oct-2018 12:25:30
    ModifiedBy: 'Jane Doe'
      Revision: 6
      Comments: [0×0 struct]
```

See Also

`getImplementationStatus` | `addJustification`

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 executed without error.

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

`slreq.Requirement` | `copy` | `moveDown` | `moveUp`

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 executed without error. 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

`slreq.Requirement` | `copy` | `move` | `moveUp`

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 executed without error. 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` — Move success status
`0 | 1`

Move 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

`slreq.Requirement` | `copy` | `move` | `moveDown`

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 requirement 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

`slreq.Requirement` | `slreq.ReqSet` | `children`

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 requirement 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

`slreq.Requirement` | `slreq.ReqSet` | `demote`

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
    IndexEnabled
    IndexNumber
    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
    IndexEnabled
    IndexNumber
    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

`slreq.Requirement` | `add` | `slreq.find`

Introduced in R2018a

reqSet

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

Return parent requirement set

Syntax

```
rsout = reqSet(req)
```

Description

`rsout = reqSet(req)` returns the parent requirement 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 requirement set

`slreq.ReqSet` object

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

Examples

Query Requirement Set Information

```
% Load a new requirement 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 requirement 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

`slreq.Requirement` | `slreq.ReqSet` | `parent`

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

`slreq.Requirement` | `slreq.ReqSet` | `getAttribute`

Topics

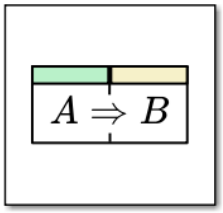
“Manage Custom Attributes for Requirements by Using the Requirements Toolbox API”

Introduced in R2018a

Blocks

Requirements Table

Formally model requirements with input conditions
Library: Requirements Toolbox



Description

The Requirements Table block models formal requirements. The block starts with evaluating conditions listed in the **Preconditions** tab. If the conditions are satisfied, you can check if other simulation data meet specified conditions in the **Postconditions** column, or execute desired actions, such as block outputs or functions, in the **Action** column. For more information, see “Use a Requirements Table Block to Create Formal Requirements”.

You can also constrain requirements based on physical limitations of your model by defining assumptions in the **Assumptions** tab. See “Add Assumptions to Requirements”.

You can configure this block only if you have Requirements Toolbox.

Ports

Input

`Port_1` — Input port

scalar | vector | matrix

Input port, specified as a scalar, vector, or matrix. Each input data that you define has a corresponding input port.

Dependencies

To create input ports, open the block and create input data in the **Symbols** pane. See “Define Data in Requirements Table Blocks”.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point` | `enumerated` | `bus`

Output

`Port_1` — Output port

scalar | vector | matrix

Output port, specified as a scalar, vector, or matrix. Each output data that you define has a corresponding output port.

Dependencies

To create output ports, open the block and create output data in the **Symbols** pane. See “Define Data in Requirements Table Blocks”.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point` | `enumerated` | `bus`

Parameters

Main

`Show port labels` — Display options for port labels

`FromPortIcon` (default) | `none` | `FromPortBlockName` | `SignalName`

Select how to display port labels on the Requirements Table block icon.

- `none` – Do not display port labels.
- `FromPortIcon` – Display the name of the input and output data.
- `FromPortBlockName` – Display the name of the input and output data.
- `SignalName` – If the signal connected to the port is named, display the signal name. Otherwise, display the name of the data.

Programmatic Use

Parameter: `ShowPortLabels`
Type: string scalar or character vector
Value: `"none"` | `"FromPortIcon"` | `"FromPortBlockName"` | `"SignalName"`
Default: `"FromPortIcon"`

`Read/Write permissions` — Levels of access to contents of block

`ReadWrite` (default) | `ReadOnly` | `NoReadOrWrite`

Control user access to the contents of the Requirements Table block.

- `ReadWrite` – Enable opening and modifying of Requirements Table block contents.
- `ReadOnly` – Enable opening of the Requirements Table block.
- `NoReadOrWrite` – Disable opening or modifying of the Requirements Table block.

Note When you attempt to view the contents of a Requirements Table block whose **Read/Write permissions** parameter is `NoReadOrWrite`, the block does not respond. For example, when you double-click the Requirements Table block, Simulink does not open the table contents and does not display messages.

Programmatic Use

Parameter: `Permissions`
Type: string scalar or character vector
Value: `"ReadWrite"` | `"ReadOnly"` | `"NoReadOrWrite"`
Default: `"ReadWrite"`

`Minimize algebraic loop occurrences` — Option to eliminate artificial algebraic loops

off (default) | on

Try to eliminate artificial algebraic loops that include the atomic unit during simulation.

- ☐ **off** – Do not try to eliminate artificial algebraic loops that include the atomic unit.
- ☑ **on** – Try to eliminate artificial algebraic loops that include the atomic unit.

Programmatic Use
Parameter: `MinAlgLoopOccurrences`
Type: string scalar or character vector
Value: `"off"` | `"on"`
Default: `"off"`

`Schedule as` — Type of scheduling

`Sample Time` (default) | `Periodic Partition` | `Aperiodic Partition`

Specify how to schedule the Requirements Table block.

- `Sample time` – Specify whether entries in the Requirements Table block must run at the same rate or can run at different rates.
- `Periodic Partition` – Schedule the Requirements Table block entries with a periodic partition. Specify a partition name and a sample time corresponding to the rate at which the partition runs.
- `Aperiodic Partition` – Schedule the Requirements Table block entries with an aperiodic partition. Specify a partition name.

Programmatic Use
Parameter: `ScheduleAs`
Type: string scalar or character vector
Value: `"SampleTime"` | `"DiscretePartition"` | `"UnconstrainedPartition"`
Default: `"SampleTime"`

`Sample time (-1 for inherited)` — Specify time interval

`-1` (default) | `[Ts 0]`

Specify whether entries in this block must run at the same rate or can run at different rates.

- If entries in the Requirements Table block can run at different rates, specify the sample time as inherited (`-1`).
- If entries must run at the same rate, specify the sample time, `Ts`, corresponding to this rate.

Programmatic Use
Parameter: `SystemSampleTime`
Type: string scalar or character vector
Value: `"-1"` | `"[Ts 0]"`
Default: `"-1"`

Code Generation

To enable these parameters, you must have Simulink Coder™ or Embedded Coder®.

