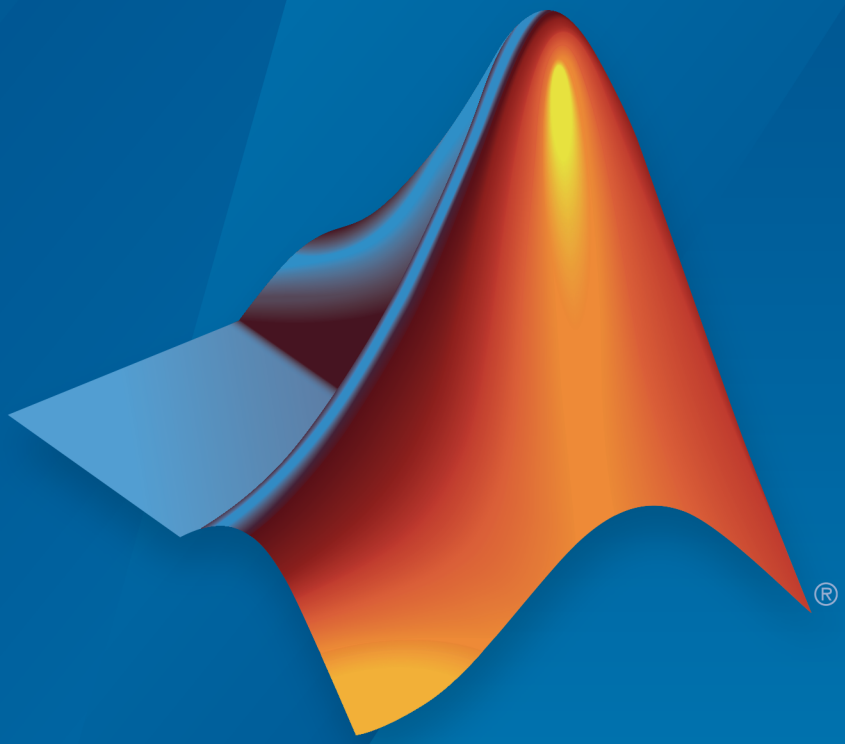
Simulink® Test™

Reference



MATLAB® & SIMULINK®

R2015b

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

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

Phone: 508-647-7000

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

Simulink® Test™ Reference

Revision History

March 2015	Online Only	New for Version 1.0 (Release 2015a)
September 2015	Online Only	Revised for Version 1.1 (Release 2015b)

Contents

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(harnessOwner,harnessName)
```

Description

`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

`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.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.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
```

See Also

`sltest.harness.check` | `sltest.harness.clone` | `sltest.harness.close` | `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.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)
harness List = 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 RegExp 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
```

'ActiveOnly' — Active harness search option
'off' (default) | 'on'

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

Example:

```
'ActiveOnly','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.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.

```
sltest.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] = sltest.import.sldvData...
('./sldv_output/ShiftLogic/ShiftLogic_sldvdata.mat',...
'TestHarnessName','DatafileHarness','TestFileName','CoverageTests',...
'CreateHarness',false)
```

Open the test harness.

```
sltest.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.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 objects.

`targetObj` — Target test file or test suite

object

The destination test file or test suite to copy to, specified as an 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 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 object.

Introduced in R2015b

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.

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 object, returned as a `sltest.testmanager.TestFile` object.

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 objects.

`targetObj` — Target test file or test suite

object

The destination test file or test suite to move to, specified as an 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 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);
```

Input Arguments

`resultObj` — Results set object

`sltest.testmanager.ResultSet` object

Results set object to get results from, specified as a 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 criteria plots
`false` (default) | `true`

Choose to include the criteria comparison plots defined under baseline or equivalence criteria 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 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.

Introduced in R2015a

sltest.testmanager.run

Run all test files in the Simulink Test manager

Syntax

```
resultObj = sltest.testmanager.run
```

Description

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

Output Arguments

`resultObj` — Results set object

`sltest.testmanager.ResultSet` object

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

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

Classes — Alphabetical List

sltest.testmanager.BaselineCriteria class

Package: sltest.testmanager

Baseline criteria object

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.

`AbsTol` — Absolute tolerance value

scalar

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

`RelTol` — 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.EquivalenceCriteria class

Package: sltest.testmanager

Equivalence criteria object

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

Parameter override object

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

any value

Value of the parameter override.

`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

Parameter set object

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.

`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.

