Simulink® Test™

Reference

MATLAB®&SIMULINK®

R2015a

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)

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 always returns `CheckResult` = `true`.

For a virtual subsystem harness, the function always 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`

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`

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','S
```

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
`false` (default) | `true`

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.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`

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.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`

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.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`

sltest.harness.find

Find test harnesses in model

Syntax

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

Description

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

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

Examples

Use 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',

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.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`

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`

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`

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`

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`

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`

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.

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.

Introduced in R2015a

sltest.testmanager.load

Load a test file in the Simulink Test manager

Syntax

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

Description

`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.

Introduced in R2015a

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`.

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, specified 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.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

Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB documentation.

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.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

Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB documentation.

Introduced in R2015a

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

Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB documentation.

Introduced in R2015a

Methods — Alphabetical List

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 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.TestCaseResult` object.

Output Arguments

`runArray` — Test case results
object

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

Introduced in R2015a

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 results set 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

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 or `sltest.testmanager.TestSuiteResult` object.

Output Arguments

`testCaseResultArray` — Test case results object
object

Contains the test case results objects that are a direct child 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)
```

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 results set object.

Input Arguments

`resultObj` — Results set object
object

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

Output Arguments

`testCaseResultArray` — Test case results object
object

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

Introduced in R2015a

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 or `sltest.testmanager.TestSuiteResult` object.

