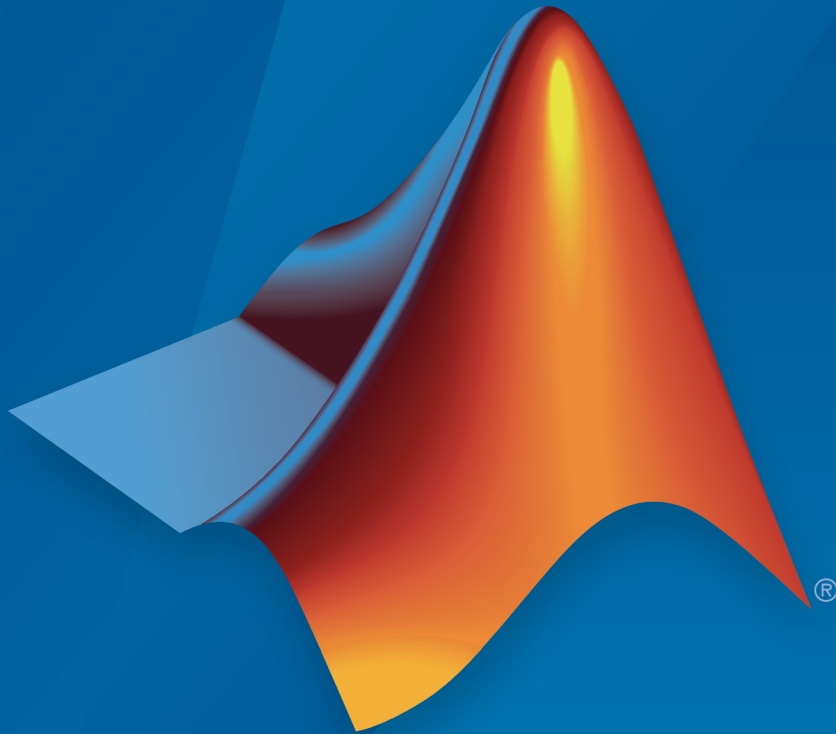


Simulink[®] Test[™]

Reference



MATLAB[®]&SIMULINK[®]

R2016a



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.
3 Apple Hill Drive
Natick, MA 01760-2098

Simulink[®] Test[™] Reference

© COPYRIGHT 2015–2016 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

March 2015	Online Only	New for Version 1.0 (Release 2015a)
September 2015	Online Only	Revised for Version 1.1 (Release 2015b)
October 2015	Online only	Rereleased for Version 1.0.1 (Release 2015aSP1)
March 2016	Online Only	Revised for Version 2.0 (Release 2016a)

1 | Functions — Alphabetical List

2 | Classes — Alphabetical List

3 | Methods — Alphabetical List

4 | Blocks — Alphabetical List

Functions — Alphabetical List

sltest.harness.check

Compare component under test between harness model and main model

Syntax

```
CheckResult = sltest.harness.check(harnessOwner, harnessName)
[CheckResult, CheckDetails] = sltest.harness.check(harnessOwner,
harnessName)
```

Description

`CheckResult = sltest.harness.check(harnessOwner, harnessName)` computes the checksum of the component under test in the harness model `harnessName` and compares it to the checksum of the component `harnessOwner` in the main model. The function returns `CheckResult` as `true` or `false`.

`[CheckResult, CheckDetails] = sltest.harness.check(harnessOwner, harnessName)` returns additional details of the check operation to the structure `CheckDetails`.

Examples

Compare Checksums for a Subsystem

Compute the checksums of the `Controller` subsystem in the model `f14` and the harness model `controller_harness`, and compare the results.

```
f14;
sltest.harness.create('f14/Controller', 'Name', 'controller_harness');
CheckResult = sltest.harness.check('f14/Controller', ...
'controller_harness')
```

```
CheckResult = 0
```

Compare Checksums for a Subsystem and Get Details

Compute the checksums of the `Controller` subsystem in the model `f14` and the harness model `controller_harness`, and compare the results.

```
f14;
sltest.harness.create('f14/Controller', 'Name', 'controller_harness');
[CheckResult, CheckDetails] = sltest.harness.check('f14/Controller', ...
    'controller_harness')

CheckResult = 0

CheckDetails =

    overall: 0
    contents: 0
    interface: 0
    reason: 'VirtualSubsystem'
```

Input Arguments

harnessOwner — Model or component

string | double

Model or component handle or path, specified as a string or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

string

The name of the harness, specified as a string.

Example: 'harness_name'

Output Arguments

CheckResult — Result of comparison

true | false

The result of the component comparison between the harness model and the system model, returned as true or false.

For a block diagram harness, the function returns `CheckResult = true`.

For a virtual subsystem harness, the function returns `CheckResult = false`.

CheckDetails — Details of the check operation

structure

Details of the check operation, returned as a structure. Fields contain the results of the overall component comparison, the results of the component interface and contents comparison, the type of component under test in the harness, and the checksum values for the main model and harness model components.

See Also

`sltest.harness.close` | `sltest.harness.create` | `sltest.harness.delete` | `sltest.harness.export` | `sltest.harness.find` | `sltest.harness.load` | `sltest.harness.open` | `sltest.harness.push` | `sltest.harness.rebuild` | `sltest.harness.set`

Introduced in R2015a

sltest.harness.clone

Copy test harness

Syntax

```
sltest.harness.clone(HarnessOwner, HarnessName)  
sltest.harness.clone(HarnessOwner, HarnessName, NewHarness)
```

Description

`sltest.harness.clone(HarnessOwner, HarnessName)` clones the test harness `HarnessName` associated with the model or component `HarnessOwner`. The cloned harness contains the source harness model contents, configuration settings, and callbacks.

`sltest.harness.clone(HarnessOwner, HarnessName, NewHarness)` uses an additional argument `NewHarness` to specify the name of the cloned harness.

Examples

Clone a Subsystem Test Harness

Create a test harness `ControllerHarness1` for the `Controller` subsystem of the model `f14`. Clone the harness and save it as `ControllerHarness2`.

```
f14  
sltest.harness.create('f14/Controller', 'Name', 'ControllerHarness1', ...  
  'EnableComponentEditing', true)  
sltest.harness.clone('f14/Controller', 'ControllerHarness1', 'ControllerHarness2')
```

Input Arguments

HarnessOwner — Model or component

string | double

Model or component handle or path, specified as a string or double.

Example: 1.9500e+03

Example: 'f14'

Example: 'f14/Controller'

HarnessName — Source harness name

string

The name of the source harness, specified as a string.

Example: 'ControllerHarness'

NewHarness — Cloned harness name

string

The name of the cloned harness, specified as a string.

Example: 'ControllerHarness2'

See Also

`sltest.harness.check` | `sltest.harness.close` | `sltest.harness.create`
| `sltest.harness.delete` | `sltest.harness.export` | `sltest.harness.find`
| `sltest.harness.load` | `sltest.harness.open` | `sltest.harness.push` |
`sltest.harness.rebuild` | `sltest.harness.set`

Introduced in R2015b

sltest.harness.close

Close test harness

Syntax

```
sltest.harness.close(modelName)
sltest.harness.close(harnessOwner)
sltest.harness.close(harnessOwner, harnessName)
```

Description

`sltest.harness.close(modelName)` closes the open test harness associated with the model `modelName`.

`sltest.harness.close(harnessOwner)` closes the open test harness associated with the model or component `harnessOwner`.

`sltest.harness.close(harnessOwner, harnessName)` closes the test harness `harnessName`, which is associated with the model or component `harnessOwner`.

Examples

Close a Harness Associated With a Subsystem

Close the test harness named `controller_harness`, associated with the subsystem `Controller` in the model `f14`.

```
f14;
sltest.harness.create('f14/Controller', 'Name', 'sample_controller_harness');
sltest.harness.open('f14/Controller', 'sample_controller_harness');
sltest.harness.close('f14/Controller', 'sample_controller_harness');
```

Close a Harness Associated With a Top-level Model

Close the test harness named `sample_harness`, which is associated with the model `f14`.

```
f14;
```

```
sltest.harness.create('f14', 'Name', 'sample_harness');  
sltest.harness.open('f14', 'sample_harness');  
sltest.harness.close('f14', 'sample_harness');
```

Input Arguments

modelName — Model name

string | double

Model handle or path, specified as a string or double.

Example: 1.9500e+03

Example: 'model_name'

harnessOwner — Model or component name

string | double

Model or component handle or path, specified as a string or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

string

The name of the harness, specified as a string.

Example: 'harness_name'

See Also

```
sltest.harness.check | sltest.harness.create | sltest.harness.delete  
| sltest.harness.export | sltest.harness.find | sltest.harness.load |  
sltest.harness.open | sltest.harness.push | sltest.harness.rebuild |  
sltest.harness.set
```

Introduced in R2015a

sltest.harness.convert

Convert test harnesses between internal and external storage

Syntax

```
sltest.harness.convert(modelName)
sltest.harness.convert(modelName,conversion)
```

Description

`sltest.harness.convert(modelName)` converts the test harnesses storage type for `modelName`.

`sltest.harness.convert(modelName,conversion)` converts test harnesses storage type using the additional option `conversion` specifying which storage type is being converted to.

Examples

Convert Test Harnesses from External to Internal

```
sltest.harness.convert('f14', 'InternalToExternal')
```

Input Arguments

modelName — Model name

string

Model handle or path, specified as a string.

Example: 'model_name'

conversion — Conversion type

'InternalToExternal' | 'ExternalToInternal'

Conversion to perform, specified as a string.

Example: `'InternalToExternal'`

See Also

`sltest.harness.check` | `sltest.harness.close` | `sltest.harness.create` |
`sltest.harness.delete` | `sltest.harness.export` | `sltest.harness.open`

Introduced in R2016a

sltest.harness.create

Create test harness

Syntax

```
sltest.harness.create(harnessOwner)
sltest.harness.create(harnessOwner,Name,Value)
```

Description

`sltest.harness.create(harnessOwner)` creates a test harness for the model component `harnessOwner`, using default properties.

`sltest.harness.create(harnessOwner,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Examples

Create Harness for a Model

Create harness for the `f14` model. The harness is called `sample_harness` and has a Signal Builder block source and a scope sink.

```
f14;
sltest.harness.create('f14','Name','sample_harness','Source',...
'Signal Builder','Sink','Scope')
```

Create Harness for a Subsystem

Create harness for the `Controller` subsystem of the `f14` model. The harness allows editing of `Controller` and uses default properties for the other options.

```
f14;
sltest.harness.create('f14/Controller','EnableComponentEditing',true);
```

Create Default Harness for a Subsystem

Create a default harness for the `Controller` subsystem of the `f14` model.

```
f14;  
sltest.harness.create('f14/Controller');
```

Input Arguments

harnessOwner — Model or component

string | double

Model or component handle or path, specified as a string or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

Name-Value Pair Options

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 (' '). You can specify several name and value pair arguments in any order as **Name1**, **Value1**, ..., **NameN**, **ValueN**.

Example: 'Name', 'controller_harness', 'Source', 'Signal Builder', 'Sink', 'To File' specifies a harness named `controller_harness`, with a signal builder block source and To File block sinks for the component under test.

'Name' — Harness name

string

The name for the harness you create, specified as the comma-separated pair consisting of 'Name' and a valid MATLAB filename.

Example:

```
'Name', 'harness_name'
```

'Description' — Harness description

string

The harness description, specified as the comma-separated pair consisting of 'Description' and a string.

Example:

```
'Description', 'A test harness'
```

'Source' — Component under test input

```
'Inport' (default) | 'Signal Builder' | 'From Workspace' | 'From File' |  
'Test Sequence' | 'None' | 'Custom'
```

The input to the component, specified as the comma-separated pair consisting of **'Source'** and one of the possible source values.

Example:

```
'Source', 'Signal Builder'
```

'CustomSourcePath' — Path to library block for custom source

string

For a custom source, the path to the library block to use as the source, specified as the comma-separated pair consisting of **'CustomSourcePath'** and the path.

Example:

```
'CustomSourcePath', 'simulink/Sources/Sine Wave'
```

'Sink' — Harness output

```
'Outport' (default) | 'Scope' | 'To Workspace' | 'To File' | 'None' |  
'Custom'
```

The output of the component, specified as the comma-separated pair consisting of **'Sink'** and one of the possible sink values.

Example:

```
'Sink', 'Scope'
```

'CustomSinkPath' — Path to library block for custom sink

string

For a custom sink, the path to the library block to use as the sink, specified as the comma-separated pair consisting of **'CustomSinkPath'** and the path.

Example:

```
'CustomSinkPath', 'simulink/Sinks/Terminator'
```

'SeparateAssessment' — Separate Test Assessment block when using Test Sequence source

false (default) | true

Option to add a separate Test Assessment block to the test harness, specified as a comma-separated pair consisting of 'SeparateAssessment' and false or true. 'Source' must be 'Test Sequence'.

Example:

```
'SeparateAssessment',true
```

'EnableComponentEditing' — Option for component editing

false (default) | true

Option to enable or disable component editing in the harness, specified as a comma-separated pair consisting of 'EnableComponentEditing' and false or true.

Example:

```
'EnableComponentEditing',true
```

'CreateWithoutCompile' — Option to create harness without compiling main model

false (default) | true

Option to specify harness creation without compiling the main model, specified as a comma-separated pair consisting of 'CreateWithoutCompile' and false or true.

false compiles the model and runs other operations to support the harness build.

true creates the harness without model compilation.

Example:

```
'CreateWithoutCompile',false
```

'VerificationMode' — Option to use normal (model), software-in-the-loop (SIL), or processor-in-the-loop (PIL) block as component under test

'Normal' (default) | 'SIL' | 'PIL'

An option to specify what type of block to use in the test harness, specified as a comma-separated pair consisting of 'VerificationMode' and the type of block to use. SIL and PIL blocks require Simulink Coder.

Example:

```
'VerificationMode', 'SIL'
```

'RebuildOnOpen' — Sets the harness rebuild command to execute when the harness opens
false (default) | true

Option to have the harness rebuild when it opens, specified as the comma-separated pair consisting of 'UseDefaultName' and false or true.

Example:

```
'RebuildOnOpen', true
```

'RebuildModelData' — Sets configuration set and model workspace entries to be updated during the test harness rebuild
false (default) | true

Option to have the configuration set and model workspace entries updated during test harness rebuild, specified as the comma-separated pair consisting of 'RebuildModelData' and true or false.

Example:

```
'RebuildModelData', true
```

'SaveExternally' — Test harnesses saved as separate SLX files
false (default) | true

Option to have each test harness saved as a separate SLX file, specified as the comma-separated pair consisting of 'SaveExternally' and true or false. A model cannot use both external and internal test harness storage. If a model already has test harnesses, a new test harness follows the storage type of the existing harnesses, which this option does not override. See “Manage Test Harnesses”.

Example:

```
'SaveExternally', true
```

See Also

sltest.harness.check | sltest.harness.clone | sltest.harness.close |
sltest.harness.convert | sltest.harness.delete | sltest.harness.export

| `sltest.harness.find` | `sltest.harness.load` | `sltest.harness.open` |
`sltest.harness.set`

Introduced in R2015a

sltest.harness.delete

Delete test harness

Syntax

```
sltest.harness.delete(harnessOwner, harnessName)
```

Description

`sltest.harness.delete(harnessOwner, harnessName)` deletes the harness `harnessName` associated with `harnessOwner`.

Examples

Delete a Harness Associated With a Subsystem

Delete the test harness `controller_harness`, which is associated with the `Controller` subsystem in the `f14` model.

```
f14;  
sltest.harness.create('f14/Controller', 'Name', 'controller_harness');  
sltest.harness.delete('f14/Controller', 'controller_harness');
```

Delete a Harness Associated With a Top-level Model

Delete the test harness `bd_harness`, which is associated with the model `f14`.

```
f14;  
sltest.harness.create('f14', 'Name', 'bd_harness');  
sltest.harness.delete('f14', 'bd_harness');
```

Input Arguments

harnessOwner — Model or component

string | double

Model or component handle or path, specified as a string or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

string

The name of the harness, specified as a string.

Example: 'harness_name'

See Also

`sltest.harness.check` | `sltest.harness.clone` | `sltest.harness.close` |
`sltest.harness.create` | `sltest.harness.export` | `sltest.harness.find` |
`sltest.harness.load` | `sltest.harness.open` | `sltest.harness.push` |
`sltest.harness.rebuild` | `sltest.harness.set`

Introduced in R2015a

sltest.harness.export

Export test harness to Simulink model

Syntax

```
sltest.harness.export(harnessOwner, harnessName, 'Name', modelName)
```

Description

`sltest.harness.export(harnessOwner, harnessName, 'Name', modelName)` exports the harness `harnessName`, associated with the model or component `harnessOwner`, to a new Simulink® model specified by the pair `'Name', modelName`.

The model must be saved prior to export.

Examples

Export a Harness to a New Model

Export the harness `controller_harness`, which is associated with the `Controller` subsystem of the `f14` model. The new model name is `model_from_harness`.

```
f14;  
sltest.harness.create('f14/Controller', 'Name', 'controller_harness');  
save_system('f14');  
sltest.harness.export('f14/Controller', 'controller_harness', 'Name', ...  
    'model_from_harness');
```

Input Arguments

harnessOwner — Model or component

string | double

Model or component handle or path, specified as a string or double

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Name of the harness from which to create the model

string

The name of the harness, specified as a string.

Example: 'harness_name'

modelName — Name of the new model

string

A valid MATLAB filename for the model generated from the harness, specified as a string.

Example: 'harness_name'

See Also

sltest.harness.check | sltest.harness.clone | sltest.harness.close |
sltest.harness.create | sltest.harness.delete | sltest.harness.find
| sltest.harness.load | sltest.harness.open | sltest.harness.push |
sltest.harness.rebuild | sltest.harness.set

Introduced in R2015a

sltest.harness.find

Find test harnesses in model

Syntax

```
harnessList = sltest.harness.find(harnessOwner)
harnessList = sltest.harness.find(harnessOwner,Name,Value)
```

Description

`harnessList = sltest.harness.find(harnessOwner)` returns a structure listing harnesses and harness properties that exist for the component or model `harnessOwner`.

`harnessList = sltest.harness.find(harnessOwner,Name,Value)` uses additional search options specified by one or more `Name,Value` pair arguments.

Examples

Use RegEx to Find Harnesses for a Model Component

Find harnesses for the `f14` model and its first-level subsystems. The function matches harness names according to a regular expression.

```
f14;
sltest.harness.create('f14','Name','model_harness');
sltest.harness.create('f14/Controller','Name','Controller_Harness1');
harnessList = sltest.harness.find('f14','SearchDepth',1,'Name','_[Hh]arnes+',...
'RegExp','on')
```

```
harnessList =
```

```
1x2 struct array with fields:
```

```
    model
    name
    description
    type
    ownerHandle
```

```
ownerFullPath  
ownerType  
isOpen  
canBeOpened  
lockMode  
verificationMode  
saveIndependently  
rebuildOnOpen  
rebuildModelData  
graphical  
origSrc  
origSink
```

Input Arguments

harnessOwner — Model or component

string | double

Model or component handle or path, specified as a string or double

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

Name-Value Pair Options

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 (' '). You can specify several name and value pair arguments in any order as **Name1**, **Value1**, ..., **NameN**, **ValueN**.

Example: 'SearchDepth', 2, 'Name', 'controller_harness' searches the model or component, and two lower hierarchy levels, for harnesses named `controller_harness`.

'Name' — Harness name to search for

verbatim string | regular expression string

Harness name to search for in the model, specified as the comma-separated pair consisting of **'Name'** and a verbatim string or a regular expression string. You can specify a regular expression only if you also use the **Name, Value** pair **'RegExp'**, **'on'**.

Example:

```
'Name', 'sample_harness'
```

```
'Name', '_[Hh]arnes+'
```

'RegExp' — Ability to search using a regular expression

'off' (default) | 'on'

Ability to search using a regular expression, specified as the comma-separated pair consisting of 'RegExp' and 'off' or 'on'. When 'RegExp' is set to 'on', you can use a regular expression with 'Name'.

Example:

```
'RegExp', 'on'
```

'SearchDepth' — Subsystem levels to search

all levels (default) | nonnegative integer

Subsystem levels into harnessOwner to search for harnesses, specified as the comma-separated pair consisting of 'SearchDepth' and an integer. For example:

0 searches harnessOwner.

1 searches harnessOwner and its subsystems.

2 searches harnessOwner, its subsystems, and their subsystems.

When you do not specify SearchDepth, the function searches all levels of harnessOwner.

Example:

```
'SearchDepth', 1
```

'OpenOnly' — Search option for open harnesses

'off' (default) | 'on'

Search option to return only active harnesses, specified as the comma-separated pair consisting of 'OpenOnly' and 'off' or 'on'.

Example:

'OpenOnly', 'on'

See Also

sltest.harness.check | sltest.harness.clone | sltest.harness.close |
sltest.harness.create | sltest.harness.delete | sltest.harness.export
| sltest.harness.load | sltest.harness.open | sltest.harness.push |
sltest.harness.rebuild | sltest.harness.set

Introduced in R2015a

sltest.harness.load

Load test harness

Syntax

```
sltest.harness.load(harnessOwner, harnessName)
```

Description

`sltest.harness.load(harnessOwner, harnessName)` loads the harness `harnessName` into memory. `harnessName` is associated with the model or component `harnessOwner`.

Examples

Load a Harness Associated With a Subsystem

Load the test harness `controller_harness`, which is associated with the `Controller` subsystem in the `f14` model.

```
f14;  
sltest.harness.create('f14/Controller', 'Name', 'controller_harness');  
save_system('f14');  
sltest.harness.load('f14/Controller', 'controller_harness');
```

Input Arguments

harnessOwner — Model or component

string | double

Model or component handle or path, specified as a string or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

string

The name of the harness, specified as a string.

Example: 'harness_name'

See Also

`sltest.harness.check` | `sltest.harness.close` | `sltest.harness.create` |
`sltest.harness.delete` | `sltest.harness.export` | `sltest.harness.find` |
`sltest.harness.open` | `sltest.harness.push` | `sltest.harness.rebuild` |
`sltest.harness.set`

Introduced in R2015a

sltest.harness.move

Move test harness from linked instance to library block

Syntax

```
sltest.harness.move(harnessOwner, harnessName)
```

Description

`sltest.harness.move(harnessOwner, harnessName)` moves the test harness `harnessName` associated with the block `harnessOwner` from the linked instance to its reference library block. Moving the test harness removes it from the linked instance.

Examples

Move Test Harness from Linked Block to Library

Move the test harness `Baseline_controller_tests` from the linked instance of the `Controller` subsystem to the library subsystem.

Open the model.

```
open_system sltestHeatpumpLibraryLinkExample
```

Move the test harness.

```
sltest.harness.move('sltestHeatpumpLibraryLinkExample/Controller', ...  
'Baseline_controller_tests')
```

Input Arguments

harnessOwner — Model or component name

string | double

Model or component handle or path, specified as a string or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

string

The name of the harness, specified as a string.

Example: 'harness_name'

See Also

sltest.harness.close | sltest.harness.create | sltest.harness.delete |
sltest.harness.find | sltest.harness.open

Introduced in R2016a

sltest.harness.open

Open test harness

Syntax

```
sltest.harness.open(harnessOwner, harnessName)
```

Description

`sltest.harness.open(harnessOwner, harnessName)` opens the harness `harnessName`, which is associated with the model or component `harnessOwner`.

Examples

Open a Harness Associated With a Subsystem

Open the test harness `controller_harness`, which is associated with the `Controller` subsystem in the `f14` model.

```
f14;  
sltest.harness.create('f14/Controller', 'Name', 'controller_harness');  
sltest.harness.open('f14/Controller', 'controller_harness');
```

Open a Harness Associated With a Model

Open the test harness `sample_harness`, which is associated with the `f14` model.

```
f14;  
sltest.harness.create('f14', 'Name', 'sample_harness');  
sltest.harness.open('f14', 'sample_harness');
```

Input Arguments

harnessOwner — Model or component

string | double

Model or component handle or path, specified as a string or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

string

The name of the harness, specified as a string.

Example: 'harness_name'

See Also

`sltest.harness.check` | `sltest.harness.close` | `sltest.harness.create` |
`sltest.harness.delete` | `sltest.harness.export` | `sltest.harness.find` |
`sltest.harness.load` | `sltest.harness.push` | `sltest.harness.rebuild` |
`sltest.harness.set`

Introduced in R2015a

sltest.harness.push

Push test harness workspace entries and configuration set to model

Syntax

```
sltest.harness.push(harnessOwner, harnessName)
```

Description

`sltest.harness.push(harnessOwner, harnessName)` pushes the configuration parameter set and workspace entries associated with the component under test from the test harness `harnessName` to the main model containing the model or component `harnessOwner`.

Examples

Push Parameters from Harness to Model

Push the parameters of the harness `controller_harness`, which is associated with the Controller subsystem in the `f14` model, to the `f14` model.

```
f14;  
sltest.harness.create('f14/Controller', 'Name', 'controller_harness');  
sltest.harness.push('f14/Controller', 'controller_harness')
```

Input Arguments

harnessOwner — Model or component

string | double

Model or component handle, or path, specified as a string or double

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — **Harness name**

verbatim string

The name of the harness, specified as a string.

Example: 'harness_name'

See Also

`sltest.harness.check` | `sltest.harness.close` | `sltest.harness.create` |
`sltest.harness.delete` | `sltest.harness.export` | `sltest.harness.find` |
`sltest.harness.load` | `sltest.harness.open` | `sltest.harness.rebuild` |
`sltest.harness.set`

Introduced in R2015a

sltest.harness.rebuild

Rebuild test harness and update workspace entries and configuration parameter set based on main model

Syntax

```
sltest.harness.rebuild(harnessOwner, harnessName)
```

Description

`sltest.harness.rebuild(harnessOwner, harnessName)` rebuilds the test harness `harnessName` based on the main model containing `harnessOwner`. The function transfers the configuration set and workspace entries associated with `harnessOwner` to the test harness `harnessName`. The function also rebuilds conversion subsystems in the test harness.