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;
```

Introduced in R2015a

sltest.testmanager.SignalCriteria class

Package: sltest.testmanager

Signal criteria object

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.

`AbsTol` — Absolute tolerance value

scalar

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

`RelTol` — Relative tolerance value

scalar

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

`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

Test case object

Description

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

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.

Properties

`Name` — Test case name
string

Name of the test case.

`Description` — Test case description
string

Test case description text.

`Enabled` — Test execution indicator
`true | false`

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

`ReasonForDisabling` — Test case disabling reason
string

You can provide a descriptive reason why the test case is disabled. This property only appears if the `Enabled` property is set to `false`.

`TestFile` — Parent test file
object

Test file object that is the parent of the test case.

`TestPath` — Test hierarchy
string

Test file, test suite, and test case hierarchy.

`Parent` — Parent object
object

Test suite object that is the parent of the specified test case.

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'
    Description: ''
        Enabled: 1
       TestFile: [1x1 sltest.testmanager.TestFile]
       TestPath: 'test_file > My Test Suite > Equivalence Test Case'
       TestType: 'equivalence'
         Parent: [1x1 sltest.testmanager.TestSuite]
```

- “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.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` — 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 result set.

`TestCasePath` — Hierarchy path in the result set

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.

Methods

Introduced in R2015a

sltest.testmanager.TestFile class

Package: sltest.testmanager

Test file object

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.

Input Arguments

`filePath` — File name and path

string

The file name and path of the test file, specified as a string.

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

`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.

`FilePath` — File path and name

string

File path and name of the test file.

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

`Name` — Test file name

string

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

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: 0
```

- “Automate Tests Programmatically”

See Also

sltest.testmanager.TestCase | sltest.testmanager.TestSuite

Introduced in R2015b

sltest.testmanager.TestInput class

Package: sltest.testmanager

Test input object

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.

`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.TestSuite class

Package: sltest.testmanager

Test suite object

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, specified as an object. The parent object can be a test file or a test suite.

`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.

`Description` — Test suite description
string

Test suite description text.

`Enabled` — Test 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` — Test suite disabling reason
string

You can provide a descriptive reason why the test suite is disabled. This property only appears if the `Enabled` property is set to `false`.

`TestFile` — Parent test file
object

Test file object that is the parent of the test suite.

`TestPath` — Test hierarchy
string

Test file and test suite hierarchy.

`Parent` — Parent object
object

Test file or test suite object that is the parent of the specified test suite.

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

- “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.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

`TestFilePath` — Test file path

string

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

`TestSuitePath` — Hierarchy path in the result set

string

The hierarchy path in the parent result set.

`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.

Methods

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. 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 a 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.

Input Arguments

`filPath` — 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 a object.

Related Examples

- “Automate Tests Programmatically”

Introduced in R2015b

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.

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 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.

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 object.

Related Examples

- "Automate Tests Programmatically"

Introduced in R2015b

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. 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 a 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. 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 a 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)
```

Description

`tc = createTestCase(type,name)` creates a new test case within the test suite. You can specify the test case name and type: baseline, equivalence, and simulation.

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.

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'
    Description: ''
        Enabled: 1
       TestFile: [1x1 sltest.testmanager.TestFile]
       TestPath: 'test_file > My Test Suite > My Baseline Test'
       TestType: 'baseline'
         Parent: [1x1 sltest.testmanager.TestSuite]
```

- “Automate Tests Programmatically”

Introduced in R2015b

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

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.

Output Arguments

`baselines` — Baseline criteria object

object array

Baseline criteria that are in the baseline test case, returned as an array of objects.

Related Examples

- "Automate Tests Programmatically"

Introduced in R2015b

getComparisonRun

Class: sltest.testmanager.TestCaseResult
Package: sltest.testmanager

Get test case comparison results

Syntax

```
runArray = getComparisonRun(resultObj)
```

Description

`runArray = getComparisonRun(resultObj)` gets all of the test case comparison results that belong to the test case results object.

Input Arguments

`resultObj` — Results set object

object

Test case results object to get results from, specified as a `sltest.testmanager.TestCaseResult` object.

Output Arguments

`runArray` — Test case results

object array

Test case comparison results, returned as an object array.

Introduced in R2015a

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. 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 a 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.

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 objects.

Related Examples

- "Automate Tests Programmatically"

Introduced in R2015b

getOutputRuns

Class: sltest.testmanager.TestCaseResult
Package: sltest.testmanager

Get test case output results

Syntax

```
runArray = getOutputRuns(resultObj)
```

Description

`runArray = getOutputRuns(resultObj)` gets all of the test case output results that are direct children of the test case results object.

Input Arguments

`resultObj` — Results set object

object

Results set object to get results from, specified as a `sltest.testmanager.TestCaseResult` object.

Output Arguments

`runArray` — Test case results

object

Contains the test case output results that are a direct child of the results set object.

Introduced in R2015a

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.

Output Arguments

`ovrs` — Parameter override object

object array

Parameter overrides that are in the parameter set object, returned as an array of 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.

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 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 available properties are specified in the method.

`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 | Boolean | scalar

The content of the test case property, returned as a string, Boolean, or scalar value.

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.

Output Arguments

`sigCriteria` — Signal criteria object

object array

Signal criteria that are in the baseline criteria object, returned as an array of 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.

Output Arguments

`sigCriteria` — Signal criteria object

object array

Signal criteria that are in the equivalence criteria object, returned as an array of 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

object

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

Output Arguments

`testCaseResultArray` — Test case results object

object array

Test case results objects, returned as an object array. 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);
```

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

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

object array

Test case results objects, returned as an object array. The function returns objects that are direct children of the test suite results object.