`Function packaging` — Code format

`Auto` (default) | `Inline` | `Nonreusable function` | `Reusable function`

Select the code format the block uses to generate code for an atomic (nonvirtual) unit.

- `Auto` – Simulink Coder and Embedded Coder choose the optimal code format based on the type and number of instances of the Requirements Table block in the model.

- `Inline` – Simulink Coder and Embedded Coder inline the Requirements Table block unconditionally.
- `Nonreusable function` – Simulink Coder explicitly generates a separate function in a separate file.
- `Reusable function` – Simulink Coder and Embedded Coder generate a function with arguments that allows reuse of block code when a model includes multiple instances of the block.

  This option also generates a function with arguments that allows the Requirements Table block to be reused in the generated code of a model reference hierarchy that includes multiple instances of a Requirements Table block across referenced models. In this case, the block must be in a library.

Tips

- When you want to represent multiple instances of a Requirements Table block as one reusable function, you can designate each of the instances as `Auto` or as `Reusable function`. It is best to use one or the other, as using both creates two reusable functions, one for each designation. The outcomes of these choices differ only when reuse is not possible. Selecting `Auto` does not allow control of the function or file name for the Requirements Table block code.
- The `Reusable function` and `Auto` options both try to determine if multiple instances of a Requirements Table block exist and if the code can be reused. The options differ only when reuse is not possible:
  - `Auto` yields inlined code, or if circumstances prohibit inlining, the setting separates functions for each Requirements Table block instance.
  - `Reusable function` yields a separate function with arguments for each Requirements Table block instance in the model.
- If you select `Reusable function` while your generated code is under source control, set **File name options** to `Use subsystem name`, `Use function name`, or `User specified`. Otherwise, the names of your code files change when you modify your model, which prevents source control on your files.

Programmatic Use

Parameter: `RTWSystemCode`
Type: string scalar or character vector
Value: `"Auto"` | `"Inline"` | `"Nonreusable function"` | `"Reusable function"`
Default: `"Auto"`

`Function name options` — How to name generated function

`Auto` (default) | `Use subsystem name` | `User specified`

Select how Simulink Coder names the function it generates for the block.

If you have Embedded Coder, you can control function names with options on the Configuration Parameter **Code Generation** > **Identifiers** pane.

- `Auto` – Assign a unique function name using the default naming convention, *`model_block`*`()`, where *`model`* is the name of the model and *`block`* is the name of the block (or that of an identical one when code is being reused).
- `Use subsystem name` – Use the Requirements Table block name as the function name. By default, the function name uses the naming convention *`model_block`*.

Note When a Requirements Table block is in a library block and the **Function packaging** parameter is set to `Reusable function`, if you set the `Use subsystem name` option, the code generator uses the name of the library block for the function name and file name.

- `User specified` – Enable the **Function name** field. Enter a legal C or C++ function name, which must be unique.

For more information, see “Generate Subsystem Code as Separate Function and Files” (Simulink Coder).

Dependencies

To enable this parameter, set **Function packaging** to `Nonreusable function` or `Reusable function`.

Programmatic Use

Parameter: `RTWFcnNameOpts`
Type: string scalar or character vector
Value: `"Auto"` | `"Use subsystem name"` | `"User specified"`
Default: `"Auto"`

`Function name` — Name of function for block code

`""` (default) | function name

Name of the function for the block code.

Use this parameter if you want to give the function a specific name instead of using an autogenerated name or the block name. For more information, see “Generate Subsystem Code as Separate Function and Files” (Simulink Coder).

Dependencies

To enable this parameter, set the **Function name options** parameter to `User specified`.

Programmatic Use

Parameter: `RTWFcnName`
Type: string scalar or character vector
Value: `""` | `"<function name>"`
Default: `""`

`File name options` — How to name generated file

`Auto` (default) | `Use subsystem name` | `Use function name` | `User specified`

How Simulink Coder names the separate file for the function it generates for the block.

- `Auto` – Depending on the configuration of the block and how many instances are in the model, `Auto` yields different results:
  - If the code generator does *not* generate a separate file for the block, the block code is generated within the code module generated from the block parent system. If the block parent is the model itself, the block code is generated within *`model`*`.c` or *`model`*`.cpp`.
  - If you select `Reusable function` for the **Function packaging** parameter and your generated code is under source control, consider specifying a **File name options** value other than `Auto`. This prevents the generated file name from changing due to unrelated model modifications, which is problematic for using source control to manage configurations.

  - If you select `Reusable function` for the **Function packaging** parameter and there are multiple instances of the block in a model reference hierarchy, in order to generate reusable code for the block, **File name options** must be set to `Auto`.
- `Use subsystem name` – The code generator generates a separate file, using the block name as the file name.

  Note When **File name options** is set to `Use subsystem name`, the block file changes if the model contains Model blocks, or if a model reference target is being generated for the model. In these situations, the file name for the Requirements Table block consists of the block name prefixed by the model name.
- `Use function name` – The code generator uses the function name specified by **Function name options** as the file name.
- `User specified` – This option enables the **File name (no extension)** text entry field. The code generator uses the name you enter as the file name. Enter a file name, but do not include the `.c` or `.cpp` (or another) extension. This file name need not be unique.

  Note While a Requirements Table block source file name need not be unique, you must avoid giving nonunique names that result in cyclic dependencies (for example, `sys_a.h` includes `sys_b.h`, `sys_b.h` includes `sys_c.h`, and `sys_c.h` includes `sys_a.h`).

Dependencies

To enable this parameter, set **Function packaging** to `Nonreusable function` or `Reusable function`.

Programmatic Use

Parameter: `RTWFileNameOpts`
Type: string scalar or character vector
Value: `"Auto"` | `"Use subsystem name"` | `"Use function name"` | `"User specified"`
Default: `"Auto"`

`File name (no extension)` — Name of generated file

`""` (default) | file name

Name of the generated file. The file name that you specify does not have to be unique. However, avoid giving non-unique names that result in cyclic dependencies (for example, `sys_a.h` includes `sys_b.h`, `sys_b.h` includes `sys_c.h`, and `sys_c.h` includes `sys_a.h`).

For more information, see “Generate Subsystem Code as Separate Function and Files” (Simulink Coder).