Examples

Rebuild Parameters from Harness to Model

Rebuild the harness `controller_harness`, which is associated with the `Controller` subsystem in the `f14` model.

```
f14;  
sltest.harness.create('f14/Controller', 'Name', 'controller_harness');  
sltest.harness.rebuild('f14/Controller', 'controller_harness');
```

Input Arguments

harnessOwner — Model or component

string | double

Model or component handle, or path, specified as a string or double

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

verbatim string

The name of the harness, specified as a string.

Example: 'harness_name'

See Also

`sltest.harness.check` | `sltest.harness.close` | `sltest.harness.create`
| `sltest.harness.delete` | `sltest.harness.export` | `sltest.harness.find`
| `sltest.harness.load` | `sltest.harness.open` | `sltest.harness.push` |
`sltest.harness.set`

Introduced in R2015a

sltest.harness.set

Change test harness property

Syntax

```
sltest.harness.set(harnessOwner, harnessName, Name, Value)
```

Description

`sltest.harness.set(harnessOwner, harnessName, Name, Value)` changes a property, specified by one `Name, Value` pair argument, for the test harness `harnessName` owned by the model or component `harnessOwner`.

Examples

Change a test harness name

Change the name of the test harness `controller_harness`, which is associated with the `Controller` subsystem in the `f14` model. The new name is `new_name`.

```
f14;  
sltest.harness.create('f14/Controller', 'Name', 'controller_harness');  
sltest.harness.set('f14/Controller', 'controller_harness', 'Name', 'new_name')
```

Enable component editing in the test harness

Set the test harness `controller_harness` to allow editing of the component under test.

```
f14;  
sltest.harness.create('f14/Controller', 'Name', 'controller_harness');  
sltest.harness.set('f14/Controller', 'controller_harness', ...  
'EnableComponentEditing', true);
```

Input Arguments

harnessOwner — Model or component

string | double

Model or component handle, or path, specified as a string or double

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

string

The name of the harness, specified as a string.

Example: 'harness_name'

Name-Value Pair Options

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 (' '). You can specify several name and value pair arguments in any order as **Name1**, **Value1**, ..., **NameN**, **ValueN**.

Example: 'Name', 'updated_harness' specifies a new harness name 'updated_harness'.

'Name' — New harness name

string

The new name for the harness, specified as the comma-separated pair consisting of 'Name' and a valid MATLAB filename.

Example:

'Name', 'new_harness_name'

'Description' — New harness description

string

The new description for the harness, specified by the comma-separated pair consisting of 'Description' and a string.

Example:

'Description', 'An updated test harness'

'EnableComponentEditing' — Set the option for component editing`false | true`

An option to enable or disable component editing in the harness, specified by a comma-separated pair consisting of 'EnableComponentEditing' and false or true.

Example:

`'EnableComponentEditing',true`**'RebuildOnOpen' — Sets the harness rebuild command to execute when the harness opens**`false (default) | true`

Option to have the harness rebuild when it opens, specified as the comma-separated pair consisting of 'UseDefaultName' and false or true.

Example:

`'RebuildOnOpen',true`**'RebuildModelData' — Sets configuration set and model workspace entries to be updated during the test harness rebuild**`false (default) | true`

Option to have the configuration set and model workspace entries updated during test harness rebuild, specified as the comma-separated pair consisting of 'RebuildModelData' and true or false.

Example:

`'RebuildModelData',true`**'RebuildWithoutCompile' — Sets the harness to rebuild without compiling the main model**`false (default) | true`

Option to rebuild the harness without compiling the main model, in which cached information from the most recent compile is used to update the test harness workspace, and conversion subsystems are not updated, specified as the comma-separated pair consisting of 'RebuildWithoutCompile' and true or false.

Example:

`'RebuildWithoutCompile',true`

See Also

`sltest.harness.check` | `sltest.harness.close` | `sltest.harness.create`
| `sltest.harness.delete` | `sltest.harness.export` | `sltest.harness.find`
| `sltest.harness.load` | `sltest.harness.open` | `sltest.harness.push` |
`sltest.harness.rebuild`

Introduced in R2015a

sltest.import.sldvData

Create test cases from Simulink Design Verifier results

Syntax

```
[owner,testHarness,testFile] = sltest.import.sldvData(dataFile)
[owner,testHarness,testFile] = sltest.import.sldvData(dataFile,Name,
Value)
```

Description

[owner,testHarness,testFile] = sltest.import.sldvData(dataFile) creates a test harness and test file using Simulink Design Verifier™ analysis results contained in dataFile. The function returns as strings the model component owner associated with the test case, the testHarness, and the testFile.

[owner,testHarness,testFile] = sltest.import.sldvData(dataFile,Name, Value) uses additional options specified by one or more Name,Value pair arguments.

Examples

Create Test Cases for ShiftLogic Subsystem

Create a test file and test harness for the ShiftLogic subsystem in the sldvdemo_autotrans model. The inputs reflect the analysis objectives.

Analyze the ShiftLogic subsystem with Simulink Design Verifier to generate test inputs for subsystem coverage. The results data file is ShiftLogic_sldvdata.mat.

Create the test case.

```
[component,harness,testfile] = sltest.import.sldvData...
('./ShiftLogic/ShiftLogic_sldvdata.mat','TestHarnessName',...
'CoverageHarness','TestFileName','CoverageTests')
```

Open the test harness.

```
sldtest.harness.open(component,harness)
```

Open the test file.

```
open([testfile '.mldatx'])
```

Create Test Cases for ShiftLogic Subsystem Using an Existing Test Harness

Create a test file and test harness for the ShiftLogic subsystem in the `sldvdemo_autotrans` model, using an existing test harness.

Analyze the ShiftLogic subsystem with Simulink Design Verifier to generate test inputs for subsystem coverage. The results data file is `ShiftLogic_sldvdata.mat`. The existing test harness is named `DatafileHarness`.

Create the test case.

```
[component,harness,testfile] = sldtest.import.sldvData...  
( './sldv_output/ShiftLogic/ShiftLogic_sldvdata.mat',...  
'TestHarnessName','DatafileHarness','TestFileName','CoverageTests',...  
'CreateHarness',false)
```

Open the test harness.

```
sldtest.harness.open(component,harness)
```

Open the test file.

```
open([testfile '.mldatx'])
```

- “Test Models Using Inputs Generated by Simulink Design Verifier”

Input Arguments

dataFile — Path and file name

string

Path and file name of the data file generated by Simulink Design Verifier analysis, specified as a string.

Example: `'ShiftLogic0/ShiftLogic0_sldvdata.mat'`

Example: `'Controller_sldvdata.mat'`

Name-Value Pair Options

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 (' '). You can specify several name and value pair arguments in any order as `Name1, Value1, . . . , NameN, ValueN`.

Example: `'TestHarnessName', 'DatafileHarness', 'CreateHarness', false`

'CreateHarness' — Create a test harness for the model or subsystem

`true` (default) | `false`

Option to add a test harness to the model or model component, which corresponds to the test cases in the test file, specified as a comma-separated pair consisting of 'CreateHarness' and `true` or `false`.

If you specify `true`, use a new test harness name with the 'TestHarnessName' name-value pair.

If you specify `false`, use an existing test harness name with the 'TestHarnessName' name-value pair.

Note: If the model under analysis is a test harness, the `CreateHarness` default value is `false`.

Example:

`'CreateHarness', false`

'TestHarnessName' — Harness name

string

The test harness used for running the test cases, specified as the comma-separated pair consisting of 'TestHarnessName' and a valid MATLAB[®] file name.

Use a new test harness name if 'CreateHarness' is `true` and an existing test harness name if 'CreateHarness' is `false`.

Example:

`'TestHarnessName', 'ModelCoverageTestHarness'`

'TestFileName' — Test file name

string

The name for the test file created for the test cases, specified as the comma-separated pair consisting of 'TestFileName' and a valid MATLAB file name.

Example:

```
'TestFileName', 'ModelCoverageTests'
```

'ExtractedModelPath' — Path of extracted model

string

The path to the model extracted from Simulink Design Verifier analysis, specified as the comma-separated pair consisting of 'ExtractedModelPath' and a path name.

Simulink Test™ uses the extracted model to generate the test harness. By default, `sltest.import.sldvData` looks for the extracted model in the output folder specified in the Design Verifier configuration parameters. Use `ExtractedModelPath` if the extracted model is in a different location.

Simulink Design Verifier does not use an extracted model when you analyze a top-level model. When you generate test cases for a top-level model, Simulink Test does not use 'ExtractedModelPath'.

Example:

```
'Tests/ExtractedModels/'
```

Introduced in R2015b

sltest.testmanager.clear

Clear all test files from the Simulink Test manager

Syntax

```
sltest.testmanager.clear
```

Description

`sltest.testmanager.clear` clears all of the test files from the Simulink Test manager. Changes to unsaved test files are not saved.

Introduced in R2015a

sltest.testmanager.clearResults

Clear all results from test manager

Syntax

```
sltest.testmanager.clearResults
```

Description

`sltest.testmanager.clearResults` clears all of the results data from the test manager **Results and Artifacts** pane.

Examples

Clear Results From Test Manager

```
% Run test files in the test manager  
sltest.testmanager.run;
```

```
% Clear results from the test manager  
sltest.testmanager.clearResults
```

- “Automate Tests Programmatically”

See Also

```
sltest.testmanager.clear
```

Introduced in R2016a

sltest.testmanager.close

Close the Simulink Test manager

Syntax

```
sltest.testmanager.close
```

Description

`sltest.testmanager.close` closes the Simulink Test manager interface. Test files and results are retained in the test manager until the MATLAB session is closed.

Introduced in R2015a

sltest.testmanager.copyTests

Copy test cases or test suites to another location

Syntax

```
objArray = sltest.testmanager.copyTests(srcObjArray,targetObj)
```

Description

`objArray = sltest.testmanager.copyTests(srcObjArray,targetObj)` copies test cases or test suites to another test file or test suite.

Examples

Copy Test Cases to a New Test Suite

```
% Create test structure
tf = sltest.testmanager.TestFile('Test File');
ts_orig = tf.createTestSuite('Original Test Suite');
tc1 = ts_orig.createTestCase('baseline','Baseline Test Case 1');
tc2 = ts_orig.createTestCase('baseline','Baseline Test Case 2');

% Create new test suite for the target location
ts_new = tf.createTestSuite('New Test Suite');

% Copy test cases to new test suite
objArray = sltest.testmanager.copyTests([tc1,tc2],ts_new)

objArray =
```

1x2 TestCase array with properties:

```
    Name
  Description
   Enabled
ReasonForDisabling
   TestFile
   TestPath
```

```

    TestType
    Parent

% Look at the details of the object array
objArray(1)

ans =

    TestCase with properties:

        Name: 'Baseline Test Case 1'
    Description: ''
        Enabled: 1
    TestFile: [1x1 sltest.testmanager.TestFile]
    TestPath: 'Test File > New Test Suite > Baseline Test Case 1'
    TestType: 'baseline'
        Parent: [1x1 sltest.testmanager.TestSuite]

```

- “Automate Tests Programmatically”

Input Arguments

srcObjArray — Test case or test suites to copy

object array

Test cases or test suites to copy, specified as an array of `sltest.testmanager.TestCase` or `sltest.testmanager.TestSuite` objects.

targetObj — Target test file or test suite

object

The destination test file or test suite to copy to, specified as an `sltest.testmanager.TestFile` or `sltest.testmanager.TestSuite` object.

Output Arguments

objArray — Test cases or test suites at new location

object array

Test cases or test suites at the target destination location, returned as an array of `sltest.testmanager.TestCase` or `sltest.testmanager.TestSuite` objects.

See Also

`sltest.testmanager.moveTests`

Introduced in R2015b

sltest.testmanager.createTestsFromModel

Generate test cases from a model

Syntax

```
testFile = sltest.testmanager.createTestsFromModel(filePath,  
modelName,testType)
```

Description

`testFile = sltest.testmanager.createTestsFromModel(filePath, modelName, testType)` generates test cases based on the model structure. The function creates test cases from test harnesses and Signal Builder groups in the model and assigns them to a test file.

Examples

Create Test Cases From a Model

Create a new test file with new test cases in the test manager. Each test case uses a Signal Builder group scenario.

```
testFile = sltest.testmanager.createTestsFromModel('C:\MATLAB\TestFile.mldatx',...  
          'sldemo_autotrans','baseline')
```

```
testFile =
```

```
TestFile with properties:
```

```
    Name: 'TestFile'  
  FilePath: 'C:\MATLAB\TestFile.mldatx'  
    Dirty: 0
```

- “Automate Tests Programmatically”

Input Arguments

filePath — Test file name and path

string

The path and name of the test file to save the generated test cases to, specified as a string.

Example: 'C:\MATLAB\TestFile.mldatx'

modelName — Model name

string

The name of the model to generate test cases from, specified as a string.

Example: 'sldemo_autotrans'

testType — Test case type

'baseline' (default) | 'simulation' | 'equivalence'

The type of test cases to generate, specified as a string. The function creates test cases using this test case type.

Output Arguments

testFile — Test file object

object

Test file that contains the generated test cases, returned as an `sltest.testmanager.TestFile` object.

Introduced in R2015b

sltest.testmanager.exportResults

Export results set from test manager

Syntax

```
sltest.testmanager.exportResults(resultObjs, filePath)
```

Description

`sltest.testmanager.exportResults(resultObjs, filePath)` exports the specified results set objects to a `.mldatx` file.

Examples

Export Results Set From Test Manager

```
% Get the results set object from the test manager
result = sltest.testmanager.getResultSets

% Export the results set object to a file
sltest.testmanager.exportResults(Obj, 'C:\MATLAB\results.mldatx')
```

- “Automate Tests Programmatically”

Input Arguments

resultObjs — Results set array

object

Result sets, specified as an array of `sltest.testmanager.ResultSet` objects.

filePath — Results file path and name

string

The file path and name of the result file you want to output, specified as a string. The results file is a `.mldatx` file.

Example: 'C:\MATLAB\results.mldatx'

See Also

`sltest.testmanager.ResultSet` | `sltest.testmanager.getResultSets`

Introduced in R2016a

sltest.testmanager.getResultSets

Returns result set objects in test manager

Syntax

```
rsList = sltest.testmanager.getResultSets
```

Description

`rsList = sltest.testmanager.getResultSets` returns an array of result set objects, `sltest.testmanager.ResultSet`, from the results currently in the test manager **Results and Artifacts** pane.

Examples

Get Test Suite Result

If there is a test suite result in the test manager that you want to interact with programmatically, then you can use this function to get the result set object, `sltest.testmanager.ResultSet`.

```
rsList = sltest.testmanager.getResultSets;  
tsrList = getTestSuiteResults(rsList(1));
```

- “Automate Tests Programmatically”

Output Arguments

rsList — Result set array

array of objects

The results currently in the test manager **Results and Artifacts** pane, returned as an array of `sltest.testmanager.ResultSet` objects.

See Also

`sltest.testmanager.ResultSet` | `sltest.testmanager.view`

Introduced in R2016a

sltest.testmanager.importResults

Import test manager results file

Syntax

```
resultObjs = sltest.testmanager.importResults(filePath)
```

Description

`resultObjs = sltest.testmanager.importResults(filePath)` imports a results set file (.mldatx) into the test manager.

Examples

Import Results and Generate Report

```
% Import results set from a file
result = sltest.testmanager.importResults('testResults.mldatx');

% Set a filepath and filename for the report
filePath = 'testreport.zip';

% Generate the report
sltest.testmanager.report(result,filePath,...
    'Author','User',...
    'Title','Test',...
    'IncludeMLVersion',true,...
    'IncludeTestResults',int32(0),...
    'LaunchReport', true);
```

- “Automate Tests Programmatically”

Input Arguments

filePath — File name and path of results set
string

File name and path of results set, specified as a string.

Example: `'testResults.mldatx'`

Output Arguments

resultObjs — Results set

object

Results set, returned as an array of `sltest.testmanager.ResultSet` objects.

See Also

`sltest.testmanager.ResultSet` | `sltest.testmanager.report`

Introduced in R2016a

sltest.testmanager.load

Load a test file in the Simulink Test manager

Syntax

```
tfObj = sltest.testmanager.load(filename)
```

Description

`tfObj = sltest.testmanager.load(filename)` loads a test file in the Simulink Test manager.

Examples

- “Automate Tests Programmatically”

Input Arguments

filename — File name of test file

string

File name of a test file, specified as a string. The string must fully specify the location of the test file.

Output Arguments

tfObj — Test file object

object

Test file, returned as an `sltest.testmanager.TestFile` object.

See Also

`sltest.testmanager.run`

Introduced in R2015a

sltest.testmanager.moveTests

Move test cases or test suites to a new location

Syntax

```
objArray = sltest.testmanager.moveTests(srcObjArray, targetObj)
```

Description

objArray = sltest.testmanager.moveTests(srcObjArray, targetObj) moves test cases or test suites to another test file or test suite.

Examples

Move Test Suite to a New Test File

```
% Create test structure
tf1 = sltest.testmanager.TestFile('Test File 1');
ts = tf1.createTestSuite('Test Suite');

% Create new test file
tf2 = sltest.testmanager.TestFile('Test File 2');

% Move test suite to Test File 2
objArray = sltest.testmanager.moveTests(ts, tf2)

objArray =

    TestSuite with properties:

        Name: 'Test Suite'
    Description: ''
        Enabled: 1
    TestFile: [1x1 sltest.testmanager.TestFile]
    TestPath: 'Test File 2 > Test Suite'
    Parent: [1x1 sltest.testmanager.TestFile]
```

- “Automate Tests Programmatically”

Input Arguments

srcObjArray — Test case or test suites to move

object array

Test cases or test suites to move, specified as an array of `sltest.testmanager.TestCase` or `sltest.testmanager.TestSuite` objects.

targetObj — Target test file or test suite

object

The destination test file or test suite to move to, specified as an `sltest.testmanager.TestFile` or `sltest.testmanager.TestSuite` object.

Output Arguments

objArray — Test cases or test suites at new location

object array

Test cases or test suites at the target destination location, returned as an array of `sltest.testmanager.TestCase` or `sltest.testmanager.TestSuite` objects.

See Also

`sltest.testmanager.copyTests`

Introduced in R2015b

sltest.testmanager.report

Generate report of test results

Syntax

```
sltest.testmanager.report(resultObj,filePath,Name,Value)
```

Description

`sltest.testmanager.report(resultObj,filePath,Name,Value)` generates a report of the specified results in `resultObj` and saves the report to the `filePath` location.

Examples

Generate a Test Report

Generate a report that includes the test author, test title, and the MATLAB version used to run the test case. The report includes only failed results.

```
filePath = 'test.pdf';  
sltest.testmanager.report(resultObj,filePath,...  
    'Author','TestAuthor',...  
    'Title','Test',...  
    'IncludeMLVersion',true,...  
    'IncludeTestResults',2);
```

- “Automate Tests Programmatically”

Input Arguments

resultObj — Results set object

object

Results set object to get results from, specified as an `sltest.testmanager.ResultSet` object.

filePath — File name and path of the generated report

string

File name and path of the generated report. File path must have file extension of pdf, docx, or zip, which are the only supported file types.

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 (' '). You can specify several name and value pair arguments in any order as `Name1`, `Value1`, ..., `NameN`, `ValueN`.

Example: 'IncludeTestRequirement', true

'Author' — Report author

empty string (default) | string

Name of the report author, specified as a string.

Example: 'Test Engineer'

'Title' — Report title

'Test' (default) | string

Title of the report, specified as a string.

Example: 'Test_Report_1'

'IncludeMLVersion' — Include the MATLAB version

true (default) | false

Choose to include the version of MATLAB used to run the test cases, specified as a Boolean value, `true` or `false`.

'IncludeTestRequirement' — Include the test requirement

true (default) | false

Choose to include the test requirement link defined under **Requirements** in the test case, specified as a Boolean value, `true` or `false`.

'IncludeSimulationSignalPlots' — Include the simulation output plots

false (default) | true

Choose to include the simulation output plots of each signal, specified as a Boolean value, `true` or `false`.

'IncludeComparisonSignalPlots' — Include the comparison plots

`false` (default) | `true`

Choose to include the signal comparison plots defined under baseline criteria, equivalence criteria, or assessments using the `verify` operator in the test case, specified as a Boolean value, `true` or `false`.

'IncludeErrorMessages' — Include error messages

`true` (default) | `false`

Choose to include any error messages from the test case simulations, specified as a Boolean value, `true` or `false`.

'IncludeTestResults' — Include all or subset of test results

2 (default) | 0 | 1

Choose to include all or a subset of test results in the report. You can select all results (passed and failed), specified as the integer value 0, select only passed results, specified as the value 1, or select only failed results, specified as the value 2.

'LaunchReport' — Open report at completion

`true` (default) | `false`

Open the report when it is finished generating, specified as a Boolean value, `true` or to not open the report, `false`.

'CustomTemplateFile' — Path to document template

string

Name and path for a Microsoft® Word template file to use for report generation, specified as a string. This is an optional argument that is only available if you have a MATLAB Report Generator™ license.

'CustomReportClass' — Class name for customized report

string

Name of the class used for report customization, specified as a string. This is an optional argument that is only available if you have a MATLAB Report Generator license.

'IncludeCoverageResult' — Include coverage result metrics

true (default) | false

Choose to include coverage metrics that are collected at test execution, specified as a Boolean value, `true` or `false`. For more information about collecting coverage, see “Collect Coverage in Tests”.

'IncludeSimulationMetadata' — Include simulation metadata

true (default) | false

Choose to include simulation metadata for each test case or iteration, specified as a Boolean value, `true` or `false`. The metadata includes: Simulink version, model version, model author, date, model user ID, model path, machine name, solver name, solver type, fixed step size, simulation start time, simulation stop time, and platform.

See Also

`sltest.testmanager.ResultSet`

Introduced in R2015a

sltest.testmanager.run

Run all test files in the Simulink Test manager

Syntax

```
resultObj = sltest.testmanager.run  
resultObj = sltest.testmanager.run('runWithPCT',value)
```

Description

`resultObj = sltest.testmanager.run` runs all of the test files in the Simulink Test manager. The function returns a `sltest.testmanager.ResultSet` object.

`resultObj = sltest.testmanager.run('runWithPCT',value)` runs all of the test files in the Simulink Test manager in parallel. The function returns a `sltest.testmanager.ResultSet` object. The function runs tests in parallel only if you have a license to Parallel Computing Toolbox™.

Examples

- “Automate Tests Programmatically”

Input Arguments

value — Enable parallel execution

false (default) | true

Enable parallel test execution, specified as `true` or `false`. The function runs tests in parallel only if you have a license to Parallel Computing Toolbox.

Example: `'runWithPCT', true`

Output Arguments

resultObj — Results set object

object

Results set object to get results from, returned as a `sltest.testmanager.ResultSet` object.

See Also

`sltest.testmanager.load`

Introduced in R2015a

sltest.testmanager.view

Launch the Simulink Test manager

Syntax

```
sltest.testmanager.view
```

Description

`sltest.testmanager.view` launches the Simulink Test manager interface. You can also use the function `sltestmgr` to launch the test manager.

Introduced in R2015a

sltest.testsequence.addStep

Add new test sequence step

Syntax

```
sltest.testsequence.addStep(blockPath,stepPath,Name,Value)
```

Description

`sltest.testsequence.addStep(blockPath,stepPath,Name,Value)` adds a step named `stepPath` to a `Test Sequence` block specified by `blockPath`. Step properties are specified by `Name,Value`.

Examples

Create a New Test Step

This example creates a test step in the Projector Fan Speed example test sequence under the parent step `SystemHeatingTest`.

Set paths and open the model.

```
Model = 'sltestProjectorFanSpeedExample';  
Harness = 'FanSpeedTestHarness';  
open_system(Model);
```

Open the test harness.

```
sltest.harness.open(Model,Harness);
```

Create a new local variable `h`.

```
sltest.testsequence.addSymbol('FanSpeedTestHarness/Test Sequence',...  
'h','Data','Local');
```

Create a step `substep1` under the step `SystemHeatingTest` and assign the value 5 to `h`.


```
sltest.testsequence.addStep('FanSpeedTestHarness/Test Sequence',...
'SystemHeatingTest.substep1', 'Label', 'h = 5')
```

Input Arguments

blockPath — Test Sequence block path

string

Path to a `Test Sequence` block, including block name, specified as a string.

Example: 'FanSpeedTestHarness/Test Sequence'

stepPath — New test step name and level in hierarchy

string

Path to the new step in the `Test Sequence` block, specified as a string. The new step must not already exist in the `Test Sequence` block. The path must include the step location in the `Test Sequence` hierarchy, using `.` to separate hierarchy levels. The new step is appended to the end of the hierarchy level.

Example: 'SystemHeatingTest.NewStep'

Name-Value Pair Options

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 (' '). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example:

'Label' — Test step actions

string

The test step action programming. To add a line, create the step actions using the `sprintf` function and the new line operator `\n`.

Example:

```
'Label', 'out = square(et)'
```

```
action = sprintf('mode = uint8(3)\nout = square(et)\n%%New step action')
```

'Label',action

'IsWhenStep' — Specifies standard or When decomposition step

false (default) | true

Specifies whether the step is a standard transition type or a When decomposition transition

Example:

'IsWhenStep',true

'WhenCondition' — When decomposition step switch condition

string

Specifies the condition that activates a when decomposition child step. This is specified as a string but must be a valid logical expression for the When step to activate.

Example:

'WhenCondition','a >= 1'

'Description' — Description for the test step

string

Test step description, specified as a string.

Example:

'This step produces a high-frequency square wave.'