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

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

object

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

Output Arguments

`testSuiteResultArray` — Test suite results object

object array

Test suite results objects, returned as an object array. 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.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

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

object array

Test suite results objects, returned as an object array. The function returns objects that are direct children of the test suite results input object.

Introduced in R2015a

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 "Import and Map Root-Level Inport 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

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

setProperty

Class: sltest.testmanager.TestCase
Package: sltest.testmanager

Set test case 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: `'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'`

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

Blocks — Alphabetical List

Test Sequence

Specify test steps, actions, and assessments in tabular format.

Description

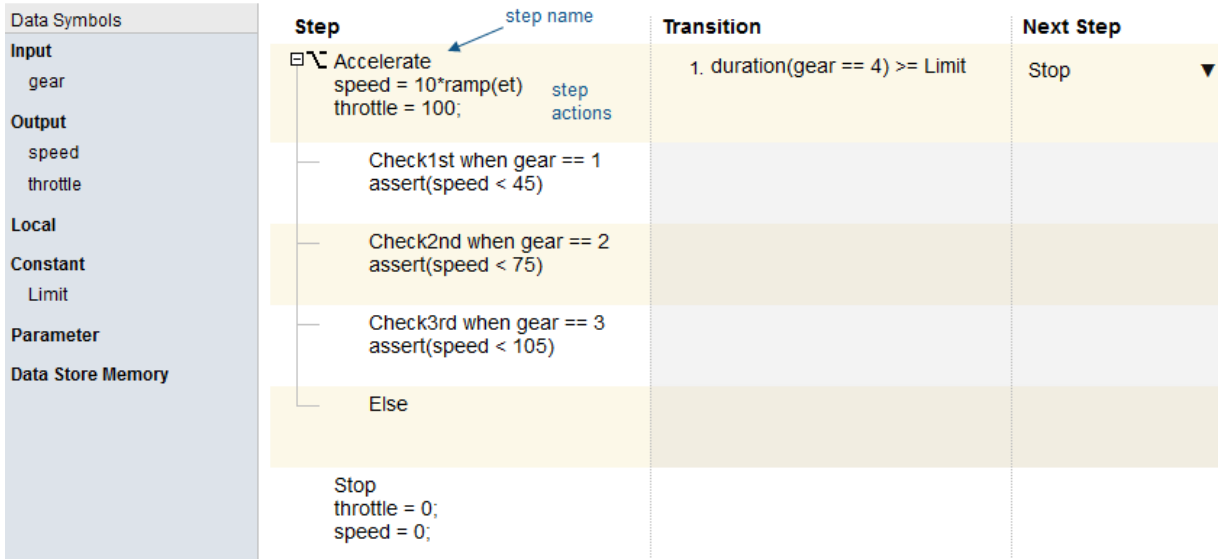
Use this block to define a test sequence using a tabular series of steps. The Test Sequence block uses MATLAB as the action language.

Test Sequence Editor

Double-click the Test Sequence block to open the Test Sequence Editor, displaying the default test step layout. The step names are the first lines in the **Step** column. Set the step names by overwriting the default names.

Step	Transition	Next Step
step_1	1. *true*	step_2 ▼
step_2		

You can undo and redo test sequence edits using the undo and redo buttons on the toolbar. While working in the test sequence editor you can use many normal Simulink keyboard shortcuts, such as save, update diagram, and start simulation.



Test Sequence Actions and Transitions

A test step consists of one or more actions and one or more transitions defined using MATLAB as the action language. You use step actions to define the test signals going to the component under test, and test step transitions to define the condition at which the test sequence executes another test step.

Initialize the block output signals in the first step of the Test Sequence block. Outputs are automatically initialized when you use a Test Sequence block in a test harness that compiles the main model.

To add a step, right-click a test 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.