Dependencies

To enable this parameter, set **File name options** to `User specified`.

Programmatic Use

Parameter: `RTWFileName`
Type: string scalar or character vector
Value: `""` | `"<file name>"`
Default: `""`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

Actual data type or capability support depends on block implementation.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Actual data type or capability support depends on block implementation.

HDL Code Generation
Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

Actual data type or capability support depends on block implementation.

Fixed-Point Conversion
Design and simulate fixed-point systems using Fixed-Point Designer™.

Actual data type or capability support depends on block implementation.

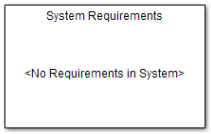
See Also
`RequirementsTable`

Topics
“Specify Requirements Table Block Properties”
“Define Data in Requirements Table Blocks”
“Set Data Types in Requirements Table Blocks”

Introduced in R2022a

System Requirements

List system requirements in Simulink models
Library: Requirements Toolbox



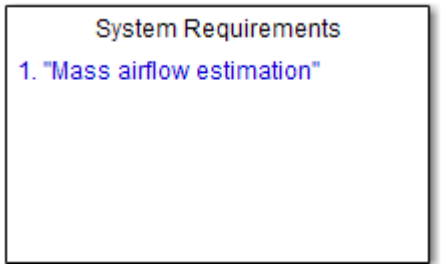
Description

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

5

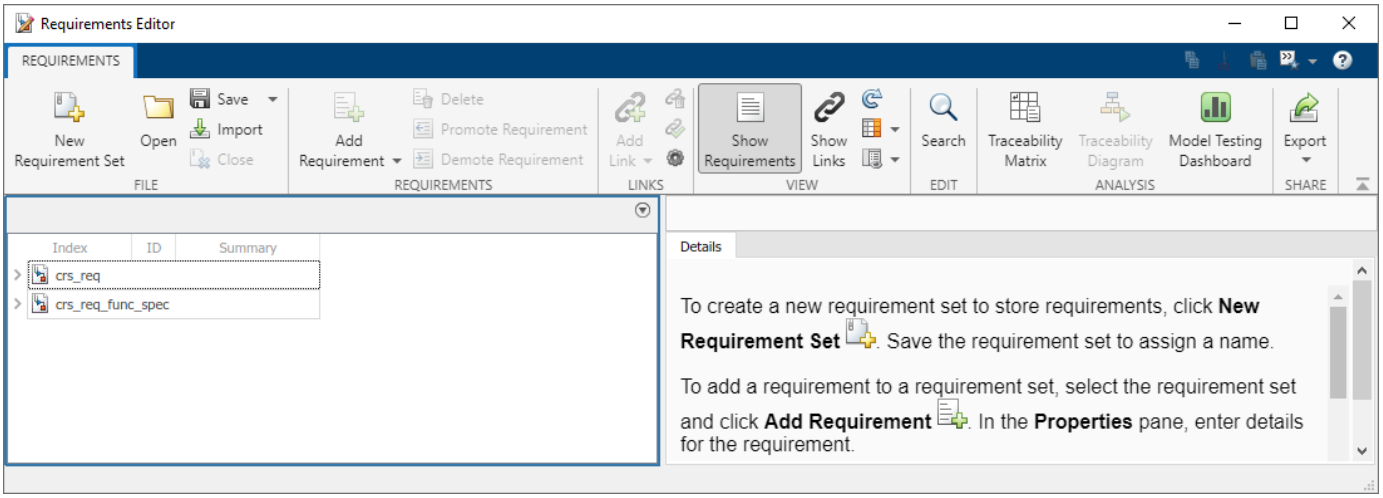
Requirements Toolbox Tools and Apps

Requirements Editor

Create and edit requirements

Description

Use the **Requirements Editor** app to create requirement sets, requirements, import and export requirements, and link requirements to blocks and other supported artifacts.



Open the Requirements Editor App

- Simulink Toolstrip: On the **Apps** tab, under **Model Verification, Validation, and Test**, click **Requirements Editor**.
- MATLAB Toolstrip: On the **Apps** tab, under **Verification, Validation, and Test**, click **Requirements Editor**.
- MATLAB command prompt: Enter `slreq.editor`.

Examples

Create, Open, and Delete Requirement Sets

To create a new requirement set:

1. In the **Requirements Editor**, click **New Requirement Set**.
2. Specify the name and file location of the requirement set. The editor saves the requirement set as a SLREQX file and the requirement set appears in the **Requirements Editor**.

You can open existing requirement sets by clicking **Open** and selecting a requirement set file. To delete a requirement set, click a requirement set and click the Close button  . Removing a requirement set does not delete the SLREQX file.

Add Requirements to a Requirement Set

To add requirements to a requirement set:

1. In the left pane, select a requirement set.
2. In the **Requirements** section, click **Add Requirement**.

Each requirement you create creates an associated `slreq.Requirement` object. You can edit the properties of the requirement in the **Requirements Editor** or programmatically. To adjust the properties in the **Requirements Editor**, click the requirement. The properties appear in the right pane of the editor.

Link Requirements to Model Artifacts

To link requirements to artifacts in models:

1. In an open Simulink model, click a model artifact. For a list of supported model artifacts, see “Supported Model Objects for Requirements Linking”.
2. In the **Requirements Editor**, click the requirement you want to link.
3. In the **Links** section, click **Add Link > Link from Selection in Simulink**.

For more information, see “Requirement Links”.

Filter Displayed Requirements

By default, the **Requirements Editor** displays loaded requirements in alphabetical order. To reduce the number of requirements displayed, apply a filter.

1. Open a requirement set.
2. In the **View** section, click **Show Requirements**.
3. In the **Edit** section, click **Search**.

When you perform a search:

- A requirement set is not visible if none of the requirements in the set pass the filter. If a child requirement passes the filter, the parent requirement set is also visible.
- The filter is not case-sensitive. For example, typing A displays the requirements whose columns contain an uppercase or lowercase A.
- The filter applies to the columns in the editor. If you add columns, the filter automatically applies to them.

Filter Displayed Links

By default, the **Requirements Editor** displays links to loaded requirement sets, in alphabetical order. To reduce the number of links displayed, apply a filter.