See Also

sltest.testsequence.addStepAfter | sltest.testsequence.addStepBefore
| sltest.testsequence.addSymbol | sltest.testsequence.addTransition |
sltest.testsequence.editStep | sltest.testsequence.find

Introduced in R2016a

sltest.testsequence.addStepAfter

Add test sequence step after existing step

Syntax

```
sltest.testsequence.addStepAfter(blockPath,newStep,  
precedingStepPath,Name,Value)
```

Description

`sltest.testsequence.addStepAfter(blockPath,newStep,precedingStepPath,Name,Value)` adds a step to a **Test Sequence** block specified by `blockPath`. The new step is named `newStep` and is inserted immediately after the step `precedingStepPath`. Step properties are specified by `Name,Value`.

Examples

Create a New Test Step

This example creates a test step `step1` in a test sequence block after the step `SetLowPhi`, which is in the second level of hierarchy under the top-level step `APEngagement_AttitudeLevels`.

Set paths and open the model.

```
filePath = fullfile(matlabroot,'toolbox','simulinktest','simulinktestdemos');  
rollModel = 'RollAutopilotMdlRef';  
testHarness = 'RollReference_Requirement1_3';  
open_system(fullfile(filePath,rollModel));
```

Open the test harness.

```
sltest.harness.open([rollModel '/Roll Reference'],testHarness)
```

Create a new local variable `h`.

```
sltest.testsequence.addSymbol( 'RollReference_Requirement1_3/Test Sequence', ...  
'h', 'Data', 'Local' );
```

Add a step named `step2` and set the value of `h` to 5.

```
sltest.testsequence.addStepAfter( 'RollReference_Requirement1_3/Test Sequence', ...  
'AttitudeLevels.APEngage_LowRoll.step2', ...  
'AttitudeLevels.APEngage_LowRoll.SetLowPhi', 'Label', 'h = 5;' )
```

Input Arguments

blockPath — Test Sequence block path

string

Path to a `Test Sequence` block, including block name, specified as a string.

Example: 'TestHarness/Test Sequence'

newStep — New test step name or handle

string

Name of a new test step in the `Test Sequence` block, specified as a string. The new step must not already exist in the `Test Sequence` block. It will be added immediately after `precedingStepPath` and have the same parent step.

Example: 'newStep'

Example: 'topStep.midStep.newStep'

precedingStepPath — Preceding test step path

string

Path to an existing step in the `Test Sequence` block, specified as a string, after which `newStep` is inserted. The path must include the step location in the `Test Sequence` hierarchy, using `.` to separate levels of hierarchy.

Example: 'topStep.midStep.afterStep'

Name-Value Pair Options

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 (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example:

'Label' — Test step actions

string

The test step action programming. To add a line, create the step actions using the `printf` function and the new line operator `\n`.

Example:

```
'Label', 'out = square(et)'
```

```
action = sprintf('mode = uint8(3)\nout = square(et)\n%New step action')
'Label', action
```

'IsWhenStep' — Specifies standard or When decomposition step

false (default) | true

Specifies whether the step is a standard transition type or a When decomposition transition

Example:

```
'IsWhenStep', true
```

'WhenCondition' — When decomposition step switch condition

string

Specifies the condition that activates a when decomposition child step. This is specified as a string but must be a valid logical expression for the When step to activate.

Example:

```
'WhenCondition', 'a >= 1'
```

'Description' — Description for the test step

string

Test step description, specified as a string.

Example:

```
'This step produces a high-frequency square wave.'
```

See Also

`sltest.testsequence.addStep` | `sltest.testsequence.addStepBefore` |
`sltest.testsequence.addSymbol` | `sltest.testsequence.addTransition` |
`sltest.testsequence.editStep` | `sltest.testsequence.find`

Introduced in R2016a

sltest.testsequence.addStepBefore

Add test sequence step before existing step

Syntax

```
sltest.testsequence.addStepBefore(blockPath,newStep,  
followingStepPath,Name,Value)
```

Description

`sltest.testsequence.addStepBefore(blockPath,newStep, followingStepPath,Name,Value)` adds a step to a **Test Sequence** block specified by `blockPath`. The new step is named `newStep` and inserted immediately before the step named `followingStepPath`. Step properties are specified by `Name,Value`.

Examples

Create a New Test Step

This example creates a test step `step1` in a test sequence block before the step `SetLowPhi`, which is in the second level of hierarchy under the top-level step `APEngagement_AttitudeLevels`.

Set paths and open the model.

```
filePath = fullfile(matlabroot,'toolbox','simulinktest','simulinktestdemos');  
rollModel = 'RollAutopilotMdlRef';  
testHarness = 'RollReference_Requirement1_3';  
open_system(fullfile(filePath,rollModel));
```

Open the test harness.

```
sltest.harness.open([rollModel '/Roll Reference'],testHarness)
```

Create a new local variable `h`.

```
sltest.testsequence.addSymbol('RollReference_Requirement1_3/Test Sequence',...
```

```
'h', 'Data', 'Local');
```

Add a step named `step1` and set the value of `h` to 5.

```
sltest.testsequence.addStepBefore('RollReference_Requirement1_3/Test Sequence',...  
'AttitudeLevels.APEngage_LowRoll.step1',...  
'AttitudeLevels.APEngage_LowRoll.SetLowPhi', 'Label', 'h = 5;')
```

Input Arguments

blockPath — Test Sequence block path

string

Path to a `Test Sequence` block, including block name, specified as a string.

Example: 'TestHarness/Test Sequence'

newStep — New test step path

string

Name of a new test step in the `Test Sequence` block, specified as a string. The new step must not already exist in the `Test Sequence` block. It will be added immediately after `followingStepPath` and have the same parent step.

Example: 'topStep.midStep.newStep'

followingStepPath — Path of test step following new step

string

Path to an existing step in the `Test Sequence` block, specified as a string, before which `newStep` is inserted. The path must include the step location in the `Test Sequence` hierarchy, using `.` to separate levels of hierarchy.

Example: 'topStep.midStep.afterStep'

Name-Value Pair Options

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 (' '). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example:

'Label' — Test step actions

string

The test step action programming. To add a line, create the step actions using the `printf` function and the new line operator `\n`.

Example:

```
'Label', 'out = square(et)'
```

```
action = sprintf('mode = uint8(3)\nout = square(et)\n%%New step action')
'Label', action
```

'IsWhenStep' — Specifies standard or When decomposition step

false (default) | true

Specifies whether the step is a standard transition type or a When decomposition transition

Example:

```
'IsWhenStep', true
```

'WhenCondition' — When decomposition step switch condition

string

Specifies the condition that activates a when decomposition child step. This is specified as a string but must be a valid logical expression for the When step to activate.

Example:

```
'WhenCondition', 'a >= 1'
```

'Description' — Description for the test step

string

Test step description, specified as a string.

Example:

```
'This step produces a high-frequency square wave.'
```

See Also

`sltest.testsequence.addStep` | `sltest.testsequence.addStepAfter` |
`sltest.testsequence.addSymbol` | `sltest.testsequence.addTransition` |
`sltest.testsequence.editStep` | `sltest.testsequence.find`

Introduced in R2016a

sltest.testsequence.addSymbol

Add new data symbol to test sequence

Syntax

```
sltest.testsequence.addSymbol(blockPath,name,kind,scope)
```

Description

`sltest.testsequence.addSymbol(blockPath,name,kind,scope)` adds a symbol name with properties specified by `scope` and `kind` to a **Test Sequence** block specified by the `blockPath`. The new symbol appears in the **Symbols** sidebar of the Test Sequence Editor.

Examples

Create a New Data Symbol

This example creates a parameter `theta` in the test sequence block.

Set paths and open the model.

```
filePath = fullfile(matlabroot,'toolbox','simulinktest','simulinktestdemos');  
rollModel = 'RollAutopilotMdlRef';  
testHarness = 'RollReference_Requirement1_3';  
open_system(fullfile(filePath,rollModel));
```

Open the test harness.

```
sltest.harness.open([rollModel '/Roll Reference'],testHarness)
```

Add a parameter.

```
sltest.testsequence.addSymbol('RollReference_Requirement1_3/Test Sequence',...
```

```
'theta', 'Data', 'Parameter')
```

Input Arguments

blockPath — Test Sequence block path

string

Path to a Test Sequence block, including block name, specified as a string.

Example: 'TestHarness/Test Sequence'

name — Name of new symbol

string

Name of the new symbol, specified as a string. This data symbol must not already exist in the Test Sequence block.

Example: 'theta'

kind — Data symbol type

'Data' | 'Message' | 'Function Call'

The scope defines how the data symbol operates in the block. It is specified as a string.

Example: 'Parameter'

scope — Data symbol scope

'Input' | 'Output' | 'Local' | 'Constant' | 'Parameter' | 'Data Store Memory'

The scope defines how the data symbol operates in the block. It is specified as a string.

Example: 'Parameter'

See Also

```
sltest.testsequence.addStep | sltest.testsequence.addStepAfter |  
sltest.testsequence.addStepBefore | sltest.testsequence.addTransition  
| sltest.testsequence.editStep | sltest.testsequence.find
```

Introduced in R2016a

sltest.testsequence.addTransition

Add new transition to test sequence step

Syntax

```
sltest.testsequence.addTransition(blockPath,fromStep,condition,  
toStep)
```

Description

`sltest.testsequence.addTransition(blockPath,fromStep,condition,toStep)` creates a test step transition in the Test Sequence block `blockPath`. The transition executes on `condition`, from the origin `fromStep`, to the destination `toStep`. `fromStep` and `toStep` must be at the same level of test step hierarchy.

Examples

Add a Transition to Stop the Test

This example adds a local error variable, then adds a transition to stop the test sequence if the error variable is 1.

Set paths and open the model.

```
filePath = fullfile(matlabroot,'toolbox','simulinktest','simulinktestdemos');  
rollModel = 'RollAutopilotMdlRef';  
testHarness = 'RollReference_Requirement1_3';  
open_system(fullfile(filePath,rollModel));
```

Open the test harness.

```
sltest.harness.open([rollModel '/Roll Reference'],testHarness)
```

Add a new local variable.

```
sltest.testsequence.addSymbol('RollReference_Requirement1_3/Test Sequence',...
```

```
'error', 'Data', 'Local')
```

Add a transition to the step `InitializeTest` which goes to the step `Stop` if error is 1.

```
sltest.testsequence.addTransition('RollReference_Requirement1_3/Test Sequence', 'Initial  
'error == 1', 'Stop')
```

Input Arguments

blockPath — Test Sequence block path

string

Path to a `Test Sequence` block, including block name, specified as a string.

Example: 'TestHarness/Test Sequence'

fromStep — Origination step path

string

Path of an existing step in the `Test Sequence` block, specified as a string, at which the transition originates. The path must include the step name and step hierarchy, using `.` to separate hierarchy levels.

Example: 'topStep.midStep.step1'

condition — Condition on which the transition executes

string

The condition on which the transition executes, specified as a string. Though specified as a string, it must be a valid logical expression for the transition to execute.

Example: 'theta == 0 && a == 1'

toStep — Destination step path

string

Path of an existing step in the `Test Sequence` block, specified as a string, which becomes the active step after the transition executes. The path must include the step name and step hierarchy, using `.` to separate hierarchy levels. This step must be at same level as `fromStep`.

Example: 'topStep.midStep.step2'

See Also

sltest.testsequence.addStep | sltest.testsequence.addStepAfter |
sltest.testsequence.addStepBefore | sltest.testsequence.addSymbol |
sltest.testsequence.editStep | sltest.testsequence.find

Introduced in R2016a

sltest.testsequence.editStep

Edit existing test sequence step

Syntax

```
sltest.testsequence.editStep(blockPath,stepPath,Name,Value)
```

Description

`sltest.testsequence.editStep(blockPath,stepPath,Name,Value)` edits the properties of an existing step specified by `stepPath` in a `Test Sequence` block specified by `blockPath`. Changes to the properties are specified by `Name,Value`.

Examples

Add and Edit a Test Step

This example adds a test step then edits the step actions of the new step.

Open the model and test harness.

```
open_system('sltestTestSequenceWhenExample')
sltest.harness.open('sltestTestSequenceWhenExample/SimpleTracker','SimpleTrackerHarness')
```

Add a test step named `SquareAndVeryQuick`.

```
sltest.testsequence.addStep('SimpleTrackerHarness/Test Sequence','Square.SquareAndVeryQuick')
```

Edit the step actions.

```
action = sprintf('mode = uint8(3);\nout = square(et);\n%% New step action')
action =

mode = uint8(3);
out = square(et);
% New step action
```



```
sltest.testsequence.editStep('SimpleTrackerHarness/Test Sequence',...
'Square.SquareAndVeryQuick','Label',action,'Description',...
'This step outputs a high-frequency square wave.')
```

Add two substeps to the new step.

```
sltest.testsequence.addStep('SimpleTrackerHarness/Test Sequence',...
'Square.SquareAndVeryQuick.Step1')
sltest.testsequence.addStep('SimpleTrackerHarness/Test Sequence',...
'Square.SquareAndVeryQuick.Step2')
```

Change the parent step to a When decomposition.

```
sltest.testsequence.editStep('SimpleTrackerHarness/Test Sequence',...
'Square.SquareAndVeryQuick','IsWhenStep',true)
```

Add a When condition to the substep Step1.

```
sltest.testsequence.editStep('SimpleTrackerHarness/Test Sequence',...
'Square.SquareAndVeryQuick.Step1','WhenCondition','a >= 1')
```

Input Arguments

blockPath — Test Sequence block path

string

Path to a Test Sequence block, including block name, specified as a string.

Example: 'TestHarness/Test Sequence'

stepPath — Path of test step to edit

string

Path to the step to edit, specified as a string. The step must already exist in the Test Sequence block. The path includes the step name and location in the Test Sequence hierarchy, using `.` to separate levels of hierarchy.

Example: 'SystemHeatingTest.NewStep'

Name-Value Pair Options

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 (' '). You can specify several name and value pair arguments in any order as Name1, Value1, . . . , NameN, ValueN.

Example:

'Name' — New test step name

string

The new name for the test step, specified as a string.

Example:

```
'Name', 'HoldOutput'
```

'Label' — Test step actions

string

The test step action programming. To add a line, create the step actions using the `sprintf` function and the new line operator `\n`.

Example:

```
'Label', 'out = square(et)'
```

```
action = sprintf('mode = uint8(3)\nout = square(et)\n%%New step action')  
'Label', action
```

'IsWhenStep' — Specifies standard or When decomposition step

false (default) | true

Specifies whether the step is a standard transition type or a When decomposition transition

Example:

```
'IsWhenStep', true
```

'WhenCondition' — When decomposition step switch condition

string

Specifies the condition that activates a when decomposition child step. This is specified as a string but must be a valid logical expression for the When step to activate.

Example:

```
'WhenCondition', 'a >= 1'
```

'Description' — Description for the test step

string

Test step description, specified as a string.

Example:

```
'This step produces a high-frequency square wave.'
```

See Also

```
sltest.testsequence.addStep | sltest.testsequence.addStepAfter |  
sltest.testsequence.addStepBefore | sltest.testsequence.addSymbol |  
sltest.testsequence.addTransition | sltest.testsequence.find
```

Introduced in R2016a

sltest.testsequence.find

Find Test Sequence blocks

Syntax

```
blocks = sltest.testsequence.find
```

Description

`blocks = sltest.testsequence.find` returns a cell array of strings `blocks` listing all Test Sequence blocks in open models.

Examples

Find Open Test Sequence Blocks

This example opens a test harness and finds the paths to the two Test Sequence blocks contained in the test harness.

Open the model and test harness.

```
open_system('sltestTestSequenceWhenExample')
sltest.harness.open('sltestTestSequenceWhenExample/SimpleTracker',...
'SimpleTrackerHarness')
```

Find the Test Sequence blocks. One of the blocks is used specifically for Test Assessment.

```
blocks = sltest.testsequence.find
```

```
blocks =
```

```
    'SimpleTrackerHarness/Tes...'    'SimpleTrackerHarness/Tes...'
```

See Also

```
sltest.testsequence.addStep | sltest.testsequence.addStepAfter |
sltest.testsequence.addStepBefore | sltest.testsequence.addSymbol |
sltest.testsequence.addTransition | sltest.testsequence.editStep
```

Introduced in R2016a

sltestiteration

Create test iteration

Syntax

```
iterObj = sltestiteration
```

Description

`iterObj = sltestiteration` returns a test iteration object, `sltest.testmanager.TestIteration`. You can use the function in the MATLAB command window, or you can use it in the context of a scripted iteration under the **Iterations** section of a test case. For more information on creating test iterations, see “Run Multiple Combinations of Tests Using Iterations”.

Examples

Iterate Over Signal Builder Groups

This example is a script that must be entered in the Scripted Iterations script text box under the **Iterations** section of a test case. Also, the system under test for this example is a model that contains **Signal Builder** groups.

```
%% Iterate Over All Signal Builder Groups

% Determine the number of possible iterations
numSteps = length(sltest_signalBuilderGroups);

% Create each iteration
for k = 1 : numSteps
    % Set up a new iteration object
    testItr = sltestiteration;

    % Set iteration settings
    setTestParam(testItr, 'SignalBuilderGroup', sltest_signalBuilderGroups{k});
end
```

```
% Add the iteration to run in this test case
% You can pass in an optional iteration name
addIteration(sltest_testCase,testItr);
end
```

- “Run Multiple Combinations of Tests Using Iterations”
- “Automate Tests Programmatically”

Output Arguments

iterObj — Test iteration

sltest.testmanager.TestIteration object

Test iteration, returned as a sltest.testmanager.TestIteration object.

See Also

sltest.testmanager.TestIteration

Introduced in R2016a

Classes — Alphabetical List

sltest.testmanager.BaselineCriteria class

Package: sltest.testmanager

Add or modify baseline criteria

Description

Instances of `sltest.testmanager.BaselineCriteria` is a set of signals in a test case that determines the pass-fail criteria in a baseline test case.

Construction

`obj = sltest.testmanager.TestCase.addBaselineCriteria` creates a `sltest.testmanager.BaselineCriteria` object for a test case object.

Properties

FilePath — File path

string

File path of the baseline criteria set, returned as a string.

AbsTo1 — Absolute tolerance value

scalar

Value of the absolute tolerance at a baseline criteria file level, specified as a scalar value.

RelTo1 — Relative tolerance value

scalar

Value of the relative tolerance at a baseline criteria file level, specified as a scalar value.

Active — Enabled indicator

0 | 1

Indicates if the baseline criteria is enabled, 0 if it is not enabled, and 1 if it is enabled.

Methods

See Also

`sltest.testmanager.TestCase`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

sltest.testmanager.CoverageSettings class

Package: sltest.testmanager

Modify coverage settings

Description

Instances of `sltest.testmanager.CoverageSettings` let you set the setting under the **Coverage Settings** section in a test file, test suite, or test case.

Construction

The `getCoverageSettings` methods for test file, test suite, and test case objects returns a `sltest.testmanager.CoverageSettings` object, which lets you access the coverage collection and metric settings.

Properties

RecordCoverage — Enable coverage collection

false (default) | true

Specify if the coverage collection is on or off, `false` for off and `true` for on.

Coverage collection is enabled or disabled in the test manager under the **Coverage Settings** section. This property controls the **Record coverage for system under test** check box. The **Record coverage for referenced models** check box must be changed manually in the test manager and not programmatically.

MetricSettings — Coverage metric setting selection

string

Selection of coverage settings that are enabled or disabled, specified as a string. For the set of possible strings, see the parameter info for `CovMetricSettings` in “Model Parameters”. Coverage metric settings can be modified only at a test-file level.

Coverage metrics are enabled or disabled in the test manager by selecting the check boxes in the **Coverage Settings** section.

Example: 'dw'

Examples

Enable MCDC and Signal Range Coverage Metrics

```
% Get coverage settings object from the test file
cov = getCoverageSettings(testfile);
cov.RecordCoverage = true;
```

```
% Enable MCDC and signal range coverage metrics
cov.MetricSettings = 'mr';
```

- “Collect Coverage in Tests”
- “Automate Tests Programmatically”

See Also

sltest.testmanager.TestFile | sltest.testmanager.TestSuite |
sltest.testmanager.TestCase

Introduced in R2016a

sltest.testmanager.EquivalenceCriteria class

Package: sltest.testmanager

Add or modify equivalence criteria

Description

Instances of `sltest.testmanager.EquivalenceCriteria` is a set of signals in a test case that determines the pass-fail criteria in an equivalence test case.

Construction

`obj = sltest.testmanager.TestCase.captureEquivalenceCriteria` creates a `sltest.testmanager.EquivalenceCriteria` object for a test case object.

Properties

Enabled — Enabled indicator

0 | 1

Indicates if the equivalence criteria is enabled, 0 if it is not enabled, and 1 if it is enabled.

Methods

See Also

`sltest.testmanager.TestCase`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

sltest.testmanager.ParameterOverride class

Package: sltest.testmanager

Add or modify parameter override

Description

Instances of `sltest.testmanager.ParameterOverride` are parameters overrides contained in a parameter set within a test case that can override model parameters.

Construction

`obj = sltest.testmanager.ParameterSet.AddParameterOverride` creates a `sltest.testmanager.ParameterOverride` object for a parameter set object.

Properties

Name — Parameter override name

string

Name of the parameter override, specified as a string.

Value — Override value

scalar | vector

Value of the parameter override, specified as a scalar or vector value.

Enabled — Enabled indicator

0 | 1

Indicates if the parameter override is enabled, 0 if it is not enabled, and 1 if it is enabled.

Source — Parameter override source

string

The source of the parameter variable, returned as a string. For example, the source could be the base workspace.

Methods

See Also

`sltest.testmanager.ParameterSet` | `sltest.testmanager.TestCase`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

sltest.testmanager.ParameterSet class

Package: sltest.testmanager

Add or modify parameter set

Description

Instances of `sltest.testmanager.ParameterSet` are sets of parameters in a test case that can override model parameters.

Construction

`obj = sltest.testmanager.TestCase.addParameterSet` creates a `sltest.testmanager.ParameterSet` object for a test case object.

Properties

Name — Parameter set name

string

Name of the parameter set, specified as a string.

FilePath — File path

string

File path of the parameter set if parameters were added from a file, returned as a string.

Enable — Enabled indicator

0 | 1

Indicates if the parameter set is enabled, 0 if it is not enabled, and 1 if it is enabled.

Methods

See Also

`sltest.testmanager.TestCase`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

sltest.testmanager.ResultSet class

Package: sltest.testmanager

Access results set data

Description

Instances of `sltest.testmanager.ResultSet` enable you to access the results from test execution performed by the test manager.

Construction

The function `sltest.testmanager.run` creates a `sltest.testmanager.ResultSet` object.

Properties

NumPassed — Number of passed tests

integer

The number of passed tests contained in the results set.

NumFailed — Number of failed tests

integer

The number of failed tests contained in the results set.

NumDisabled — Number of disabled tests

integer

The number of test cases that were disabled in the results set.

NumTotal — Total number of tests

integer

The total number of tests in the results set.

NumTestCaseResults — Number of test case result children

integer

The number of test case results that are direct children of the results set object.

NumTestSuiteResults — Number of test suite result children

integer

The number of test suite results that are direct children of the results set object.

NumTestFileResults — Number of test file result children

integer

The number of test file results that are direct children of the results set object.

Methods

Examples

Get Test Result Set Data

Get results from running a test file with `sltest.testmanager.run`.

```
result = sltest.testmanager.run;
testCaseResultArray = result.getTestCaseResults;
testSuiteResultArray = result.getTestSuiteResults;
```

- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestFileResult` | `sltest.testmanager.TestSuiteResult` | `sltest.testmanager.TestCaseResult`

Introduced in R2015a

sltest.testmanager.SignalCriteria class

Package: sltest.testmanager

Add or modify signal criteria

Description

Instances of `sltest.testmanager.SignalCriteria` is an individual signal in a criteria set in a test case that determines the pass-fail criteria.

Construction

`obj = getAllSignalCriteria` creates a `sltest.testmanager.SignalCriteria` object for a test case object. This can be constructed from baseline or equivalence criteria.

Properties

Name — Signal name

string

Signal name, returned as a string.

AbsTo1 — Absolute tolerance value

scalar

Value of the absolute tolerance at a signal level, specified as a scalar value.

RelTo1 — Relative tolerance value

scalar

Value of the relative tolerance at a signal level, specified as a scalar value.

SyncMethod — Time synchronization method

'union' | 'intersection' | 'uniform'

The method of time synchronization used when a signal is compared to another signal, specified as a string. The method can be one of the following:

- `'union'` — Compare using a time vector that is the union of the time vectors of both timeseries. This method of time synchronization might require value interpolation.
- `'intersection'` — Compare using a time vector that is the intersection of the time vectors of both timeseries. This method of time synchronization does not require value interpolation because only time points common to both time series are considered.
- `'uniform'` — Compare using a time vector that is constructed using uniform time steps. This method of time synchronization requires value interpolation in both timeseries.

InterpMethod — Interpolation method

`'zoh'` | `'linear'`

The method of interpolation used to align signal data, specified as a string. The method can be one of the following:

- `'zoh'` — Zero-order hold. Data values are interpolated by holding their value at the previous time point.
- `'linear'` — Interpolated data values are determined by taking the data values at the previous and next time points. These two points form the linear interpolant, which becomes a straight line between these points. The interpolated data value is the point at which the linear interpolant and time point meet.

Enabled — Enabled indicator

0 | 1

Indicates if the signal criteria is enabled, 0 if it is not enabled, and 1 if it is enabled.

DataSource — Signal data source

string

Signal data source, returned as a string.

SID — Signal identifier

string

Signal identifier, returned as a string.

BlockPath — Signal block path

string

Signal block path, returned as a string.

Methods

See Also

sltest.testmanager.BaselineCriteria | sltest.testmanager.EquivalenceCriteria |
sltest.testmanager.TestCase

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

sltest.testmanager.TestCase class

Package: sltest.testmanager

Create or modify test case

Description

Instances of `sltest.testmanager.TestCase` are test case objects.