Test Sequence Hierarchy

You can arrange test sequences in a hierarchy of parent and child sequences. Child steps are only active when the parent step is active. For each sequence level, you can define the transition type:

Standard Transition

For each sequence level, the default step entered is the first step listed in the sequence. For each step:

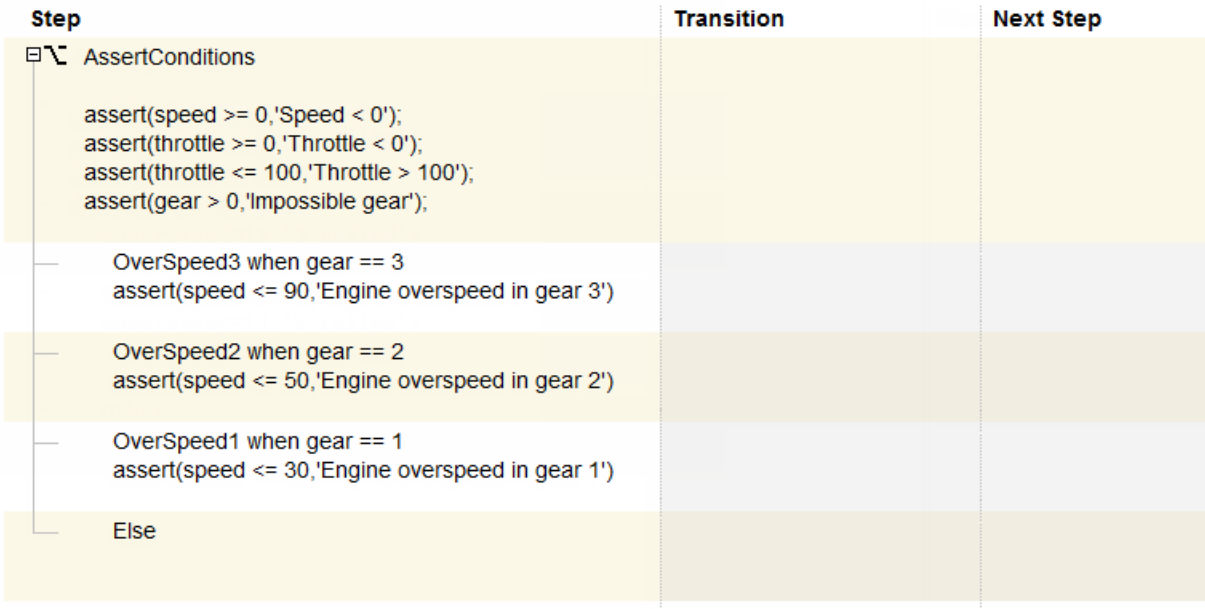
- Define the outputs of the step in the Step column.
- Define the exit condition from that step in the Transition column.
- Choose the next test step in the Next Step column.

When Decomposition

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.

In a **When decomposition** sequence, steps execute based on the signal condition defined in the **Step** column preceded by the `when` operator. At each time step, the `when` conditions evaluate from top to bottom, and the first step with a matching condition executes.

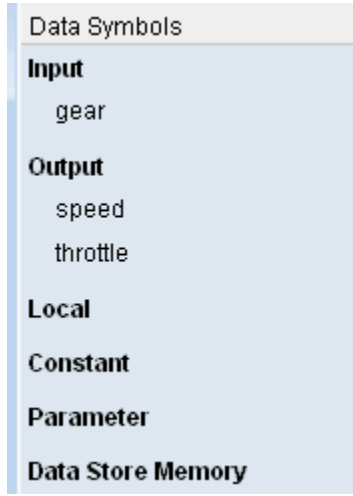
Do not include the `when` operator in the final step in the sequence. This step handles conditions which do not match the other **When decomposition** steps.



Input, Output, and Data Management

Manage inputs, outputs, and data objects using the **Data Symbols** sidebar of the Test Sequence Editor. Click the data symbols button on the toolbar to show or hide the sidebar. To add a data symbol, mouse over the data 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 data symbol to the test sequence block, you can access that data symbol from test steps at any hierarchy level.



Data Symbol Type	Description	Procedure for Adding
Input	Test Sequence block inputs.	Click **Add** in the **Data Symbols** pane and enter the input name.
Output	Test Sequence block outputs.	Click **Add** in the **Data Symbols** pane and enter the output name.
Local	Local variables are available inside the test sequence block in which they are defined.	Add a local variable in the **Data Symbols** pane 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 **Data Symbols** pane and set the constant value in the Data dialog box. Click **Edit** and enter the constant value in **Initial Value**.
Parameter	Parameters are available inside and outside the Test Sequence block.	Using the Model Explorer, add a Simulink parameter in the workspace of the model containing the Test

Data Symbol Type	Description	Procedure for Adding
		Sequence block. Then add the parameter name to the **Data Symbols > Parameter** list.
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 Symbols > Data Store Memory** list.

Related Examples

- "Evaluate Temporal or Signal Conditions in a Test Sequence Transition"
- "Generate Function-Based Test Signals"
- "Test a Model Component Using Signal Functions"
- "Test Downshift Points of a Transmission Controller"
- "Debug a Test Sequence"

Introduced in R2015a