1. Open a requirement set.
2. In the **View** section, click **Show Links**.
3. In the **Edit** section, click **Search**.

Display Additional Columns

To display additional columns in the left pane:

1. Decide if you want to view requirement or link sets. To view requirement sets, in the **View** section, click **Show Requirements**. To view link sets, in the **View** section, click **Show Links**.
2. In the **View** section, click the **Columns** button  .

If you selected **Show Requirements**, you can select from these options:

- **Implementation Status**: Displays the implementation status summaries for your requirement sets. For more information, see “Review Requirements Implementation Status”.
- **Verification Status**: Displays the verification status summaries for your requirement sets. For more information, see “Review Requirements Verification Status”.
- **Select Attributes**: Select additional attributes to display. You can display the **Index**, **ID**, **Summary**,**Type**, **Keywords**, **SID**, **CreatedOn**, **CreatedBy**, **ModifiedOn**, **SyncronizedOn**, **ModifiedBy**, **Revision**, **Verified**, **Implemented**, **Description**, **Rationale**. The default attributes are **Index**, **ID**, and **Summary**.

If you selected **Show Links**, you can only click **Select Attributes**. You can then select the following attributes: **Label**, **Source**, **Type**,**Destination**, **Keywords**, **SID**, **CreatedOn**, **CreatedBy**, **ModifiedOn**, **ModifiedBy**, **Revision**, **Description**, and **Rationale**. The default attributes are **Label**, **Source**, **Type**, and **Destination**.

Once you display the attributes, you can filter them with the **Search** feature.

Import Requirements in Other Formats

To import requirements from a third-party requirements application:

1. In the **File** section, click **Import** to open the Import Requirements window.
2. In the **Document Type** property, select the file format. You can select Microsoft Word, Microsoft Excel, ReqIF File, and IBM DOORS Next.
3. In the **Document Location** property, select the location of the file.
4. Set the import options. Each format has different import options.

If you import the requirements, Requirements Toolbox creates an `slreq.Requirement` object for each requirement. If you import the requirements as referenced requirements, Requirements Toolbox

creates an `slreq.Reference` object for each requirement. For more information, see “Import Requirements from Third-Party Applications”.

Create Report from Requirements Information

To create a report for one or more requirement sets:

1. In the **Share** section, click **Export > Generate Report**. The Report Generation Options window opens.
2. Set the file name and location of the report by clicking the **Select** button.
3. Select the report content options.
4. Select the requirement sets to include in the report. The **Included Requirement Sets** section displays the loaded requirement sets. To add a requirement set, open the requirement set using the **Requirements Editor**.
5. Click **Generate Report**.

For more information, see “Report Requirements Information”.

Open the Traceability Matrix Window

To access the Traceability Matrix window:

In the **Analyze** section, click **Traceability Matrix**. You can then create a traceability matrix in the window. For more information, see “Track Requirement Links with a Traceability Matrix”.

Create a Traceability Diagram

To create a traceability diagram:

1. Click a requirement set.
2. In the **Analyze** section, click **Traceability Diagram**.

For more information, see “Visualize Links with a Traceability Diagram”.

Open the model testing dashboard

If you have a license for Simulink Check™, you can also open the model testing dashboard. To open the model testing dashboard:

In the **Analyze** section, click **Model Testing Dashboard**. For more information, see “Assess Requirements-Based Testing Quality by Using the Model Testing Dashboard” (Simulink Check) and

"Explore Status and Quality of Testing Activities Using the Model Testing Dashboard" (Simulink Check).

Parameters

View

`Show Requirements` — Show requirements and requirement sets
on (default) | off

Show the loaded requirements and requirement sets. To enable this parameter, in the **View** section, click **Show Requirements**.You can enable this parameter or the **Show Links** parameter.

`Show Links` — Show requirements links
off (default) | on

Show the loaded links and link sets. To enable this parameter, in the **View** section, click **Show Links**. You can enable this parameter or the **Show Requirements** parameter.

`Columns` — Select displayed columns in requirement and link sets
Select Attributes

Select attributes and information to display when viewing loaded requirement and link sets. In the **View** section, click the **Columns** button  . Once you display the attributes, you can filter them with the **Search** feature.

`Information` — Select displayed information for selected requirements
Change Information | **Comments** | **Code Traceability**

Select information you want to display in individual requirements. To access this parameter, in the **View** section, click the **Information** button  . You can then select the following information types:

- **Change Information**: Indicates changes to requirements. For more information, see "Track Changes to Requirement Links".
- **Comments**: Adds the comment section in the right pane of selected requirements.
- **Code Traceability**: Displays code traceability information of requirements. For more information, see "Requirements Traceability for MATLAB Code Lines".

The default information types displayed are **Change Information** and **Comments**.

Tips

- Open the **Requirements Manager** app in a Simulink model by navigating to the **Apps** tab and, under **Model Verification, Validation, and Test**, clicking **Requirements Manager**. You can use the **Requirements Manager** to edit and link requirements without leaving the Simulink model.

See Also

Functions
`slreq.ReqSet` | `slreq.Link` | `slreq.LinkSet` | `slreq.clear` | `slreq.import` | `slreq.load` | `slreq.new` | `slreq.open`

Topics
“Work with Requirements in the Requirements Editor”
“Access Frequently Used Features and Commands from the Requirements Editor”
“Assess Allocation and Impact”
“Define Custom Requirement and Link Types”

Introduced in R2017b

Operators

duration

Time during which condition is valid

Syntax

```
duration(C)
```

Description

`duration(C)` returns the length of time, in seconds, that has elapsed since the conditional expression `C` became `true`. You can use this operator only in the Requirements Table block.

Examples

Compare Duration Length to Input Data

Execute the action when the input data `u` is greater than or equal to `0` for longer than the value of the input data `y`. Otherwise, the action sets the output data `a` to `0`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Requirement 1	duration(u>=0) > y	a =1
2 D		Else	a = 0

Tips

- The Requirements Table block resets the value of the `duration` operator if the conditional expression `C` becomes `false` or if the block becomes inactive.
- The `duration` operator does not support conditions that depend on local or output data.