If you want to modify the test case settings that define how the test case executes, use the methods `sltest.testmanager.TestCase.setProperty` and `sltest.testmanager.TestCase.getProperty`.

Construction

`obj = sltest.testmanager.TestCase(parent, type, name)` creates a `sltest.testmanager.TestCase` object as a child of the specified parent. You can specify the name of the test case and the test case type: baseline, equivalence, or simulation.

Input Arguments

parent — Parent test suite

object

Parent test suite for the test case to reside in, specified as an `sltest.testmanager.TestSuite` object.

type — Test case type

'baseline' (default) | 'equivalence' | 'simulation'

Test case type, specified as a string. There are three test case types: baseline, equivalence, and simulation.

name — Test case name

string

Name of the test suite, specified as a string. If this is empty, a unique name is created.

runOnTarget — Run simulation on target

cell array of Booleans

Specify if you want to run the test case simulation on a target, specified as a cell array of Booleans. This is an optional argument. For more information on real-time testing, see “Test Models in Real Time”.

Properties

Name — Test case name

string

Name of the test case, returned as a string.

Description — Test case description

string

Test case description text, specified as a string.

Enabled — Test execution indicator

true | false

Indicates if the test case will execute, specified as a logical value `true` or `false`.

ReasonForDisabling — Disabled description

string

Description text for why the test file was disabled, specified as a string. This property is visible only when the `Enabled` property is set to `false`.

TestType — Test case type

string

The test case type, returned as a string. The test case type can be one of the three types: `simulation`, `equivalence`, or `baseline`.

TestFile — Parent test file

object

Test file that is the parent of the test case, returned as an `sltest.testmanager.TestFile` object.

TestPath — Test hierarchy

string

Test file, test suite, and test case hierarchy, returned as a string.

Parent — Parent object

object

Test suite that is the parent of the specified test case, returned as an `sltest.testmanager.TestSuite` object.

Requirements — Test file requirements

structure array

The requirements that are attached at the test-file level, returned as a structure.

RunOnTarget — Target indicator

cell array

Indicates if the simulation runs on a target, returned as a cell array of Booleans.

Methods

Examples

Create New Test File, Test Suite, and Test Case

```
% Create test file
testfile = sltest.testmanager.TestFile('C:\MATLAB\test_file.mldatx');

% Create test suite
testsuite = sltest.testmanager.TestSuite(testfile, 'My Test Suite');

% Create test case
testcase = sltest.testmanager.TestCase(testsuite, 'equivalence', ...
    'Equivalence Test Case')

testcase =
```

TestCase with properties:

```
    Name: 'Equivalence Test Case'  
    TestFile: [1x1 sltest.testmanager.TestFile]  
    TestPath: 'test_file > My Test Suite > Equivalence Test Case'  
    TestType: 'equivalence'  
    RunOnTarget: {2x1 cell}  
        Parent: [1x1 sltest.testmanager.TestSuite]  
    Requirements: [0x1 struct]  
    Description: ''  
    Enabled: 1
```

- “Automate Tests Programmatically”

See Also

[sltest.testmanager.TestFile](#) | [sltest.testmanager.TestSuite](#)

Introduced in R2015b

sltest.testmanager.TestCaseResult class

Package: sltest.testmanager

Access test case results data

Description

Instances of `sltest.testmanager.TestCaseResult` enable you to access the results from test execution performed by the test manager at a test-case level.

Construction

The function `sltest.testmanager.run` creates a `sltest.testmanager.ResultSet` object, which contains the test case result object. You can also create a test case result directly if you use `sltest.testmanager.TestCase.run` to execute an individual test case.

Properties

NumPassedIterations — Number of passed tests

`integer`

The number of passed tests in an individual test case result, returned as an integer.

NumFailedIterations — Number of failed tests

`integer`

The number of failed tests in an individual test case result, returned as an integer.

NumDisabledIterations — Number of disabled tests

`integer`

The number of disabled tests in an individual test case result, returned as an integer.

NumIncompleteIterations — Number of incomplete tests

`integer`

The number of incomplete tests in an individual test case result, returned as an integer.

NumTotalIterations — Total number of tests

integer

The total number of tests in an individual test case result, returned as an integer.

Outcome — Outcome of test case result

0 | 1 | 2 | 3

The outcome of an individual test case result. The integer 0 means the test case was disabled, 1 means the test case execution was incomplete, 2 means the test case passed, and 3 means the test case failed.

TestFilePath — Test file path

string

The path of the test file used to create the test case result.

TestCasePath — Result hierarchy path

string

The hierarchy path in the parent result set.

TestCaseType — Type of test case

'Simulation' | 'Baseline' | 'Equivalence'

The type of test case from the three available test cases in the test manager: simulation, baseline, and equivalence.

RunOnTarget — Target indicator

cell array

Indicates if the simulation runs on a target, returned as a cell array of Booleans.

Methods

Examples

Get Test Case Result From Results Set

```
% Run test file in test manager and output results set
```

```
resultset = sltest.testmanager.run;

% Get the test file result object
tfr = getTestFileResults(resultset);

% Get the test suite result object
tsr = getTestSuiteResults(tfr);

% Get the test case result object
tcr = getTestCaseResults(tsr);
```

- “Automate Tests Programmatically”

See Also

sltest.testmanager.TestSuiteResult | sltest.testmanager.TestFileResult

Introduced in R2015a

sltest.testmanager.TestFile class

Package: sltest.testmanager

Create or modify test file

Description

Instances of `sltest.testmanager.TestFile` are files that can contain test suites and test cases.

Construction

`obj = sltest.testmanager.TestFile(filePath,mode)` creates a `sltest.testmanager.TestFile` object with a default test suite and test case as children of the test file. The default test case type is a baseline test case. If the test file exists in the test manager, then a new test file is not created.

Input Arguments

filePath — File name and path

string

The file name and path of the test file, specified as a string. The specified file name determines the name of the test file as seen in the test manager.

Example: 'C:\MATLAB\TestFile.mldatx'

mode — Override existing test files

'false' (default) | 'true'

Indicate if you want to override any test files with the same file name and path, specified as either 'true' or 'false'.

Properties

Description — Test file description

string

Test file description text, specified as a string.

Dirty — Unsaved changes indicator

0 | 1

Indicates if the test file has any unsaved changes, 0 if there are not any unsaved changes, and 1 if there are unsaved changes.

Enabled — Test file execution indicator

true | false

Indicates if test cases that are children of the test file will execute, specified as a logical value `true` or `false`.

FilePath — File path and name

string

File path and name of the test file, returned as a string.

Example: 'C:\MATLAB\test_file.mldatx'

Name — Test file name

string

Name of the test file without the file path and file extension, returned.

ReasonForDisabling — Disabled description

string

Description text for why the test file was disabled, specified as a string. This property is visible only when the `Enabled` property is set to `false`.

Requirements — Test file requirements

structure array

The requirements that are attached at the test-file level, returned as a structure.

Methods

Examples

Create New Test File

Create a new test file and return the test file object.

```
testfile = sltest.testmanager.TestFile('C:\MATLAB\test_file.mldatx')
```

```
testfile =
```

```
    TestFile with properties:
```

```
        Name: 'test_file'  
    FilePath: 'C:\MATLAB\test_file.mldatx'  
        Dirty: 1  
  Requirements: [0x1 struct]  
    Description: ''  
        Enabled: 1
```

- “Automate Tests Programmatically”

See Also

[sltest.testmanager.TestCase](#) | [sltest.testmanager.TestSuite](#)

Introduced in R2015b

sltest.testmanager.TestFileResult class

Package: sltest.testmanager

Access test file results data

Description

Instances of `sltest.testmanager.TestFileResult` enable you to access the results from test execution performed by the test manager at a test-file level.

Construction

The function `sltest.testmanager.run` creates a `sltest.testmanager.ResultSet` object, which contains the test file result object. You can also create a test file result directly if you use `sltest.testmanager.TestFile.run` to execute tests in an individual test file.

Properties

TestFileResultPath — Result hierarchy path

string

The hierarchy path in the result set.

TestFilePath — Test file path

string

The path of the test file used to create the result.

NumPassed — Number of passed tests

integer

The number of passed tests contained in the test file result.

NumFailed — Number of failed tests

integer

The number of failed tests contained in the test file result.

NumDisabled — Number of disabled tests

integer

The number of test cases that were disabled in the test file result.

NumIncomplete — Number of incomplete tests

integer

The number of test cases that did not execute in the test file result.

NumTotal — Total number of tests

integer

The total number of tests in the test file result.

NumTestCaseResults — Number of test case result children

integer

The number of test case results that are direct children of the test file result.

NumTestSuiteResults — Number of test suite result children

integer

The number of test suite results that are direct children of the test file result.

Methods

Examples

Get Test File Result From Results Set

```
% Run test file in test manager and output results set  
resultset = sltest.testmanager.run;
```

```
% Get the test file result object  
tfr = getTestFileResults(resultset);
```

- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestSuiteResult` | `sltest.testmanager.TestCaseResult`

Introduced in R2016a

sltest.testmanager.TestInput class

Package: sltest.testmanager

Add or modify test input

Description

Instances of `sltest.testmanager.TestInput` are sets of signal input data that can be mapped to override the inputs in the System Under Test.

Construction

`obj = sltest.testmanager.TestCase.addInput` creates a `sltest.testmanager.TestInput` object for a test case object.

Properties

Name — Test input name

string

Name of the test input, returned as a string.

FilePath — File path

string

File path of the test input, returned as a string.

SheetName — Spreadsheet name

string

If the input file is a Microsoft Excel[®] spreadsheet file, then this is the name of the spreadsheet from which the inputs are derived, specified as a string.

MappingStatus — Input mapping status

string

Mapping status to indicate if the inport mapping was successful. For more information about troubleshooting the mapping status, see “Understand Mapping Results”.

InputString — Input string

`string`

Evaluated during test case execution in the `LoadExternalInput` configuration parameter of the System Under Test, specified as a string.

Active — Enabled indicator

`0 | 1`

Indicates if the input is set to override in the test case, `0` if it is not enabled, and `1` if it is enabled.

Methods

See Also

`sltest.testmanager.TestCase`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

sltest.testmanager.TestIteration class

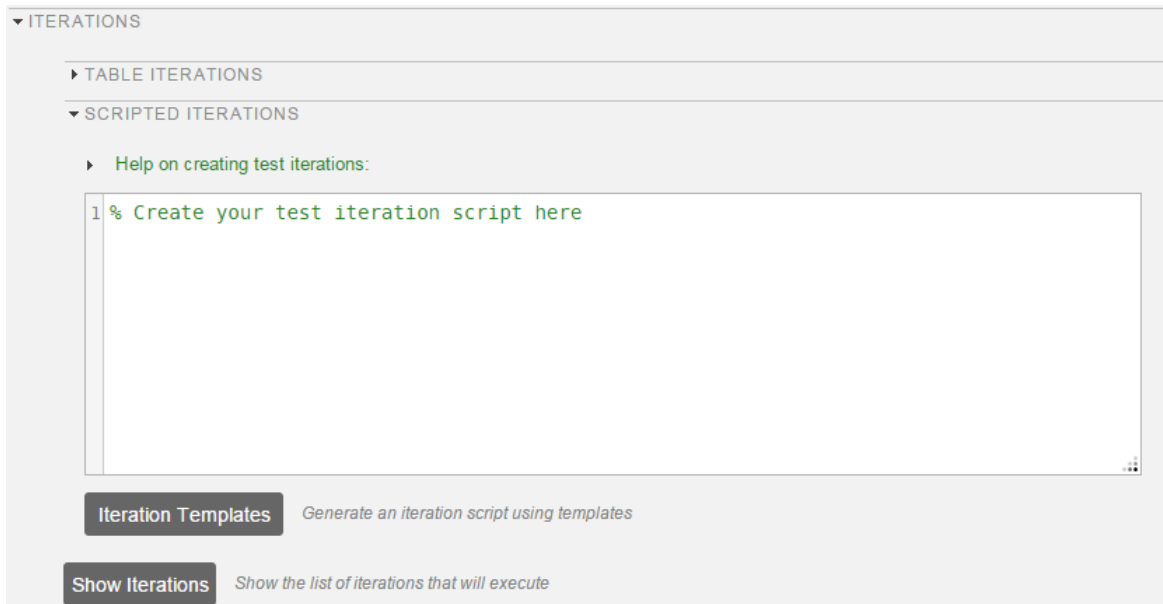
Package: sltest.testmanager

Create or modify test iteration

Description

Iterations let you test a combination of model settings for testing methods such as Monte Carlo and parameter sweeping. Iterations initialize during test execution but before any model callbacks and test callbacks. Once you create a test iteration object, you can override aspects of the test case for each iteration using the class methods.

You create your iteration script in the text window under the **Iterations** section of a test case. Iteration scripts will not run successfully in the MATLAB command window.



The screenshot shows the 'ITERATIONS' section of a test case editor. It contains a 'TABLE ITERATIONS' section and a 'SCRIPTED ITERATIONS' section. Under 'SCRIPTED ITERATIONS', there is a 'Help on creating test iterations:' link and a text area with the placeholder text '1 % Create your test iteration script here'. Below the text area are two buttons: 'Iteration Templates' with the tooltip 'Generate an iteration script using templates' and 'Show Iterations' with the tooltip 'Show the list of iterations that will execute'.

The examples scripts in this reference page must be inserted into this section and other sections of the test case must be defined. For more information on iterations and scripted iterations, see “Run Multiple Combinations of Tests Using Iterations”.

Construction

`iterationObj = sltest.testmanager.TestIteration` returns a test iteration object. The object is used to construct a single iteration in a test case. Each iteration you want to create in the test must use a single iteration object.

You can also create a test iteration within a iteration script using the `sltestiteration` function.

If you use a `for` loop in the MATLAB command window to add many iterations to a test case, then the MATLAB command window might become temporarily unusable. The recommended way to add iterations to a test case using the MATLAB command window is by vectorization. For example:

```
iterations(100) = sltest.testmanager.TestIteration;  
addIteration(tc,iterations);
```

Properties

Name — Iteration name

empty (default) | string

Name of the test iteration, specified as a string. The iteration name must be unique from other iterations in the test case.

Example: 'Iteration 1a'

ModelParams — Model parameter overrides

empty (default) | cell array

Set of model parameter overrides for the iteration, returned as a cell array of strings.

TestParams — Test parameter settings

empty (default) | cell array

Set of test parameter settings for the iteration, returned as a cell array of strings.

Variables — Model variable overrides

empty (default) | cell array

Set of model variable overrides for the iteration, returned as a cell array of strings.

Methods

Examples

Model Parameter Sweep

In this example of a scripted iteration, specify the model in the test case to be `sldemo_absbrake`. The iterations are generated during test execution. This section of script is in the **Scripted Iterations** section of the test case. It will execute only in the **Scripted Iterations** section. The `sltest_testCase` is a variable defined for you in the **Scripted Iterations** section, which is the parent test case object of the iteration.

```
% Specify the parameter sweep
vars = 32 : 0.5 : 34;

% Create iteration for each parameter using a loop
for k = 1 : length(vars)

    % Create test iteration object
    testItr = sltest.testmanager.TestIteration;

    % Set the parameter value for this iteration
    setVariable(testItr,'Name','g','Source','base workspace','Value',vars(k));

    str = sprintf('Iteration %d',k);

    % Add the iteration object to the test case
    addIteration(sltest_testCase,testItr,str);
end
```

Iterate Over Parameter Sets

In this example of a scripted iteration, there must be parameter sets defined in the **Parameter Overrides** section of the test case. The iterations are generated during test execution. This section of script is in the **Scripted Iterations** section of the test case. It will execute only in the **Scripted Iterations** section. The `sltest_testCase` is a variable defined for you in the **Scripted Iterations** section, which is the parent test case object of the iteration.

```
% Define parameter sets for a test case and add this code in the
```

```
% Scripted iterations section of the test case
for k = 1 : length(sltest_parameterSets)

    % Create test iteration object
    testItr = sltest.testmanager.TestIteration;

    % Use the parameter set in this iteration
    testItr.setTestParam('ParameterSet',sltest_parameterSets{k});

    str = sprintf('ParameterSet %d',k);

    % Add the iteration object to the test case
    addIteration(sltest_testCase,testItr,str);
end
```

- “Run Multiple Combinations of Tests Using Iterations”
- “Automate Tests Programmatically”

Alternatives

If you do not want to use a script to create iterations, then you can use table iterations in the test case. For more information about table iterations, see “Run Multiple Combinations of Tests Using Iterations”.

See Also

`sltest.testmanager.TestIterationResult`

Introduced in R2016a

sltest.testmanager.TestIterationResult class

Package: sltest.testmanager

Access test iteration result data

Description

Instances of `sltest.testmanager.TestIterationResult` enable you to access the results from test execution performed by the test manager at a test-iteration level. The hierarchy of test results is Result Set > Test File Result > Test Suite Result > Test Case Result > Test Iteration Result.

Construction

The function `sltest.testmanager.run` creates a `sltest.testmanager.ResultSet` object, which contains the test case result object.

Properties

Outcome — Outcome of test iteration result

0 | 1 | 2 | 3

The outcome of an individual test iteration result. The integer 0 means the test iteration was disabled, 1 means the test iteration execution was incomplete, 2 means the test iteration passed, and 3 means the test iteration failed.

TestFilePath — Test file path

string

The path of the test file used to create the test iteration result.

TestCasePath — Result hierarchy path

string

The hierarchy path in the parent result set.

TestCaseType — Type of test case

'Simulation' | 'Baseline' | 'Equivalence'

The type of test case from the three available test cases in the test manager: simulation, baseline, and equivalence.

RunOnTarget — Target indicator

cell array

Indicates if the simulation ran on the target or not, returned as an array of Booleans.

Methods

Examples

- “Run Multiple Combinations of Tests Using Iterations”
- “Automate Tests Programmatically”

See Also

sltest.testmanager.TestIteration

Introduced in R2016a

sltest.testmanager.TestResultReport class

Package: sltest.testmanager

Customize generated results report

Description

`sltest.testmanager.TestResultReport` is a class that enables you to customize report generation of result from the test manager. You can derive the class and override various methods to customize your report. By customizing the methods, you can change the report title, plots, tables, headers, icons, and more.

For more information and examples on customizing reports, see “Customize Generated Reports”.

Construction

`Obj = sltest.testmanager.TestResultReport(resultObjects, reportFilePath)` creates a report generation object.

To use this class, you must inherit from the class. Use the following code as the first lines in your class definition code to inherit from the class.

```
% class definition
classdef CustomReport < sltest.testmanager.TestResultReport
    %
    % Report customization code here
    %
end
```

Input Arguments

resultObjects — Results set object

object

Results set object to get results from, specified as an `sltest.testmanager.ResultSet` object.

reportFilePath — File name and path of the generated report

string

File name and path of the generated report. File path must have file extension of pdf, docx, or zip, which are the only supported file types.

Example: 'C:\MATLAB\Report.pdf'

Properties

AuthorName — Author name

empty string (default) | string

The name of the author or the generated report, specified as a string.

Example: 'Test Engineer'

BodyFontColor — Body paragraph font color

'Black' (default) | string

Body paragraph text font color, specified as a string.

Example: 'Red'

BodyFontName — Body paragraph font style name

'Arial' (default) | string

Body paragraph text font-style name, specified as a string.

Example: 'Times New Roman'

BodyFontSize — Body paragraph font size

'12pt' (default) | string

Body paragraph text font size, specified in points as a string.

Example: '14pt'

ChapterIndent — First level indentation width

'3mm' (default) | string

First level section indentation width, specified in millimeters as a string.

Example: '5mm'

ChapterIndentL2 — Second level indentation width

'6mm' (default) | string

Second level section indentation width, specified in millimeters as a string.

Example: '8mm'

ChapterIndentL3 — Third level indentation width

'8mm' (default) | string

Third level section indentation width, specified in millimeters as a string.

Example: '10mm'

CustomTemplateFile — Template file name and path

empty string (default) | string

The file name and path to a Microsoft Word template file for report customization, specified as a string. For more information about using template files, see “Generate Report Using Microsoft Word Template”. The use of this argument is valid only available if you have a MATLAB Report Generator license.

Example: 'C:\MATLAB\CustomReportTemplate.dotx'

HeadingFontColor — Section heading font color

'Black' (default) | string

Section heading text font color, specified as a string.

Example: 'Blue'

HeadingFontName — Section heading font style name

'Arial' (default) | string

Section heading text font-style name, specified as a string.

Example: 'Times New Roman'

HeadingFontSize — Section heading font size

'14pt' (default) | string

Section heading text font color, specified in points as a string.

Example: '16pt'

IconFileOutcomeDisabled — Disabled test result icon

empty string (default) | string

File name and path of an icon image for a disabled test result, specified as a string. The icon file specified replaces the default icon image. The icon image is reduced to 16x16 pixels.

Example: 'C:\MATLAB\disabled_test_icon.png'

IconFileOutcomeFailed — Failed test result icon

empty string (default) | string

File name and path of an icon image for a failed test result, specified as a string. The icon file specified replaces the default icon image. The icon image is reduced to 16x16 pixels.

Example: 'C:\MATLAB\failed_test_icon.png'

IconFileOutcomeIncomplete — Incomplete test result icon

empty string (default) | string

File name and path of an icon image for an incomplete test result, specified as a string. The icon file specified replaces the default icon image. The icon image is reduced to 16x16 pixels.

Example: 'C:\MATLAB\incomplete_test_icon.png'

IconFileOutcomeMisaligned — Misaligned test result icon

empty string (default) | string

File name and path of an icon image for a misaligned test result, specified as a string. The icon file specified replaces the default icon image. The icon image is reduced to 16x16 pixels.

Example: 'C:\MATLAB\misaligned_test_icon.png'

IconFileOutcomePassed — Passed test result icon

empty string (default) | string

File name and path of an icon image for a passed test result, specified as a string. The icon file specified replaces the default icon image. The icon image is reduced to 16x16 pixels.

Example: 'C:\MATLAB\passed_test_icon.png'

IconFileTestCaseResult — Test case result icon

empty string (default) | string

File name and path of an icon image for a test case result, specified as a string. The icon file specified replaces the default icon image. The icon image is reduced to 16x16 pixels.

Example: 'C:\MATLAB\test_case_result_icon.png'

IconFileTestFileResult — Test file result icon

empty string (default) | string

File name and path of an icon image for a test file result, specified as a string. The icon file specified replaces the default icon image. The icon image is reduced to 16x16 pixels.

Example: 'C:\MATLAB\test_file_result_icon.png'

IconFileTestIterationResult — Iteration result icon

empty string (default) | string

File name and path of an icon image for an iteration result, specified as a string. The icon file specified replaces the default icon image. The icon image is reduced to 16x16 pixels.

Example: 'C:\MATLAB\iteration_result_icon.png'

IconFileTestSuiteResult — Test suite result icon

empty string (default) | string

File name and path of an icon image for a test suite result, specified as a string. The icon file specified replaces the default icon image. The icon image is reduced to 16x16 pixels.

Example: 'C:\MATLAB\test_suite_result_icon.png'

IconModelReference — Model reference icon

empty string (default) | string

File name and path of an icon image for a model reference in the coverage report, specified as a string. The icon file specified replaces the default icon image. The icon image is reduced to 16x16 pixels.

Example: 'C:\MATLAB\model_reference_icon.png'

IconTopLevelModel — Top-level model icon

empty string (default) | string

File name and path of an icon image for a top-level model in the coverage report, specified as a string. The icon file specified replaces the default icon image. The icon image is reduced to 16x16 pixels.

Example: 'C:\MATLAB\top_level_model_icon.png'

IncludeComparisonSignalPlots — Include comparison signal plots

false (default) | true

Include the signal comparison plots in the report, specified as `true` or `false`.

IncludeCoverageResult — Include coverage results

false (default) | true

Include the coverage results in the report, specified as `true` or `false`.

IncludeErrorMessages — Include error messages

true (default) | false

Include error messages in the report, specified as `true` or `false`.

IncludeMWVersion — Include MATLAB version

true (default) | false

Include the version of MATLAB used to run the tests in the report, specified as `true` or `false`.

IncludeSimulationMetaData — Include simulation metadata

false (default) | true

Include simulation metadata in the report, specified as `true` or `false`. The metadata includes: Simulink version, model version, model author, date, model user ID, model path, machine name, solver name, solver type, fixed step size, simulation start time, simulation stop time, and platform.

IncludeSimulationSignalPlots — Include simulation signal plots

false (default) | true

Include the simulation signal output plots in the report, specified as `true` or `false`.

IncludeTestRequirement — Include test requirement

true (default) | false

Include the test requirements linked to the test file, test suite, or test case in the report, specified as `true` or `false`.

IncludeTestResults — Include all or subset of test results

2 (default) | 0 | 1

Include all or a subset of test results in the report. You can select all results (passed and failed), specified as the value 0, select only passed results, specified as the value 1, or select only failed results, specified as the value 2.

LaunchReport — Open report at completion

true (default) | false

Open the report when it is finished generating, specified as a boolean value, `true` or to not open the report, `false`.

ReportTitle — Report title

empty string (default) | string

Title of the report, specified as a string

Example: 'Test Case Report'

SectionSpacing — Spacing between sections

'2mm' (default) | string

Spacing between sections, specified in millimeters as a string.

Example: '5mm'

SignalPlotHeight — Plot height

'600px' (default) | string

Plot height, specified in pixels as a string.

Example: '500px'

SignalPlotWidth — Plot width

'500px' (default) | string

Plot width, specified in pixels as a string.

Example: '400px'

TableFontColor — Table font color

'Black' (default) | string

Table font color, specified as a string.

Example: 'Blue'

TableFontName — Table font style name

'Arial' (default) | string

Table font-style name, specified as a string.