Output Arguments

`testSuiteResultArray` — Test suite results object

object

Contains the test suite results objects that are a direct child 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)
```

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 results set object.

Input Arguments

`resultObj` — Results set object
object

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

Output Arguments

`testSuiteResultArray` — Test suite results object
object

Contains the test suite results objects that are a direct child of the results set object.

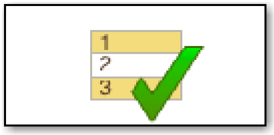
Introduced in R2015a

Blocks — Alphabetical List

Test Sequence

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

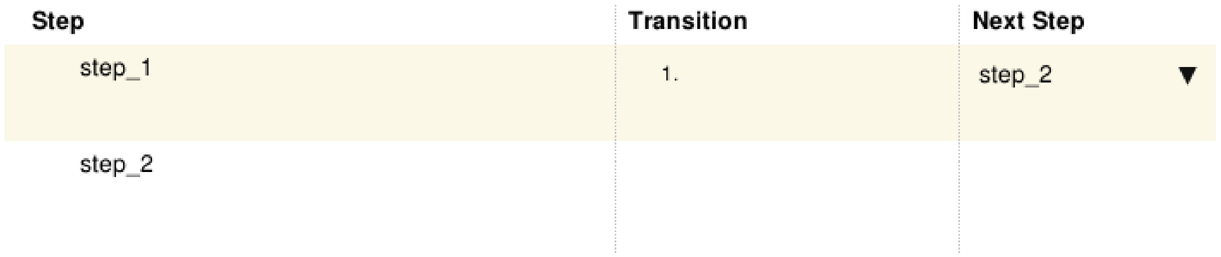
Description



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

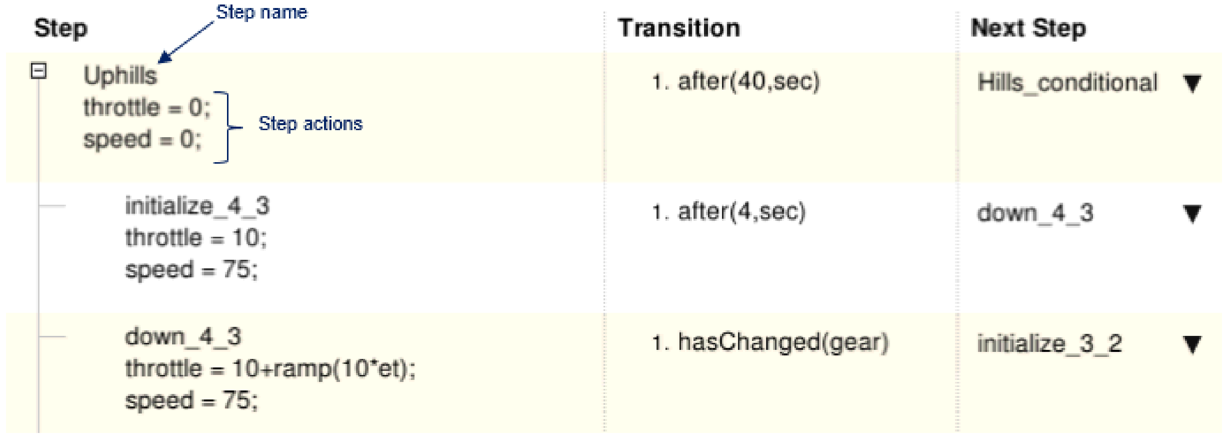
Test Sequence Editor

Double-click the Test Sequence block to open the Test Sequence Editor, displaying the default test step layout.



Step	Transition	Next Step
step_1	1.	step_2 ▼
step_2		

The step names are the first lines in the **Step** column. Set the step names by overwriting the default names.



Step	Transition	Next Step

Uphills
throttle = 0;
speed = 0;
	1. after(40,sec)	Hills_conditional

initialize_4_3
throttle = 10;
speed = 75;
	1. after(4,sec)	down_4_3

down_4_3
throttle = 10+ramp(10*et);
speed = 75;
	1. hasChanged(gear)	initialize_3_2

Test Sequence Actions and Transitions

A test step consists of one or more actions and one or more transitions defined using MATLAB 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.

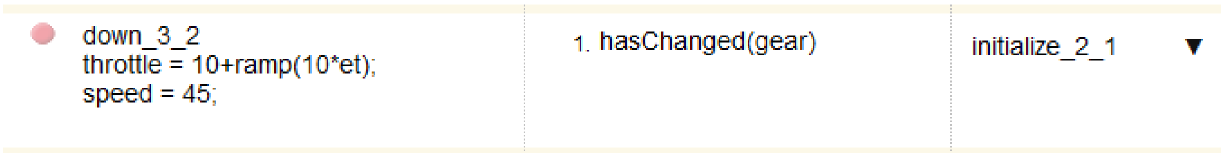
In the first step of the Test Sequence block, initialize the output signals. Outputs are automatically initialized when you use a Test Sequence block in a test harness that compiles the main model.

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.

Breakpoints

Add a breakpoint to stop simulation at the entrance to a particular test step. Right-click a test step, and select **Break while executing step**. A breakpoint stops simulation after the transition condition to that step, and before any commands or outputs in the step execute.

When you add a breakpoint to a step, the editor displays a breakpoint marker.



down_3_2
throttle = 10+ramp(10*et);
speed = 45;

1. hasChanged(gear)

initialize_2_1

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:

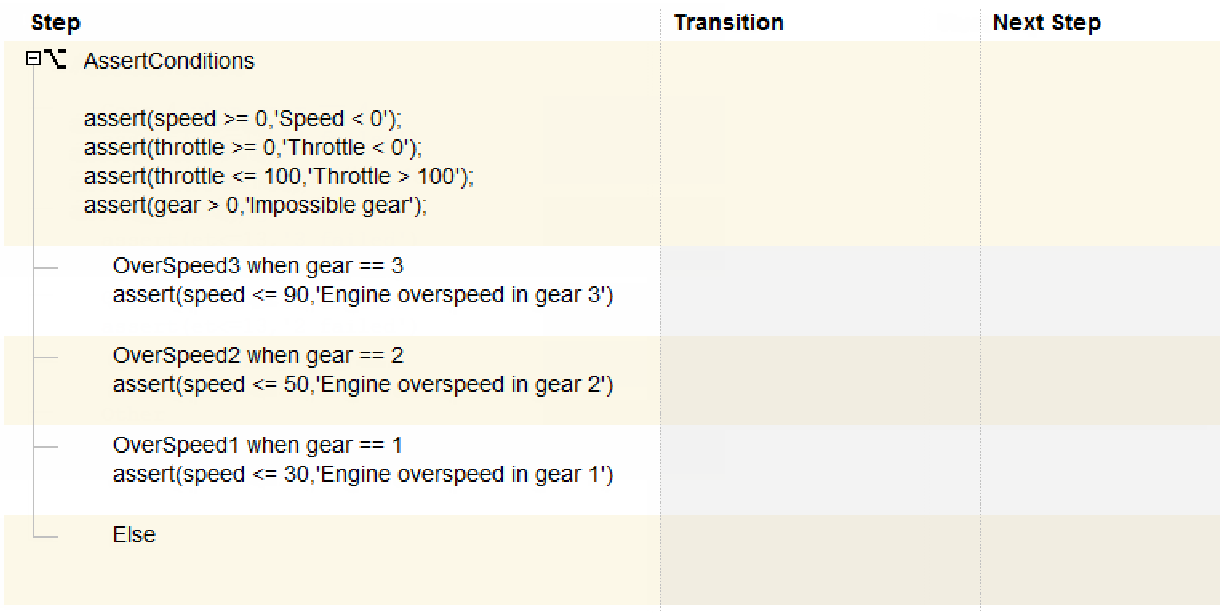
- Define the outputs of the step in the Step column.
- Define the exit condition from that step in the Transition column.
- Choose the next test step in the Next Step column.

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.



Step	Transition	Next Step

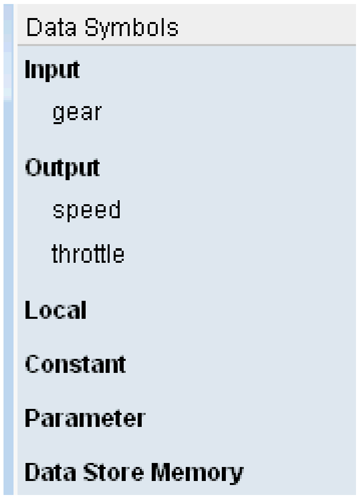
AssertConditions
 	
assert(speed >= 0,'Speed < 0');
assert(throttle >= 0,'Throttle < 0');
assert(throttle <= 100,'Throttle > 100');
assert(gear > 0,'Impossible gear');		

OverSpeed3 when gear == 3 assert(speed <= 90,'Engine overspeed in gear 3')		
OverSpeed2 when gear == 2 assert(speed <= 50,'Engine overspeed in gear 2')		
OverSpeed1 when gear == 1 assert(speed <= 30,'Engine overspeed in gear 1')		
Else		

Input, Output, and Data Management

Manage inputs, outputs, and data objects using the **Data Symbols** pane of the Test Sequence Editor. 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.


Data Symbols

Input

gear

Output

speed

throttle

Local

Constant

Parameter

Data Store Memory

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.

Test Sequence Operators and Functions

Operators for Absolute Time Temporal Logic

Operator	Syntax	Description
`after`	`after(n,TimeUnits)`	Returns true if `n` specified units of time have elapsed since the beginning of the current test step. The timer resets if the sequence exits the test step.
`before`	`before(n,TimeUnits)`	Returns true until `n` specified units of time elapse since the beginning of the current test step. The timer resets if the sequence exits the test step.
`duration`	`ElapsedTime = durati`	`duration` uses temporal logic and signal conditions to return the time in seconds since SignalCondition became true, within the period of the step in which the instance of `duration` is used.

Operator	Syntax	Description

`elapsed`
Abbreviation: `et`
	`elapsed(TimeUnits)`
	
	`et(TimeUnits)`
		Returns the elapsed time of the test step in the units specified. Specifying no time units returns the value in seconds.

`getSimulationTim`
Abbreviation: `t`
	`getSimulationTime(Ti`
	
	`t(TimeUnits)`
		Returns the elapsed time of the simulation in the units specified. Specifying no time units returns the value in seconds.

Syntax in the table uses these arguments:

`TimeUnits`

The units of time.

Value: `sec|msec|usec`

Examples:

```
msec
```

`SignalCondition`

Logical expression of the condition to trigger the temporal operator. Variables used in the signal condition must be inputs, parameters, or constants in the Test Sequence block.

Examples:

```
u > 0
x <= 1.56
```

Signal Output Functions

You can use these mathematical functions to generate output for test signals.

Note: These functions are not equivalent to signal generators. Consider the effect of scaling, rounding, and other approximations in your application.

Syntax	Description	Additional Information
`square(x)`	Represents a square wave output of period 1 and	`square(x)` ≡ `4*floor(x)-2*floor(2*x)+1`

Syntax	Description	Additional Information
	range `-1` to `1`, returning the value of the square wave at time `x`. Within the period `0 <= x < 1`, `square(x)` returns the value `1` for `0 <= x < 0.5`, and `0` for `0.5 <= x < 1`.	
`sawtooth(x)`	Represents a sawtooth wave output of period `1` and range `-1` to `1`, returning the value of the sawtooth wave at time `x`. Within the period `0 <= x < 1`, `sawtooth(x)` increases.	`sawtooth(x)` ≡ `2*(x-floor(x))-1`
`triangle(x)`	Represents a triangle wave output of period `1` and range `-1` to `1`, returning the value of the triangle wave at time `x`. Within the period `0 <= x < 0.5`, `triangle(x)` increases.	`triangle(x)` ≡ `2*abs(sawtooth(x+0.5))-1`
`ramp(x)`	Represents a ramp signal of slope `1`, returning the value of the ramp at time `x`.	`ramp(x)` ≡ `x`
`heaviside(x)`	Represents a heaviside step signal, returning `0` for `x < 0` and `1` for `x >= 0`.	`heaviside(x)` ≡ `x < 0 ? 0 : 1`

Syntax	Description	Additional Information
`latch(x)`	Returns the current value of `x` and holds that value for the duration of the test step.	For example, in `TestStep2`, `latch(x)` holds the value of `x` upon entry of `TestStep2`, and generates a ramp signal descending from that value.

Step	Transition	Next Step
`TestStep1` `x = ramp(t`	`after(5,se`	`TestStep2`
`TestStep2` `x = latch(`		

Related Examples

- "Test a Model Component Using Signal Functions"
- "Test Downshift Points of a Transmission Controller"