See Also

Requirements Table | isStartup | prev | t

Topics

“Use a Requirements Table Block to Create Formal Requirements”
“Control Requirement Execution by Using Temporal Logic”

Introduced in R2022a

hasChanged

Detect change in data since last time step

Syntax

```
tf = hasChanged(data_name)
```

Description

`tf = hasChanged(data_name)` returns `1` (`true`) if the value of `data_name` at the beginning of the current time step is different from the value of `data_name` at the beginning of the previous time step. Otherwise, the operator returns `0` (`false`). You can use this operator only in the Requirements Table block.

The argument `data_name` can be a:

- Scalar
- Matrix or an element of a matrix
- Structure or a field in a structure
- Valid combination of structure fields or matrix elements

If `data_name` is a matrix, the operator returns `true` when it detects a change in one of the elements of element of `data_name`. You can also index elements of a matrix by using numbers or expressions that evaluate to a constant integer.

If `data_name` is a structure, the operator returns `true` when it detects a change in one of the fields of `data_name`. You can also index fields in a structure by using dot notation.

The argument `data_name` cannot be a nontrivial expression or a custom code variable.

Examples

Detect Change in Input Data

Set the output data `a` to `1` if the input data `M` has changed since the last time step. Otherwise, set `a` to `0`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Requirement 1	hasChanged(M)	a = 1
2 D		Else	a = 0

Detect Change in Matrix Element

Set the output data `a` to `1` if the element in row 1 and column 3 of input data `M` has changed since the last time step. Otherwise, set `a` to `0`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Requirement 1	hasChanged(M(1,3))	a = 1
2 D		Else	a = 0

Detect Change in Structure

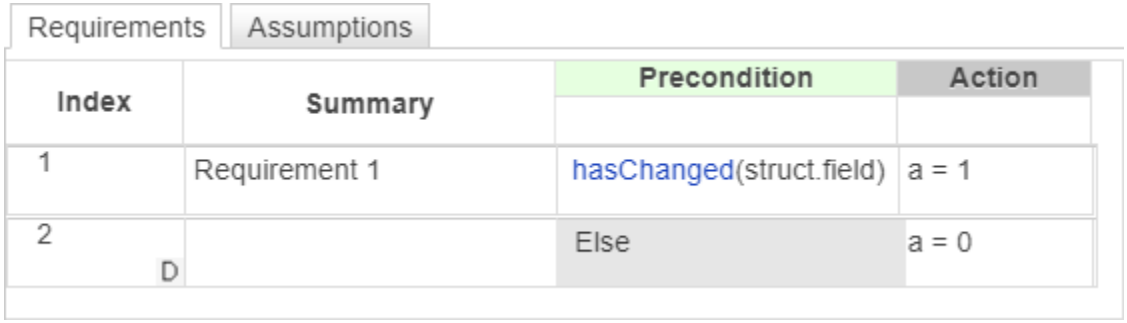
Set the output data `a` to `1` if one of the fields of the structure `struct` has changed value since the last time step. Otherwise, set `a` to `0`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Requirement 1	hasChanged(struct)	a = 1
2 D		Else	a = 0

Detect Change in Structure Field

Set the output data `a` to `1` if the field `struct.field` has changed value since the last time step. Otherwise, set `a` to `0`.



Tips

- If the Requirements Table block writes to the specified data but does not change the value, the `hasChanged` operator returns `false`.

See Also

Requirements Table | hasChangedTo | hasChangedFrom

Topics

“Use a Requirements Table Block to Create Formal Requirements”
“Detect Data Changes by Using Requirements Table Blocks”

Introduced in R2022a

hasChangedFrom

Detect change in data from specified value

Syntax

```
tf = hasChangedFrom(data_name,value)
```

Description

`tf = hasChangedFrom(data_name,value)` returns `1` (`true`) if the value of `data_name` was equal to the specified `value` at the beginning of the previous time step and is a different value at the beginning of the current time step. Otherwise, the operator returns `0` (`false`). You can use this operator only in the Requirements Table block.

The argument `data_name` can be a:

- Scalar
- Matrix or an element of a matrix
- Structure or a field in a structure
- Valid combination of structure fields or matrix elements

If `data_name` is a matrix, the operator returns `true` when it detects a change in one of the elements of `data_name`. You can also index elements of a matrix by using numbers or expressions that evaluate to a constant integer.

If `data_name` is a structure, the operator returns `true` when it detects a change in one of the fields of `data_name`. You can also index fields in a structure by using dot notation.

The argument `data_name` cannot be a nontrivial expression or a custom code variable.

The argument `value` must be an expression that resolves to a value that is comparable with `data_name`:

- If `data_name` is a scalar, then `value` must resolve to a scalar.
- If `data_name` is a matrix, then `value` must resolve to a matrix with the same dimensions as `data_name`.
- If `data_name` is a structure, then `value` must resolve to a structure whose field specification matches `data_name` exactly.

Examples

Detect Change in Input Data

Set the output data `a` to `1` if the input data `M` has changed from `1` since the last time step. Otherwise, set `a` to `0`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Requirement 1	hasChangedFrom(M,1)	a = 1
2 D		Else	a = 0

Detect Change in Matrix Element

Set the output data `a` to `1` if the element in row 1 and column 3 of input data `M` has changed from `1` since the last time step. Otherwise, set `a` to `0`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Requirement 1	hasChangedFrom(M(1,3),1)	a = 1
2 D		Else	a = 0

Detect Change in Structure

Set the output data `a` to `1` if one of the fields of the structure `struct` has changed from the value of `structValue` since the last time step. Otherwise, set `a` to `0`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Requirement 1	hasChangedFrom(struct,structValue)	a = 1
2 D		Else	a = 0

Detect Change in Structure Field

Set the output data `a` to `1` if the field `struct.field` has changed from the value of `1` since the last time step. Otherwise, set `a` to `0`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Requirement 1	hasChangedFrom(struct.field,1)	a = 1
2 D		Else	a = 0

Tips

- If the Requirements Table block writes to the specified data but does not change the value, the `hasChangedFrom` operator returns `false`.

See Also

Requirements Table | hasChanged | hasChangedTo

Topics

“Use a Requirements Table Block to Create Formal Requirements”
“Detect Data Changes by Using Requirements Table Blocks”

Introduced in R2022a

hasChangedTo

Detect change in data to specified value

Syntax

```
tf = hasChangedTo(data_name,value)
```

Description

`tf = hasChangedTo(data_name,value)` returns `1` (`true`) if the value of `data_name` was not equal to the specified `value` at the beginning of the previous time step and is equal to `value` at the beginning of the current time step. Otherwise, the operator returns `0` (`false`). You can use this operator only in the Requirements Table block.