Example: 'Times New Roman'

TableFontSize — Table font size

'7.5pt' (default) | string

Table font size, specified in points as a string.

Example: '10pt'

TitleFontColor — Title font color

'Black' (default) | string

Title font color, specified as a string.

Example: 'Blue'

TitleFontName — Title font style name

'Arial' (default) | string

Title font-style name, specified as a string.

Example: 'Times New Roman'

TitleFontSize — Title font size

'16pt' (default) | string

Title font size, specified in points as a string.

Example: '20pt'

Methods

Examples

Inherit Class and Customize a Report

```
% class definition
classdef CustomReport < sltest.testmanager.TestResultReport
```

```

% This custom class used by Test Manager adds a custom message in
% the title page

% Class constructor
methods
    function this = CustomReport(resultObjects, reportFilePath)
        this@sltest.testmanager.TestResultReport(resultObjects,reportFilePath);
    end
end

methods(Access=protected)
    function addTitlePage(obj)
        import mlreportgen.dom.*;

        % Add a custom message
        label = Text('Some custom content can be added here');
        append(obj.titlePart,label);

        % Call the superclass method to get the default behavior
        addTitlePage@sltest.testmanager.TestResultReport(obj);
    end
end
end

```

Use Custom Report Class to Generate Report

```

% import existing results or use sltest.testmanager.run to run tests
% and collect results
result = sltest.testmanager.importResults('testResults.mldatx');
filePath = 'testreport.zip';
sltest.testmanager.report(result,filePath,...
    'Author','User',...
    'Title','Test',...
    'IncludeMLVersion',true,...
    'IncludeTestResults',int32(0),...
    'CustomReportClass','CustomReport',...
    'LaunchReport',true);

```

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

sltest.testmanager.ResultSet | sltest.testmanager.report

Introduced in R2016a

sltest.testmanager.TestSuite class

Package: sltest.testmanager

Create or modify test suite

Description

Instances of `sltest.testmanager.TestSuite` can contain other test suites and test cases.

Construction

`obj = sltest.testmanager.TestSuite(parent, name)` creates a `sltest.testmanager.TestSuite` object as a child of the specified parent. You can use test files or other test suites as the parent.

Input Arguments

parent — Parent object
object

Parent object for the test suite to reside in. The parent object can be a test file or a test suite, specified as an `sltest.testmanager.TestFile` or `sltest.testmanager.TestSuite` object.

name — Test suite name
string

Name of the test suite, specified as a string. If this is empty, a unique name is created.

Properties

Name — Test suite name
string

Name of the test file without the file path and file extension, returned as a string.

Description — Test suite description

string

Test suite description text, specified as a string.

Enabled — Test suite execution indicator

true | false

Indicates if test cases that are children of the test suite will execute, specified as a logical value `true` or `false`.

ReasonForDisabling — Disabled description

string

Description text for why the test suite was disabled, specified as a string. This property is visible only when the `Enabled` property is set to `false`.

TestFile — Parent test file

object

Test file that is the parent of the test suite, returned as a `sltest.testmanager.TestFile` object.

TestPath — Test hierarchy

string

Test file and test suite hierarchy, returned as a string.

Parent — Parent object

object

Test file or test suite that is the parent of the specified test suite, returned as a `sltest.testmanager.TestFile` or `sltest.testmanager.TestSuite` object.

Requirements — Test suite requirements

structure array

The requirements that are attached at the test-suite level, returned as a structure.

Methods

Examples

Create New Test File and Suite

```
% Create test file
testfile = sltest.testmanager.TestFile('C:\MATLAB\test_file.mldatx');

% Create test suite
testsuite = sltest.testmanager.TestSuite(testfile, 'My Test Suite')

testsuite =
```

```
    TestSuite with properties:
```

```
        Name: 'My Test Suite'
      TestFile: [1x1 sltest.testmanager.TestFile]
     TestPath: 'test_file > My Test Suite'
        Parent: [1x1 sltest.testmanager.TestFile]
  Requirements: [0x1 struct]
   Description: ''
        Enabled: 1
```

- “Automate Tests Programmatically”

See Also

sltest.testmanager.TestCase | sltest.testmanager.TestFile

Introduced in R2015b

sltest.testmanager.TestSuiteResult class

Package: sltest.testmanager

Access test suite results data

Description

Instances of `sltest.testmanager.TestSuiteResult` enable you to access the results from test execution performed by the test manager at a test-suite level.

Construction

The function `sltest.testmanager.run` creates a `sltest.testmanager.ResultSet` object, which contains the test suite result object. You can also create a test suite result directly if you use `sltest.testmanager.TestSuite.run` to execute tests in an individual test suite.

Properties

TestSuitePath — Result hierarchy path

string

The hierarchy path in the parent result set.

TestFilePath — Test file path

string

The path of the test file used to create the result.

NumPassed — Number of passed tests

integer

The number of passed tests contained in the test suite result.

NumFailed — Number of failed tests

integer

The number of failed tests contained in the test suite result.

NumDisabled — Number of disabled tests

integer

The number of test cases that were disabled in the test suite result.

NumTotal — Total number of tests

integer

The total number of tests in the test suite result.

NumTestCaseResults — Number of test case result children

integer

The number of test case results that are direct children of the test suite result.

NumTestSuiteResults — Number of test suite result children

integer

The number of test suite results that are direct children of the test suite result.

Methods

Examples

Get Test Suite Result From Results Set

```
% Run test file in test manager and output results set  
resultset = sltest.testmanager.run;
```

```
% Get the test file result object  
tfr = getTestFileResults(resultset);
```

```
% Get the test suite result object  
tsr = getTestSuiteResults(tfr);
```

- “Automate Tests Programmatically”

See Also

sltest.testmanager.TestFileResult | sltest.testmanager.TestCaseResult

Introduced in R2015a

Methods — Alphabetical List

addBaselineCriteria

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Add baseline criteria to test case

Syntax

```
baseline = addBaselineCriteria(filePath,refreshIfExists)
```

Description

`baseline = addBaselineCriteria(filePath,refreshIfExists)` adds a baseline criteria set to the test case and returns a baseline criteria object, `sltest.testmanager.BaselineCriteria`. This function can be used only if the test type is a baseline test case.

Input Arguments

filePath — File name and path

string

Name and file path of the baseline criteria file, specified as a string.

refreshIfExists — Refresh signal criteria

false (default) | true

Refresh if the baseline criteria already exists, specified as a Boolean. The Boolean `false` errors if signal criteria is already loaded, and `true` refreshes signal criteria from the file data.

Output Arguments

baseline — Baseline criteria object

object

Baseline criteria added to the test case, returned as an `sltest.testmanager.BaselineCriteria` object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

addInput

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Add input file to test case

Syntax

```
input = addInput(filePath,simulationIndex)
```

Description

`input = addInput(filePath,simulationIndex)` adds a file to the Inputs section of the test case and returns a test input object, `sltest.testmanager.TestInput`.

Input Arguments

filePath — Input file name and path

string

Name and file path of the input file, specified as a string.

simulationIndex — Test case simulation number

1 | 2

Simulation number that the parameter sets apply to, specified as an integer, 1 or 2. This setting applies to the simulation test case where there are two simulations. For baseline and simulation test cases, the simulation index is 1.

Output Arguments

input — Test input object

object

Test input, returned as an `sltest.testmanager.TestInput` object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

addIteration

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Add test iteration to test case

Syntax

```
addIteration(tcObj, iterObj)  
addIteration(tcObj, iterObj, name)
```

Description

`addIteration(tcObj, iterObj)` adds a test iteration to the test case. The test manager gives the iteration a unique name.

`addIteration(tcObj, iterObj, name)` adds a test iteration to the test case with a specified name, which must be unique.

Input Arguments

tcObj — Test case to add iteration to
object

Test case that you want to add the iteration to, specified as a `sltest.testmanager.TestCase` object.

iterObj — Test iteration to add
object

Test iteration that you want to add to the test case, specified as a `sltest.testmanager.TestIteration` object.

name — Test iteration name
string

Test iteration name, specified as a string. The name must be unique with respect to other iterations in the test case. This is an optional argument.

Example: 'Test Iteration 5'

Examples

Iterate Over Parameter Sets

In this example, there must be parameter sets defined in the **Parameter Overrides** section of the test case. The iterations are generated during test execution. This section of script is in the Scripted Iterations section of the test case. It will execute only in the Scripted Iterations section.

```
% Define parameter sets for a test case and add this code in the
% Scripted iterations section of the test case
for k = 1 : length(sltest_parameterSets)

    % Create test iteration object
    testItr = sltest.testmanager.TestIteration();

    % Use the parameter set in this iteration
    testItr.setTestParam('ParameterSet', sltest_parameterSets{k});

    str = sprintf('ParameterSet %d', k);

    % Add the iteration object to the test case
    addIteration(sltest_testCase, testItr, str);
end
```

- “Run Multiple Combinations of Tests Using Iterations”
- “Automate Tests Programmatically”

See Also

sltest.testmanager.TestIteration

Introduced in R2016a

addParameterOverride

Class: sltest.testmanager.ParameterSet

Package: sltest.testmanager

Add parameter override to set

Syntax

```
ovr = addParameterOverride(Name,Value)
```

Description

`ovr = addParameterOverride(Name,Value)` adds a parameter override to parameter set and returns a parameter override object, `sltest.testmanager.ParameterOverride`.

Input Arguments

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 (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: 'Value',5.2

'Name' — Variable name

string

Name of the variable you want to override, specified as a string.

'Value' — Variable value

any value

Value of the variable you want to override.

'MaskedBlockPath' — Masked block path

string

If the parameter you want to override is contained in a masked block, then provide the block path, specified as a string.

Output Arguments

ovr — Parameter override object

object

Parameter override added to the parameter set, returned as an `sltest.testmanager.ParameterOverride` object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

addParameterSet

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Add parameter set

Syntax

```
pset = addParameterSet(Name, Value)
```

Description

`pset = addParameterSet(Name, Value)` adds a parameter set to the test case and returns a parameter set object, `sltest.testmanager.ParameterSet`.

Input Arguments

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 (' '). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `'SimulationIndex', 2`

'Name' — Parameter set name

auto-generated unique name (default) | string

Name of the parameter set, specified as a string. This name is the label shown in the test case parameter set table. If you do not specify a name, the function creates an auto-generated unique name.

'FilePath' — Parameter set name and file path

string

The full name and path of the MAT-file, which contains the parameter values, specified as a string. If no parameter file path is given, then the function creates an empty parameter set.

'SimulationIndex' — Simulation number

1 (default) | 2

Simulation number that the parameter set applies to, specified as an integer, 1 or 2. This parameter applies to the simulation test case where there are two simulations. For baseline and simulation test cases, the simulation index is 1.

Output Arguments

pset — Parameter set object

object

Parameter set, returned as an `sltest.testmanager.ParameterSet` object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

addReportBody

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Add main report body

Syntax

```
addReportBody(obj)
```

Description

`addReportBody(obj)` adds the main body pages to the report.

This method also calls:

- `sltest.testmanager.TestResultReport.genResultSetBlock`
- `sltest.testmanager.TestResultReport.genTestSuiteResultBlock`
- `sltest.testmanager.TestResultReport.genTestCaseResultBlock`

Input Arguments

obj — Test report object

`sltest.testmanager.TestResultReport` object

Test report, specified as a `sltest.testmanager.TestResultReport` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport`

Introduced in R2016a

addReportTOC

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Add report table of contents



Syntax

addReportTOC(obj)

Description

addReportTOC(obj) adds the table of contents page to the report.

Summary

Name	Outcome	Duration (Seconds)
Results: 2015-Dec-01 11:46:00	2 ✓	11
 test1	2 ✓	11
 New Test Suite 1	2 ✓	11
 New Test Case 1	2 ✓	10
 Iteration	✓	8
 Iteration 1	✓	2

Input Arguments

obj — Test report object

sltest.testmanager.TestResultReport object

Test report, specified as a sltest.testmanager.TestResultReport object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport`

Introduced in R2016a

addTitlePage

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Add report title page

Syntax

addTitlePage(obj)

Description

addTitlePage(obj) adds the title page to the report.

Report Generated by Test Manager

Title:	Test
Author:	Test Author
Date:	01-Dec-2015 11:50:23

Test Environment

Platform:	PCWIN64
MATLAB:	(R2016b)

Input Arguments

obj — Test report object

sltest.testmanager.TestResultReport object

Test report, specified as a `sltest.testmanager.TestResultReport` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport`

Introduced in R2016a

captureBaselineCriteria

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Capture baseline criteria and add to test case

Syntax

```
baseline = captureBaselineCriteria(filePath,append)
```

Description

`baseline = captureBaselineCriteria(filePath,append)` runs the System Under Test and captures a baseline criteria set. The function returns a baseline criteria object, `sltest.testmanager.BaselineCriteria`. This function can be used only if the test type is a baseline test case.

Input Arguments

filePath — Input file name and path

string

Name and file path to save the baseline criteria file to, specified as a string.

append — Append baseline criteria

true | false

Append baseline criteria if criteria already exists, specified as a Boolean. The Boolean `true` appends to existing criteria, and `false` replaces all existing criteria.

Output Arguments

baseline — Baseline criteria object

object

Baseline criteria added to the test case, returned as an `sltest.testmanager.BaselineCriteria` object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

captureEquivalenceCriteria

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Capture equivalence criteria and add to test case

Syntax

```
eq = captureEquivalenceCriteria(replaceAll)
```

Description

`eq = captureEquivalenceCriteria(replaceAll)` runs the System Under Test in Simulation 1 and captures an equivalence criteria set. The function returns an equivalence criteria object, `sltest.testmanager.EquivalenceCriteria`. This function can be used only if the test type is an equivalence test case.

Input Arguments

replaceAll — Replace equivalence criteria

`true` | `false`

Replace existing equivalence criteria if criteria already exists in the test case, specified as a Boolean. The Boolean `true` replaces all existing criteria, and `false` errors if criteria already exists in the test case.

Output Arguments

eq — Equivalence criteria object

object

Equivalence criteria added to the test case, returned as an `sltest.testmanager.EquivalenceCriteria` object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

close

Class: sltest.testmanager.TestFile

Package: sltest.testmanager

Close test file in test manager

Syntax

```
close
```

Description

close closes the test file in the test manager and does not save unsaved changes.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

copySimulationSetting

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Copy simulation setting in equivalence test case

Syntax

```
copySimulationSetting(fromSimIndex,toSimIndex)
```

Description

`copySimulationSetting(fromSimIndex,toSimIndex)` copies the simulation setting from one simulation number to another within an equivalence test case. This function works only for equivalence test case types.

Input Arguments

fromSimIndex — Copy from simulation number

1 | 2

Simulation number you want to copy the settings from, specified as an integer, 1 or 2. This is the source simulation.

toSimIndex — Copy to simulation number

1 | 2

Simulation number you want to copy the settings to, specified as an integer, 1 or 2. This is the target simulation.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

createTestCase

Class: sltest.testmanager.TestSuite

Package: sltest.testmanager

Create test case

Syntax

```
tc = createTestCase(type,name,runOnTarget)
```

Description

`tc = createTestCase(type,name,runOnTarget)` creates a new test case within the test suite. You can specify the test case name and `type`: `baseline`, `equivalence`, and `simulation`. Also, if you are using the test case for real-time testing, you can specify this using the `runOnTarget` argument.

Input Arguments

type — Test case type

'baseline' (default) | 'equivalence' | 'simulation'

Test case type, specified as a string.

name — Test case name

string

Test case name, specified as a string. If this input argument is empty, then the test manager gives the test case a unique name.

runOnTarget — Run simulation on target

cell array of Booleans

Specify if you want to run the test case simulation on a target, specified as a cell array of Booleans. This is an optional argument. For more information on real-time testing, see “Test Models in Real Time”.

Output Arguments

tc – Test case object

object

Test case, returned as an `sltest.testmanager.TestCase` object.

Examples

Create Test Case

```
% Create test file
tf = sltest.testmanager.TestFile('C:\MATLAB\test_file.mldatx');

% Create test suite
ts = sltest.testmanager.TestSuite(tf, 'My Test Suite');

% Create test case
tc = ts.createTestCase('baseline', 'My Baseline Test')

tc =
```

TestCase with properties:

```
    Name: 'My Baseline Test'
   TestFile: [1x1 sltest.testmanager.TestFile]
  TestPath: 'test_file > My Test Suite > My Baseline Test'
   TestType: 'baseline'
 RunOnTarget: {[0]}
     Parent: [1x1 sltest.testmanager.TestSuite]
 Requirements: [0x1 struct]
  Description: ''
     Enabled: 1
```

- “Automate Tests Programmatically”

Introduced in R2015b

createTestForSubsystem

Class: sltest.testmanager.TestFile

Package: sltest.testmanager

Create test case for subsystem

Syntax

```
createTestForSubsystem(tfObj, subsystemFullPath, testType,  
inputsLocation, baselineLocation)
```

Description

`createTestForSubsystem(tfObj, subsystemFullPath, testType, inputsLocation, baselineLocation)` creates a new test case for a model subsystem within the test file in a new test suite. Specify the test case name and type: baseline, equivalence, and simulation. Also, specify where the input and baseline datasets are saved.

Input Arguments

tfObj — Test file

string

Test file, specified as an `sltest.testmanager.TestFile` object.

subsystemFullPath — Subsystem path

string

Full path of the subsystem starting at the top model, specified as a string.

Example: 'f14/Controller'

testType — Test case type

'baseline' | 'equivalence' | 'simulation'

Test case type, specified as a string.

Example: 'baseline'

inputsLocation — Inputs file location

string

File name and path of where to save the inputs data, specified as a string. The inputs file is saved as a MAT-file with the extension `.mat`.

Example: 'C:\MATLAB\inputs_data.mat'

baselineLocation — Baseline file location

string

File name and path of where to save the baseline data, specified as a string. The baseline file is saved as a MAT-file with the extension `.mat`.

Example: 'C:\MATLAB\baseline_data.mat'

See Also

`sltest.testmanager.TestFile`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

createTestForSubsystem

Class: sltest.testmanager.TestSuite

Package: sltest.testmanager

Create test case for subsystem

Syntax

```
createTestForSubsystem(tsObj, subsystemFullPath, testType,  
inputsLocation, baselineLocation)
```

Description

`createTestForSubsystem(tsObj, subsystemFullPath, testType, inputsLocation, baselineLocation)` creates a new test case for a model subsystem within the test suite. Specify the test case name and type: baseline, equivalence, and simulation. Also, specify where the input and baseline datasets are saved.

Input Arguments

tsObj — Test suite

string

Test suite, specified as an `sltest.testmanager.TestSuite` object.

subsystemFullPath — Subsystem path

string

Full path of the subsystem starting at the top model, specified as a string.

Example: 'f14/Controller'

testType — Test case type

'baseline' | 'equivalence' | 'simulation'

Test case type, specified as a string.

Example: 'baseline'

inputsLocation — Inputs file location

string

File name and path of where to save the inputs data, specified as a string. The inputs file is saved as a MAT-file with the extension `.mat`.

Example: 'C:\MATLAB\inputs_data.mat'

baselineLocation — Baseline file location

string

File name and path of where to save the baseline data, specified as a string. The baseline file is saved as a MAT-file with the extension `.mat`.

Example: 'C:\MATLAB\baseline_data.mat'

See Also

`sltest.testmanager.TestSuite`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

createTestSuite

Class: sltest.testmanager.TestFile

Package: sltest.testmanager

Create new test suite

Syntax

```
ts = createTestSuite(suiteName)
```

Description

`ts = createTestSuite(suiteName)` creates a test suite and adds it to the test file.

Input Arguments

suiteName — Test suite name

string

Name of the test suite, specified as a string.

Output Arguments

ts — Test suite object

object

Test suite, returned as an `sltest.testmanager.TestSuite` object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

createTestSuite

Class: sltest.testmanager.TestSuite

Package: sltest.testmanager

Create test suite

Syntax

```
ts = createTestSuite(suiteName)
```

Description

`ts = createTestSuite(suiteName)` creates a new test suite.

Input Arguments

suiteName — Test suite name

string

Test suite name, specified as a string. If this input argument is empty, then the test manager gives the test suite a unique name.

Output Arguments

ts — Test suite object

object

Test suite, returned as an `sltest.testmanager.TestSuite` object.

Examples

Create Test Suite

```
% Create test file
```

```
tf = sltest.testmanager.TestFile('C:\MATLAB\test_file.mldatx');  
  
% Create test suite  
ts = sltest.testmanager.TestSuite(tf,'My Test Suite');  
  
% Create another new test suite using method  
ts2 = ts.createTestSuite('Baseline Tests')  
  
ts2 =  
  
    TestSuite with properties:  
  
        Name: 'Baseline Tests'  
    Description: ''  
        Enabled: 1  
    TestFile: [1x1 sltest.testmanager.TestFile]  
    TestPath: 'test_file > My Test Suite > Baseline Tests'  
    Parent: [1x1 sltest.testmanager.TestSuite]
```

- “Automate Tests Programmatically”

Introduced in R2015b

deleteliterations

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Delete test iterations that belong to test case

Syntax

```
deleteIterations(tcObj, iterObj)
```

Description

`deleteIterations(tcObj, iterObj)` deletes one or more test iterations from the test case.

Input Arguments

tcObj — **Test case to delete iteration from**
object

Test case that you want to delete the iteration from, specified as a `sltest.testmanager.TestCase` object.

iterObj — **Test iteration to delete**
object array

Test iterations that you want to delete from the test case, specified as an array of `sltest.testmanager.TestIteration` objects.

Examples

- “Run Multiple Combinations of Tests Using Iterations”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestIteration`

Introduced in R2016a

genBaselineInfoTable

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate baseline dataset information table

Syntax

```
baselineTable = genBaselineInfoTable(obj,result)
```

Description

`baselineTable = genBaselineInfoTable(obj,result)` generates a section for the baseline dataset information used in the test case.

Baseline Information

Baseline Name: baseline1.mat

Baseline File: Z:\12\aabhishe.Dec1\baseline1.mat

Name	Data Type	Units	Sample Time	Interp	Sync	Link to Plot
yout.Ww	double		Continuous	linear	union	Link
yout.Vs	double		Continuous	linear	union	Link
yout.Sd	double		Continuous	linear	union	Link
slp	double		Continuous	linear	union	Link

Input Arguments

obj — Test report object

sltest.testmanager.TestResultReport object

Test report, specified as a sltest.testmanager.TestResultReport object.

result — Result set

`sltest.testmanager.TestCaseResult` object |
`sltest.testmanager.TestIterationResult` object

Result set, specified as a `sltest.testmanager.TestCaseResult` or `sltest.testmanager.TestIterationResult` object.

Output Arguments

baselineTable — Table

`mlreportgen.dom.FormalTable` object

The test case baseline dataset table generated by the method, returned as a `mlreportgen.dom.FormalTable` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `mlreportgen.dom.FormalTable`

Introduced in R2016a

genCoverageTable

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate coverage collection table



Syntax

```
docPart = genCoverageTable(obj, resultObj)
```

Description

`docPart = genCoverageTable(obj, resultObj)` generates a section for coverage that was collected during the test.

Aggregated Coverage Results

Model Name	C1	TBL	Execution
 sldemo_absbrake	--	100%	100%
 sldemo_wheelspeed_absbrake	100%	--	100%

Input Arguments

obj — Test report object

`sltest.testmanager.TestResultReport` object

Test report, specified as a `sltest.testmanager.TestResultReport` object.

resultObj — Result set

`sltest.testmanager.TestResult` object

Result set, specified as a `sltest.testmanager.TestResult` object.

Output Arguments

docPart — Document part

`mlreportgen.dom.DocumentPart` object

The coverage section document part generated by the method, returned as a `mlreportgen.dom.DocumentPart` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `mlreportgen.dom.DocumentPart`

Introduced in R2016a

genHyperLinkToToC

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate link to table of contents

Syntax

```
para = genHyperLinkToToC(obj,indent)
```

Description

para = genHyperLinkToToC(obj,indent) generates link to the report table of contents.

The link appears at the end of sections:

[Back to Report Summary](#)

Input Arguments

obj — Test report object

sltest.testmanager.TestResultReport object

Test report, specified as a sltest.testmanager.TestResultReport object.

indent — Link indentation

string

Space between the left outer boundary of this paragraph and the left inner boundary of the link, specified as a string.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- cm — centimeters
- in — inches
- mm — millimeters
- pi — picas
- pt — points
- px — pixels

Example: '5mm'

Output Arguments

para — Paragraph

mlreportgen.dom.Paragraph object

The link paragraph generated by the method, returned as a mlreportgen.dom.Paragraph object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

sltest.testmanager.TestResultReport | mlreportgen.dom.Paragraph

Introduced in R2016a

genIterationSettingTable

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate iteration settings table

Syntax

```
docPart = genIterationSettingTable(obj,result)
```

Description

`docPart = genIterationSettingTable(obj,result)` generates a section for the iteration settings used to override the parent test case settings.

Iteration Settings

Test Overrides

Parameter Name	Value
ParameterSet	Parameter Set 1

Input Arguments

obj — Test report object

sltest.testmanager.TestResultReport object

Test report, specified as a sltest.testmanager.TestResultReport object.

result — Result set

sltest.testmanager.ReportUtility.ReportResultData object

Result set, specified as a sltest.testmanager.ReportUtility.ReportResultData object.

Output Arguments

docPart — Document part

`mlreportgen.dom.DocumentPart` object

The iteration settings section document part generated by the method, returned as a `mlreportgen.dom.DocumentPart` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `mlreportgen.dom.DocumentPart`

Introduced in R2016a

genMetadataBlockForTestResult

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate result metadata section

Syntax

```
docPart = genMetadataBlockForTestResult(obj, result,  
isTestSuiteResult)
```

Description

`docPart = genMetadataBlockForTestResult(obj, result, isTestSuiteResult)` generates a result metadata section for a test suite or test case result.

If called from `sltest.testmanager.TestResultReport.genTestSuiteResultBlock`, then this method also calls:

- `sltest.testmanager.TestResultReport.genTableRowsForResultMetaInfo`
- `sltest.testmanager.TestResultReport.genRequirementLinksTable`

If called from `sltest.testmanager.TestResultReport.genTestCaseResultBlock`, then this method also calls:

- `sltest.testmanager.TestResultReport.genTableRowsForResultMetaInfo`
- `sltest.testmanager.TestResultReport.genRequirementLinksTable`
- `sltest.testmanager.TestResultReport.genIterationSettingTable`

Input Arguments

obj — Test report object

`sltest.testmanager.TestResultReport` object

Test report, specified as a `sltest.testmanager.TestResultReport` object.

result — Result set

`sltest.testmanager.ReportUtility.ReportResultData` object

Result set, specified as a `sltest.testmanager.ReportUtility.ReportResultData` object.

isTestSuiteResult — Test suite indicator

`true` | `false`

Flag to indicate whether the test result metadata block is for a test suite result or not, specified as a Boolean, `true` or `false`.

Output Arguments

docPart — Document part

`mlreportgen.dom.DocumentPart` object

The result set section document part generated by the method, returned as a `mlreportgen.dom.DocumentPart` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `mlreportgen.dom.DocumentPart`

Introduced in R2016a

genParameterOverridesTable

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate test case parameter overrides table

Syntax

```
overridesTable = genParameterOverridesTable(obj,result,simIndex)
```

Description

overridesTable = genParameterOverridesTable(obj,result,simIndex)
generates a section for parameter overrides used in the test case.

Parameter Overrides

Workspace Variable	Value	Source	Model Element
Parameter Set 1			
Rr	1.5	base workspace	sldemo_absbrake/Rr, sldemo_absbrake/Vehicle speed (angular)

Input Arguments

obj — Test report object

sltest.testmanager.TestResultReport object

Test report, specified as a sltest.testmanager.TestResultReport object.

result — Result set

`sltest.testmanager.TestResult` object

Result set, specified as a `sltest.testmanager.TestResult` object.

simIndex — Simulation index

1 | 2

Simulation number that the test case parameter overrides table applies to, specified as an integer, 1 or 2. This setting applies to the simulation test case.

Output Arguments

overridesTable — Table

`mlreportgen.dom.FormalTable` object

The test case parameter overrides table generated by the method, returned as a `mlreportgen.dom.FormalTable` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `mlreportgen.dom.FormalTable`

Introduced in R2016a

genRequirementLinksTable

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate requirement links table

Syntax

```
reqTable = genRequirementLinksTable(obj,resultObj,isTestSuiteResult)
```

Description

`reqTable = genRequirementLinksTable(obj,resultObj,isTestSuiteResult)` generates a section for table of requirement links.

Test Case Requirements

Description: requirement

Document: <http://www.mathworks.com>

Input Arguments

obj — Test report object

sltest.testmanager.TestResultReport object

Test report, specified as a sltest.testmanager.TestResultReport object.

resultObj — Result set

sltest.testmanager.TestResult object

Result set, specified as a sltest.testmanager.TestResult object.

isTestSuiteResult — Test suite indicator

true | false

Flag to indicate whether the requirement links table is for a test suite result or not, specified as a Boolean, `true` or `false`.

Output Arguments

reqTable — Table

`mlreportgen.dom.FormalTable` object

The requirements links table generated by the method, returned as a `mlreportgen.dom.FormalTable` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `mlreportgen.dom.FormalTable`

Introduced in R2016a

genResultSetBlock

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate results set section

Syntax

```
docPart = genResultSetBlock(obj,result)
```



Description

docPart = genResultSetBlock(obj,result) generates the results set section.

Results: 2015-Dec-01 11:46:00

Result Type: Result Set
 Parent: None
 Start Time: 2015-Dec-01 11:46:00
 End Time: 2015-Dec-01 11:46:11
 Outcome: Total: 2, Passed: 2

Aggregated Coverage Results

Model Name	C1	TBL	Execution
 slidemo_absbrake	--	100%	100%
 slidemo_wheelspeed_absbrake	100%	--	100%

[Back to Report Summary](#)

This method also calls:

- sltest.testmanager.TestResultReport.genTableRowsForResultMetaInfo
- sltest.testmanager.TestResultReport.genCoverageTable
- sltest.testmanager.TestResultReport.genHyperLinkToToC

Input Arguments

obj — Test report object

`sltest.testmanager.TestResultReport` object

Test report, specified as a `sltest.testmanager.TestResultReport` object.

result — Result set

`sltest.testmanager.ReportUtility.ReportResultData` object

Result set, specified as a `sltest.testmanager.ReportUtility.ReportResultData` object.

Output Arguments

docPart — Document part

`mlreportgen.dom.DocumentPart` object

The result set section document part generated by the method, returned as a `mlreportgen.dom.DocumentPart` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `mlreportgen.dom.DocumentPart`

Introduced in R2016a

genRunBlockForTestCaseResult

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate test case configuration and results section

Syntax

```
docPart = genRunBlockForTestCaseResult(obj,run,runType,result,
simIndex)
```

Description

docPart = genRunBlockForTestCaseResult(obj,run,runType,result,simIndex) generates a combined section for baseline data, simulation configuration, parameter overrides, simulation output, criteria comparison, and verify run data.

This method also calls:

- sltest.testmanager.TestResultReport.genBaselineInfoTable
- sltest.testmanager.TestResultReport.genSimulationConfigurationTable
- sltest.testmanager.TestResultReport.genParameterOverridesTable
- sltest.testmanager.TestResultReport.genSignalSummaryTable
- sltest.testmanager.TestResultReport.plotOneSignalToFile
- sltest.testmanager.TestResultReport.genHyperLinkToToC

Input Arguments

obj — Test report object

sltest.testmanager.TestResultReport object

Test report, specified as a sltest.testmanager.TestResultReport object.

run — Run data

Simulink.sdi.Run object

Test case result run data, specified as a `Simulink.sdi.Run` object.

runType — Run type

`sltest.testmanager.RunTypes` object

The run type, specified as a `sltest.testmanager.RunTypes` object.

result — Result

`sltest.testmanager.ReportUtility.ReportResultData` object

Run result, specified as a

`sltest.testmanager.ReportUtility.ReportResultData` object.

simIndex — Simulation index

1 | 2

Simulation number that the result applies to, specified as an integer, 1 or 2. This setting applies to the simulation test case.

Output Arguments

docPart — Document part

`mlreportgen.dom.DocumentPart` object

The result set section document part generated by the method, returned as a `mlreportgen.dom.DocumentPart` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `Simulink.sdi.Run` | `mlreportgen.dom.DocumentPart`

Introduced in R2016a

genSignalSummaryTable

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate signal output and comparison data

Syntax

```
docPart = genSignalSummaryTable(obj,signalList,isComparison,
isSummaryTable)
```

Description

docPart = genSignalSummaryTable(obj,signalList,isComparison, isSummaryTable) generates a section for signal output and comparison data.

Name	Data Type	Units	Sample Time	Interp	Sync	Link to Plot
yout.Ww	double		Continuous	linear	union	Link
yout.Vs	double		Continuous	linear	union	Link
yout.Sd	double		Continuous	linear	union	Link
slp	double		Continuous	linear	union	Link

Input Arguments

obj — Test report object

sltest.testmanager.TestResultReport object

Test report, specified as a sltest.testmanager.TestResultReport object.

signalList — Signal list

sltest.testmanager.ReportUtility.Signal object

Signals in the summary table, specified as an array of sltest.testmanager.ReportUtility.Signal objects.

isComparison — Comparison indicator

true | false

Flag to indicate whether the signal is from a comparison run or not, specified as a Boolean, `true` or `false`.

isSummaryTable — Summary table indicator

true | false

Flag to indicate whether this is for a signal summary table in the report or an individual signal plot, specified as a Boolean. `true` for the signal summary table or `false` for an individual signal plot.

Output Arguments

docPart — Document part

`mlreportgen.dom.DocumentPart` object

The signal summary section document part generated by the method, returned as a `mlreportgen.dom.DocumentPart` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `mlreportgen.dom.DocumentPart`

Introduced in R2016a

genSimulationConfigurationTable

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate test case simulation configuration table

Syntax

```
simCfgTable = genSimulationConfigurationTable(obj,result,simIndex)
```

Description

`simCfgTable = genSimulationConfigurationTable(obj,result,simIndex)` generates a section for simulation configuration data used in the test case.

Simulation

System Under Test Information

Model:	sldemo_absbrake
Simulation Mode	Normal
Configuration Set:	Configuration
Start Time:	0
Stop Time:	14.635438874554231
Checksum:	1514901952 3523488043 2320696550 3824373991
Simulink Version:	8.7

Input Arguments

obj — Test report object

`sltest.testmanager.TestResultReport` object

Test report, specified as a `sltest.testmanager.TestResultReport` object.

result — Result set

`sltest.testmanager.ReportUtility.ReportResultData` object

Result set, specified as a `sltest.testmanager.ReportUtility.ReportResultData` object.

simIndex — Simulation index

1 | 2

Simulation number that the test case configuration table applies to, specified as an integer, 1 or 2. This setting applies to the simulation test case.

Output Arguments

simCfgTable — Table

`mlreportgen.dom.FormalTable` object

The test case simulation configuration table generated by the method, returned as a `mlreportgen.dom.FormalTable` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `mlreportgen.dom.FormalTable`

Introduced in R2016a

genTableRowsForResultMetaInfo

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate test result metadata table

Syntax

```
rowList = genTableRowsForResultMetaInfo(obj,result)
```

Description

`rowList = genTableRowsForResultMetaInfo(obj,result)` generates a section for test result metadata used in a result set, test file, test suite, test case, or test iteration.

Test Result Information

Result Type:	Test Suite Result
Parent:	test1
Start Time:	2015-Dec-01 14:14:16
End Time:	2015-Dec-01 14:14:20
Outcome:	Total: 2, Passed: 1, Failed: 1

Input Arguments

obj — Test report object

`sltest.testmanager.TestResultReport` object

Test report, specified as a `sltest.testmanager.TestResultReport` object.

result — Result set

`sltest.testmanager.ReportUtility.ReportResultData` object

Result set, specified as a `sltest.testmanager.ReportUtility.ReportResultData` object.

Output Arguments

rowList — Table row

`mlreportgen.dom.TableRow` object

The metadata information table row generated by the method, returned as a `mlreportgen.dom.TableRow` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `mlreportgen.dom.TableRow`

Introduced in R2016a

genTestCaseResultBlock

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate test case result section

Syntax

```
docPart = genTestCaseResultBlock(obj, result)
```

Description

`docPart = genTestCaseResultBlock(obj, result)` generates a test case result section.

This method also calls:

- `sltest.testmanager.TestResultReport.genMetadataBlockForTestResult`
- `sltest.testmanager.TestResultReport.genCoverageTable`
- `sltest.testmanager.TestResultReport.genHyperLinkToToC`

Input Arguments

obj — Test report object

`sltest.testmanager.TestResultReport` object

Test report, specified as a `sltest.testmanager.TestResultReport` object.

result — Result set

`sltest.testmanager.ReportUtility.ReportResultData` object

Result set, specified as a `sltest.testmanager.ReportUtility.ReportResultData` object.

Output Arguments

docPart — Document part

`mlreportgen.dom.DocumentPart` object

The result set section document part generated by the method, returned as a `mlreportgen.dom.DocumentPart` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `mlreportgen.dom.DocumentPart`

Introduced in R2016a

genTestSuiteResultBlock

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Generate test suite result section

Syntax

```
docPart = genTestSuiteResultBlock(obj,result)
```

Description

docPart = genTestSuiteResultBlock(obj,result) generates a test suite result section.

New Test Suite 1

Test Result Information

Result Type:	Test Suite Result
Parent:	test1
Start Time:	2015-Dec-01 11:46:00
End Time:	2015-Dec-01 11:46:11
Outcome:	Total: 2, Passed: 2

Test Suite Information

Name: New Test Suite 1
[Back to Report Summary](#)

This method also calls:

- sltest.testmanager.TestResultReport.genMetadataBlockForTestResult
- sltest.testmanager.TestResultReport.genCoverageTable
- sltest.testmanager.TestResultReport.genHyperLinkToToC

Input Arguments

obj — Test report object

`sltest.testmanager.TestResultReport` object

Test report, specified as a `sltest.testmanager.TestResultReport` object.

result — Result set

`sltest.testmanager.ReportUtility.ReportResultData` object

Result set, specified as a `sltest.testmanager.ReportUtility.ReportResultData` object.

Output Arguments

docPart — Document part

`mlreportgen.dom.DocumentPart` object

The result set section document part generated by the method, returned as a `mlreportgen.dom.DocumentPart` object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport` | `mlreportgen.dom.DocumentPart`

Introduced in R2016a

getBaselineCriteria

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Get baseline criteria

Syntax

```
baselines = getBaselineCriteria
```

Description

`baselines = getBaselineCriteria` gets all of the baseline criteria sets in a test case and returns them as an array of baseline criteria objects, `sltest.testmanager.BaselineCriteria`.

Output Arguments

baselines — Baseline criteria object

object array

Baseline criteria that are in the baseline test case, returned as an array of `sltest.testmanager.BaselineCriteria` objects.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getBaselineRun

Class: sltest.testmanager.TestCaseResult

Package: sltest.testmanager

Get test case baseline dataset

Syntax

```
baseline = getBaselineRun(resultObj)
```

Description

`baseline = getBaselineRun(resultObj)` gets the baseline dataset used in a test case, which belongs to the test case results object. The baseline dataset is saved with the test case result only if the **Save baseline data in test result** check box is selected in the test case under the **Baseline Criteria** section.

Input Arguments

resultObj — Test case result

object

Test case results object to get baseline dataset from, specified as a `sltest.testmanager.TestCaseResult` object.

Output Arguments

baseline — Baseline dataset

object

Test case baseline dataset, returned as a `Simulink.sdi.Run` object. If the **Save baseline data in test result** check box is not selected in the test case, then the function returns an empty array.

See Also

Simulink.sdi.Run

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

getBaselineRun

Class: sltest.testmanager.TestIterationResult

Package: sltest.testmanager

Get test iteration baseline dataset

Syntax

```
baseline = getBaselineRun(resultObj)
```

Description

`baseline = getBaselineRun(resultObj)` gets the baseline dataset used in a test iteration, which belongs to the test iteration results object. The baseline dataset is saved with the test iteration result only if the **Save baseline data in test result** check box is selected in the parent test case under the **Baseline Criteria** section.

Input Arguments

resultObj — Test iteration result
object

Test iteration results object to get baseline dataset from, specified as a `sltest.testmanager.TestIterationResult` object.

Output Arguments

baseline — Baseline dataset
object

Test iteration baseline dataset, returned as a `Simulink.sdi.Run` object. If the **Save baseline data in test result** check box is not selected in the parent test case, then the function returns an empty array.

See Also

Simulink.sdi.Run

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

getComparisonRun

Class: sltest.testmanager.TestCaseResult

Package: sltest.testmanager

Get test case comparison results

Syntax

```
run = getComparisonRun(resultObj)
```

Description

`run = getComparisonRun(resultObj)` gets all of the test case comparison results that belong to the test case results object.

Input Arguments

resultObj — Test case result

object

Test case results object to get results from, specified as a `sltest.testmanager.TestCaseResult` object.

Output Arguments

runArray — Comparison results

object

Test case comparison results, returned as a `Simulink.sdi.Run` object.

See Also

`Simulink.sdi.Run`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015a

getComparisonRun

Class: sltest.testmanager.TestIterationResult

Package: sltest.testmanager

Get test iteration comparison results

Syntax

```
run = getComparisonRun(resultObj)
```

Description

`run = getComparisonRun(resultObj)` gets all of the test iteration comparison results that belong to the test iteration results object.

Input Arguments

resultObj — Test iteration result

object

Test iteration results object to get results from, specified as a `sltest.testmanager.TestIterationResult` object.

Output Arguments

run — Comparison results

object

Test iteration comparison results, returned as a `Simulink.sdi.Run` object.

See Also

`sltest.testmanager.TestIterationResult`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

getCoverageResults

Class: sltest.testmanager.ResultSet

Package: sltest.testmanager

Get coverage results

Syntax

```
covResult = getCoverageResults(resultObj)  
covResult = getCoverageResults(resultObj,model)
```

Description

`covResult = getCoverageResults(resultObj)` gets all of the coverage results that belong to the result set object.

`covResult = getCoverageResults(resultObj,model)` gets all of the coverage results that belong to the result set object and the specified model.

Input Arguments

resultObj — Result set

object

Result set object to get coverage results from, specified as a `sltest.testmanager.ResultSet` object.

model — Model name

string

Name of a model within the set of coverage results, specified as a string.

Output Arguments

covResult — Coverage results

object array

Coverage results contained in the result set, returned as an array of `cvdata` objects. For more information on `cvdata` objects, see `cv.cvdatagroup`.

See Also

`sltest.testmanager.CoverageSettings`

Related Examples

- “Automate Tests Programmatically”
- “Collect Coverage in Tests”

Introduced in R2016a

getCoverageResults

Class: sltest.testmanager.TestCaseResult

Package: sltest.testmanager

Get coverage results

Syntax

```
covResult = getCoverageResults(resultObj)  
covResult = getCoverageResults(resultObj,model)
```

Description

`covResult = getCoverageResults(resultObj)` gets all of the coverage results that belong to the test case results object.

`covResult = getCoverageResults(resultObj,model)` gets all of the coverage results that belong to the test case results object and the specified model.

Input Arguments

resultObj — Test case result

object

Test case results object to get coverage results from, specified as a `sltest.testmanager.TestCaseResult` object.

model — Model name

string

Name of a model within the set of coverage results, specified as a string.

Output Arguments

covResult — Coverage results

object array

Coverage results contained in the test case result, returned as an array of `cvdata` objects. For more information on `cvdata` objects, see `cv.cvdatagroup`.

See Also

`sltest.testmanager.CoverageSettings`

Related Examples

- “Automate Tests Programmatically”
- “Collect Coverage in Tests”

Introduced in R2016a

getCoverageResults

Class: sltest.testmanager.TestFileResult

Package: sltest.testmanager

Get coverage results

Syntax

```
covResult = getCoverageResults(resultObj)
covResult = getCoverageResults(resultObj,model)
```

Description

`covResult = getCoverageResults(resultObj)` gets all of the coverage results that belong to the test file results object.

`covResult = getCoverageResults(resultObj,model)` gets all of the coverage results that belong to the test file results object and the specified model.

Input Arguments

resultObj — Test file result

object

Test file results object to get coverage results from, specified as a `sltest.testmanager.TestFileResult` object.

model — Model name

string

Name of a model within the set of coverage results, specified as a string.

Output Arguments

covResult — Coverage results

object array

Coverage results contained in the test file result, returned as an array of `cvdata` objects. For more information on `cvdata` objects, see `cv.cvdatagroup`.

See Also

`sltest.testmanager.CoverageSettings`

Related Examples

- “Automate Tests Programmatically”
- “Collect Coverage in Tests”

Introduced in R2016a

getCoverageResults

Class: sltest.testmanager.TestIterationResult

Package: sltest.testmanager

Get coverage results

Syntax

```
covResult = getCoverageResults(resultObj)
covResult = getCoverageResults(resultObj,model)
```

Description

`covResult = getCoverageResults(resultObj)` gets all of the coverage results that belong to the test iteration results object.

`covResult = getCoverageResults(resultObj,model)` gets all of the coverage results that belong to the test iteration results object and the specified model.

Input Arguments

resultObj — Test iteration result

object

Test iteration results object to get coverage results from, specified as a `sltest.testmanager.TestIterationResult` object.

model — Model name

string

Name of a model within the set of coverage results, specified as a string.

Output Arguments

covResult — Coverage results

object array

Coverage results contained in the test iteration result, returned as an array of `cvdata` objects. For more information on `cvdata` objects, see `cv.cvdatagroup`.

See Also

`sltest.testmanager.CoverageSettings`

Related Examples

- “Automate Tests Programmatically”
- “Collect Coverage in Tests”

Introduced in R2016a

getCoverageResults

Class: sltest.testmanager.TestSuiteResult

Package: sltest.testmanager

Get coverage results

Syntax

```
covResult = getCoverageResults(resultObj)
covResult = getCoverageResults(resultObj,model)
```

Description

`covResult = getCoverageResults(resultObj)` gets all of the coverage results that belong to the test suite results object.

`covResult = getCoverageResults(resultObj,model)` gets all of the coverage results that belong to the test suite results object and the specified model.

Input Arguments

resultObj — Test suite result

object

Test suite results object to get coverage results from, specified as a `sltest.testmanager.TestSuiteResult` object.

model — Model name

string

Name of a model within the set of coverage results, specified as a string.

Output Arguments

covResult — Coverage results

object array

Coverage results contained in the test suite result, returned as an array of `cvdata` objects. For more information on `cvdata` objects, see `cv.cvdatagroup`.

See Also

`sltest.testmanager.CoverageSettings`

Related Examples

- “Automate Tests Programmatically”
- “Collect Coverage in Tests”

Introduced in R2016a

getCoverageSettings

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Get coverage settings

Syntax

```
covSettings = getCoverageSettings(tcObj)
```

Description

`covSettings = getCoverageSettings(tcObj)` gets the coverage settings for a test case and returns an `sltest.testmanager.CoverageSettings` object.

Input Arguments

tcObj — Test case

object

Test case object to get coverage settings from, specified as an `sltest.testmanager.TestCase` object.

Output Arguments

covSettings — Coverage settings

object

Coverage settings for the test case, returned as an `sltest.testmanager.CoverageSettings` object.

See Also

`sltest.testmanager.CoverageSettings`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

getCoverageSettings

Class: sltest.testmanager.TestFile

Package: sltest.testmanager

Get coverage settings

Syntax

```
covSettings = getCoverageSettings(tfObj)
```

Description

`covSettings = getCoverageSettings(tfObj)` gets the coverage settings for a test file and returns an `sltest.testmanager.CoverageSettings` object.

Input Arguments

tfObj — Test file

object

Test file object to get coverage settings from, specified as an `sltest.testmanager.TestFile` object.

Output Arguments

covSettings — Coverage settings

object

Coverage settings for the test file, returned as an `sltest.testmanager.CoverageSettings` object.

See Also

`sltest.testmanager.CoverageSettings`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

getCoverageSettings

Class: sltest.testmanager.TestSuite

Package: sltest.testmanager

Get coverage settings

Syntax

```
covSettings = getCoverageSettings(tsObj)
```

Description

`covSettings = getCoverageSettings(tsObj)` gets the coverage settings for a test suite and returns an `sltest.testmanager.CoverageSettings` object.

Input Arguments

tsObj — Test suite

object

Test suite object to get coverage settings from, specified as an `sltest.testmanager.TestSuite` object.

Output Arguments

covSettings — Coverage settings

object

Coverage settings for the test suite, returned as an `sltest.testmanager.CoverageSettings` object.

See Also

`sltest.testmanager.CoverageSettings`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

getEquivalenceCriteria

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Get equivalence criteria from test case

Syntax

```
eq = getEquivalenceCriteria
```

Description

`eq = getEquivalenceCriteria` gets the equivalence criteria set from the test case. The function returns an equivalence criteria object, `sltest.testmanager.EquivalenceCriteria`. This function can be used only if the test type is an equivalence test case.

Output Arguments

eq — Equivalence criteria object

object

Equivalence criteria in the test case, returned as an `sltest.testmanager.EquivalenceCriteria` object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getInputs

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Get test case inputs

Syntax

```
inputs = getInputs(simulationIndex)
```

Description

`inputs = getInputs(simulationIndex)` gets all of the input sets in a test case and returns them as an array of test input objects, `sltest.testmanager.TestInput`.

Input Arguments

simulationIndex — Test case simulation number

1 | 2

Simulation number that the parameter sets apply to, specified as an integer, 1 or 2. This setting applies to the simulation test case where there are two simulations. For baseline and simulation test cases, the simulation index is 1.

Output Arguments

inputs — Test input object

object array

Test inputs that belong to the test case, returned as an array of `sltest.testmanager.TestInput` objects.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getIterationResults

Class: sltest.testmanager.TestCaseResult

Package: sltest.testmanager

Get iteration results

Syntax

```
iterArray = getIterationResults(resultObj)
```

Description

`iterArray = getIterationResults(resultObj)` returns the test iteration results that are children of the test case result.

Input Arguments

resultObj — Test case result

object

Test case results object to get the iteration results from, specified as a `sltest.testmanager.TestCaseResult` object.

Output Arguments

iterArray — Iteration result

object array

Iteration result set, returned as an array of `sltest.testmanager.TestIterationResult` objects.

See Also

`sltest.testmanager.TestIterationResult`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

getIterations

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Get test iterations that belong to test case

Syntax

```
iterArray = getIterations(tcObj)  
iterArray = getIterations(tcObj, iterName)
```

Description

`iterArray = getIterations(tcObj)` get one or more test iteration objects that belong to the test case.

`iterArray = getIterations(tcObj, iterName)` get one or more test iteration objects with the specified name that belong to the test case.

Input Arguments

tcObj — Test case to get iteration from

object

Test case that you want to get the iteration from, specified as a `sltest.testmanager.TestCase` object.

iterName — Test iteration name

string

Test iteration name, specified as a string. This is an optional argument.

Example: 'Test Iteration 5'

Output Arguments

iterArray — Test iteration objects

object array

Test iterations that belong to the test case, returned as an array of `sltest.testmanager.TestIteration` objects.

Examples

- “Run Multiple Combinations of Tests Using Iterations”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestIteration`

Introduced in R2016a

getOutputRuns

Class: sltest.testmanager.TestCaseResult

Package: sltest.testmanager

Get test case simulation output results

Syntax

```
runArray = getOutputRuns(resultObj)
```

Description

`runArray = getOutputRuns(resultObj)` gets all of the test case simulation output results that are direct children of the test case results object.