The argument `data_name` can be a:

- Scalar
- Matrix or an element of a matrix
- Structure or a field in a structure
- Valid combination of structure fields or matrix elements

If `data_name` is a matrix, the operator returns `true` when it detects a change in one of the elements of `data_name`. You can also index elements of a matrix by using numbers or expressions that evaluate to a constant integer.

If `data_name` is a structure, the operator returns `true` when it detects a change in one of the fields of `data_name`. You can also index fields in a structure by using dot notation.

The argument `data_name` cannot be a nontrivial expression or a custom code variable.

The argument `value` must be an expression that resolves to a value that is comparable with `data_name`:

- If `data_name` is a scalar, then `value` must resolve to a scalar.
- If `data_name` is a matrix, then `value` must resolve to a matrix with the same dimensions as `data_name`.
- If `data_name` is a structure, then `value` must resolve to a structure whose field specification matches `data_name` exactly.

Examples

Detect Change in Input Data

Set the output data `a` to `1` if the input data `M` has changed to `1` since the last time step. Otherwise, set `a` to `0`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Requirement 1	hasChangedTo(M,1)	a = 1
2 D		Else	a = 0

Detect Change in Matrix Element

Set the output data `a` to `1` if the element in row 1 and column 3 of input data `M` has changed to `1` since the last time step. Otherwise, set `a` to `0`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Requirement 1	hasChangedTo(M(1,3),1)	a = 1
2 D		Else	a = 0

Detect Change in Structure

Set the output data `a` to `1` if one of the fields of the structure `struct` has changed value since the last time step and the current value of `struct` is equal to `structValue`. Otherwise, set `a` to `0`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Requirement 1	hasChangedTo(struct,structValue)	a = 1
2 D		Else	a = 0

Detect Change in Structure Field

Set the output data `a` to `1` if the field `struct.field` has changed to the value `1` since the last time step. Otherwise, set `a` to `0`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Requirement 1	hasChangedTo(struct.field,1)	a = 1
2 D		Else	a = 0

Tips

- If the Requirements Table block writes to the specified data but does not change the value, the `hasChangedTo` operator returns `false`.

See Also

Requirements Table | hasChanged | hasChangedFrom

Topics

“Use a Requirements Table Block to Create Formal Requirements”
“Detect Data Changes by Using Requirements Table Blocks”

Introduced in R2022a

isStartup

Whether simulation time is `0`

Syntax

```
isStartup
isStartup()
```

Description

`isStartup` returns `true` if the simulation time equals `0` and returns `false` at all other simulation times. You can use this operator only in the Requirements Table block.

`isStartup()` is an alternative way to execute `isStartup`.

Examples

Change Requirement Evaluation Due to Start Time

Use `isStartup` to check when the block input data `y` is greater than or equal to `0` when the simulation time equals `0`, and check that `y` is less than or equal to `0` at other times. The second requirement checks the logical opposite of `isStartup` with the `~` operator.

Requirements | Assumptions

Index	Summary	Precondition	Postcondition
1	Requirement 1	isStartup	y >= 0
2	Requirement 2	~isStartup	y <= 0

Tips

- Because `isStartup` returns a Boolean value, you can use it as the only entry in a requirement precondition of the Requirements Table block.
- You can use `isStartup` with `prev` to specify time-dependent requirement execution.

See Also

Requirements Table | duration | prev | t

Topics

“Use a Requirements Table Block to Create Formal Requirements”
“Control Requirement Execution by Using Temporal Logic”

“Establish Hierarchy in Requirements Table Blocks”

Introduced in R2022a

getPrevious, prev

Previous value of data

Syntax

```
z = getPrevious(u)
z = prev(u)
```

Description

`z = getPrevious(u)` returns the value of the data at the previous time step. This operator works only in the Requirements Table block.

`u` must be specified as input or output data.

`z = prev(u)` is an alternative way to execute `getPrevious(u)`.

Examples

Check Previous Data Values

At the start time, set `y` equal to `0`. After the start time, recall the value of the input data `u` in the precondition at the previous time step. One requirement checks if the previous value of `u` is greater than or equal to the current value, and another checks if the previous value is less than the current value. The block assigns different values for the output data `y`.

Requirements | Assumptions

Index	Summary	Precondition	Action
1	Startup Requirement	isStartup	y = 0
2	After Startup Requirement	~isStartup	
2.1		u >= prev(u)	y = 1
2.2		u < prev(u)	y = 2

Tips

- If `getPrevious` attempts to return the value of the data at a time step when it was not defined, it returns undefined behavior. For example, data is not defined before the simulation time is `0`. For this situation, use the `isStartup` operator to define additional requirements at a simulation time of `0` and `~isStartup` at the other time steps.

- You can use this operator only in the **Requirements** tab.

See Also

Requirements Table | duration | isStartup | t

Topics

“Use a Requirements Table Block to Create Formal Requirements”
“Control Requirement Execution by Using Temporal Logic”

Introduced in R2022a

t

Elapsed time of simulation

Syntax

`t`

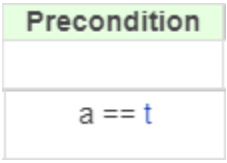
Description

`t` returns the simulation time in seconds. You can use this operator only in the Requirements Table block.

Examples

Check if Variable Equals Simulation Time

In a precondition, check if the variable `a` is equal to the simulation time in seconds.



Tips

- `t` captures the time of the highest model in the model hierarchy. As a result, `t` is the same value in each Requirements Table block used in a simulation, including disabled blocks in Enabled Subsystem blocks.

See Also

Requirements Table | isStartup | duration | prev

Topics

“Use a Requirements Table Block to Create Formal Requirements”
“Control Requirement Execution by Using Temporal Logic”

Introduced in R2022a

Objects

AssumptionRow

Work with assumptions in Requirements Table block

Description

`AssumptionRow` objects represent assumptions in Requirements Table blocks Use `AssumptionRow` objects to programmatically adjust the assumption properties.

Creation

There are several ways to create a `AssumptionRow` object:

- Create a new assumption in a Requirements Table block by using the `addAssumptionRow` object function.
- Create an assumption interactively in the Requirements Table block, then get the associated `AssumptionRow` object by using the `getAssumptionRows` object function.

Properties

`Index` — Index of assumption
character vector (default)

This property is read-only.

Index of the assumption, returned as a character vector. When you create a new assumption, the software automatically assigns the assumption a unique index.

`Preconditions` — Precondition expression
`{''}` (default) | cell array of character vectors

Precondition expression, specified as a cell array of a character vector. For more information on preconditions in assumptions, see “Add Assumptions to Requirements”.

Data Types: `char` | `cell`

`Postconditions` — Postcondition expression
`{''}` (default) | cell array of character vectors

Postcondition expression, specified as a cell array of a character vector. For more information on postconditions in assumptions, see “Add Assumptions to Requirements”.

Data Types: `char` | `cell`

`Summary` — Assumption summary text
`""` (default) | string scalar | character vector

Assumption summary text, specified as a string scalar or character vector. Use this property to add text to the **Summary** column in the **Assumptions** tab of the Requirements Table block.

Data Types: `char` | `string`

Object Functions

addChild	Add child requirement or assumption to Requirements Table block
getChildren	Retrieve child requirements and assumptions in Requirements Table block
clear	Clear row in Requirements Table block
removeRow	Remove Requirements Table block row

Examples

Create Assumptions and Set Preconditions and Postconditions

In a `RequirementsTable` object named `reqTable`, add two assumptions.

```
addAssumptionRow(reqTable);
addAssumptionRow(reqTable);
```

Retrieve the `AssumptionRow` objects.

```
aRow = getAssumptionRows(reqTable);
```

Set the preconditions for the assumptions.

```
aRow(1).Preconditions = {'u1 > 1'};
aRow(2).Preconditions = {'u1 > 0'};
aRow(3).Preconditions = {'u1 > -1'};
```

Set the postconditions for the assumptions.

```
aRow(1).Postconditions = {'u2 > 1'};
aRow(2).Postconditions = {'u2 > 0'};
aRow(3).Postconditions = {'u2 < -1'};
```

See Also

Objects

`RequirementsTable` | `RequirementRow`

Functions

`addAssumptionRow` | `getAssumptionRows`

Introduced in R2022a

RequirementRow

Work with requirements in Requirements Table block

Description

`RequirementRow` objects represent requirements in Requirements Table blocks. Use the objects to programmatically adjust the requirement properties.

Creation

There are several ways to create a `RequirementRow` object:

- Create a new requirement in a Requirements Table block by using the `addRequirementRow` object function.
- Create a requirement interactively in the Requirements Table block, then get the associated `RequirementRow` object by using the `getRequirementRows` object function.

Properties

`Actions` — Action expression
`{''}` (default) | cell array of character vectors

Action expressions, specified as a cell array of character vectors. For more information on actions, see “Use a Requirements Table Block to Create Formal Requirements”.

Data Types: `cell` | `char`

`Duration` — Duration expression
`""` (default) | string scalar | character vector

Duration expression, entered as a string scalar or character vector.

Data Types: `char` | `string`

`Index` — Index of requirement
character vector (default)

This property is read-only.

Index of the requirement, returned as a character vector. When you create a new requirement, the software automatically assigns the requirement a unique index.

`Preconditions` — Precondition expression
`{''}` (default) | cell array of character vectors

Precondition expressions, specified as a cell array of character vectors. You can also use the `addRequirementRow` object function to set the Precondition property when you create the `RequirementRow` object.

Example: `reqRow.Preconditions = {'u1 > 0','','u3 > 0'}` specifies the preconditions in a requirement with `u1 > 0` in the first **Precondition** column, nothing in the second **Precondition** column, and `u3 > 0` in the third **Precondition** column.

Data Types: `cell` | `char`

`Postconditions` — Postcondition expression

`{''}` (default) | cell array of character vectors

Postcondition expressions, specified as a cell array of character vectors.

Example: `reqRow.Postconditions = {'u1 > 0','','u3 > 0'}` specifies the postconditions in a requirement with `u1 > 0` in the first **Postcondition** column, nothing in the second **Postcondition** column, and `u3 > 0` in the third **Postcondition** column.

Data Types: `cell` | `char`

`Summary` — Requirement summary text

`""` (default) | string scalar | character vector

Requirement summary text, specified as a string scalar or character vector. Use this property to add text to the **Summary** column in the **Requirements** tab of the Requirements Table block.

Data Types: `char` | `string`

Object Functions

addChild	Add child requirement or assumption to Requirements Table block
getChildren	Retrieve child requirements and assumptions in Requirements Table block
clear	Clear row in Requirements Table block
removeRow	Remove Requirements Table block row

Examples

Create Requirements and Set Preconditions and Postconditions

In a `RequirementsTable` object named `reqTable`, add two additional requirements.

```
addRequirementRow(reqTable);
addRequirementRow(reqTable);
```

Retrieve the `RequirementRow` objects.

```
rRow = getRequirementRows(reqTable);
```

Set the preconditions for the requirements.

```
rRow(1).Preconditions = {'u1 > 1'};
rRow(2).Preconditions = {'u1 > 0'};
rRow(3).Preconditions = {'u1 > -1'};
```

Set the postconditions for the requirements.

```
rRow(1).Postconditions = {'u2 > 1'};
rRow(2).Postconditions = {'u2 > 0'};
rRow(3).Postconditions = {'u2 < -1'};
```

See Also

Objects
RequirementsTable | AssumptionRow

Functions
addRequirementRow | getRequirementRows

Introduced in R2022a

RequirementsTable

Configure Requirements Table blocks

Description

Use `RequirementsTable` objects to configure Requirements Table blocks.

Creation

There are several ways to create a `RequirementsTable` object:

- Use the `slreq.modeling.create` function to create a new Simulink model that contains a Requirements Table block.
- Add a Requirements Table block to an existing model using `add_block` and retrieve the object with the `slreq.modeling.find` function.

Properties

`Name` — Name of Requirements Table block
`"Requirements Table"` (default) | string scalar | character vector

Name of the Requirements Table block, specified as a string scalar or character vector.