Input Arguments

resultObj — Test case result
object

Test case results object to get results from, specified as a `sltest.testmanager.TestCaseResult` object.

Output Arguments

runArray — Simulation output results
object

Test case simulation output results, returned as a `Simulink.sdi.Run` object array.

See Also

`Simulink.sdi.Run`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015a

getOutputRuns

Class: sltest.testmanager.TestIterationResult

Package: sltest.testmanager

Get test iteration simulation output results

Syntax

```
runArray = getOutputRuns(resultObj)
```

Description

`runArray = getOutputRuns(resultObj)` gets all of the test iteration simulation output results that are direct children of the test iteration results object.

Input Arguments

resultObj — Test iteration result

object

Test iteration results object to get results from, specified as a `sltest.testmanager.TestIterationResult` object.

Output Arguments

runArray — Simulation output results

object

Test iteration simulation output results, returned as a `Simulink.sdi.Run` object array.

See Also

`sltest.testmanager.TestIterationResult`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

getParameterOverrides

Class: sltest.testmanager.ParameterSet

Package: sltest.testmanager

Get parameter overrides

Syntax

```
ovrs = getParameterOverrides
```

Description

`ovrs = getParameterOverrides` gets all of the parameter overrides in a parameter set and returns them as an array of parameter override objects, `sltest.testmanager.ParameterOverride`.

Output Arguments

ovrs — Parameter override object

object array

Parameter overrides that are in the parameter set object, returned as an array of `sltest.testmanager.ParameterOverride` objects.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getParameterSets

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Get test case parameter sets

Syntax

```
psets = getParameterSets(simulationIndex)
```

Description

`psets = getParameterSets(simulationIndex)` gets all of the parameter sets in a test case and returns them as an array of parameter set objects, `sltest.testmanager.ParameterSet`.

Input Arguments

simulationIndex — Test case simulation number

1 | 2

Simulation number that the parameter sets apply to, specified as an integer, 1 or 2. This setting applies to the simulation test case where there are two simulations. For baseline and simulation test cases, the simulation index is 1.

Output Arguments

psets — Parameter set object

object array

Parameter sets that belong to the test case, returned as an array of `sltest.testmanager.ParameterSet` objects.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getProperty

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Get test case property

Syntax

```
val = getProperty(propertyName, simulationIndex)
```

Description

`val = getProperty(propertyName, simulationIndex)` gets a test case property.

Input Arguments

propertyName — Test case property

string

Test suite property names, specified as a string. The properties are set using `thesltest.testmanager.TestCase.setProperty` method. The available properties are:

- 'Model' — name of the model to be tested
- 'SimulationMode' — simulation mode of the model during the test
- 'HarnessName' — harness name used for the test
- 'HarnessOwner' — harness owner name
- 'OverrideStartTime' — override start time
- 'StartTime' — start time override value of simulation
- 'OverrideStopTime' — override stop time
- 'StopTime' — stop time override value of simulation
- 'OverrideInitialState' — override initial state

- 'InitialState' — string evaluated to specify the initial state of the system under test
- 'PreloadCallback' — string evaluated before the model loads and any model callbacks
- 'PostloadCallback' — string evaluated after the system under test loads and PostLoadFcn callback completes
- 'PreStartRealTimeApplicationCallback' — string evaluated before the real-time application is started on target computer
- 'CleanupCallback' — string evaluated after simulation completes and all model callbacks have run
- 'UseSignalBuilderGroups' — use signal builder groups for test input
- 'SignalBuilderGroup' — signal builder group name to use
- 'OverrideModelOutputSettings' — override model output settings
- 'SaveOutput' — save simulation output
- 'SaveState' — save model states during simulation
- 'SaveFinalState' — save final state of simulation
- 'SignalLogging' — log signals
- 'DSMLogging' — log data store
- 'ConfigsetOverrideSetting' — value to determine override of configuration set
- 'ConfigsetName' — configuration set override name
- 'ConfigsetFileLocation' — path to a MAT-file that contains a configuration set object
- 'ConfigsetVarName' — name of the variable in ConfigsetFileLocation that is a configuration set
- 'IterationScript' — string evaluated for test case iteration script
- 'SimulationIndex' — determines which simulation a property applies to, applicable to the equivalence test case type
- 'FastRestart' — indicates if test iterations run using fast restart mode
- 'SaveBaselineRunInTestResult' — enable saving the baseline run used in the test case, saved in the test result
- 'LoadAppFrom' — location to load real-time application from
- 'TargetComputer' — target computer name

- 'TargetApplication' — target application name

simulationIndex — Test case simulation number

1 | 2

Simulation number that the property applies to, specified as an integer, 1 or 2. This setting applies to the simulation test case where there are two simulations. For baseline and simulation test cases, the simulation index is 1.

Output Arguments

val — Property content

string | logical | scalar

The content of the test case property, returned as a string, logical, or scalar value.

See Also

sltest.testmanager.TestCase.setProperty

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getProperty

Class: sltest.testmanager.TestSuite

Package: sltest.testmanager

Get test suite property

Syntax

```
val = getProperty(propertyName)
```

Description

`val = getProperty(propertyName)` gets a test suite property.

Input Arguments

propertyName — Test suite property

string

Test suite property names, specified as a string. The available properties are 'SetupCallback' and 'CleanupCallback'.

Output Arguments

val — Property content

string

The content of the test suite property, returned as a string.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getSignalCriteria

Class: sltest.testmanager.BaselineCriteria

Package: sltest.testmanager

Get signal criteria

Syntax

```
sigCriteria = getSignalCriteria
```

Description

`sigCriteria = getSignalCriteria` gets the list of the signal criteria in a baseline criteria set and returns them as an array of signal criteria objects, `sltest.testmanager.SignalCriteria`.

Output Arguments

sigCriteria — Signal criteria object

object array

Signal criteria that are in the baseline criteria object, returned as an array of `sltest.testmanager.SignalCriteria` objects.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getSignalCriteria

Class: sltest.testmanager.EquivalenceCriteria

Package: sltest.testmanager

Get signal criteria

Syntax

```
sigCriteria = getSignalCriteria
```

Description

`sigCriteria = getSignalCriteria` gets the list of the signal criteria in an equivalence criteria set and returns them as an array of signal criteria objects, `sltest.testmanager.SignalCriteria`.

Output Arguments

sigCriteria — Signal criteria object

object array

Signal criteria that are in the equivalence criteria object, returned as an array of `sltest.testmanager.SignalCriteria` objects.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getTestCaseByName

Class: sltest.testmanager.TestSuite

Package: sltest.testmanager

Get test case object by name

Syntax

```
tc = getTestCaseByName(name)
```

Description

tc = getTestCaseByName(name) returns a test case with the specified name.

Input Arguments

name — Test suite name

string

The name of the test case within a test suite object, specified as a string. If the name does not match a test case, then the function returns an empty test case object.

Output Arguments

tc — Test case object

object

Test case, returned as an sltest.testmanager.TestCase object. If the name does not match a test case, then the function returns an empty test case object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getTestCaseResults

Class: sltest.testmanager.ResultSet

Package: sltest.testmanager

Get test case results object

Syntax

```
testCaseResultArray = getTestCaseResults(resultObj)
```

Description

`testCaseResultArray = getTestCaseResults(resultObj)` gets all of the test case results that are direct children of the results set object.

Input Arguments

resultObj — Results set

object

Results set object to get results from, specified as a `sltest.testmanager.ResultSet` object.

Output Arguments

testCaseResultArray — Test case results

object array

Test case results, returned as an array of `sltest.testmanager.TestCase` objects. The function returns objects that are direct children of the results set object.

Examples

Get Test Result Set Data

Get results from running a test file with `sltest.testmanager.run`.

```
result = sltest.testmanager.run;  
testCaseResultArray = getTestCaseResults(result);
```

- “Automate Tests Programmatically”

Introduced in R2015a

getTestCaseResults

Class: sltest.testmanager.TestSuiteResult

Package: sltest.testmanager

Get test case results object

Syntax

```
testCaseResultArray = getTestCaseResults(resultObj)
```

Description

`testCaseResultArray = getTestCaseResults(resultObj)` gets all of the test case results that are direct children of the test suite results object.

Input Arguments

resultObj — Results set

object

Test suite results object to get test case results from, specified as a `sltest.testmanager.TestSuiteResult` object.

Output Arguments

testCaseResultArray — Test case results

object array

Test case results objects, returned as an array of `sltest.testmanager.TestCaseResult` objects. The function returns objects that are direct children of the test suite results object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015a

getTestCases

Class: sltest.testmanager.TestSuite

Package: sltest.testmanager

Get test cases at first level of test suite

Syntax

```
tcArray = getTestCases
```

Description

`tcArray = getTestCases` returns an array of test case objects that are at the first level of the specified test suite.

Output Arguments

tcArray — Test case array

object array

Array of test cases at the first level of the specified test suite, returned as an array of `sltest.testmanager.TestCase` objects.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getTestSuiteByName

Class: sltest.testmanager.TestFile

Package: sltest.testmanager

Get test suite object by name

Syntax

```
ts = getTestSuiteByName(name)
```

Description

ts = getTestSuiteByName(name) returns a test suite with the specified name.

Input Arguments

name — Test suite name

string

The name of the test suite within the test file, specified as a string. If the name does not match a test suite, then the function returns an empty test suite object.

Output Arguments

ts — Test suite object

object

Test suite, returned as an sltest.testmanager.TestSuite object. If the name does not match a test suite, then the function returns an empty test suite object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getTestFileResults

Class: sltest.testmanager.ResultSet

Package: sltest.testmanager

Get test suite results object

Syntax

```
testFileResultArray = getTestFileResults(resultObj)
```

Description

`testFileResultArray = getTestFileResults(resultObj)` gets all of the test file results that are direct children of the results set object.

Input Arguments

resultObj — Results set

object

Results set object to get test file results from, specified as a `sltest.testmanager.ResultSet` object.

Output Arguments

testSuiteResultArray — Test suite results

object array

Test file results objects, returned as an array of `sltest.testmanager.TestFileResult` objects. The function returns objects that are direct children of the results set input object.

See Also

`sltest.testmanager.ResultSet` | `sltest.testmanager.TestSuiteResult` | `sltest.testmanager.TestCaseResult`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

getTestSuiteByName

Class: sltest.testmanager.TestSuite

Package: sltest.testmanager

Get test suite object by name

Syntax

```
ts = getTestSuiteByName(name)
```

Description

ts = getTestSuiteByName(name) returns a test suite with the specified name.

Input Arguments

name — Test suite name

string

The name of the test suite within a test suite object, specified as a string. If the name does not match a test suite, then the function returns an empty test suite object.

Output Arguments

ts — Test suite object

object

Test suite, returned as an sltest.testmanager.TestSuite object. If the name does not match a test suite, then the function returns an empty test suite object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getTestSuites

Class: sltest.testmanager.TestFile

Package: sltest.testmanager

Get test suites at first level of test file

Syntax

```
tsArray = getTestSuites
```

Description

`tsArray = getTestSuites` returns an array of test suite objects that are at the first level of the test file.

Output Arguments

tsArray — Test suite array

object array

Array of test suites at the first level of the test file, returned as an array of `sltest.testmanager.TestSuite` objects.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getTestSuites

Class: sltest.testmanager.TestSuite

Package: sltest.testmanager

Get test suites at first level of test suite

Syntax

```
tsArray = getTestSuites
```

Description

`tsArray = getTestSuites` returns an array of test suite objects that are at the first level of the specified test suite.

Output Arguments

tsArray — Test suite array

object array

Array of test suites at the first level of the specified test suite, returned as an array of `sltest.testmanager.TestSuite` objects.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

getTestSuiteResults

Class: sltest.testmanager.ResultSet

Package: sltest.testmanager

Get test suite results object

Syntax

```
testSuiteResultArray = getTestSuiteResults(resultObj)
```

Description

`testSuiteResultArray = getTestSuiteResults(resultObj)` gets all of the test suite results that are direct children of the results set object.

Input Arguments

resultObj — Results set

object

Results set object to get results from, specified as a `sltest.testmanager.ResultSet` object.

Output Arguments

testSuiteResultArray — Test suite results

object array

Test suite results objects, returned as an array of `sltest.testmanager.TestSuite` objects. The function returns objects that are direct children of the results set object.

Examples

Get Test Result Set Data

Get results from running a test file with `sltest.testmanager.run`.

```
result = sltest.testmanager.run;  
testCaseResultArray = getTestCaseResults(result);  
testSuiteResultArray = getTestSuiteResults(result);
```

- “Automate Tests Programmatically”

Introduced in R2015a

getTestSuiteResults

Class: sltest.testmanager.TestFileResult

Package: sltest.testmanager

Get test suite results object

Syntax

```
testSuiteResultArray = getTestSuiteResults(tfObj)
```

Description

`testSuiteResultArray = getTestSuiteResults(tfObj)` gets all of the test suite results that are direct children of the test file results object.

Input Arguments

tfObj — Test file results

object

Test file results object to get test suite results from, specified as a `sltest.testmanager.TestFileResult` object.

Output Arguments

testSuiteResultArray — Test suite results

object array

Test suite results objects, returned as an array of `sltest.testmanager.TestSuiteResult` objects. The function returns objects that are direct children of the test suite results input object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

getTestSuiteResults

Class: sltest.testmanager.TestSuiteResult

Package: sltest.testmanager

Get test suite results object

Syntax

```
testSuiteResultArray = getTestSuiteResults(resultObj)
```

Description

`testSuiteResultArray = getTestSuiteResults(resultObj)` gets all of the test suite results that are direct children of the test suite results object.

Input Arguments

resultObj — Test suite results
object

Test suite results object to get test suite results from, specified as a `sltest.testmanager.TestSuiteResult` object.

Output Arguments

testSuiteResultArray — Test suite results
object array

Test suite results objects, returned as an array of `sltest.testmanager.TestSuiteResult` objects. The function returns objects that are direct children of the test suite results input object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015a

getVerifyRun

Class: sltest.testmanager.TestCaseResult

Package: sltest.testmanager

Get test case verify statement

Syntax

```
dataset = getVerifyRun(resultObj)
```

Description

`dataset = getVerifyRun(resultObj)` gets the verify statement dataset from a test case result. Verify statements are made in the **Test Sequence** or **Test Assessment** blocks in the system under test.

Input Arguments

resultObj — Test case result
object

Test case results object to get verify statement dataset from, specified as a `sltest.testmanager.TestCaseResult` object.

Output Arguments

dataset — Verify statement dataset
object

Test case verify statement dataset, returned as a `Simulink.sdi.Run` object.

See Also

`Simulink.sdi.Run`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

getVerifyRun

Class: sltest.testmanager.TestIterationResult

Package: sltest.testmanager

Get test iteration verify statement

Syntax

```
dataset = getVerifyRun(resultObj)
```

Description

`dataset = getVerifyRun(resultObj)` gets the verify statement dataset from a test iteration result. Verify statements are made in the **Test Sequence** or **Test Assessment** blocks in the system under test.

Input Arguments

resultObj — Test iteration result

object

Test iteration results object to get verify statement dataset from, specified as a `sltest.testmanager.TestIterationResult` object.

Output Arguments

dataset — Verify statement dataset

object

Test iteration verify statement dataset, returned as a `Simulink.sdi.Run` object.

See Also

`Simulink.sdi.Run`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2016a

layoutReport

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

Incorporates parts of report into one document

Syntax

layoutReport(obj)

Description

layoutReport(obj) incorporates all of the report parts into one document. The report is divided into three main parts: title page, table of contents, and the main body.

This method also calls:

- sltest.testmanager.TestResultReport.addTitlePage
- sltest.testmanager.TestResultReport.addReportTOC
- sltest.testmanager.TestResultReport.addReportBody

Input Arguments

obj — Test report object

sltest.testmanager.TestResultReport object

Test report, specified as a sltest.testmanager.TestResultReport object.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport`

Introduced in R2016a

map

Class: sltest.testmanager.TestInput

Package: sltest.testmanager

Maps test input to System Under Test

Syntax

```
map(mode,customFunction)
```

Description

map(mode,customFunction) maps the test input data to the System Under Test.

Input Arguments

mode — Mapping mode

0 | 1 | 2 | 3 | 4

Mapping mode, specified as an integer.

- 0 — Block name
- 1 — Block path
- 2 — Signal name
- 3 — Port order (index)
- 4 — Custom

For more information on mapping modes, see “Map Root Inport Signal Data”.

customFunction — Custom mapping function name

string

Name of function used for custom mapping, specified as a string. This argument is valid only when **mode** is set to 4, custom.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

plotOneSignalToFile

Class: sltest.testmanager.TestResultReport

Package: sltest.testmanager

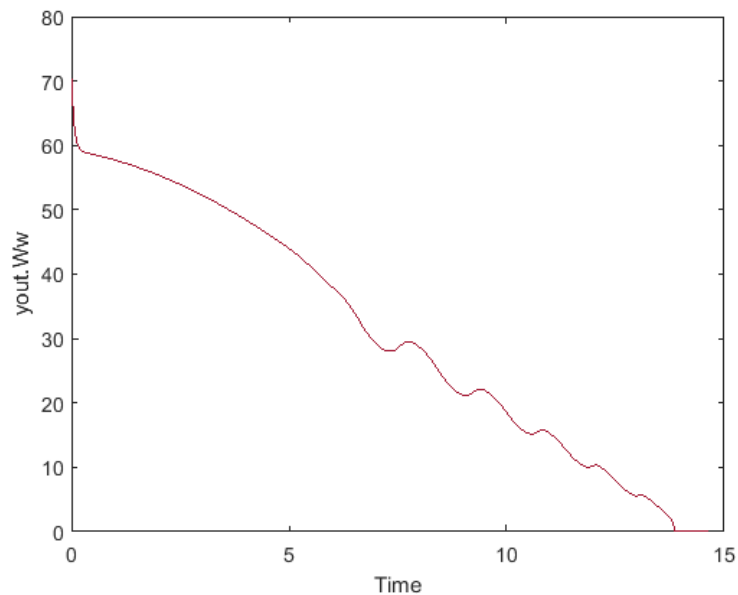
Save signal plot to file

Syntax

```
plotOneSignalToFile(obj,filePath,onesig,isComparison)
```

Description

`plotOneSignalToFile(obj,filePath,onesig,isComparison)` saves a signal plot to a PNG image file. If the signal plot is a comparison, then the baseline signal, the difference, and the tolerance are plotted in the same plot.



Input Arguments

obj — Test report object

`sltest.testmanager.TestResultReport` object

Test report, specified as a `sltest.testmanager.TestResultReport` object.

filePath — Image file path

string

File path and name of the image you want to save, specified as a string.

onesig — Result signal

`sltest.testmanager.ReportUtility.Signal` object

The result signal, specified as a `sltest.testmanager.ReportUtility.Signal` object.

isComparison — Comparison indicator

`true` | `false`

Flag to indicate whether the signal is from a comparison run or not, specified as a Boolean, `true` or `false`.

Examples

- “Export Test Results and Generate Reports”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestResultReport`

Introduced in R2016a

remove

Class: sltest.testmanager.BaselineCriteria

Package: sltest.testmanager

Remove baseline criteria

Syntax

```
remove
```

Description

`remove` removes the baseline criteria from a test case. The baseline criteria object is empty after a call to this function.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

remove

Class: sltest.testmanager.EquivalenceCriteria

Package: sltest.testmanager

Remove equivalence criteria

Syntax

remove

Description

remove removes the equivalence criteria from a test case. The equivalence criteria object is empty after a call to this function.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

remove

Class: sltest.testmanager.ParameterOverride

Package: sltest.testmanager

Remove parameter override

Syntax

```
remove
```

Description

`remove` removes the parameter override from the parameter set. The parameter override object is empty after a call to this function.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

remove

Class: sltest.testmanager.ParameterSet

Package: sltest.testmanager

Remove parameter set

Syntax

remove

Description

remove removes the parameter set from a test case. The parameter set object is empty after a call to this function.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

remove

Class: sltest.testmanager.SignalCriteria

Package: sltest.testmanager

Remove signal criteria

Syntax

```
remove
```

Description

`remove` removes signal criteria from the baseline or equivalence criteria set. The signal criteria object is empty after a call to this function.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

remove

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Remove test case

Syntax

remove

Description

remove removes the test case. The test case object is empty after a call to this function. All parameter overrides, baseline criteria, or equivalence criteria associated with the test case become invalid.

Examples

Remove Test Case

```
% Create test file
tf = sltest.testmanager.TestFile('C:\MATLAB\test_file.mldatx');

% Create test suite
ts = sltest.testmanager.TestSuite(tf,'My Test Suite');

% Create test case
testcase = sltest.testmanager.TestCase(ts,'equivalence',...
'Eq Test Case');

% Remove the test case
testcase.remove;
```

- “Automate Tests Programmatically”

Introduced in R2015b

remove

Class: sltest.testmanager.TestInput

Package: sltest.testmanager

Remove test input

Syntax

remove

Description

remove removes the test input from a test case. The test input object is empty after a call to this function.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

remove

Class: sltest.testmanager.TestSuite

Package: sltest.testmanager

Remove test suite

Syntax

remove

Description

remove removes the test suite. The test suite object is empty after a call to this function.

Examples

Remove Test Suite

```
% Create test file
tf = sltest.testmanager.TestFile('C:\MATLAB\test_file.mldatx');

% Create test suite
ts = sltest.testmanager.TestSuite(tf, 'My Test Suite');

% Remove the test suite
ts.remove;
```

- “Automate Tests Programmatically”

Introduced in R2015b

run

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Run test case

Syntax

```
resultObj = run(tcObj)
```

Description

`resultObj = run(tcObj)` runs the test case and returns a results set object.

Input Arguments

tcObj — Test case object
object

Test case you want to run, specified as an `sltest.testmanager.TestCase` object.

Output Arguments

resultObj — Results set object
object

Test results, returned as a results set object, `sltest.testmanager.ResultSet`.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

run

Class: sltest.testmanager.TestFile

Package: sltest.testmanager

Run test cases in test file

Syntax

```
resultObj = run(tfObj)
```

Description

`resultObj = run(tfObj)` runs all of the enabled test cases in the test file and returns a results set object.

Input Arguments

tfObj — Test file object

object

Test file with the test cases you want to run, specified as an `sltest.testmanager.TestFile` object.

Output Arguments

resultObj — Results set object

object

Test results, returned as a results set object, `sltest.testmanager.ResultSet`.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

run

Class: sltest.testmanager.TestSuite

Package: sltest.testmanager

Run test cases in test suite

Syntax

```
resultObj = run(tsObj)
```

Description

`resultObj = run(tsObj)` runs all of the enabled test cases in the test suite and returns a results set object.

Input Arguments

tsObj — Test suite object

object

Test suite with the test cases you want to run, specified as an `sltest.testmanager.TestSuite` object.

Output Arguments

resultObj — Results set object

object

Test results, returned as a results set object, `sltest.testmanager.ResultSet`.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

saveToFile

Class: sltest.testmanager.TestFile

Package: sltest.testmanager

Save test file

Syntax

```
saveToFile(filePath)
```

Description

`saveToFile(filePath)` saves the test file to the specified file path.

Input Arguments

filePath — File path

string

The file path to save the test file at, specified as a string. If no file path is given, the functions saves the test file at the original location.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

setModelParam

Class: sltest.testmanager.TestIteration

Package: sltest.testmanager

Set model parameter for iteration

Syntax

```
setModelParam(obj,modelObject,paramName,value)
```

Description

setModelParam(obj,modelObject,paramName,value) sets a model parameter for the test iteration object.

Input Arguments

obj — Test iteration object

object

Test iteration that you want to set the model parameter, specified as a sltest.testmanager.TestIteration object.

modelObject — Name or handle of a model or block

string | handle

Handle or name of a model or block, specified as a numeric handle or a string. A numeric handle must be a scalar. You can also set parameters of lines and ports, but you must use numeric handles to specify them.

Example: 'vdp/Fcn'

paramName — Model or block parameter name

string

Model or block parameter name, specified as the comma-separated pair consisting of the parameter name, specified as a string, and the value, specified in the format determined

by the parameter type. Case is ignored for parameter names. Value strings are case sensitive. Values are often strings, but they can also be numeric, arrays, and other types. Many block parameter values are specified as strings, but two exceptions are these parameters: `Position`, specified as a vector, and `UserData`, which can be any data type.

For more information on parameter name and value pairs, see the function reference page for `set_param`.

Example: `'Solver', 'ode15s'`

Data Types: `char`

Examples

Override Gain Value

```
setModelParam(obj,[sltest_bdroot '/Mu'], 'Gain', '1000')
```

- “Run Multiple Combinations of Tests Using Iterations”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestIteration` | `set_param`

Introduced in R2016a

setProperty

Class: sltest.testmanager.TestCase

Package: sltest.testmanager

Set test case property

Syntax

```
setProperty(Name,Value)
```

Description

setProperty(Name,Value) sets a test case property.

Input Arguments

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 (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'StopTime',100

'Model' — System Under Test model name

empty string (default)

The model name in the System Under Test section, specified as a string.

Example: 'sldemo_absbrake'

'SimulationMode' — Simulation mode

empty string (default) | 'Normal' | 'Accelerator' | 'Rapid Accelerator' | 'Software-in-the-Loop (SIL)' | 'Processor-in-the-Loop (PIL)'

The simulation mode of the model or harness, specified as a string. To return to the default model settings, specify an empty string, ''.

Example: 'SimulationMode', 'Rapid Accelerator'

'OverrideStartTime' — Override model start time

false (default) | true

Indicate if the test case overrides the model start time, specified as a Boolean, true or false.

'StartTime' — Model start time

0 (default) | scalar

Model start time, specified as a scalar value.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

'OverrideStopTime' — Override model stop time

false (default) | true

Indicate if the test case overrides the model start time, specified as a Boolean, true or false.

'StopTime' — Model stop time

10 (default) | scalar

Model stop time, specified as a scalar value.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

'OverrideInitialState' — Override model initial state

false (default) | true

Indicate if the test case overrides the model initial state, specified as a Boolean, true or false.

'InitialState' — Model initial state

empty string (default) | string

Model initial state from a workspace variable, specified as a string.

'HarnessName' — Test harness name

empty string (default) | string

Name of a test harness to use in the System Under Test section, specified as a string.

'HarnessOwner' — Test harness owner name

empty string (default) | string

Name of the test harness owner, specified as a string.

'UseSignalBuilderGroup' — Override Signal Builder group

false (default) | true

Indicate if the test case overrides the model and uses a different Signal Builder group in the Inputs section, specified as a Boolean, `true` or `false`.

'SignalBuilderGroup' — Signal Builder group name

empty string (default) | string

Signal Builder group name, specified as a string. To return to the default model settings, specify an empty string, `''`.

'OverrideModelOutputSettings' — Override model output settings

false (default) | true

Indicate if the test case overrides the model settings under the Outputs section, specified as a Boolean, `true` or `false`.

'SaveOutput' — Override saving output

false (default) | true

Indicate if the test case overrides saving model output, specified as a Boolean, `true` or `false`.

'SaveState' — Save output state values

false (default) | true

Indicate if the test case is set to save output state values, specified as a Boolean, `true` or `false`.

'SignalLogging' — Log signals

true (default) | false

Indicate if the test case is set to log signals marked for logging in the model, specified as a Boolean, `true` or `false`.

'DSMLogging' — Log Data Store variables

`true` (default) | `false`

Indicate if the test case is set to log Data Store variables, specified as a Boolean, `true` or `false`.

'SaveFinalState' — Save final state

`false` (default) | `true`

Indicate if the test case is set to store final state values, specified as a Boolean, `true` or `false`.

'SimulationIndex' — Equivalence test case simulation

1 (default) | 2

Simulation number that the property applies to, specified as an integer, 1 or 2. This setting applies to the simulation test case.

'ConfigSetOverrideSetting' — Configuration setting override

1 (default) | 2 | 3

Override the configuration settings, specified as an integer.

- 1 — No override
- 2 — Use a named configuration set in the model
- 3 — Use a configuration set specified in a file

'ConfigSetName' — Configuration set name

empty string (default) | string

Name of the configuration setting in a model, specified as a string.

'ConfigSetVarName' — Configuration set variable name

empty string (default) | string

Variable name in a configuration set file, specified as a string.

'ConfigSetFileLocation' — Configuration set file path

empty string (default) | string

File name and path of the configuration set, specified as a string.

'PreloadCallback' — Pre-load callback script

string

Pre-load callback script, specified as a string.

'PostloadCallback' — Post-load callback script

string

Post-load callback script, specified as a string.

'CleanupCallback' — Cleanup callback script

string

Test-case level cleanup callback script, specified as a string. The function deletes any existing callback script and replaces it with the specified string.

Example: 'clear a % clear value from workspace'

'PreStartRealTimeApplicationCallback' — Real-time pre-start callback

string

String evaluated before the real-time application is started on the target computer, specified as a string. For more information on real-time testing, see “Test Models in Real Time”.

'IterationScript' — Iteration script

string

Iteration script evaluated to create test case iterations, specified as a string. For more information about test iteration scripts, see “Run Multiple Combinations of Tests Using Iterations”.

'FastRestart' — Run iterations using fast restart

false (default) | true

Indicate if the test iterations run using fast restart mode, specified as a Boolean, true or false.

'SaveBaselineRunInTestResult' — Save baseline in test result

false (default) | true

Indicate if the test case saves the baseline used in the test result after test execution, specified as a Boolean, `true` or `false`.

'LoadAppFrom' — Application location

1 (default) | 2 | 3

Location from which to load the application, specified as an integer. This property is available only in real-time test cases.

- 1 — Model
- 2 — Target application
- 3 — Target computer

For more information on real-time testing, see “Test Models in Real Time”.

'TargetComputer' — Target computer name

string

Name of the target computer, specified as a string. This property is available only in real-time test cases. For more information on real-time testing, see “Test Models in Real Time”.

'TargetApplication' — Target application name and path

string

Name and path of the target application, specified as a string. This property is available only in real-time test cases. For more information on real-time testing, see “Test Models in Real Time”.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

setProperty

Class: sltest.testmanager.TestSuite

Package: sltest.testmanager

Set test suite property

Syntax

```
setProperty(Name,Value)
```

Description

setProperty(Name,Value) sets a test suite property.

Input Arguments

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 (' '). You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: 'SetupCallback', 'a = 300; % set nominal value'

'SetupCallback' — Setup callback script

empty (default) | string

Test-suite level setup callback script, specified as a string. The function deletes any existing callback script and replaces it with the specified string.

Example: 'a = 300; % set nominal value'

'CleanupCallback' — Cleanup callback script

empty (default) | string

Test-suite level cleanup callback script, specified as a string. The function deletes any existing callback script and replaces it with the specified string.

Example: `'clear a % clear value from workspace'`

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

setTestParam

Class: sltest.testmanager.TestIteration

Package: sltest.testmanager

Set test case parameter

Syntax

```
setTestParam(obj, Name, Value)
```

Description

`setTestParam(obj, Name, Value)` sets the test iteration parameter with additional options specified by one or more `Name, Value` pair arguments.

Input Arguments

obj — Test iteration object

object

Test iteration object that you want to apply the test parameter to, specified as a `sltest.testmanager.TestIteration` object.

Name-Value Pair Arguments

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

'ParameterSet' — Parameter set

Current test case setting (default) | string

Parameter set name, specified as the comma-separated pair consisting of `'ParameterSet'` and a string. Parameter set overrides are setup in the **Parameter Overrides** section of the test case.

Example: 'ParameterSet', 'Param Set 1'

'Baseline' — Baseline criteria dataset

Current test case setting (default) | string

Baseline criteria dataset name, specified as the comma-separated pair consisting of 'Baseline' and a string. Baseline criteria datasets are setup in the **Baseline Criteria** section of the test case. It is available only for baseline test cases.

Example: 'Baseline', 'BaselineSet_High'

'Input' — External input

Current test case setting (default) | string

External input name, specified as the comma-separated pair consisting of 'Input' and a string. External input overrides are setup in the **Inputs** section of the test case.

Example: 'Input', 'Run1'

'ConfigSet' — Configuration setting

Current test case setting (default) | string

Configuration setting name, specified as the comma-separated pair consisting of 'ConfigSet' and a string. Configuration setting overrides are setup in the **Configuration Settings Overrides** section of the test case.

Example: 'ConfigSet', 'Solver 3'

'SignalBuilderGroup' — Signal Builder group

Current test case setting (default) | string

Signal Builder group input name, specified as the comma-separated pair consisting of 'SignalBuilderGroup' and a string. Signal Builder group overrides are setup in the **Inputs** section of the test case.

Example: 'SignalBuilderGroup', 'Acceleration'

'PreLoadFcn' — Pre-load callback script

Current test case setting (default) | string

Pre-load callback script, specified as the comma-separated pair consisting of 'PreLoadFcn' and a string. The pre-load callback script is setup in the **Callbacks** section of the test case.

'PostLoadFcn' — Post-load callback script

Current test case setting (default) | string

Post-load callback script, specified as the comma-separated pair consisting of 'PostLoadFcn' and a string. The post-load callback script is setup in the **Callbacks** section of the test case.

'PreStartRealTimeApplicationFcn' — Pre-start real-time application callback script

Current test case setting (default) | string

Pre-start real-time application callback script, specified as the comma-separated pair consisting of 'PreStartRealTimeApplicationFcn' and a string. The pre-start real-time application callback script is setup in the **Callbacks** section of the test case.

'CleanupFcn' — Cleanup callback script

Current test case setting (default) | string

Cleanup callback script, specified as the comma-separated pair consisting of 'CleanupFcn' and a string. The cleanup callback script is setup in the **Callbacks** section of the test case.

'Description' — Description

Current test case setting (default) | string

Test description text, specified as the comma-separated pair consisting of 'Description' and a string. The Description is setup in the **Description** section of the test case.

Example: 'Description', 'Test the autopilot controller for wind gusts'

Examples

Set Test Parameter

```
setTestParam(obj, 'Description', 'Test the autopilot controller for wind gusts');
```

- “Run Multiple Combinations of Tests Using Iterations”
- “Automate Tests Programmatically”

See Also

sltest.testmanager.TestIteration

Introduced in R2016a

setVariable

Class: sltest.testmanager.TestIteration

Package: sltest.testmanager

Set model variable override

Syntax

```
setVariable(obj, 'Name', varName, 'Source', srcName, 'Value', value)
```

Description

`setVariable(obj, 'Name', varName, 'Source', srcName, 'Value', value)` sets a model variable override for the test iteration. Specify the `sltest.testmanager.TestIteration` object, and then specify the variable name, source, and override value. The method overrides the variable in the test iteration and does not permanently change the model variable.

Input Arguments

obj — Test iteration object

object

The test iteration you want to apply the override to, specified as a `sltest.testmanager.TestIteration` object.

varName — Variable name

string

Name of the variable you want to override, specified as a string.

srcName — Variable source

'base workspace' | 'model workspace' | 'mask workspace' | 'data dictionary' | model element paths

The source of the variable to override, specified as a string. For non-real-time test cases, the possible sources can be the base workspace, model workspace, mask workspace, and

data dictionary. For real-time test cases, the possible sources are model element paths, which might be an empty string.

value — Override value

scalar

Value of the variable you want to override, specified as a scalar value.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Examples

Set Variable in Base Workspace

```
setVariable(obj, 'Name', 'g', 'Source', 'base workspace', 'Value', 33);
```

- “Run Multiple Combinations of Tests Using Iterations”
- “Automate Tests Programmatically”

See Also

`sltest.testmanager.TestIteration`

Introduced in R2016a

Blocks — Alphabetical List

Test Assessment

Specify test assessments, steps, and actions in tabular format

Description



Use this block to define test assessments in a tabular series of steps. Like the **Test Sequence** block, the **Test Assessment** block uses MATLAB as the action language.

Test Sequence Editor

Default Configuration

Double-click the **Test Assessment** block to open the **Test Sequence Editor**, displaying the default test step layout. The top-level step of a **Test Assessment** block uses **When decomposition** transition by default. To change the transition type to a standard transition, right-click the top-level step and clear **When decomposition**.

Step	Transition	Next Step	Description
<input type="checkbox"/> step_1			
<input type="checkbox"/> step_1_1 when t < 5			
<input type="checkbox"/> step_1_2			

Add step after • Add sub-step

Adding, Deleting, and Editing Test Steps

To add a test step, right-click a step in the editor and select **Add step before** or **Add step after**. Select **Add sub-step** to create test steps in a lower hierarchy level. Select **Delete step** to delete the selected step.

To cut or copy a test step, select the step and type **Ctrl+X** or **Ctrl+C**. To paste the test step, select another test step, type **Ctrl+V**, and select whether to paste before or after the step.

Actions and Transitions

The step names are the first lines in the **Step** column. To enter new step names, overwrite the default names. You can undo and redo test sequence edits using the undo and redo buttons on the toolbar. The Test Sequence Editor offers common Simulink keyboard shortcuts, such as save, update diagram, and start simulation.

A test step consists of actions and transitions. Use test step actions to define assessments and signals. Use transitions to define conditions at which the test sequence exits the test step and enters another test step. For a When decomposition sequence, **when** statements accompany the test step name and define the active step.

Initialize the output signals in the first test step. Outputs are automatically initialized when you use a **Test Sequence** block in a test harness that compiles the main model.

Test Sequence Hierarchy and Transitions

You can arrange test sequences in a hierarchy of parent steps and substeps. Substeps can activate only when the parent step is active. For each sequence level, you can define the transition type as a standard transition or a When decomposition.

Standard Transition

In a standard transition, the default step is the first step listed in the sequence. The sequence progresses according to the transition conditions and next steps. To create a sequence using standard transitions:

- Add new steps to the sequence.
- Define outputs and assessments in the **Step** cell. For example, this code sets `on_off` to `false` and verifies that the `FanOn` signal is `true`.

```
on_off = false;
verify(FanOn == true);
```

- For each step that requires a transition, hover over the **Transition** cell and click **Add transition**. Define the step exit conditions in the transition. For example, this code transitions to another step after the current step has been active for 20 seconds.

```
after(20,sec)
```

- Choose the next test step in the **Next Step** cell.


Step	Transition	Next Step
<pre>initialize on_off = false; Tproj = single(0);</pre>	1. true	Normal_on_off ▼
<div style="border-left: 1px solid black; padding-left: 5px;"> <pre>Normal_on_off end_test = 0;</pre> </div>		
<div style="border-left: 1px solid black; padding-left: 5px;"> <pre>On on_off = true;</pre> </div>	1. FanOn == true	Wait ▼
<div style="border-left: 1px solid black; padding-left: 5px;"> <pre>Wait on_off = false; verify(FanOn == true,... 'Simulink:verify_scenario1',... 'Fan should be active');</pre> </div>	1. after(20,sec)	Off ▼
<div style="border-left: 1px solid black; padding-left: 5px;"> <pre>Off on_off = true;</pre> </div>	1. FanOn == false	End ▼
<pre>End on_off = false; end_test = 1;</pre>		

When Decomposition

In a **When decomposition** sequence, steps activate based on a signal condition defined on the same line as the test step name, preceded by the **when** operator. For example, a step named `OverSpeed2` activates when the signal `gear` is 2:

OverSpeed2 when gear == 2

At each time step, the when statements evaluate from top to bottom, and the first step with a valid statement executes.


A **When decomposition** requires a parent step. To change to a **When decomposition** sequence, right-click the parent step and select **When decomposition**. The parent step displays the When decomposition icon . Add substeps to define the when conditions.

The last step of a When decomposition handles conditions which make no other when condition valid. Do not include a when statement in the final step in the sequence. For example, this sequence includes the step **Else** with no when statement.

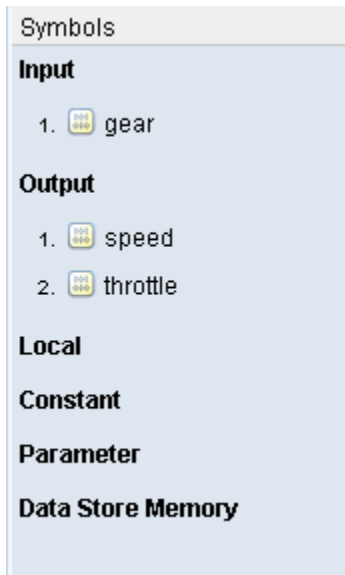
Step	Transition	Next Step
<div style="border: 1px solid #ccc; padding: 5px;"> <div style="display: flex; align-items: center;"> ▢ ↘ HighLevelAssessment </div> <pre style="margin-top: 5px;"> assert(speed >= 0); assert(throttle >= 0); assert(throttle <= 100); assert(gear > 0); </pre> </div>		
<div style="border-left: 1px solid #ccc; border-right: 1px solid #ccc; padding: 5px;"> <div style="display: flex; align-items: center;"> — <div> <p>OverSpeed3 when gear == 3</p> <pre>verify(speed <= 90, 'SLTest:Gear3_overspeed')</pre> </div> </div> </div>		
<div style="border-left: 1px solid #ccc; border-right: 1px solid #ccc; padding: 5px;"> <div style="display: flex; align-items: center;"> — <div> <p>OverSpeed2 when gear == 2</p> <pre>verify(speed <= 50, 'SLTest:Gear2_overspeed')</pre> </div> </div> </div>		
<div style="border-left: 1px solid #ccc; border-right: 1px solid #ccc; padding: 5px;"> <div style="display: flex; align-items: center;"> — <div> <p>OverSpeed5 when gear == 3</p> <pre>verify(speed <= 30, 'SLTest:Gear1_overspeed')</pre> </div> </div> </div>		
<div style="border-left: 1px solid #ccc; border-right: 1px solid #ccc; padding: 5px;"> <div style="display: flex; align-items: center;"> — <div> <p>Else</p> </div> </div> </div>		

Input, Output, and Data Management

Manage inputs, outputs, and data objects using the **Symbols** sidebar of the Test

Sequence Editor. Click the symbols sidebar button  on the toolbar to show or hide the sidebar. To add a symbol, mouse over the symbol type and click **Add**. To edit or delete a data symbol, mouse over the data symbol and click **Edit** or **Delete**.

If you add a symbol to the test sequence block, you can access that symbol from test steps at any hierarchy level. For information on using messages, see “Test Sequence Action and Transition Operations”.



Symbol Type	Description	Procedure for Adding
Input	Inputs can be data or messages.	Click Add in the sidebar and enter the input name.
Output	Outputs can be data, messages, or function calls.	Click Add in the sidebar and enter the output name.
Local	Local data entries are available inside the test sequence block in which they are defined.	Add a local variable in the sidebar and initialize the local variable in the first test step.
Constant	Constants are read-only data entries available inside the test sequence block in which they are defined.	Add a constant in the sidebar. Click Edit and enter the constant value in the dialog box, in the Initial Value field.

Symbol Type	Description	Procedure for Adding
Parameter	Parameters are data available inside and outside the Test Sequence block.	Using the Model Explorer, add a parameter in the workspace of the model containing the Test Sequence block. Then add the parameter name to the Parameter symbols.
Data Store Memory	Data Store Memory entries are available inside and outside the Test Sequence block.	Using the Model Explorer, add a Simulink.signal entry in the workspace of the model containing the Test Sequence block. Alternatively, add a Data Store Memory block to the model. Then add the data store memory name to the Data Store Memory symbols.

See Also

“Syntax for Test Sequences and Assessments” | “Debug a Test Sequence”

Related Examples

- “Test Sequence Action and Transition Operations”
- “Generate Function-Based Test Signals”
- “Assess Simulation Using Logical Statements”

Introduced in R2016a

Test Sequence

Specify test steps, actions, and assessments in tabular format

Description



Use this block to define a test sequence using a tabular series of steps. Like the `Test Assessment` block, the `Test Sequence` block uses MATLAB as the action language.

Test Sequence Editor

Default Configuration

Double-click the `Test Sequence` block to open the `Test Sequence Editor`, displaying the default test step layout.

Step	Transition	Next Step
step_1	1. <i>true</i>	step_2 ▼
step_2		

You can undo and redo test sequence edits using the undo and redo buttons on the toolbar. The `Test Sequence Editor` offers many Simulink keyboard shortcuts, such as save, update diagram, and start simulation.

Adding, Deleting, and Editing Test Steps

To add a test step, right-click a step in the editor and select **Add step before** or **Add step after**. Select **Add sub-step** to create test steps in a lower hierarchy level. Select **Delete step** to delete the selected step.

To cut or copy a test step, select the step and type **Ctrl+X** or **Ctrl+C**. To paste the test step, select another test step, type **Ctrl+V**, and select whether to paste before or after the step.

Actions and Transitions

The step names are the first lines in the **Step** column. To enter new step names, overwrite the default names. You can undo and redo test sequence edits using the undo and redo buttons on the toolbar. The Test Sequence Editor offers common Simulink keyboard shortcuts, such as save, update diagram, and start simulation.

A test step consists of actions and transitions. Use test step actions to define assessments and signals. Use transitions to define conditions at which the test sequence exits the test step and enters another test step. For a When decomposition sequence, when statements accompany the test step name and define the active step.

Initialize the output signals in the first test step. Outputs are automatically initialized when you use a **Test Sequence** block in a test harness that compiles the main model.

Test Sequence Hierarchy and Transitions

You can arrange test sequences in a hierarchy of parent steps and substeps. Substeps can activate only when the parent step is active. For each sequence level, you can define the transition type as a standard transition or a When decomposition.

Standard Transition

In a standard transition, the default step is the first step listed in the sequence. The sequence progresses according to the transition conditions and next steps. To create a sequence using standard transitions:

- Add new steps to the sequence.
- Define outputs and assessments in the **Step** cell. For example, this code sets `on_off` to `false` and verifies that the `FanOn` signal is `true`.

```
on_off = false;
verify(FanOn == true);
```

- For each step that requires a transition, hover over the **Transition** cell and click **Add transition**. Define the step exit conditions in the transition. For example, this code transitions to another step after the current step has been active for 20 seconds.

```
after(20,sec)
```

- Choose the next test step in the **Next Step** cell.


Step	Transition	Next Step
<pre>initialize on_off = false; Tproj = single(0);</pre>	1. true	Normal_on_off ▼
<div style="border-left: 1px solid black; padding-left: 5px;"> <pre>Normal_on_off end_test = 0;</pre> </div>		
<div style="border-left: 1px solid black; padding-left: 5px;"> <pre>On on_off = true;</pre> </div>	1. FanOn == true	Wait ▼
<div style="border-left: 1px solid black; padding-left: 5px;"> <pre>Wait on_off = false; verify(FanOn == true,... 'Simulink:verify_scenario1',... 'Fan should be active');</pre> </div>	1. after(20,sec)	Off ▼
<div style="border-left: 1px solid black; padding-left: 5px;"> <pre>Off on_off = true;</pre> </div>	1. FanOn == false	End ▼
<pre>End on_off = false; end_test = 1;</pre>		

When Decomposition

In a **When decomposition** sequence, steps activate based on a signal condition defined on the same line as the test step name, preceded by the **when** operator. For example, a step named `OverSpeed2` activates when the signal `gear` is 2:

OverSpeed2 when gear == 2

At each time step, the when statements evaluate from top to bottom, and the first step with a valid statement executes.


A **When decomposition** requires a parent step. To change to a **When decomposition** sequence, right-click the parent step and select **When decomposition**. The parent step displays the When decomposition icon . Add substeps to define the when conditions.

The last step of a When decomposition handles conditions which make no other when condition valid. Do not include a when statement in the final step in the sequence. For example, this sequence includes the step **Else** with no when statement.

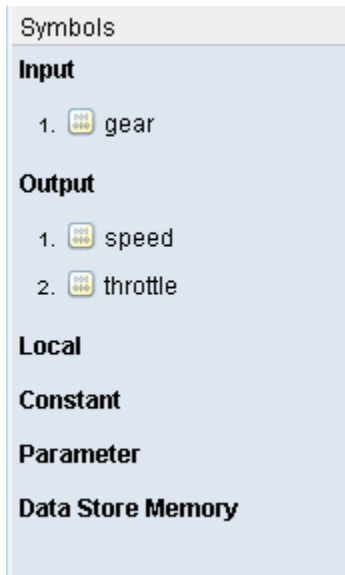
Step	Transition	Next Step
<div style="border: 1px solid #ccc; padding: 5px;"> <div style="display: flex; align-items: center;"> ▢ ↘ HighLevelAssessment </div> <pre style="margin-top: 5px;"> assert(speed >= 0); assert(throttle >= 0); assert(throttle <= 100); assert(gear > 0); </pre> </div>		
<div style="border-left: 1px solid #ccc; border-right: 1px solid #ccc; padding: 5px;"> <div style="display: flex; align-items: center;"> — <div> <p>OverSpeed3 when gear == 3</p> <pre>verify(speed <= 90, 'SLTest:Gear3_overspeed')</pre> </div> </div> </div>		
<div style="border-left: 1px solid #ccc; border-right: 1px solid #ccc; padding: 5px;"> <div style="display: flex; align-items: center;"> — <div> <p>OverSpeed2 when gear == 2</p> <pre>verify(speed <= 50, 'SLTest:Gear2_overspeed')</pre> </div> </div> </div>		
<div style="border-left: 1px solid #ccc; border-right: 1px solid #ccc; padding: 5px;"> <div style="display: flex; align-items: center;"> — <div> <p>OverSpeed5 when gear == 3</p> <pre>verify(speed <= 30, 'SLTest:Gear1_overspeed')</pre> </div> </div> </div>		
<div style="border-left: 1px solid #ccc; border-right: 1px solid #ccc; padding: 5px;"> <div style="display: flex; align-items: center;"> — <div> <p>Else</p> </div> </div> </div>		

Input, Output, and Data Management

Manage inputs, outputs, and data objects using the **Symbols** sidebar of the Test

Sequence Editor. Click the symbols sidebar button  on the toolbar to show or hide the sidebar. To add a symbol, mouse over the symbol type and click **Add**. To edit or delete a data symbol, mouse over the data symbol and click **Edit** or **Delete**.

If you add a symbol to the test sequence block, you can access that symbol from test steps at any hierarchy level. For information on using messages, see “Test Sequence Action and Transition Operations”.



Symbol Type	Description	Procedure for Adding
Input	Inputs can be data or messages.	Click Add in the sidebar and enter the input name.
Output	Outputs can be data, messages, or function calls.	Click Add in the sidebar and enter the output name.
Local	Local data entries are available inside the test sequence block in which they are defined.	Add a local variable in the sidebar and initialize the local variable in the first test step.
Constant	Constants are read-only data entries available inside the test sequence block in which they are defined.	Add a constant in the sidebar. Click Edit and enter the constant value in the dialog box, in the Initial Value field.

Symbol Type	Description	Procedure for Adding
Parameter	Parameters are data available inside and outside the Test Sequence block.	Using the Model Explorer, add a parameter in the workspace of the model containing the Test Sequence block. Then add the parameter name to the Parameter symbols.
Data Store Memory	Data Store Memory entries are available inside and outside the Test Sequence block.	Using the Model Explorer, add a Simulink.signal entry in the workspace of the model containing the Test Sequence block. Alternatively, add a Data Store Memory block to the model. Then add the data store memory name to the Data Store Memory symbols.

See Also

“Syntax for Test Sequences and Assessments” | “Debug a Test Sequence”

Related Examples

- “Test Sequence Action and Transition Operations”
- “Generate Function-Based Test Signals”
- “Assess Simulation Using Logical Statements”

Introduced in R2015a