Example: `table.Name = "tableName"` changes the block name to `tableName`

Data Types: `char` | `string`

`Path` — Path of Requirements Table block
string scalar | character vector

This property is read-only.

Path of the Requirements Table block, specified as a string scalar or character vector.

Data Types: `char` | `string`

`RequirementHeaders` — Requirement Table block headers
structure array

Requirement Table block headers, specified as a structure array. Specify headers to add under the **Precondition**, **Postcondition**, and **Action** columns in the **Requirements** tab by setting the `preconditions`, `postconditions`, and `actions` fields to a string vector or cell array of character vectors. Use a cell array to add multiple columns under the **Precondition**, **Postcondition**, and **Action** columns.

Example: `table.RequirementHeaders.preconditions = ["u1","",""]` changes the **Precondition** column header where one header is `u1` and the other two are empty.

Data Types: `struct`

Object Functions

addRequirementRow	Add requirement to Requirements Table block
addAssumptionRow	Add assumption to Requirements Table block
addSymbol	Add data to Requirements Table block
clear	Clear row in Requirements Table block
getAssumptionRows	Retrieve assumptions in Requirements Table block
getRequirementRows	Retrieve requirements in Requirements Table block
findSymbol	Retrieve data in Requirements Table block
hideAssumptionColumn	Hide Precondition column in Assumptions tab
hideRequirementColumn	Hide columns in Requirements tab
removeRow	Remove Requirements Table block row
showAssumptionColumn	Show Precondition column in Assumptions tab
showRequirementColumn	Show columns in Requirements tab

Examples

Change Name of a Requirements Table Block

Create a new model called `myModel` that contains a Requirements Table block.

```
table = slreq.modeling.create("myModel");
```

Change the name of the block to `newTableName`.

```
table.Name = "newTableName";
```

Specify Precondition, Postcondition, and Action Columns

Create a new model called `myModel` that contains a Requirements Table block.

```
table = slreq.modeling.create("myModel");
```

Specify three **Precondition** columns with empty headers.

```
table.RequirementHeaders.preconditions = ["","",""];
```

Specify two **Postcondition** columns where one header is `u1` and the other is empty.

```
table.RequirementHeaders.postconditions = ["u1",""];
```

Specify two **Action** columns with the headers `u2` and `u3`.

```
table.RequirementHeaders.actions = ["u2","u3"];
```

See Also

Blocks
Requirements Table

Objects
`AssumptionRow` | `RequirementRow` | `Symbol`

Functions
`slreq.modeling.create` | `slreq.modeling.find`

Introduced in R2022a

Symbol

Configure data in Requirements Table blocks

Description

`Symbol` objects represent the data in Requirements Table blocks. Use `Symbol` objects to configure the input, output, parameter, local, and constant data in a Requirements Table block.

Creation

There are several ways to create a `Symbol` object:

- Create new data in a Requirements Table block by using the `addSymbol` object function.
- Create new data interactively in the Requirements Table block, then get the associated `Symbol` object by using the `findSymbol` object function.

Properties

`Complexity` — Whether data accepts complex values
`"Off"` (default) | `"On"` | `"Inherited"`

Whether the data accepts complex values, specified as one of these values:

Complexity	Description
`"Inherited"`	The data inherits complexity based on the `Scope` property. Input and output data inherit complexity from the Simulink signals connected to the associated input and output ports. Local and parameter data inherit complexity from the parameter to which the data is bound.
`"Off"`	The data is a real number.
`"On"`	The data is a complex number.

Data Types: `enumerated`

`isDesignOutput` — Whether data is design model output
`false` or `0` (default) | `true` or `1`

Whether the data is a design model output, specified as a numeric or logical `1` (`true`) or `0` (`false`). This property applies only when the `Scope` property is `Input`. For more information, see “Treat as design model output for analysis”.

Data Types: `logical`

`Name` — Name of data
`"data"` (default) | string scalar | character vector

Name of the data, specified as a string scalar or character vector.

Data Types: `char` | `string`

Scope — Scope of data
`"Input"` (default) | `"Output"` | `"Local"` | `"Constant"` | `"Parameter"`

Scope of the data that specifies where the data resides in memory relative to the block, specified as one of these values:

Scope	Description
`"Input"`	The data is an input signal to a Requirements Table block.
`"Output"`	The data is an output signal of a Requirements Table block.
`"Local"`	The data is defined in the current block only.
`"Constant"`	The data is a read-only constant value that is visible to the block.
`"Parameter"`	The data resides in a variable of the same name in the MATLAB workspace, the model workspace, or in the workspace of a masked subsystem that contains this block.

Data Types: `enumerated`

Size — Size of data
`"-1"` (default) | string scalar | character vector

Size of the data, specified as a string scalar or character vector. This property must resolve to a scalar value or a MATLAB vector of values. The default value is `"-1"`, which means that the size is inherited. For more information, see “Inherit Size from Simulink” (Simulink).

Data Types: `char` | `string`

Type — Data type
`"Inherit: Same as Simulink"` (default) | `"double"` | `"single"` | `"int8"` | ...

Data type, specified as:

- `"Inherit: Same as Simulink"`
- `"double"`
- `"single"`
- `"half"`
- `"int64"`
- `"int32"`
- `"int16"`
- `"int8"`
- `"uint64"`
- `"uint32"`

- "uint16"
- "uint8"
- "boolean"
- "fixdt(1,16,0)"
- "fixdt(1,16,2^0,0)"
- "Enum: <class name>"
- "Bus: <object name>"

To modify the data type properties, use the **Symbols** pane and Property Inspector. For more information, see “Set Data Types in Requirements Table Blocks”.

Data Types: enumerated

Examples

Add Data to a Requirement Table Block

Create a new model called myModel that contains a Requirements Table block.

```
table = slreq.modeling.create("myModel");
```

Add data named u1 to the block.

```
data = addSymbol(table,Name="u1");
```

Retrieve Data and Change It

From a model named myModel that contains a Requirements Table block, retrieve the RequirementsTable object.

```
table = slreq.modeling.find("myModel");
```

Retrieve the Symbol objects from the block.

```
data = findSymbols(table);
```

Change the properties of the first Symbol object in the array.

```
data(1).Name = "u1";
data(1).Scope = "Output";
```

See Also

addSymbol | findSymbol

Introduced in R2022a